AI-Powered Quiz Generator

Technical Assessment for PwC

Aseel Alzubaidi

12th, December 2023

Contents

1. Introduction

Quizzet was initiated with the vision of creating a dynamic and interactive platform for users to enhance their knowledge through a quiz-based learning approach. The application leverages the powerful capabilities of OpenAI's language model, providing a tool for users to engage with quizzes tailored to their chosen topics. This initiative aligns to deliver a solution that fosters knowledge acquisition, through an accessible and user-friendly interface.

2. Goals and Objectives

2.1. Goals

The primary goal of the Interactive MCQ Quiz Application is to create an accessible and engaging learning environment for users who wish to test their knowledge on various subjects.

2.2. Objectives

- To provide a dynamic and interactive user experience using Streamlit for the front end.
- To implement robust application logic that integrates with Langchain and OpenAI Chat Completion API for real-time question generation.
- To ensure a scalable and maintainable codebase that allows for easy updates and feature additions.
- To document the system thoroughly, ensuring ease of use, deployment, and collaboration for future developers and users.

3. System Architecture

The system architecture is designed to be modular and scalable, comprising three main components:

- **Front End**: Developed using Streamlit, it offers an intuitive user interface for quiz participation, topic selection, and results display.
- **Back End**: A Python-based logic layer that handles API calls, question generation, answer validation, and scoring.
- **OpenAI Integration**: Utilizes the OpenAI API to generate quiz content dynamically based on user input, leveraging the advanced language models provided by OpenAI.

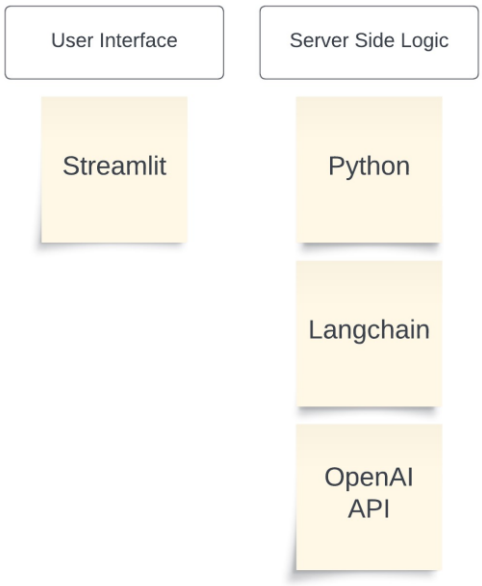


Figure 1-Front End and Back End

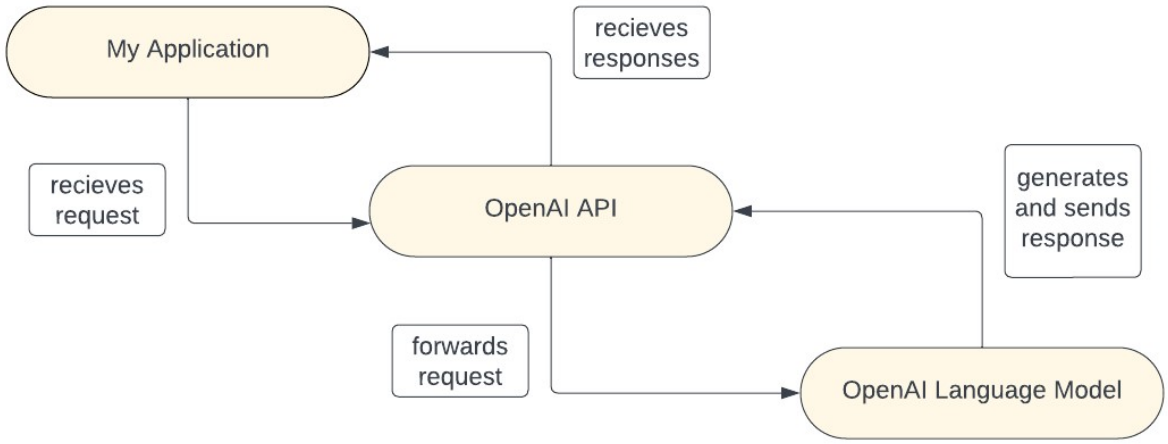


Figure 2-Logic Diagram

4. Environment Setup and Library Installation

To establish a robust development environment for **Quizzet**, it is crucial to install several key libraries. This installation process should be carried out through the command prompt (cmd). The necessary libraries include Streamlit for the front-end interface, OpenAI for quiz content generation, and Langchain for enhanced language processing.

- **Streamlit Installation**: Streamlit is essential for building the user interface. Install it using the command: `pip install streamlit`
- **OpenAI API Library**: The OpenAI library is crucial for dynamically generating quiz questions and answers. Install it with: `pip install openai`
- **Langchain**: To augment language processing capabilities, Langchain is a necessary addition. Install it via: `pip install langchain`

5. Application Logic

The Application Logic section forms the backbone of Quizzet, orchestrating a sophisticated interplay of Python, Langchain, and the OpenAI Chat Completion API to dynamically generate quiz questions and answers. The primary goal is to create a seamless and interactive quiz experience for users based on their chosen topic of interest.

a. Backend Logic with Python:

At the core of the application lies the Python backend logic, responsible for handling user requests, interfacing with external APIs, and orchestrating the entire quiz generation process. The generate_quiz_data function is a pivotal component, iterating through the specified number of questions and leveraging the OpenAI Chat Completion API to dynamically generate quiz content related to the user's chosen topic.

b. Integration of Langchain and OpenAI Chat Completion API:

The integration of Langchain and the OpenAI Chat Completion API is a key aspect of the Application Logic. The generate_quiz_data function utilizes the OpenAI API to generate responses to prompts related to the user's specified topic. Langchain comes into play during the processing of these responses, extracting the relevant question and answer options.

c. Dynamic Quiz Question Generation:

The heart of the application's logic lies in dynamically generating quiz questions. For each question, a prompt is constructed, requesting the generation of a quiz question and answer options based on the user's chosen topic. The response from the OpenAI Chat Completion API is then processed to extract the question and options using the extract_question_and_options function.

d. Question, Answer Options, and Correct Answer Identification:

The Application Logic ensures that each generated quiz question is accompanied by a set of answer options. The correct answer is identified and organized within the quiz data structure. This comprehensive logic guarantees that users are presented with meaningful and contextually relevant quiz content, enhancing the overall quiz-taking experience.

6. Front-End Design: Streamlit Interface Overview

The "Front-End Design: Streamlit Interface Overview" section encapsulates the user-facing aspect of Quizzet, leveraging the Streamlit framework to craft an intuitive and visually appealing interface. The design is geared towards providing users with a seamless experience from entering their preferred quiz topic to submitting answers and receiving scores.

a. Streamlit Interface Initialization:

The Streamlit interface is initialized using st.set_page_config to configure the page title, icon, layout, and sidebar state. This ensures a cohesive and visually pleasing layout for users engaging with Quizzet.

b. Styling and Theming:

Styling elements are introduced using HTML and CSS within the st.markdown block. This section enhances the visual aesthetics of the interface, setting the background color and text style to create an engaging and readable environment for users.

c. Logo Display:

A logo image, represented by "logo.png," is displayed at the top of the interface, adding a personalized touch to the application. This visual element contributes to brand identity and recognition.

d. Title and Input Fields:

The main title "MCQ Quiz Application" is presented prominently, setting the stage for users. The application prompts users to enter their preferred quiz topic via the st.text_input function. This establishes a clear and straightforward process for users to initiate the quiz.

e. Dynamic Quiz Generation:

Conditional logic is implemented to check if a user has entered a quiz topic. If a topic is provided, the interface dynamically adjusts to display additional input fields, such as the number of questions the user wants to answer (st.number_input). This ensures flexibility and user control over the quiz parameters.

f. Caching Quiz Data:

The @st.cache decorator is employed to efficiently cache the quiz data, preventing unnecessary recalculations of quiz content when the user interacts with the application. This enhances the application's responsiveness and performance.

g. Quiz Presentation and Answer Submission:

For each generated quiz question, the interface dynamically presents the question and multiple-choice answer options using st.write and st.radio. Users can select their answers interactively, providing a seamless quiz-taking experience. The design promotes clarity and ease of navigation, enhancing user engagement.

h. Scoring and Correct Answers:

Upon quiz submission (triggered by the "Submit Quiz" button), the application calculates and displays the user's score. Correct answers are revealed, allowing users to review their performance. This feedback mechanism enhances the learning experience and provides valuable insights into quiz results.

7. Back-End Architecture: System Logic

The "Back-End Architecture: System Logic" section delves into the intricacies of Quizzet's backend, emphasizing the orchestration of Python, Langchain, and the OpenAI Chat Completion API. This section explores the foundational elements that power the application, from backend logic to the integration of external services.

a. Python Backend Logic:

At the heart of the application's backend lies the Python logic responsible for handling user requests, coordinating data flow, and executing the core functionalities of Quizzet. The generate_quiz_data function stands as a central component, iteratively invoking the OpenAI Chat Completion API to dynamically generate quiz questions based on the user's selected topic.

b. Integration of Langchain and OpenAI Chat Completion API:

The synergy between Langchain and the OpenAI Chat Completion API is a critical aspect of the backend architecture. Langchain plays a pivotal role in processing the responses obtained from the OpenAI API. Specifically, the extract_question_and_options function meticulously dissects the API response, extracting the generated quiz question and multiple-choice answer options.

c. Dynamic Quiz Question Generation:

The backend logic focuses on the dynamic generation of quiz questions tailored to the user's specified topic. For each question, a prompt is constructed, invoking the OpenAI Chat Completion API to generate contextually relevant quiz content. This ensures a personalized and engaging quiz experience for users.

d. Question, Answer Options, and Correct Answer Identification:

The backend system orchestrates the generation of not only questions but also multiple-choice answer options for each question. The identification of the correct answer is a key step, ensuring that the quiz data structure encapsulates all necessary information for a seamless quiz-taking experience on the frontend.

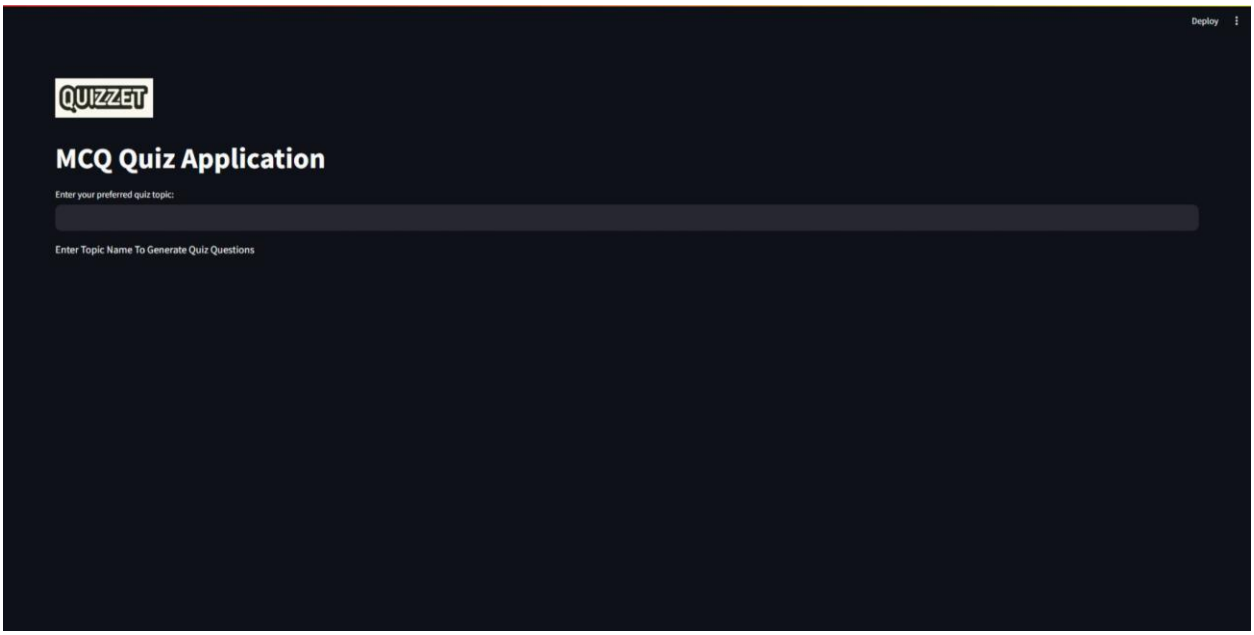
e. Streamlit Integration:

The backend logic seamlessly integrates with the Streamlit frontend, providing the necessary quiz data for presentation. The load_quiz_data function, decorated with @st.cache, efficiently manages the caching of quiz data. This optimization enhances the application's responsiveness by avoiding redundant computations when users interact with the frontend.

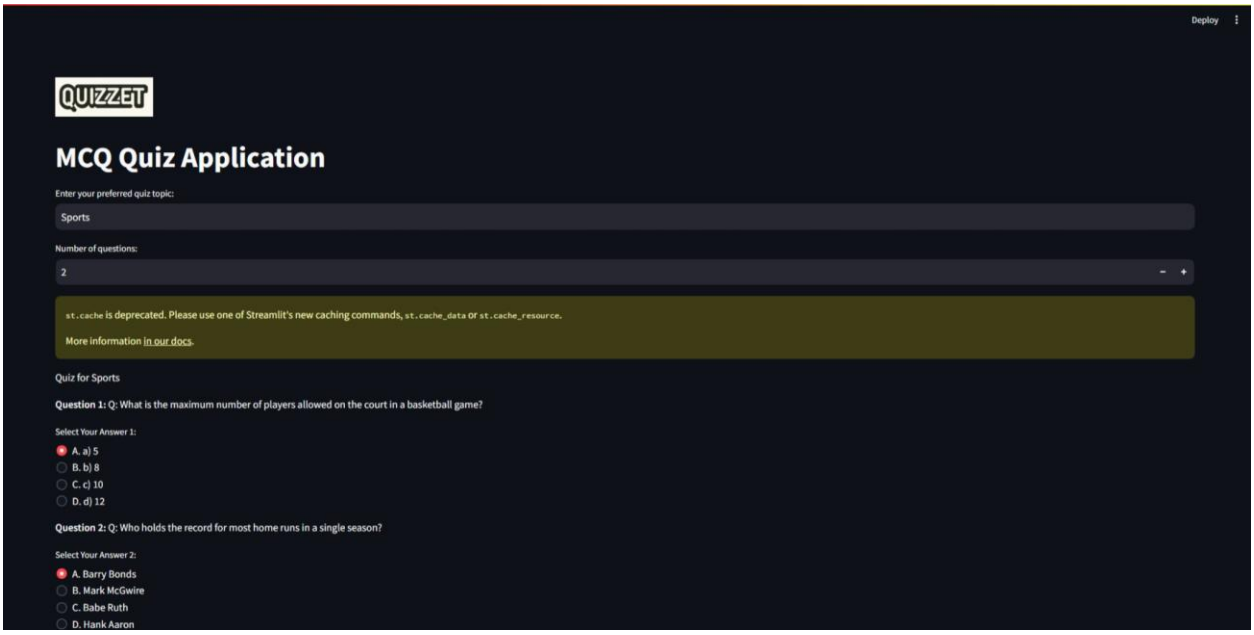
f. User Interaction and Quiz Submission:

The backend logic facilitates user interactions by dynamically adjusting to user inputs, such as the selected quiz topic and the number of questions. Upon submission of the quiz, the backend calculates the user's score by comparing their selected answers with the correct answers stored in the quiz data structure.

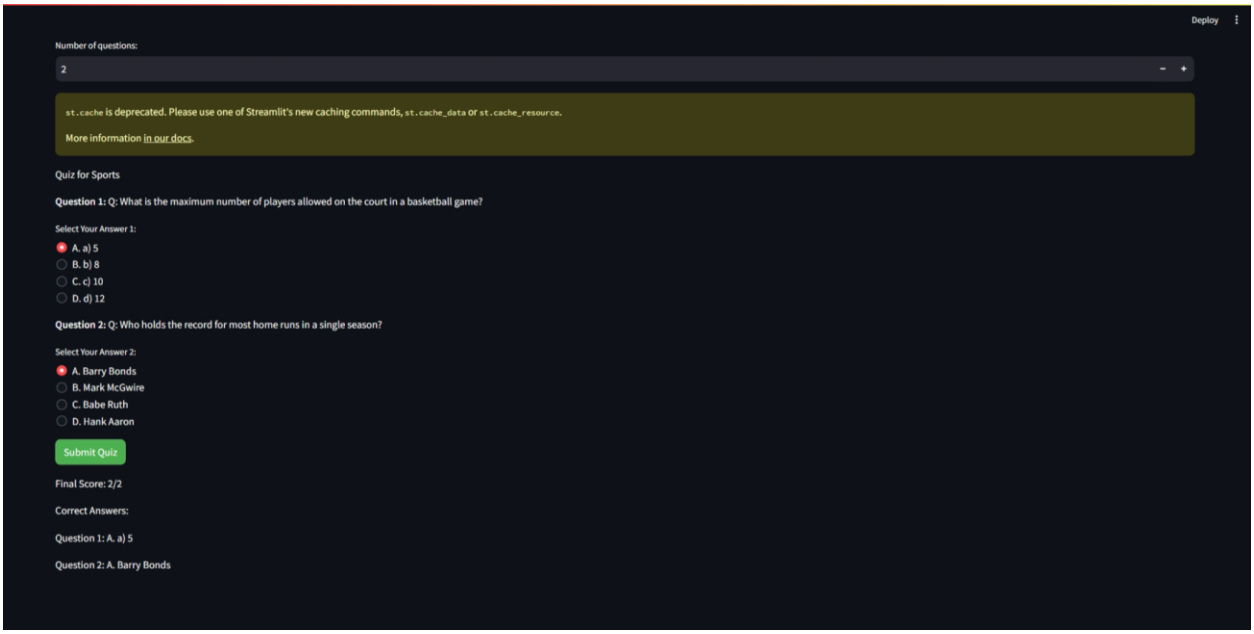
8. User Guide



The first page response will request the user to enter the topic name of the quiz to prepare the module of the topic.



Then the page will request the user to enter the number of questions to generate random questions and multiple-choice answers to let the user start to answer.



Then the user will submit the answers to get his score calculated from the backend code based on the correct and wrong answers selected by the user.

9. Code Documentation

```python
#Importing necessary libraries and modules
import openai
import langchain
import streamlit as st

#Setting OpenAI API key
openai.api_key = "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"

#Function to generate quiz data dynamically based on user's topic of interest
def generate_quiz_data(topic, num_questions):
    quiz_data = []

    #Iterating through the specified number of questions
    for _ in range(num_questions):
        #Constructing a prompt for the OpenAI Chat Completion API
        prompt = f"Generate a quiz question and answer options about {topic}"

        #Making a request to the OpenAI API to generate quiz content
        response = openai.Completion.create(
            engine="text-davinci-003",
            prompt=prompt,
            max_tokens=150,
            n=1,
            stop=None,
            temperature=0.7,
        )

        #Extracting the question and answer options from the API response
        question, options = extract_question_and_options(response.choices[0].text.strip())

        #Appending the generated quiz data to the list
        quiz_data.append({"question": question, "options": options})

    return quiz_data

#Function to extract question and answer options from the API response
def extract_question_and_options(response_text):
    lines = response_text.split("\n")

    #The first line is considered the question
    question = lines[0]

    #Extracting options excluding empty lines and those containing '?'
    options = [option for option in lines[1:] if option.strip() and '?' not in option]

    return question, options

#Main function to set up Streamlit interface and execute the application
def main():
    #Configuring Streamlit page settings
    st.set_page_config(
        page_title="MCQ Quiz App",
        page_icon=":bulb:",
        layout="wide",
        initial_sidebar_state="collapsed",
    )
```

```
57
58      #Adding custom CSS styles to the interface
59      st.markdown(
60          """
61          <style>
62          body {
63              color: #333;
64              background-color: #f0f0f0;
65          }
66          .stButton>button {
67              background-color: #4CAF50;
68              color: white;
69              font-size: 16px;
70          }
71          </style>
72          """,
73          unsafe_allow_html=True,
74      )
75
76      #Displaying the logo image
77      st.image("logo.png", width=150)
78
79      #Setting the main title for the application
80      st.title("MCQ Quiz Application")
81
82      #Text input for the user to enter their preferred quiz topic
83      topic = st.text_input("Enter Quiz Topic:")
84
85      #Checking if a topic has been entered
86      if topic:
87          #Number input for the user to specify the number of questions
88          num_questions = st.number_input("Number of questions:", min_value=1, step=1, value=2)
89
90          #Caching the quiz data to improve performance
91          @st.cache(allow_output_mutation=True)
92          def load_quiz_data():
93              return generate_quiz_data(topic, num_questions)
94
95          #Loading the quiz data
96          quiz_data = load_quiz_data()
97
98          #Displaying the quiz title
99          st.write(f"Quiz For {topic}")
100
101         #List to store user answers
102         user_answers = []
103
104         #Iterating through the quiz data to present questions and collect user answers
105         for i, question_data in enumerate(quiz_data):
106             question = question_data["question"]
107             options = question_data["options"]
108
109             #Displaying each quiz question and options
110             st.write(f"\n**Question {i + 1}:** {question}")
111             user_answer = st.radio(f"Select Answer {i + 1}:", options, key=f"question_{i}")
112             user_answers.append(user_answer)
113
```

```
        #Button to submit the quiz
        submit_button = st.button("Submit Quiz")

        #Handling quiz submission
        if submit_button:
            score = 0

            #Calculating the user's score by comparing answers with correct answers
            for i, question_data in enumerate(quiz_data):
                correct_answer = question_data["options"][0]
                if user_answers[i] == correct_answer:
                    score += 1

            #Displaying the final score
            st.write(f"Final Score: {score}/{len(quiz_data)}")

            #Displaying correct answers
            st.write("Correct Answers:")
            for i, question_data in enumerate(quiz_data):
                st.write(f"Question {i + 1}: {question_data['options'][0]}")
    else:
        #Prompting the user to enter a topic if none is provided
        st.write("Enter Topic Name To Generate Quiz Questions")

#Executing the main function if the script is run directly
if __name__ == "__main__":
    main()
```

10. Conclusion

Quizzet successfully brings together the power of Python, Langchain, and the OpenAI Chat Completion API to create a dynamic and engaging quiz experience for users. The integration of these technologies allows the application to intelligently generate contextually relevant quiz questions and answer options based on the user's chosen topic. The Streamlit framework further enhances the user experience by providing an intuitive and visually appealing interface.

Achievements:

- **Dynamic Quiz Generation:** The application excels in dynamically generating quiz content, allowing users to interactively participate in quizzes tailored to their specific areas of interest.

- **User-Friendly Interface:** The Streamlit interface offers a seamless and user-friendly experience. Users can easily input their preferred quiz topics, specify the number of questions, and navigate through the quiz presentation.

- **Real-Time Feedback:** The application provides real-time feedback to users by calculating and displaying their scores upon quiz submission. Additionally, correct answers are revealed, fostering a learning experience.

Areas for Improvement:

- **Scalability:** As the application grows, scalability considerations should be addressed. This includes optimizing backend processes and potentially exploring ways to parallelize or distribute the quiz generation tasks.

- **Enhanced User Engagement:** Future iterations of the application could explore features to enhance user engagement, such as timed quizzes, multimedia elements, or a broader range of question types.

- **Customization Options:** Introducing more customization options for users, such as the ability to select difficulty levels or quiz formats, can further personalize the quiz-taking experience.

Future Developments:

- **Integration of Additional APIs:** Consideration could be given to integrating additional APIs or services to expand the types of questions and content that can be included in the quizzes.

- **User Accounts and Tracking:** Implementing user accounts and tracking functionality could enable users to save their quiz history, track progress, and revisit previous quizzes.

- **Community Features:** Introducing community features, such as leaderboards, sharing quiz results, or collaborative quizzes, could enhance the social aspect of the application.

Quizzet lays a solid foundation for interactive and personalized quiz experiences. It successfully meets the current project requirements and serves as a launching point for future developments and enhancements. By leveraging cutting-edge technologies and prioritizing user experience, the application offers a compelling platform for knowledge exploration and engagement.