MCQ Generator

Table of contents

1. What is MCQ Generator?
    a. Key Features
    b. Purpose
2. Sequence Diagram for DevOps
3. Sequence Diagram
4. Setting Up the MCQ Generator
    a. The UI
5. Test Case

What is MCQ?

The Multiple-Choice Question (MCQ) Generator is an interactive quiz generation tool designed to create quizzes for a wide range of topics based on user preferences. Developed using the LangChain programming language and integrated with the Streamlit framework for a user-friendly interface, this generator empowers users to quickly and effortlessly generate quizzes tailored to their specific interests or learning needs.

Key Features:

Topic Customization:

Users can input their desired quiz topic, allowing the generator to focus on generating questions relevant to the chosen subject matter.

Dynamic Question Quantity:

The generator provides flexibility in the number of questions to include in the quiz, enabling users to specify the desired quantity for a comprehensive or concise quiz experience.

Effortless User Interaction:

The user interface, built with Streamlit, offers a seamless experience. The sidebar provides an intuitive input form for topic selection and question quantity, ensuring a user-friendly and engaging interaction.

Automatic Question Generation:

Leveraging the LangChain programming language, the MCQ Generator automatically generates multiple-choice questions based on the user's input. This functionality streamlines the quiz creation process, saving time and effort.

Real-time Answer Evaluation:

As users progress through the quiz, the MCQ Generator evaluates and scores their responses in real-time. This immediate feedback enhances the learning experience, allowing users to track their performance.

Clear Scoring Display:

At the end of the quiz, the generator provides a clear and concise summary of the user's performance, displaying the total score achieved out of the specified number of questions.

Purpose:

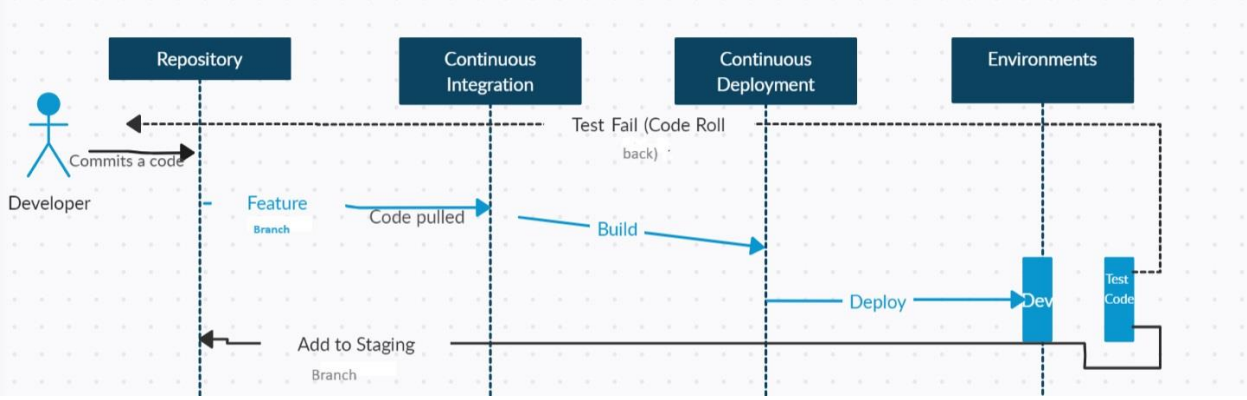
The MCQ Generator serves as a valuable tool for educators, learners, and quiz enthusiasts by providing a quick and efficient means of creating quizzes on diverse topics. Whether for educational purposes, self-assessment, or simply for fun, this generator simplifies the quiz creation process, making learning and testing knowledge an enjoyable experience.

Sequence Diagram for DevOps

Streamlit App Deployment with Azure Container Registry and GitHub Actions

Introduction

This document outlines the process of deploying a Streamlit app containerized using Docker on Azure Container Registry (ACR) and automated using GitHub Actions.



1. Created an Azure Container Registry:

Navigated to the Azure Portal and initiated the creation of a new ACR resource.

Assigned a unique name to the ACR and selected the appropriate subscription and resource group.

Configured access control for the ACR.

2. Built and Pushed Docker Image:

Utilized the Dockerfile provided with the Streamlit app to build the Docker image.

Tagged the Docker image with the ACR repository URL and a unique tag.

Executed the docker push command to push the Docker image to the designated ACR repository.

3. Created GitHub Actions Workflow:

Created a new GitHub repository to house the Streamlit app.

Accessed the Actions tab within the GitHub repository.

Established a new workflow file (e.g., deploy.yml) for automated deployment.

Defined the workflow steps, encompassing:

Checking out the code from the repository.

Setting up the Docker environment.

Retrieving the Docker image from ACR.

Deploying the Docker image to Azure App Service.

4. Connected GitHub Actions to Azure App Service:

Integrated the azure/webapp-deploy@v2 action into the GitHub Actions workflow for deployment tasks.

Supplied the Azure App Service connection details, including subscription ID, resource group name, and app service name.

Configured deployment parameters, such as the deployment method and target environment.

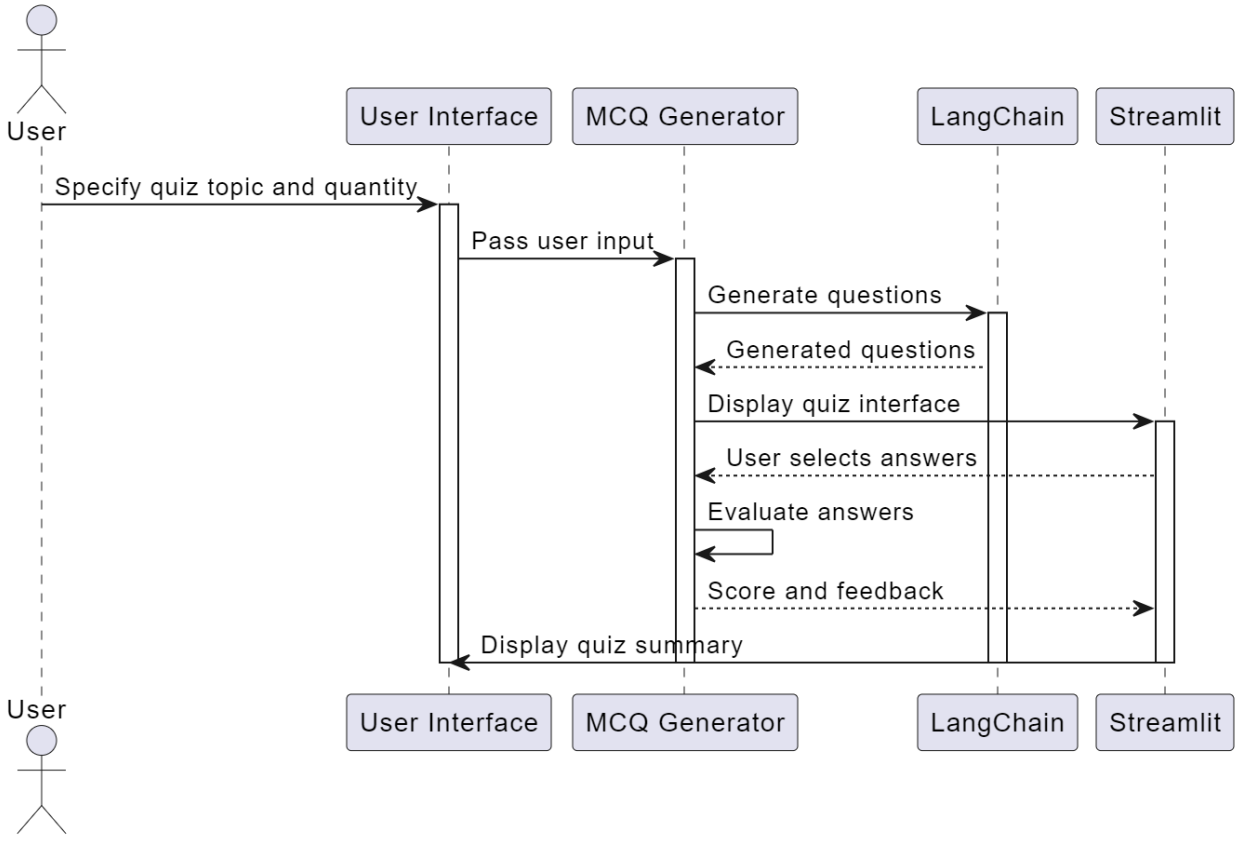
5. Triggered Automated Deployment:

Initiated a commit and push of any changes made to the Streamlit app code.

GitHub Actions automatically detected the changes and initiated the deployment workflow.

The workflow built the Docker image, pushed it to ACR, and deployed it to Azure App Service.

Sequence Diagram



Setting Up the MCQ Generator(locally)

Step 1: Downloading the Project

Obtain the project either from the GitHub repository or through email.

Step 2: Setting Up the Environment

Navigate to the project directory and locate the requirements.txt file.

Download all the required packages listed in the requirements.txt file to set up the project environment.

Step 3: API Key Configuration

Open the main.py file and locate line 14.

Add the necessary API key for the openAI API in the designated section.

Step 4: Terminal Setup

Open a terminal in the project directory.

Step 5: Running the Script

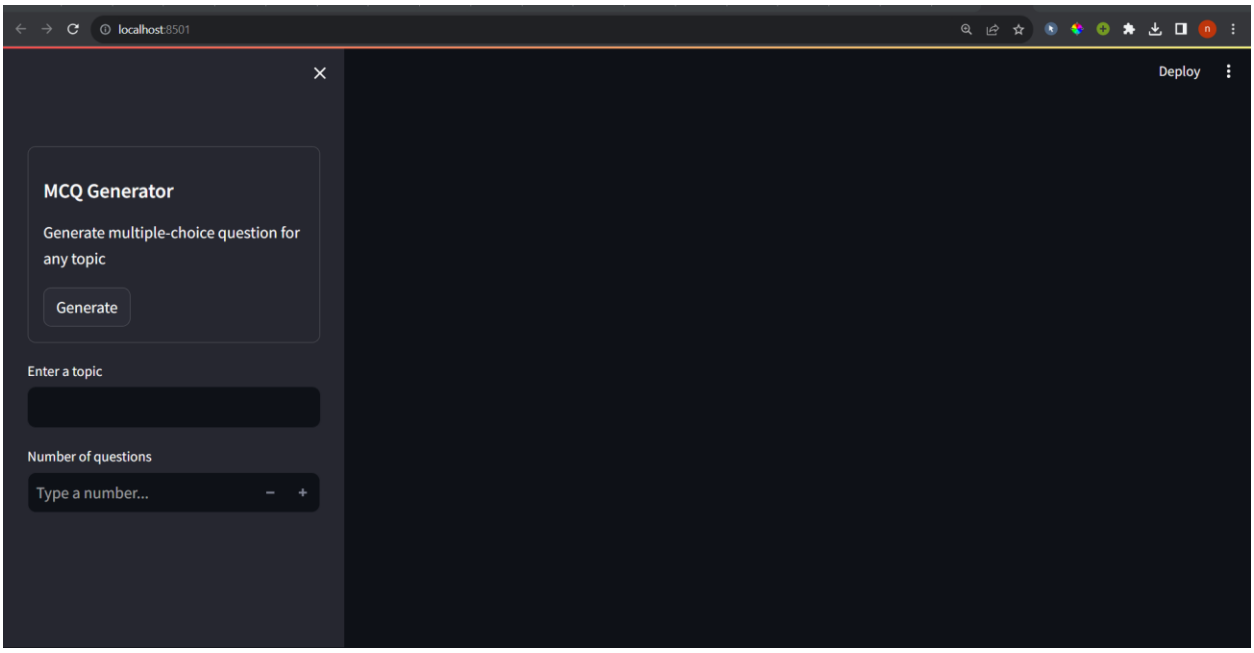
Execute the following command in the terminal:

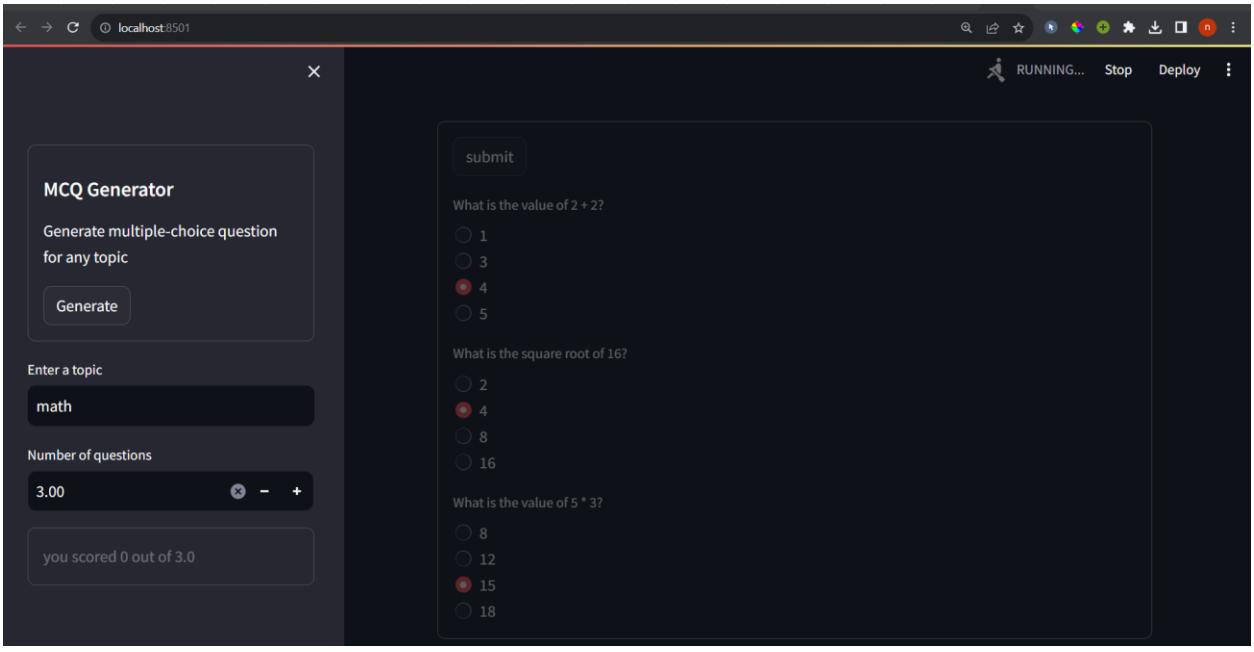
```
streamlit run ui.py
```

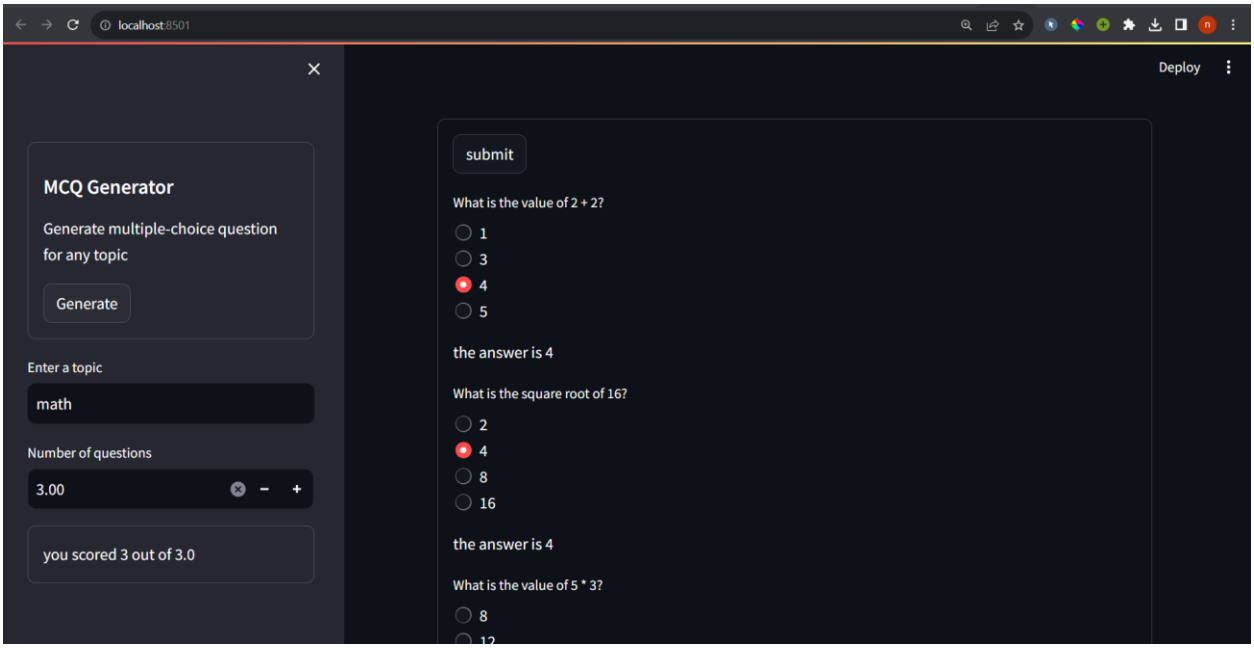
This command initiates the execution of the Streamlit application.

Following these steps ensures that you have the required packages installed, the API key configured, and the Streamlit application ready to run, providing a seamless setup for the MCQ Generator.

The UI


localhost:8501
Deploy
MCQ Generator
Generate multiple-choice question for any topic
Generate
Enter a topic
Number of questions
Type a number...


localhost:8501
RUNNING...
Stop
Deploy
MCQ Generator
Generate multiple-choice question for any topic
Generate
Enter a topic
math
Number of questions
3.00
you scored 0 out of 3.0
submit
What is the value of 2 + 2?
1
3
4
5
What is the square root of 16?
2
4
8
16
What is the value of 5 * 3?
8
12
15
18


localhost:8501
Deploy
MCQ Generator
Generate multiple-choice question for any topic
Generate
Enter a topic
math
Number of questions
3.00
you scored 3 out of 3.0
submit
What is the value of 2 + 2?
1
3
4
5
the answer is 4
What is the square root of 16?
2
4
8
16
the answer is 4
What is the value of 5 * 3?
8

Test Case

Test Case	Objective	Procedure	Expected Output	Observation
Random Subject Test	Verify behavior for a random subject	Selected a highly random subject	OpenAI returns outputs in the correct JSON format	Outputs may not consistently adhere to the expected JSON format for extremely random subjects.
Duplicate Question Test	Examine handling of the same topic	Entered the same topic for multiple quiz generations	Diverse questions despite the same topic, accounting for variations due to the model's temperature setting	Some questions may be repeated due to the model's temperature setting when the same topic is entered.
Basic Input Test	Validate functionality with fundamental input	Provided basic input parameters for quiz generation	Flawless generation of questions and accurate scoring	Performs well with basic input, generating diverse questions and providing accurate scoring.

Notes:

Consider the randomness inherent in the model's responses, especially for highly diverse or obscure topics.

Duplicated questions may occur when entering the same topic due to the OpenAI model's temperature parameter.