Author

Name : KAUSHIK K S

Email ID : kaushikkanduri883@gmail.com

Hi,I am Kaushik K S! I am passionate about tech, always looking for ways to solve real-world problems and create meaningful solutions. I enjoy coding, exploring new technologies, and diving into creative projects that push me to learn and grow.

Project Report: Quiz Management System

Description:

Quiz Management System is a web application that allows one to take quizzes, keep records of their score, and even view extensive reports of performance. It has capabilities to create, answer, and score quizzes along with result interpretation, thus delivering a full-scale platform for carrying out quizzes and assessments.

Technologies Used:

Flask (Web Framework)

Flask-SQLAlchemy (Database ORM)

Flask-Login (Authentication)

HTML, CSS, JavaScript (Frontend)

SQLite (Database)

Jinja2 (Templating Engine)

Werkzeug (Security)

Purpose of Technologies:

Flask: Delivers an extensible, flexible, yet lightweight web app framework.

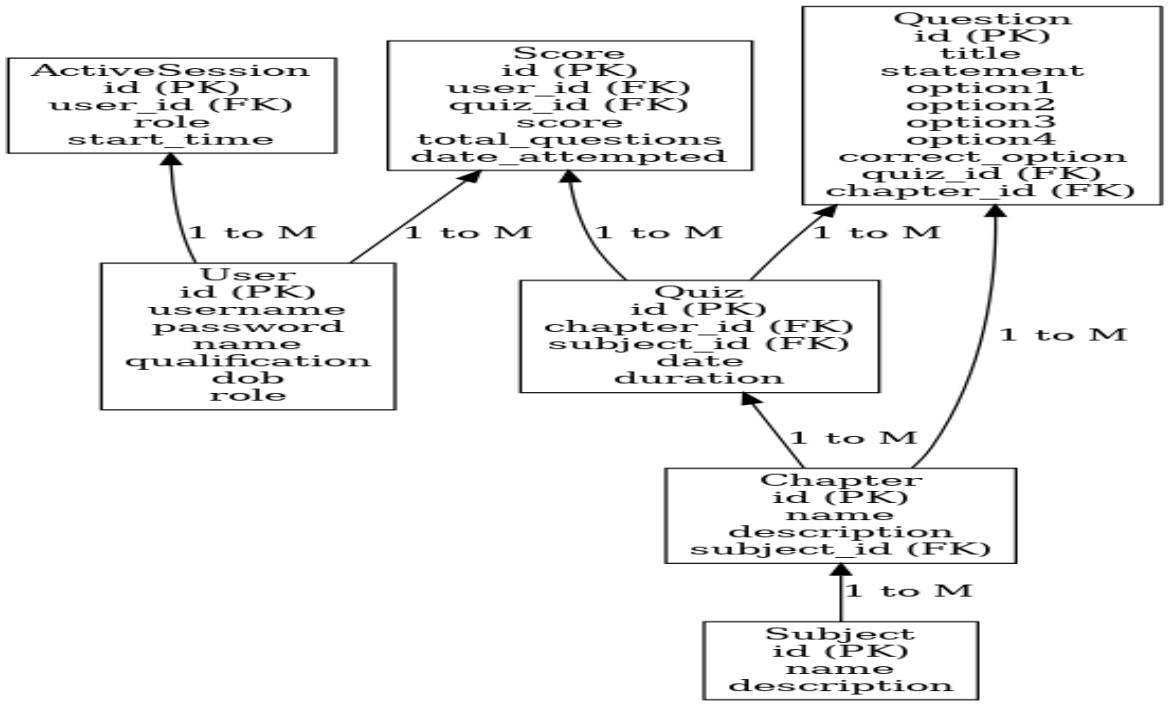
Flask-SQLAlchemy: ORM for simple database interaction with Python classes.

Flask-Login: Handles user authentication, session management, and access control.

Jinja2: Facilitates dynamic HTML rendering and templating.

Werkzeug: Offers security functions and password hashing.

ER Diagram of my Models



The ER diagram illustrates how the models are related in your project:

User and ActiveSession:

- A User may have many ActiveSessions (One-to-Many relationship).
- user_id in ActiveSession points to User table.

Subject, Chapter, Quiz, Question:

- A Subject may have many Chapters (One-to-Many relationship).
- A Chapter may have many Quizzes and Questions (One-to-Many relationship).
- subject_id in Chapter and Quiz tables point to Subject.
- chapter_id in Quiz and Question tables point to Chapter.

Quiz and Question:

- A Quiz may have many Questions (One-to-Many relationship).
- quiz_id in Question points to Quiz.

Score:

- Each Score is associated with a User and a Quiz (Many-to-One relationship for both).
- user_id and quiz_id in Score point to User and Quiz, respectively.

These relationships impose logical constraints among the entities, supporting consistent storage and retrieval of data. Let me know if you have any further requirements or if anything's unclear!

Design Approach:

- The project is based on the MVC (Model–View–Controller) pattern, where controllers manage routing, views manage rendering, and models manage data.
- Routes are logically structured for quiz viewing, starting, submitting, and showing scores.
- Error handling provides smooth user experience.
- Session–based data storage provides temporary data management.

API Design: The Quizmaster API follows the OpenAPI 3.0.3 standard, with well–defined schemas and endpoints:

- User: Manages user authentication and profile data, including roles and qualifications.
- ActiveSession: Manages user sessions with role and start time tracking.
- Subject and Chapter: Handle quiz categorization and organization.
- Quiz: Stores quiz metadata such as date and duration.
- Question: Manages quiz questions, options, and correct answers.
- Score: Stores quiz attempt data, including scores and dates.
- Each schema defines properties and required fields, ensuring consistency and validation.
- CRUD operations are performed using methods like GET, POST, and DELETE for handling quizzes, questions, and scores.

Architecture and Features:

Route files/Controllers are inside the controllers folder, templates in templates folder, and database models are in models folder in a models.py file.

Features Implemented:

- Dynamic quiz generation based on the database.
- Real–time tracking of answers via sessions.
- Auto–save option across questions.
- Display of final result with feedback.
- Persistent history and performance tracking for quizzes.

Other Features:

- Handling of invalid quiz/question access error.
- Responsive user interface with HTML and CSS.
- Session–based management for handling quiz responses.

Conclusion:

The Quiz Management System effectively manages quiz interactions, tracking, and reporting. It takes advantage of Flask and its extensions to implement a seamless and secure user experience.