Assignment

AI Powered Quiz Generator

Submitted By ***Nivedita Ajit***

MCQ QUIZ APPLICATION

GITHUB LINK: https://github.com/NiveditaAjit27/PWC.git

In the ever-changing landscape of technology, the significance of interactive applications that captivate users in real-time has grown exponentially, playing a crucial role in both educational and entertainment domains. An example of such applications is the creation of a quiz platform that allows users to actively engage in quizzes tailored to their selected subjects. This code makes use of Python for backend processing, Langchain for the dynamic generation of questions and answers, and Streamlit for the development of a user-friendly front-end interface.

DESIGN

1. **Objective**: Design a Quiz Application: The primary goal is to create an interactive quiz application where users can input their preferred quiz topic, set the number of questions, and answer multiple-choice questions generated dynamically using the OpenAI API.

2. **Libraries Used: OpenAI:**
The code utilizes the OpenAI GPT-3 language model for generating questions and answer options dynamically. Streamlit: Streamlit is employed for building the interactive web application. It simplifies the process of creating user interfaces with Python.

3. **User Interface:**
Streamlit Title and Subtitle: The application starts with a title ("MCQ Quiz Application") and a subtitle where users can input their preferred quiz topic. User Inputs: The user is prompted to enter their preferred quiz topic and set the number of questions using a text input and a slider, respectively.

4. **Question Generation:**
OpenAI API Integration: The generate_question_answer function utilizes the OpenAI API to generate diverse and contextually relevant questions and answer options based on user-specified topics.

5. **Quiz Generation:**
Dynamic Question and Answer Options: Within a Streamlit form, a loop generates a set number of questions, each with multiple-choice answer options. Answer Shuffling: Answer options are shuffled randomly to avoid any bias in the quiz structure.

6. **User Interaction:**
User Input for Answers: Users provide their answers through text input fields, with validation to accept only A, B, C, or D as responses.

7. **Display: Question Display:**
Each question is displayed along with the shuffled answer options in the user interface. Correct Answer Display: The correct answer is also shown for each question.

8. **Scoring: Scoring Mechanism:**
The application dynamically updates and displays the user's score based on the correctness of their answers.

9. **Submission:**
Form Submission: A submit button is included to submit the quiz for evaluation.

10. **Result Display**: Final Score: Upon submission, the application reveals the user's final score.

11. **Code Organization: Modular Functions:**
The code is organized into functions for question generation, answer validation, and score calculation, promoting modularity and readability.

12. **User Feedback: Feedback Display:**
The application provides immediate feedback on the correctness of the user's answers.

13. **Conclusion: Closing Message:**
The application concludes by displaying the user's final score.

14. **Future Improvements: Scalability:**
Considerations for scalability and efficiency in question generation could be discussed for potential future improvements.

15. **Code Security: API Key Handling:**
The importance of securely handling API keys, such as using environment variables or secure storage, should be highlighted in the context of code security.

16. **User Experience: Visual Elements**:
The inclusion of visual elements, such as question separators, enhances the user experience.

USER GUIDE

1. Introduction

1.1 Purpose

The MCQ Quiz Application is designed to provide an interactive and educational quiz-taking experience. Users can choose a preferred quiz topic, set the number of questions, and answer dynamically generated multiple-choice questions.

1.2 Key Features

- Dynamic question generation using OpenAI GPT-3.
-
- User-friendly interface with Streamlit.
-
- Randomized answer options for each question.
-
- Immediate feedback on correctness.
-
- Final score display at the end of the quiz.

2. Getting Started

2.1 Accessing the Application

Navigate to the MCQ Quiz Application through the provided link. Ensure that your browser is up-to-date for optimal performance.

2.2 Application Interface Overview

Title: "MCQ Quiz Application"
Subtitle: "Topic to quiz on!"
User Inputs:
Text input for entering the preferred quiz topic.
Slider for setting the number of questions (1 to 10).
Questions and Answers Section: Displayed within a form.
User Answer Input: Text input for entering the answer (A, B, C, or D).
Submit Button: "Submit Quiz"
Final Score Display: Shown after submitting the quiz.

3. Taking the Quiz

3.1 Entering Quiz Topic

Topic Input:
Enter your preferred quiz topic in the provided text input.

3.2 Setting the Number of Questions

Number of Questions:
Use the slider to set the desired number of questions (between 1 and 10).

3.3 Answering Questions

Question Display:
Questions are displayed numerically, along with randomized answer options (A, B, C, D).
Enter your chosen answer in the corresponding text input.

3.4 Submitting the Quiz

Submit Button:
Click the "Submit Quiz" button after answering all questions.

4. Results

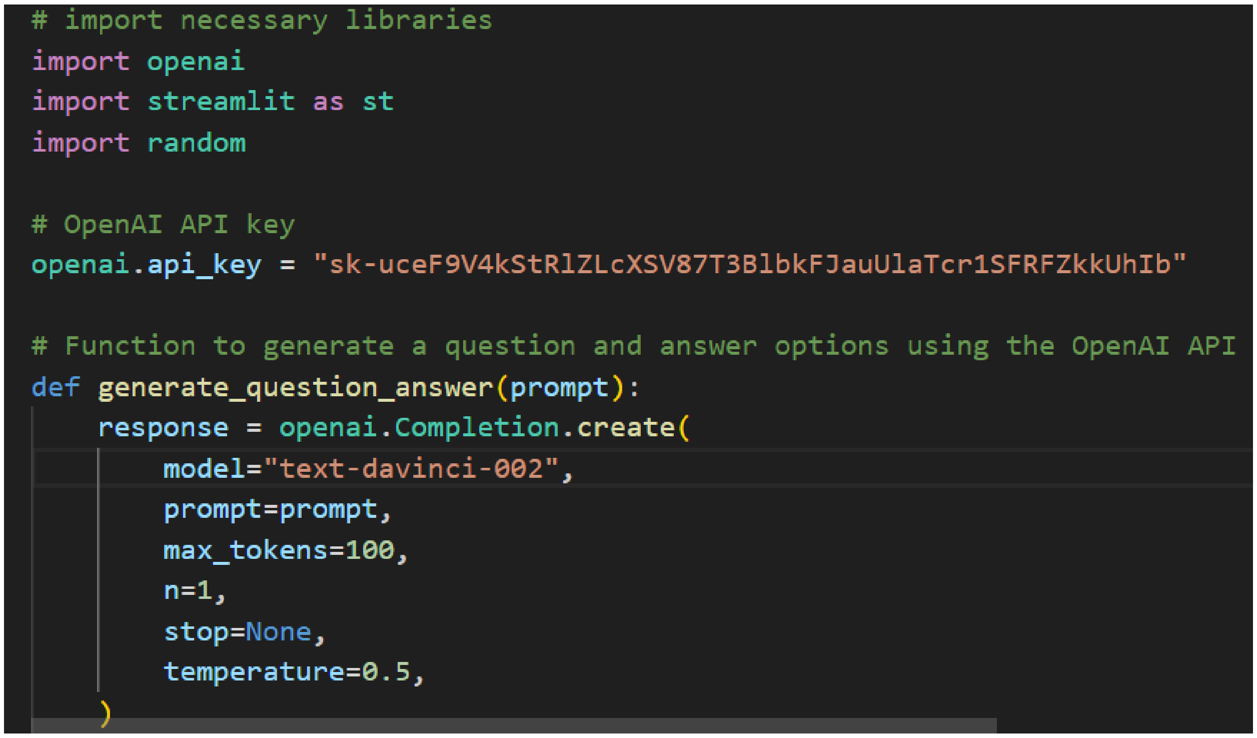
4.1 Viewing the Final Score

Final Score Display:
After submitting the quiz, the application will display your final score out of the total number of questions.

4.2 Reviewing Correct Answers

Correct Answers:
The correct answer for each question is displayed along with your submitted answer.

CODE IMPLEMENTATION

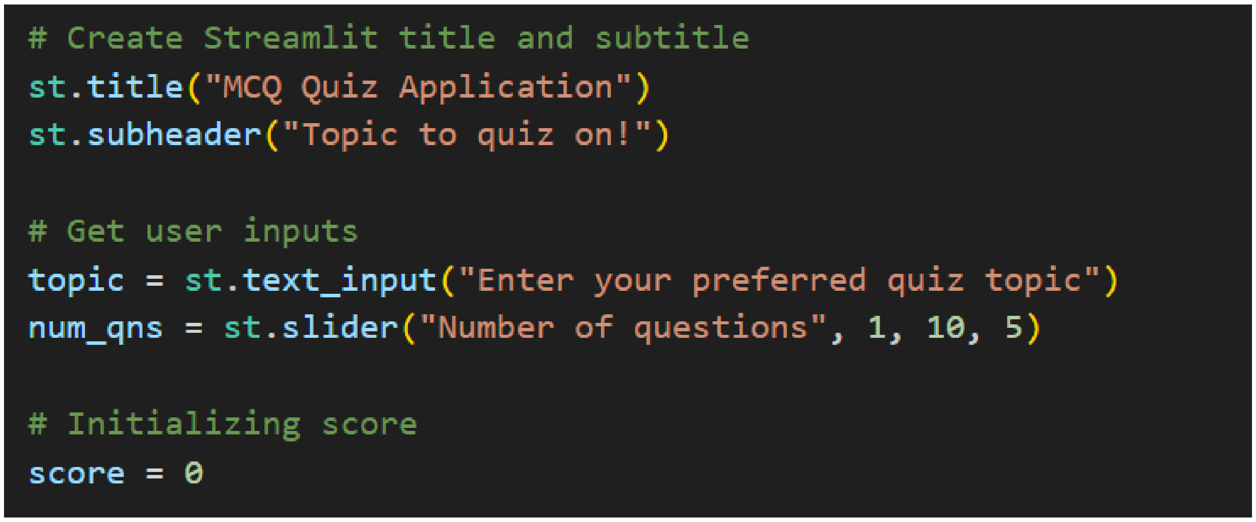


```python
# import necessary libraries
import openai
import streamlit as st
import random

# OpenAI API key
openai.api_key = "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"

# Function to generate a question and answer options using the OpenAI API
def generate_question_answer(prompt):
    response = openai.Completion.create(
        model="text-davinci-002",
        prompt=prompt,
        max_tokens=100,
        n=1,
        stop=None,
        temperature=0.5,
    )
```

- Firstly we import the necessary files like openai and streamlit.
- Then we open the API key that's being provided
- The 'generate_question_answers' helps with generation of questions and answers with the help of OpenAI
- openai.Completion.create is used to interact with the OpenAI API
- 'model="text-davinci-002' tells which language model to use
- 'prompt' is the input text that serves as the starting point for the model to generate the completion.
- 'max_tokens=100' limits the generated responses to only 100
- 'n=1' means only one completion
- 'stop=None' means there's no stopping point
- 'temperature=0.5' checks the model's output randomness
- 'response.choices[0].text.strip()' ,the function extracts the text generated by the API response and eliminates any leading or trailing white spaces from it.

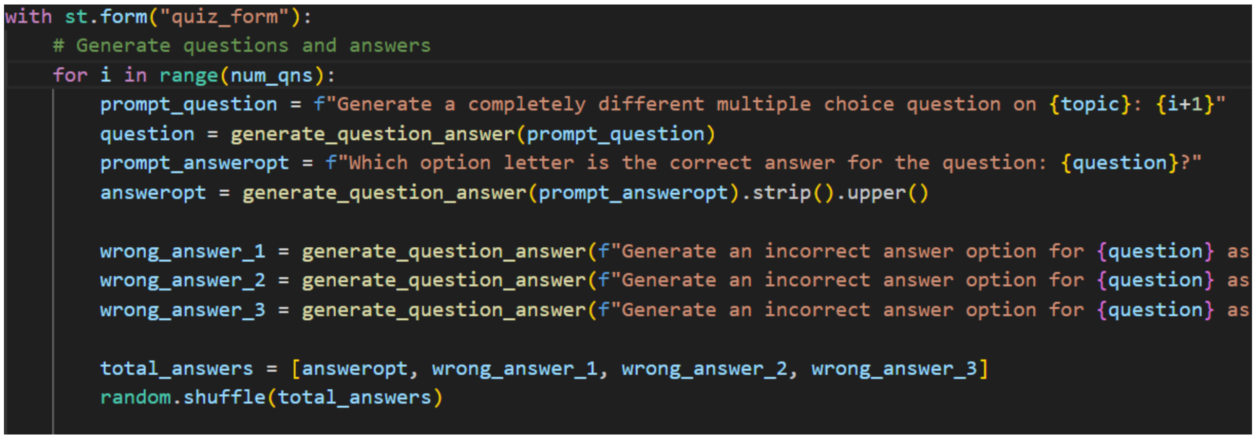


```python
# Create Streamlit title and subtitle
st.title("MCQ Quiz Application")
st.subheader("Topic to quiz on!")

# Get user inputs
topic = st.text_input("Enter your preferred quiz topic")
num_qns = st.slider("Number of questions", 1, 10, 5)

# Initializing score
score = 0
```

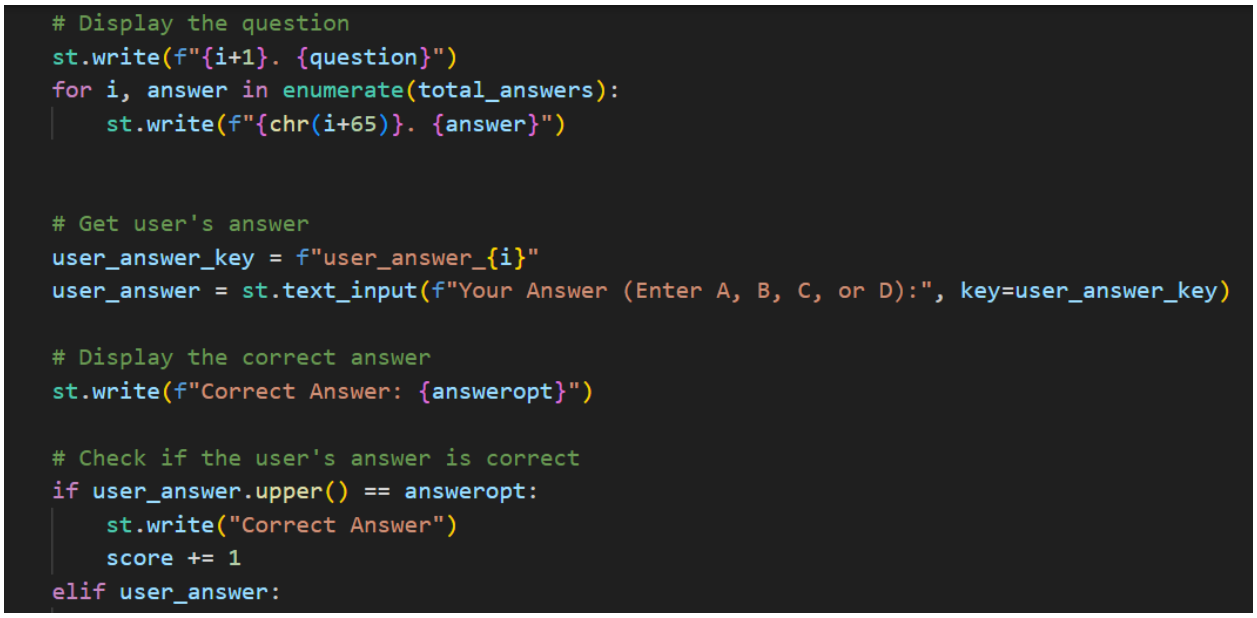
In this part we create the title and subtitle that needs to be shown in the Streamlit Application. Then we mention a subtitle below the main saying ‘Topic to quiz on’. Then we take the user input for the topic and we also ask for the number of questions with the help of a slider. The number of questions is stored in ‘num_qns’. The initialization of the score is done to calculate the final score. Initialization of score is 0 in the beginning. It will be updated as the user answers every question.



```
with st.form("quiz_form"):
    # Generate questions and answers
    for i in range(num_qns):
        prompt_question = f"Generate a completely different multiple choice question on {topic}: {i+1}"
        question = generate_question_answer(prompt_question)
        prompt_answeropt = f"Which option letter is the correct answer for the question: {question}?"
        answeropt = generate_question_answer(prompt_answeropt).strip().upper()

        wrong_answer_1 = generate_question_answer(f"Generate an incorrect answer option for {question} as
        wrong_answer_2 = generate_question_answer(f"Generate an incorrect answer option for {question} as
        wrong_answer_3 = generate_question_answer(f"Generate an incorrect answer option for {question} as

        total_answers = [answeropt, wrong_answer_1, wrong_answer_2, wrong_answer_3]
        random.shuffle(total_answers)
```

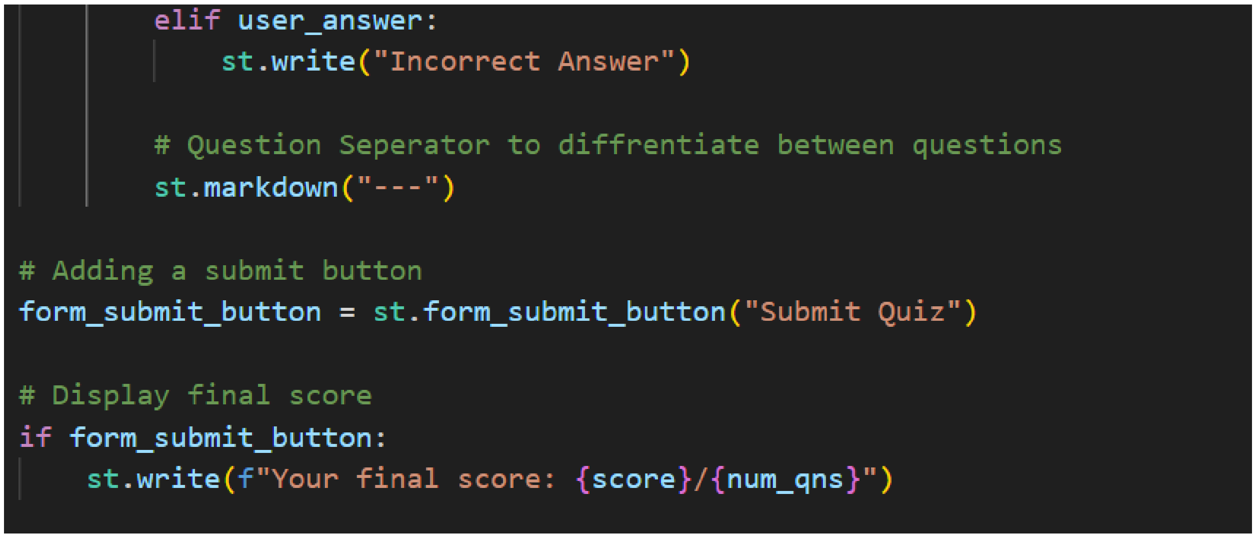


```
        # Display the question
        st.write(f"{i+1}. {question}")
        for i, answer in enumerate(total_answers):
            st.write(f"{chr(i+65)}. {answer}")


        # Get user's answer
        user_answer_key = f"user_answer_{i}"
        user_answer = st.text_input(f"Your Answer (Enter A, B, C, or D):", key=user_answer_key)

        # Display the correct answer
        st.write(f"Correct Answer: {answeropt}")

        # Check if the user's answer is correct
        if user_answer.upper() == answeropt:
            st.write("Correct Answer")
            score += 1
        elif user_answer:
```



```
        elif user_answer:
            st.write("Incorrect Answer")

        # Question Seperator to diffrentiate between questions
        st.markdown("---")

    # Adding a submit button
    form_submit_button = st.form_submit_button("Submit Quiz")

    # Display final score
    if form_submit_button:
        st.write(f"Your final score: {score}/{num_qns}")
```

In this part of the code we create a form using st.form("quiz form"). We set up a form using a special tool in Streamlit called "st.form." This tool helps organize the different parts of the form neatly. As we go through each question in a loop based on the chosen number of questions, we create unique questions and their possible answers. The questions are presented using "st.write," which arranges them with numbers and displays the answer options labeled A, B, C, and D. To get answers from users, we use "st.text_input," asking them to type A, B, C, or D. Additionally, we show the correct answer below the user's input with "st.write(f'Correct Answer: {answeropt}')." Scoring is done by checking if the user's answer matches the correct one, updating the score accordingly. We use a visual separator, "st.markdown('---')," between questions. Finally, we include a submit button with "st.form_submit_button('Submit Quiz')." When users click submit, the app reveals their final score.