Quiz Portal

Introduction

The project aims at creating an online Quiz application. The application allows users to view the assigned quizzes and attempt those as well. Now Admin will add questions to the question bank and create the quiz accordingly by tagging the questions with the created quiz. Admin can perform the activities like view all questions, add questions, view all quizzes, assign quizzes to the user. Users can perform activities like view assigned quizzes, attempt the assigned quiz, view quiz results.

The project would leverage on concepts of RESTful API design, deployment using Flask, and DBMS concepts with SQL queries.

Housekeeping points

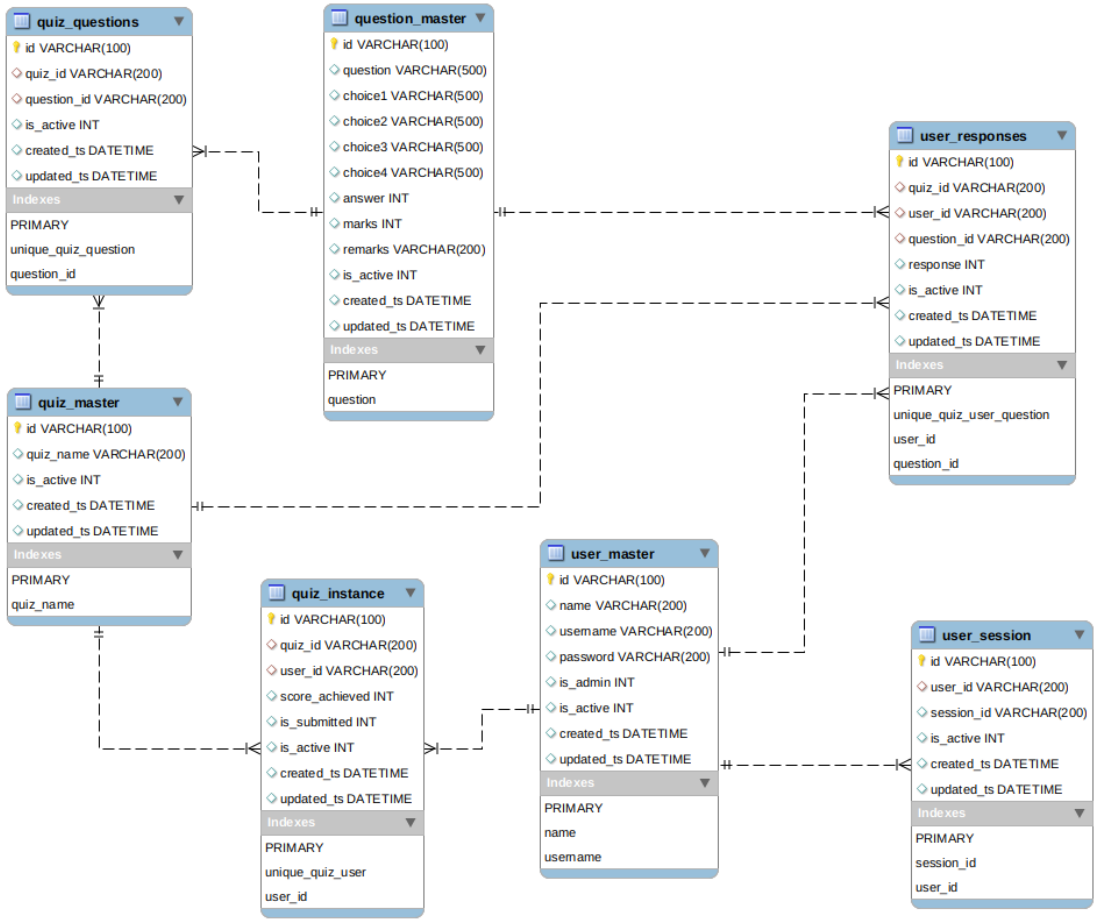
- This is a minimal example and not intended to follow some standard practices.
- We focus on the main flow, and some interesting cases of data validation if not all.
- When a new user is created, a unique user id in uuid format is shared to the user. When the user logs in the application then a session token in uuid format is created and saved at the session management end. The authentication of users is done via this newly created session token assigned at the time of login the application.. The server verifies if such a token is available in the database and infers the sender ID.

Program Statement and tasks

The given program can be designed in following ways:

1. Performing the setup to create the database and table structure for data insertion. This python script should be executed only once to create the tables in the sqlite.
    a. *__init__.py (inside app folder):* This file contains the initial database creation, creating appropriate tables and inserting the necessary records. The unique uuid tokens are created for 1000 users and stored in the database. Admin will always be the first user created who has id 10000. Subsequent users are created/deleted by admin only which starts from 10001 and goes till 11000.

2. Following is the Entity Relationship Diagram of the application:



3. The application can be structured as an ***app*** folder which will be responsible for various apis implementations. Inside the ***app*** folder we have various folders as per their areas of implementation. Below is the tree structure of the code structuring look like

app

__init__.py
apis.py
models.py
services.py
schemas.py
setup.py

main.py
requirements.txt

a. Models: Now Models.py file is responsible for creating various class models which can be mapped to the various tables as mentioned above.

b. Apis: The apis.py file, will contain all RESTful implementations of various apis to implement the problem statement.

c. Services: In the Services.py file, we have the helpers implementation of apis which will be responsible for implementing the exact logic of the respective apis.

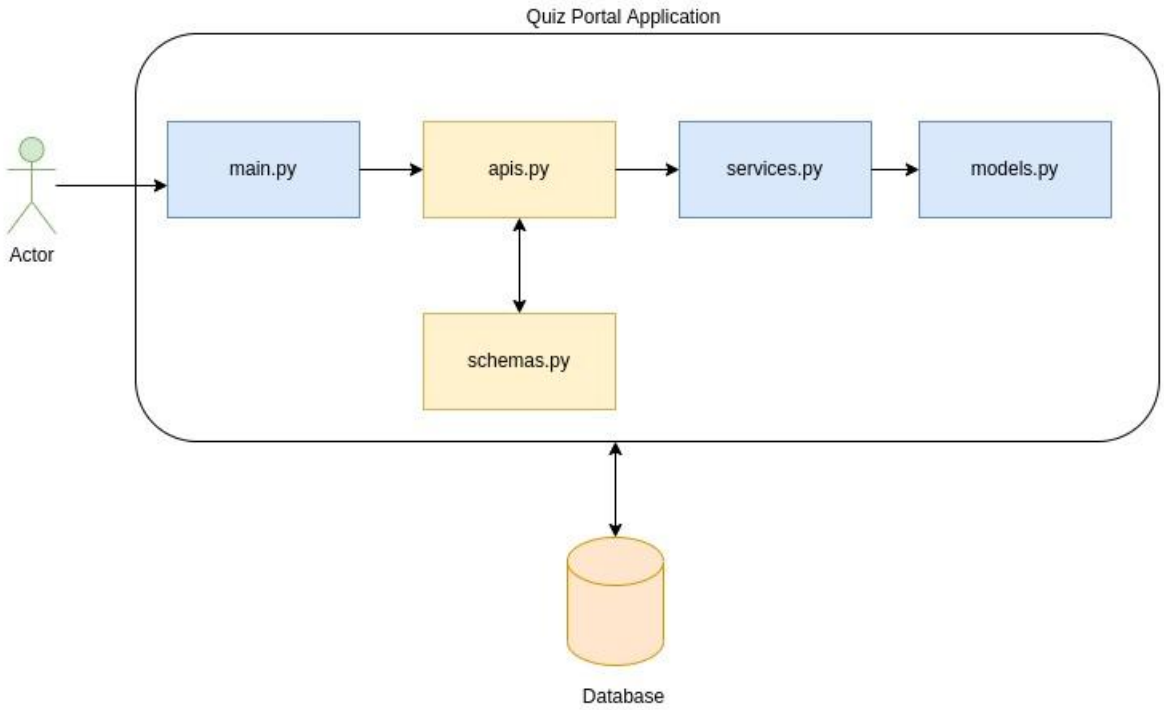
d. Schemas: In this file, we will be having the request and response schemas needed to build respective apis.

e. Setup.py : In this file, we have a function that will store questions in the database as a part of initial data.

f. Main.py : This file responsible for starting the flask application and will be an entry point for the apis for serving.

g. requirements.txt : This file will be having all libraries with their versions, required for serving the apis correctly.

Below is the flow diagram for execution in Quiz Portal Application:



Execution Flow for Quiz Portal Application

4. Let's discuss the various functions to be implemented as a part of the quiz portal application:

1. Sign Up : Users will be signed up for using the application. Users will be Admin or Normal Users.

2. Login: User can be able to login using the credentials.In this session_id will be created which will be further used for all subsequent activities.

3. Add Questions : Admin can add new questions in the questions table.

4. List All Questions : Admin can list all questions persisted in the database.

5. Creating Quiz : Admin can create a new quiz and tag the required questions to the created quiz

6. Assigning Quiz to Users : Admin can assign the required quiz to the user.

7. View quiz : Users are able to view the list of all questions for the particular quiz id.

8. List all Assigned Quizzes: Users can able to view the list of all assigned quizzes with the respective status and with respective scores (submitted/not submitted)

9. Attempt Quiz : Users can attempt the assigned quiz only once by submitting the responses in the given json format. Response will be the score achieved from the quiz

10. List All Quizzes : Admin can list all created quizzes.

11. Quiz Results : For a given quiz id, Admin can fetch quiz results in which the results are sorted in decreasing order of the achieved scores and the quiz instances where users are yet to attempt the quiz, will be displayed at last.

12. Logout : User will be perform a logout action in case of taking break from the application

Evaluation Rubrics

Total Project Points: 240

- **Basic compilation without errors (10%) : 24 Points**
- **Correctness:**
- **Correctness of implementation**
    - **Problem statement - point 1.a (20%) : 48 Points**
    - **Problem statement - point 2.a (20%) : 48 Points**
    - **Problem statement - point 3.a (25%) : 60 Points**
    - **Problem statement - point 4.a (25%) : 60 Points**

Program Instructions

1. Download the zipped folder named **Quiz-Portal-Mini-Project.zip**, unzip it on your local machine, and save it. Go into the directory named **Quiz-Portal-Mini-Project.**

2. Make sure you have Python 3.6 or higher installed. At your command prompt, run:

```
$ python --version
Python 3.7.3
```

If not installed, install the latest available version of Python 3.

3. Open the unzipped folder, and make sure that you have access to the config package, src package and the empty python files that are saved in these packages. Once you have added the code based on the

given a task, you can run quiz_app.py to verify if the edited code is working successfully or not.

```
$ python3 quiz_app.py (On many Linux platforms)
                    OR
$ python quiz_app.py (On Windows platforms)
```

In any case, one of these two commands should work.

4. After running **quiz_app.py,** you can examine the result. This file will currently run various simple calls that will communicate with different classes and methods in those classes. As you solve the problems, you'll be frequently modifying and running this file. You can comment or modify the initial code as needed.

5. Alternatively, you could install a popular Python **IDE**, such as **PyCharm** or **Visual Studio Code**, and select a command to build the project from there.A helper file/document that has necessary information about how to use PyCharm for the first time will also be available.

6. There will be another file that will help you to understand the import mechanism in python will also be available.Please take help of these files accordingly.

7. Once the program is ready to submit, zip the parent folder **Quiz-Portal-Mini-Project**, and upload the new zip file as **Quiz-Portal-Mini-Project.zip**. It is now ready for evaluation.

8. Code submission where the files are not executable and can not be accessed will not be evaluated.

9. After building the apis, swagger docs can be accessed using the below url :
http://127.0.0.1:8000/swagger-ui/

This will be useful for apis testing purposes. Below is the example of how swagger docs will look like.