Author

Anurag Pandey
22F1000812
22f1000812@ds.study.iitm.ac.in
I have a strong enthusiasm for artificial intelligence and data science, with a keen interest in developing innovative solutions and exploring emerging technologies in these fields.

Description

This project involves developing a multi-user quiz platform using Flask for the backend, Jinja2, HTML, CSS, and Bootstrap for the frontend, and SQLite for the database. The system will have an admin with full control over quiz management and users who can register, log in, attempt quizzes, and view scores. The goal is to implement core functionalities such as user authentication, quiz creation, and score tracking while ensuring a seamless user experience.

Technologies used

Backend Technologies

- **Flask**: A lightweight web framework for building the application backend. It provides routing, request handling, and template rendering.
- **Flask-SQLAlchemy**: An ORM (Object Relational Mapper) that simplifies database interactions with SQLite, making it easier to handle queries and data models.
- **Flask-Migrate**: Helps manage database migrations using Alembic, allowing schema changes without data loss.
- **SQLite**: A lightweight, file-based relational database used for storing user, quiz, and score data.

Frontend Technologies

- **Jinja2**: A templating engine used for dynamic content rendering in HTML pages.
- **HTML, CSS, Bootstrap**: Used for designing the frontend and ensuring a responsive, user-friendly interface.

Data Handling and Visualization

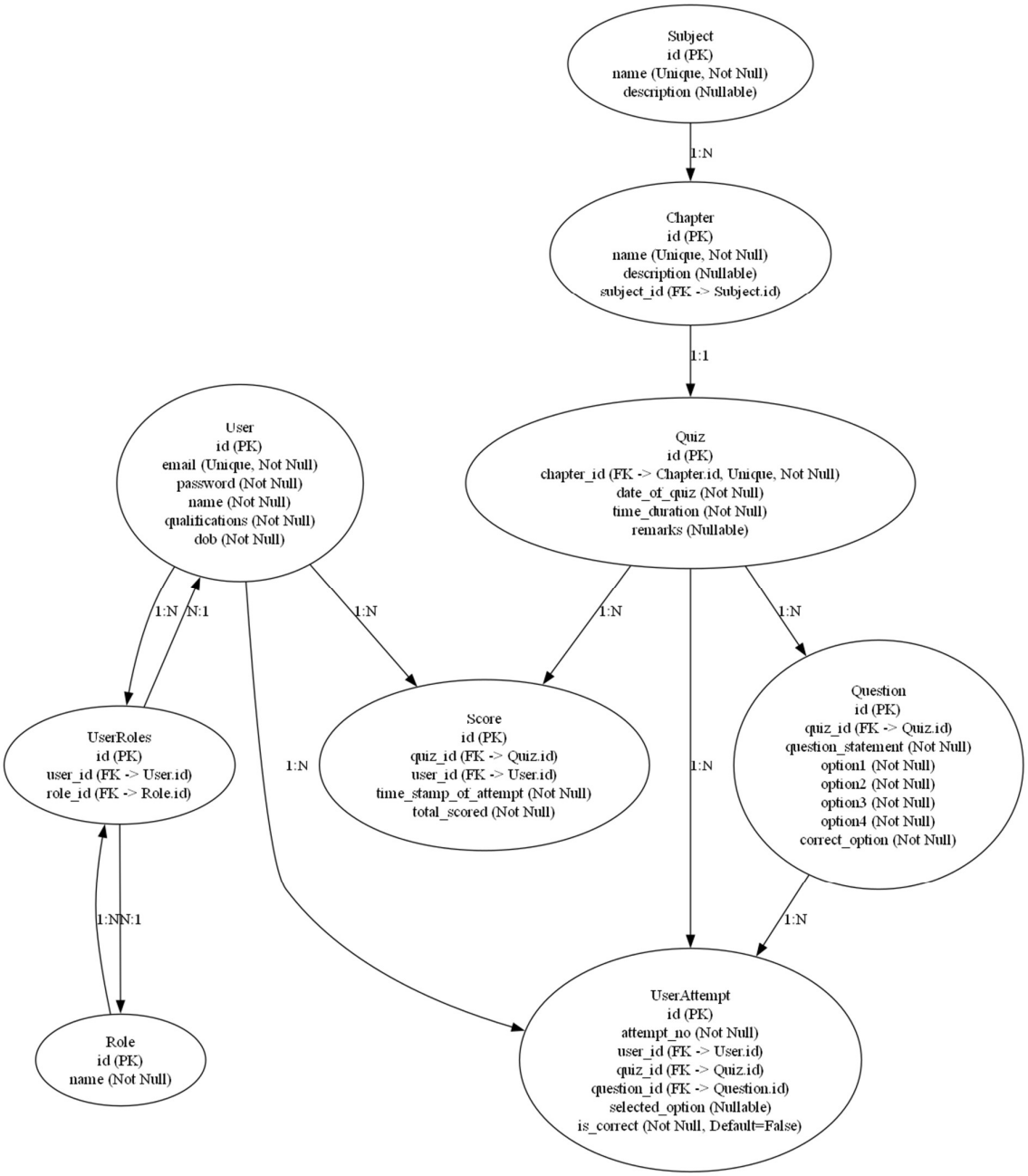
- **Plotly**: A library used to create interactive charts and visualizations for quiz statistics and user performance insights.
- **Pandas**: Used for data manipulation and analysis, particularly in handling quiz scores and generating statistics.
- **JSON & Plotly.utils**: Used for converting data into JSON format for visualization.

Other Libraries and Utilities

- **Flask's built-in modules (request, session, render_template, flash, redirect, url_for)**: Used for handling form submissions, user sessions, flash messages, and page redirections.
- **Datetime**: Used for managing quiz timestamps and duration calculations.

- **Collections (Counter)**: Helps count and analyse quiz-related data efficiently.

DB Schema Design (Image is also attached in the root folder)



The Quiz Master API was designed to maximize efficiency and maintainability within the constraints of using Flask, as mandated by the college. The architecture follows a modular approach, separating authentication, quiz management, and user handling into distinct routes for clarity. SQLAlchemy ORM was used for database management, ensuring smooth interactions with relational data. Session-based authentication was implemented to manage user roles securely. Jinja2 templating was integrated for dynamic content rendering.

API Design

The API facilitates user authentication, quiz management, and role-based access control for admins and users. Admins can manage subjects, chapters, quizzes, and questions, while users can attempt quizzes. It includes endpoints for login, registration, dashboard access, and CRUD operations on quizzes and questions. Sessions handle user authentication and authorization. The API is implemented using Flask, SQLAlchemy, and Jinja templates.

Architecture and Features

Project Organization

The **Quiz Master** project follows a structured Flask application architecture. The **controllers (routes and business logic)** are located in the routes/ directory, handling user authentication, quiz management, and role-based access. The **models** reside in the models/ folder, defining the database schema using SQLAlchemy. The **templates**, including HTML pages for the frontend, are placed inside the templates/ directory, while static assets like CSS, JavaScript, and images are stored in static/. The services/ directory contains utility functions, such as authentication and database operations, while the config/ folder manages environment settings. The main.py file initializes the Flask app, and the application is modularized for easy scalability.

Features Implemented

The project includes **default features** such as **user authentication (login/register), role-based access control (admin/user), and CRUD operations for quizzes, chapters, subjects, and users**. Additional functionalities include **search and summary** endpoints for retrieving quiz details efficiently. Users can **attempt quizzes on the /quiz_master route**, which records their responses, redirects to scores upon submission, and stores attempts in the database. The project follows a **RESTful API structure**, with routes secured by session authentication and validation. Future enhancements could include **leaderboards, analytics, and AI-based quiz recommendations** to improve the user experience.

Video

https://drive.google.com/file/d/1cDDL6eQI8OD02BDw5N7n1MTwIBpv0MDW/view?usp=sharing