Quiz Master V1 - MAD I Jan 2025

Modern Application Development I

Project Statement

It is a multi-user app (one requires an administrator and other users) that acts as an exam preparation site for multiple courses.

Frameworks to be used

These are the mandatory frameworks on which the project has to be built.

- Flask for application back-end
- Jinja2 templating, HTML, CSS and Bootstraps for application front-end
- SQLite for database (No other database is permitted)

-

Note: All demos should be possible on your local machine.

Roles

The platform will have two roles:

1. Admin - root access - It is the superuser of the app and requires no registration

- Admin is also known as the quiz master
- There is only one admin to this application
- The administrator login redirects to the quiz master/admin dashboard
- The administrator will manage all the other users
- The administrator will create a new subject
- The administrator will add various chapters under a subject
- The administrator will add quiz questions under a chapter

2. User - Can attempt any quiz of its choice

- User Registration and Login
- Each user may have:

1. id - primary key
2. Username (email)
3. Password
4. Full Name
5. Qualification
6. DOB

- To be able to choose the subject as well as the chapter name

- Start the quiz

- View the quiz scores

Terminologies

User: The user will register/login and attempt any quiz of his/her interest.

Admin: The superuser with full control over other users and data. Registration is not allowed for the admin: The admin account must pre-exist in the database when the application is initialized.

Subject: The field of study in which the user wishes to give the quiz. The admin will be creating one or many subjects in the application. Every subject can possibly have the following fields:

1. id - primary key
2. Name
3. Description
4. etc: Additional fields (if any)

Chapter: Each subject can be subdivided into multiple modules called chapters. The possible fields of a chapter can be the following:

1. id - primary key
2. Name
3. Description

4. etc: Additional fields (if any)

Quiz: A quiz is a test that is used to evaluate the user’s understanding of any particular chapter of any particular subject. A test may contain the following attributes:

1. id - primary key
2. chapter_id (foreign key-chapter)
3. date_of_quiz
4. time_duration(hh:mm)
5. remarks (if any)
6. etc: Additional fields (if any)

Questions: Every quiz will have a set of questions created by the admin. Possible fields for a question include:

1. id - primary key
2. quiz_id (foreign key-quiz)
3. question_statement
4. Option1, option2, … etc.
5. etc: Additional fields (if any)

Scores: Stores the scores and details of a user's quiz attempt. Possible fields for scores include:

1. id - primary key
2. quiz_id (foreign key-quiz)
3. user_id (foreign key-user)
4. time_stamp_of_attempt
5. total_scored
6. etc: Additional fields (if any)

Note: The above fields are not exhaustive. Students can add more fields as per their specific requirements.

Application Wireframe

Quiz Master

Note:
The provided wireframe is intended only to illustrate the application's flow and demonstrate what should appear when a user navigates between pages.

- Replication of the exact views is NOT mandatory.
- Students are encouraged to work on their own front-end ideas and designs while maintaining the application's intended functionality and flow.

Core Functionalities

1. Admin login and User login

- A login/register form with fields like username, password etc. for user and admin login
- You can either use a proper login framework or just use a simple HTML form with username and password (we are not concerned with how secure the login or the app is)
- The app must have a suitable model to store and differentiate all types of users
-

2. Admin Dashboard - for the Admin

- The admin should be added, whenever a new database is created
- The admin creates/edits/deletes a subject
- The admin creates/edits/deletes a chapter under the subject
- The admin will create a new quiz under a chapter
- Each quiz contains a set of questions (MCQ - only one option correct)
- The admin can search the users/subjects/quizzes
- Shows the summary charts

3. Quiz management - for the Admin

- Edit/delete a quiz
- The admin specifies the date and duration(HH: MM) of the quiz
- The admin creates/edits/deletes the SOC questions inside the specific quiz

4. User dashboard - for the User

- The user can attempt any quiz of his/her interest
- Every quiz has a timer
- Each quiz score is recorded
- The earlier quiz attempts are shown
- Shows the summary charts

Note: The database must be created programmatically (via table creation or model code). Manual database creation, such as using DB Browser for SQLite, is NOT allowed.

Recommended Functionalities

- API resources are created to interact with the subjects, chapters and/or quizzes. (Please note: you can choose which API resources to make from the given ones, It is NOT mandatory to create API resources for CRUD of all the components)
- APIs can either be created by returning JSON from a controller or using a flask extension like flask_restful
- External APIs/libraries for creating charts, e.g. Chart JS
- Implementing frontend validation on all the form fields using HTML5 form validation or JavaScript
- Implement backend validation within your app's controllers.

Optional Functionalities

- Provide styling and aesthetics to your application by creating a beautiful and responsive front end using simple CSS or Bootstrap
- Incorporate a proper login system to prevent unauthorized access to the app using Flask extensions like flask_login, flask_security etc.
- Any additional feature you feel is appropriate for the application

Evaluation

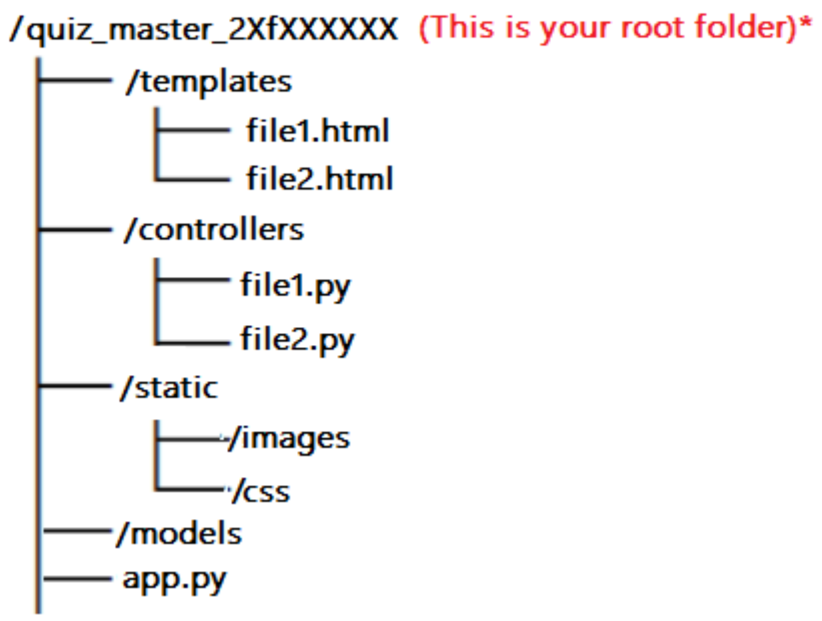
- Students have to create and submit a project report (not more than 5 pages) on the portal, along with the actual project submission
- The report must include the following things;

- Student details
- Project details, including the question statement and how you approached the problem statement
- Frameworks and libraries used
- ER diagram of your database, including all the tables and their relations
- API resource endpoints (if any)
- Drive link of the presentation video

- The project report must be included as a PDF inside the root submission folder and NOT along with it.

Click here of project report demo

Possible folder structure:



```
/quiz_master_2XfXXXXXX  (This is your root folder)*
├── /templates
│     ├── file1.html
│     └── file2.html
├── /controllers
│     ├── file1.py
│     └── file2.py
├── /static
│     ├──/images
│     └──/css
├──/models
├── app.py
```

- All code is to be submitted on the portal in a single zip file (zipping instructions are given in the project document - Project Doc T12025
- Video Presentation Guidelines (Advised):

1. A short Intro (not more than 30 sec)

2. How did you approach the problem statement? (30 sec)

3. Highlight key features of the application (90 sec)

4. Any Additional feature(s) implemented other than core requirements (30 sec)

Note:

1. The final video must not exceed 5-10 minutes.

2. Keeping your video feed on during recording (like in a screencast) is optional but recommended.

- The video must be uploaded on the student drive with access to anyone with the link and the link must be included in the report:

- This will be viewed during or before the viva, so it should be a clear explanation of your work.

- Viva: after the video explanation, you are required to give a demo of your work, and answer any questions that the examiner asks:

- This includes making changes as requested and running the code for a live demo.
- Other questions that may be unrelated to the project itself but are relevant to the course.

Instructions

- This is a live document and will be updated with more details (wireframe)
- We will freeze the problem statement on or before 20/12/2024, beyond which any modifications to the statement will be communicated via proper announcements.
- The project has to be submitted as a single zip file.