

QUIZ MASTER - V2

Quizo

Project Report by:

Yuvraj

22f3302815

Modern Application Development -2 Project

Jan - 2025 Term

Author

Name: **Yuvraj**

Roll Number: **22f3002815**

Student Email ID: 22f3002815@ds.study.iitm.ac.in

About me: Coding is my passion and developing websites have always been my interest. Learning new languages is one of the most important thing I feel which contributes to my journey in this coding world.

Description

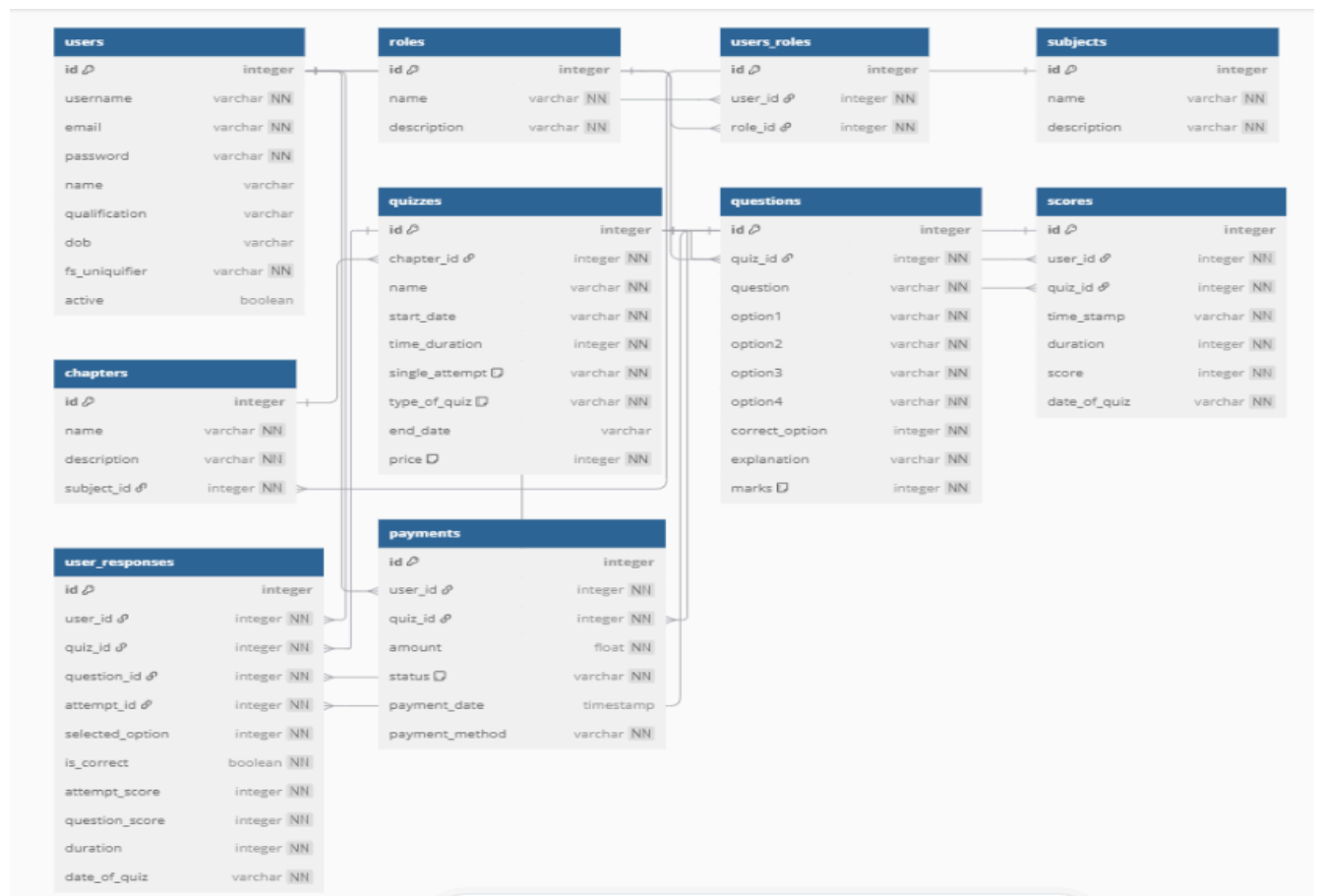
Quizo is an online quiz platform where users can attempt various quizzes on different topics of their interest. The user friendly interface and effective performance of the webiste makes it more interconnected with the user. Admins can set questions for the quizzes, can also create new subjects and chapters.

Technologies Used

- **Python:** Primary language for backend logic and server-side operations.
- **SQLite3:** Lightweight relational database for local storage.
- **HTML:** Structures web content using a markup language.
- **CSS:** Enhances visual presentation with stylesheets.
- **Vue.js:** For frontend tasks.
- **Flask** (*redirect, request, render_template, url_for, session, flash, jsonify*): Web framework for handling routing, sessions, user interactions, and message flashing.
- **Flask-Login** (*LoginManager, login_user, logout_user, current_user, UserMixin*): Manages authentication, login/logout, and access control.
- **SQLAlchemy:** ORM for Python, offering SQL flexibility and power.
- **Flask-SQLAlchemy:** Integrates SQLAlchemy with Flask for database management.
- **Flask-JWT-Extended:** Implements JWT-based authentication and authorization.
- **ChartJS:** Used to make interactive charts for user and admin dashboard.

- **Flask-Caching:** Improves performance by caching responses and reducing server load.
- **smtplib:** Enable email sending, supporting multipart messages and attachments.
- **Celery:** Task queue for asynchronous execution and scheduling using *shared_task* and *crontab*.
- **Redis:** In-memory data store for caching and batch job processing with Celery.

DB Schema Design



The database contains 10 tables. The tables are:

1. **User:** Contains users data.
2. **Role:** Contains about the role of the users namely: User and admin.
3. **Users_Role:** It contains the user id (foreign key to user table column id) and role id (foreign key to role table column id) to maintain the roles of the users.
4. **Subject:** Contains the subject data.

5. **Chapter:** Contains the chapter data for each subject.
6. **Quiz:** Contains the quiz data for each chapter.
7. **Question:** Contains the questions data for each quiz. Linked to quiz through quiz_id foreign key.
8. **Scores:** Contains the score of the user for each quiz. Linked to user and quiz table.
9. **UserResponse:** Stores in detail data of a user's attempt. Linked to user, quiz, score, question table.
10. **Payment:** Stores the transaction data of the website. Linked to user and quiz table.

API Design

- **Roles:** The API includes endpoints to check if a user has 'admin' or 'user' roles.
- **Subjects:** The API allows retrieval, creation, updating, and deletion of subjects.
- **Chapters:** The API enables retrieval, creation, updating, and deletion of chapters, associated with specific subjects.
- **Quizzes:** The API facilitates retrieval, creation, updating, and deletion of quizzes, linked to specific chapters. It also handles fetching a single quiz.
- **Questions:** The API manages the retrieval, creation, updating, and deletion of questions associated with a particular quiz.
- **Scores:** The API handles the submission, retrieval (for both admins and users), updating, and deletion of quiz scores.
- **Quiz Attempts:** The API allows users to retrieve quiz questions for attempting a quiz and to submit their answers, calculating and storing the score.
- **Quiz Results:** The API enables retrieval of detailed results for a specific quiz attempt, including question-wise evaluation.
- **Past Attempts:** The API provides an endpoint for users to view their history of quiz attempts.
- **User Summary:** The API offers a summary of a user's performance, including subject-wise averages and recent scores.
- **Admin Summary:** The API provides an overview for administrators with statistics like total users, subjects, etc., and data for charts (subject performance, qualification distribution, recent quiz participation).
- **Users (Admin Only):** The API has an endpoint for administrators to retrieve a list of all users.
- **User Details (Admin Only):** The API allows administrators to retrieve detailed information for a specific user, including their quiz history and average score.
- **Payments:** The API includes functionality to process payments for quizzes.
- **User Transactions:** The API allows users to view their payment history.

- **Admin Transactions:** The API allows administrators to view all payment transactions.

Architecture and Features

The project is organized with controllers, mail service, background tasks, celery workers, models, and the initialization file housed within the application folder. The app.py, responsible for running the application, is situated in the home directory. The components folder within the static directory contains JavaScript files representing Vue templates, facilitating modular and reusable frontend components for the application's user interface. Configuration files like celeryconfig.py and config.py centralize settings for Celery and the application, respectively. This organization separates concerns and ensures a clear structure for different components of the project.

Features of the Quiz Application:

The implemented features cater to both **users and admins**, providing a seamless quiz experience.

Users can:

- Explore available **subjects, chapters, and quizzes**.
- Attempt quizzes within the allowed **time duration**.
- View **scores and performance analytics**.
- Track **quiz history** and progress over time.
- Participate in **both free and paid quizzes**, with secure **payment integration**.

Admins have additional capabilities, including:


- **Creating, editing, and deleting quizzes, questions, and subjects**.
- Monitoring **quiz statistics, user performance, and revenue** via a dedicated **dashboard**.

The system supports **role-based access control (RBAC)**, ensuring proper access management. It also employs **Celery's scheduler** to execute two periodic tasks:

- A **daily task** for sending quiz recommendations and reminders.
- A **monthly task** for analyzing user engagement and generating reports.

These features contribute to an engaging, structured, and scalable quiz platform.

Video

Video link:  video1997437980.mp4