AUTHOR:
S Aswin
23f2002498
23f2002498@ds.study.iitm.ac.in

DESCRIPTION:
A functional web application for **Quizzes** needs to be made with secure login, record keeping with a database and tailored experiences for admin and user. Admin need the ability to create,update or delete subjects, chapter, quizzes and questions. While Users attempt quiz and receive feedback for them.

TECHNOLOGIES USED:
Backend:
- **Flask:** Web server gateway interface framework used to implement core components.
  - **Functions used:** request, render_template, redirect, url_for, flash, jsonify, session, Response
- **Flask Login:** Login management and route protection.
  - **Functions used:** LoginManager, UserMixin, login_user, login_required, logout_user, current_user
- **Flask Restful :** REST API framework used to implement fetch apis.
  - **Functions used:** Resource, Api
- **SQLite3:** Database functions.
- **Hashlib, Secrets:** Password hashing and token generation.
- **OS, Datetime:** file operations and datetime support.

Frontend:
- **Boostrap CSS:** For frontend styling.
- **HTMX:** Javascript framework used for dynamic page updates without need for reloads.
- **VueJS:** Used to implement quiz timer that is persistent across refreshes.

DATABASE:

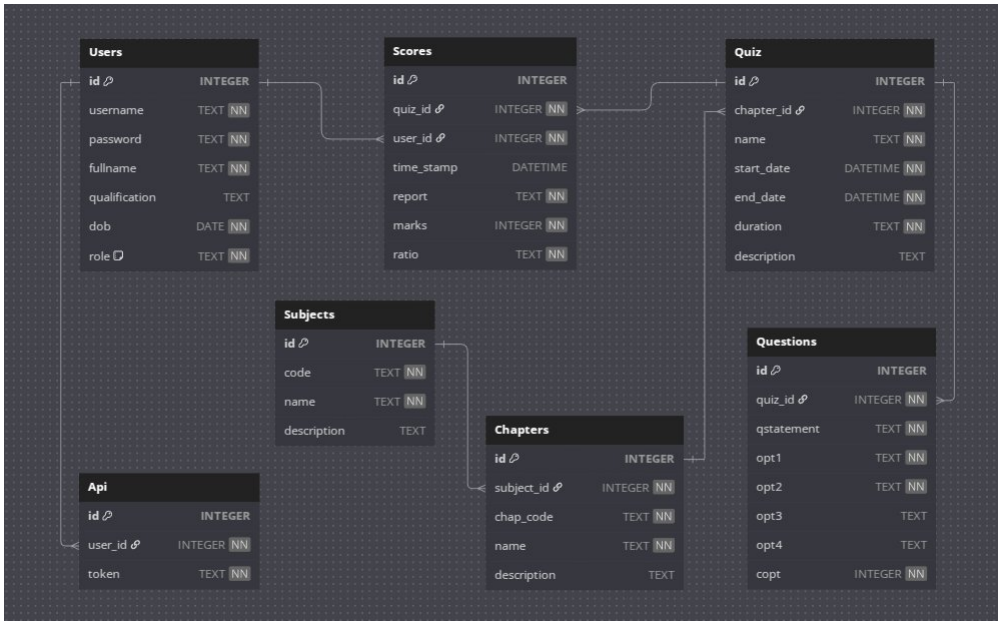

Figure1: Entitiy Relation Diagram

Quizzes needed to be grouped and identified by their chapters or subjects therefore quiz table has a foreign key reference to chapters which in turn references subjects establishing a relation chain. Questions were grouped under a quiz and therefore had quiz_id as the foreign key. Scores table uses user_id and quiz_id as foreign keys to store scores for each quiz attempt by a user and stores question wise report as TEXT for future use. **Refer ER Diagram.** The database is programmatically created by executing **“dbschema.sql”** through python automatically if database is missing, with admin defaults taken from **“secrets.json”**.

API:
REST API implementation for fetching subjects, chapters, quizzes, and scores. Implemented using **“flask_restful”** python modules, the response is serialized using **“jsonify”** function from flask to return a JSON response to client.

Resource endpoints:
- **/api/subjects:** Returns list of subject objects with id, code, name and description.
- **/api/chapters:** Returns list of chapter objects with id, chapter code, subject, name and description.
- **/api/quiz:** Returns quiz list with id, chapter, subject, name, duration, descriptions, tart and end dates
- **/api/scores:** Returns quiz scores for all attempts on a per user basis with quiz details, time of submission, scores and question report.

Authentication is done through tokens for each user. Token passed through **“X-API-KEY”** headers in request .Tokens are 64 character hex strings generated with python **“secrets”** module. Only admin token is created at first run,admin can perform further user token management.

ARCHITECTURE AND FEATURES:

The core components are split into 4 subdirectories namely data, modules, templates, static. **app.py** and **api.yaml** reside in the root directory. Contents of directory :

- **Data:** Contains **“dbschema.sql”** and **“secrets.json”** for database creation and stores the created **“instance.db”** which stores all data required for application to function.
- **Modules:** Contains 3 custom modules to implement core functionalities.
  - **Dbmanage.py** : Provides User, Subject, Chapters, Quiz, Questions and Scores classes with functions to perform CRUD operations on respective tables in the database.
  - **Utilities.py :** Functions for processing scores, parsing data, validation, search capability and special class “api_tools” to serialize and sanitize data to use with the API.
  - **Security.py :** Functions for programmatic creation of database, database checks, API token generation and management, password hashing and verifiying login credentials.
- **Static :** images and styles.css required for frontend.
- **Templates:** Jinja templates organized into admin, extra, forms, lists, scores, search and user based on their purpose with index, login and register templates residing in root of templates.

```
.
└── Root/
    ├── data/
    │   ├── dbschema.sql
    │   ├── instance.db
    │   └── secrets.json
    ├── modules/
    │   ├── dbmanage.py
    │   ├── utilities.py
    │   └── security.py
    ├── static/
    │   ├── images
    │   └── styles.css
    ├── templates/
    │   └── Templates organized into folders
    ├── app.py
    └── openapi.yaml
```

Figure2: File Structure

FEATURES:

CORE:

- **Login:** Implemented using flask login and html forms to collect credentials and verify with database credentials.
- **CRUD:** Implemented with object classes with add(),get(),update() and remove() functions to handle SQL for each table in the database. Output managed using flask and jinja templates.
- **Admin and User Dashboards:** Separate dashboard for admin with CRUD for subject, chapters, quiz and questions with user details and api management. User dashboard built to provide list of quizzes available for the user to attempt and scores for past attempts with option to get question wise report.
- **Quiz page with Duration and Timeline:** Made quiz attempt only available between a fixed timeline and ability to set duration for a quiz which is used for the quiz timer. Quiz page is programmed to automatically submit quiz if the timer runs out.
- **Search :** Search functionality for admins and users to find subjects, chapters, quiz and questions. Additionally admins can view answer for question and search for users.
- **Scores and Quiz Feedback :** Question wise report is generated when the quiz is submitted and the score calculated and feedback of correct/incorrect answers is shown to the user and saved in the database as text which is later parsed into a dictionary before accessing . Users can view these past attempts from scores tab in their dashboards.

ADDITIONAL:

- **API :** REST API built using flask restful to fetch subject, chapters, quiz and scores. Authenticated by sending unique tokens sent in request header. Only Admin can create assign tokens to user.
- **Frontend Enhancements:** Frontend enhanced and made responsive for mobile using Boostrap CSS.
- **Frontend Validation** for forms with HTML5.
- **Backend Validation** with regex.

VIDEO:

https://drive.google.com/file/d/18GL0HvSb1C2BKEDq_09NPABJwN6mFLmA/view?usp=sharing