Project Report

Date	20 July 2024
Team ID	SWTID1720448590
Project Name	**Inquisitive:** A Multilingual AI Question Generator Using PaLM’s Text-Bison-001
Maximum Marks	4 Marks

TEAM DETAILS:

NAME	REGISTRATION NO.
NIHAL JOHANN THOMAS	21MIC0180
SRIJA BASAK	21BCE3506
KIRAN R	21MIC0167
RAGUL G	21MIC0174

Overview:

- Inquisitive leverages the robust capabilities of the PaLM architecture to analyze user input text and autonomously generate questions from it.
- This multilingual application enables seamless interaction, extracting key information from the text to formulate relevant questions in various languages.
- It enhances comprehension and engagement through dynamic question generation, facilitating deeper exploration of textual content across linguistic barriers.
- Inquisitive leverages the robust capabilities of the PALM architecture to analyze user input text and autonomously generate questions from it.

- This multilingual application enables seamless interaction, extracting key information from the text to formulate relevant questions in various languages.
- It enhances comprehension and engagement through dynamic question generation, facilitating deeper exploration of textual content across linguistic barriers.

POTENTIAL APPLICATIONS OF INQUISITIVE

Scenario 1: Corporate Training Programs

Enhance corporate training programs by incorporating Inquisitive: Employees can input training materials, and the application generates quizzes, fostering active learning and reinforcing key concepts. This automated process saves time for trainers and encourages self-paced learning, leading to improved knowledge retention and skill development.

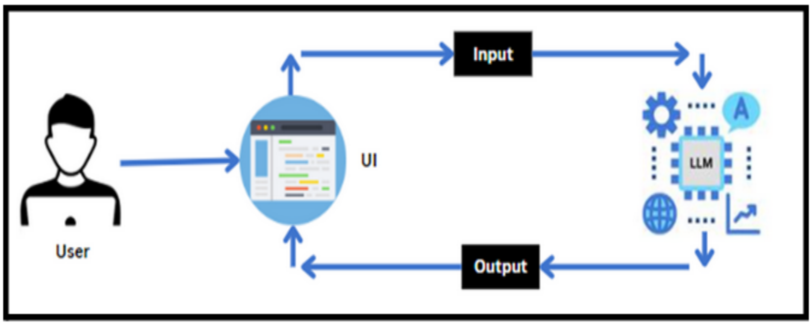
Scenario 2: Content Creation for Marketing

In marketing, utilize Inquisitive to transform product descriptions or promotional content into interactive quizzes or FAQs. This engages customers, encourages interaction with the brand, and provides valuable insights into consumer preferences and understanding. The dynamic nature of generated questions keeps content fresh and engaging, driving customer interest and retention.

Scenario 3: Knowledge Management in Meetings

During corporate meetings, employ Inquisitive to summarize discussions in real-time and generate follow-up questions. This ensures thorough understanding among participants, stimulates critical thinking, and clarifies any ambiguities. By automating question generation, the tool streamlines the meeting process, promotes active engagement, and facilitates deeper exploration of topics, ultimately leading to more productive and insightful discussions.

TECHNICAL ARCHITECTURE :



PRE-REQUISITE KNOWLEDGE

- **LLM & PALM: https://cloud.google.com/vertex-ai/docs/generative-ai/learn-resources**
- **Google Translate API: https://pypi.org/project/googletrans/**
- **Streamlit: https://www.datacamp.com/tutorial/streamlit**

PROJECT FLOW

Step 1: Users input text into the UI of Inquisitive.

Step 2: The input text is then processed and analyzed by the PaLM architecture, which is integrated into the backend.

Step 3: PaLM autonomously generates questions based on the input text.

Step 4:The generated questions are sent back to the frontend for display on the UI.

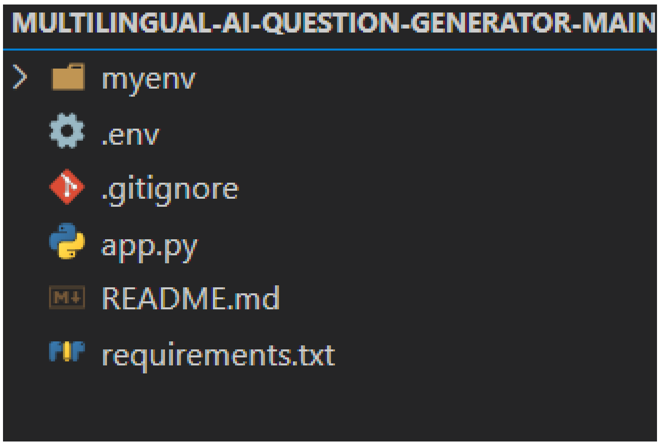
Step 5: Users can view the dynamically generated questions and interact with them to gain deeper insights into the content.

To accomplish this, the activities listed below are done for this project,

- **Initializing the PaLM**
  - Generate PaLM API
  - Initialize the pre-trained model
- **Interfacing with Pre-trained Model**
  - Questions Generator
- **Model Deployment**
  - Deploy the application using Streamlit

PROJECT STRUCTURE

The project folder containing the files as shown below is created:



```
MULTILINGUAL-AI-QUESTION-GENERATOR-MAIN
> myenv
  .env
  .gitignore
  app.py
  README.md
  requirements.txt
```

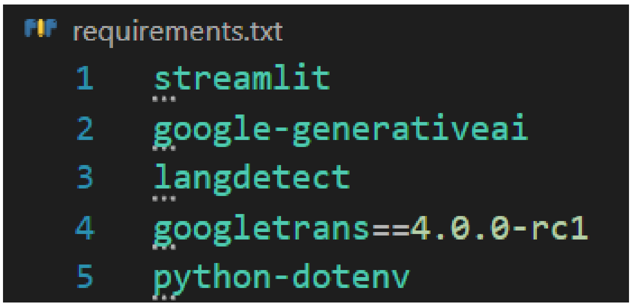
app.py: Main application file containing the code for Inquisitive. It serves as the primary application file housing both the model and Streamlit UI code.

requirements.txt: File listing all the Python dependencies required for the project.

.env: Environment file containing the `API_KEY` used for configuring the Google Generative AI.

1. REQUIREMENTS SPECIFICATIONS

Creation of a requirements.txt file is done to list the required libraries



```
streamlit
google-generativeai
langdetect
googletrans==4.0.0-rc1
python-dotenv
```

streamlit: Streamlit is a powerful framework for building interactive web applications with Python.

google-generativeai: Python client library for accessing the GenerativeAI API, facilitating interactions with pre-trained language models like Gemini Pro.

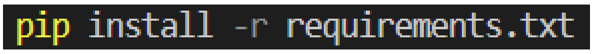
langdetect: Language detection library is used to identify the language of a given text.

googletrans==4.0.0-rc1: Google Translate API wrapper for Python allows translation of text between different languages using Google Translate service.

python-dotenv: python-dotenv is used for managing environment variables in Python projects

Installation of the required libraries

- Open the terminal.
- Run the command: pip install -r requirements.txt
- This command installs all the libraries listed in the requirements.txt file



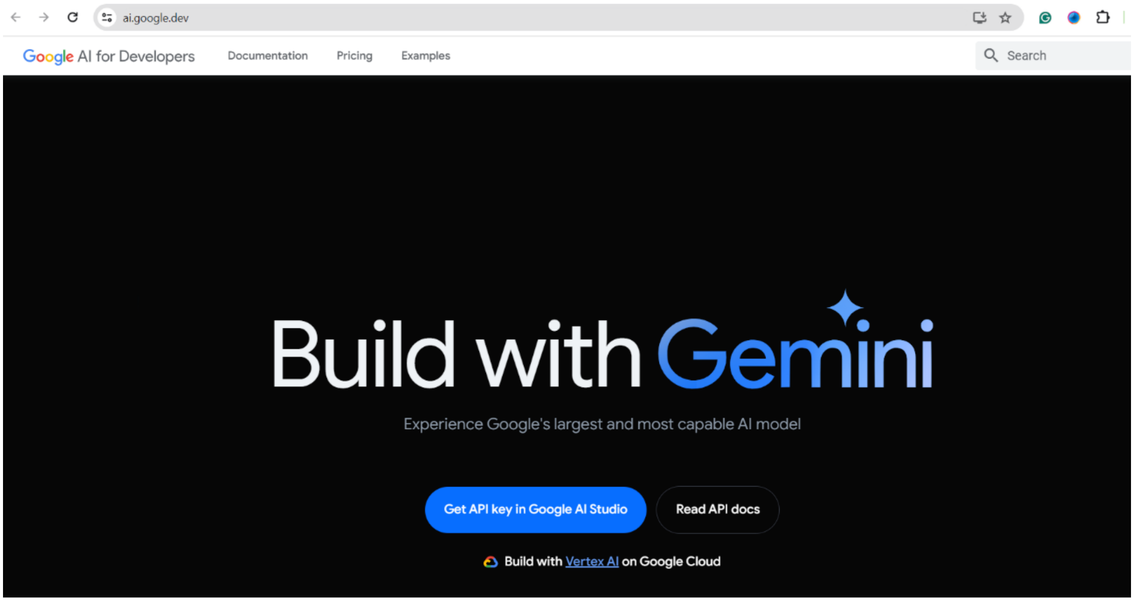
```
pip install -r requirements.txt
```

2. INITIALIZATION OF THE MODEL

- The Google API key is a secure access token provided by Google, enabling developers to authenticate and interact with various Google APIs.
- It acts as a form of identification, allowing users to access specific Google services and resources.
- This key plays a crucial role in authorizing and securing API requests, ensuring that only authorized users can access and utilize Google's services.

Generate PaLM API Key

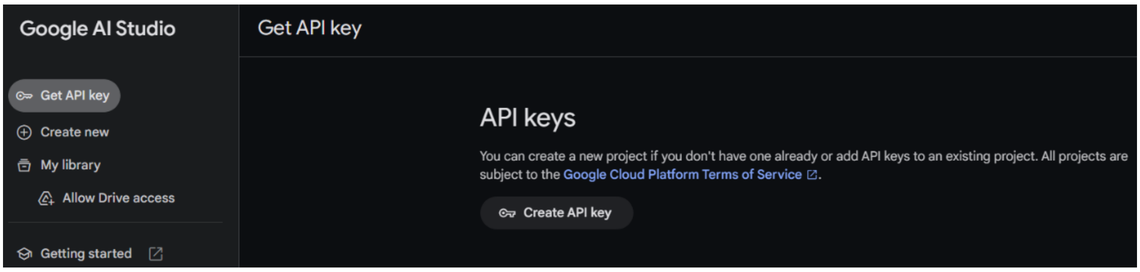
- Click on the link (https://developers.generativeai.google/).
- Then click on "Get API key in Google AI Studio".
- Click on "Get API key" from the right navigation menu.
- Now click on "Create API key". (Refer the below images)
- Copy the API key.



Build with Gemini

Experience Google's largest and most capable AI model

After signing in to your account, navigate to the 'Get an API Key' option. Clicking on this option will redirect you to another webpage as shown below.



API keys

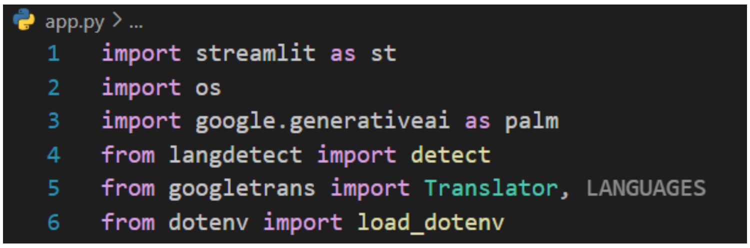
You can create a new project if you don't have one already or add API keys to an existing project. All projects are subject to the Google Cloud Platform Terms of Service.

Next, click on 'Create API Key' and choose the generative language client as the project. Then, select 'Create API key in existing project'.

Copy the newly generated API key as it is required for loading the pre-trained model.

Initialize the pre-trained model

i) Import necessary files:

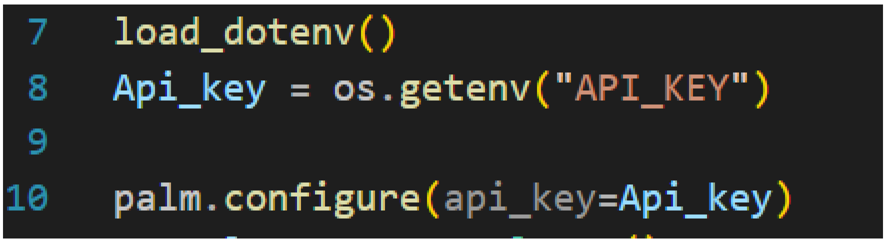


```python
import streamlit as st
import os
import google.generativeai as palm
from langdetect import detect
from googletrans import Translator, LANGUAGES
from dotenv import load_dotenv
```

- Streamlit, a popular Python library, is imported as st, enabling the creation of user interfaces directly within the Python script.
- Importing the os module: The os module in Python provides a way of using operating system-dependent functionality such as manipulating files and directories, working with processes, and environment variables
- Importing the palm module: This line imports the palm module from the google.generativeai package.
- The langdetect library is imported for language detection functionality, allowing the application to identify the language of user-input text.
- The code imports the Translator class from the googletrans library, enabling translation capabilities within the application, crucial for handling multilingual text input and output.

- Importing the dotenv module: The dotenv module in Python is used to load environment variables from a .env file into the os.getenv function. This is particularly useful in development and deployment scenarios where one wants to keep certain configuration variables (like API keys, database URIs, etc.) outside of one's main source code repository.

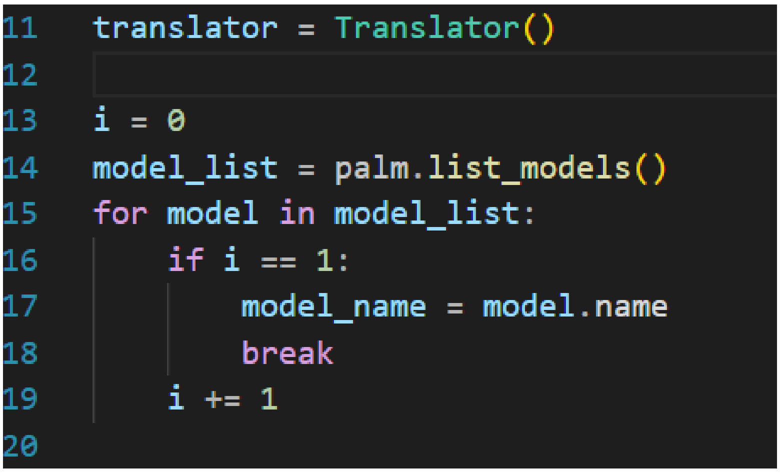
ii) Configuration of the PALM API with the API key and initialize translator:



```
load_dotenv()
Api_key = os.getenv("API_KEY")

palm.configure(api_key=Api_key)
```

- The API key is pasted to a .env file in the same project directory
- Configuring the API key: The configure function is used to set up or configure the Google API with an API key.
- The Translator class facilitates language translation capabilities within the application.

iii) Creating a translator object, then listing the available models and fetching a model name:



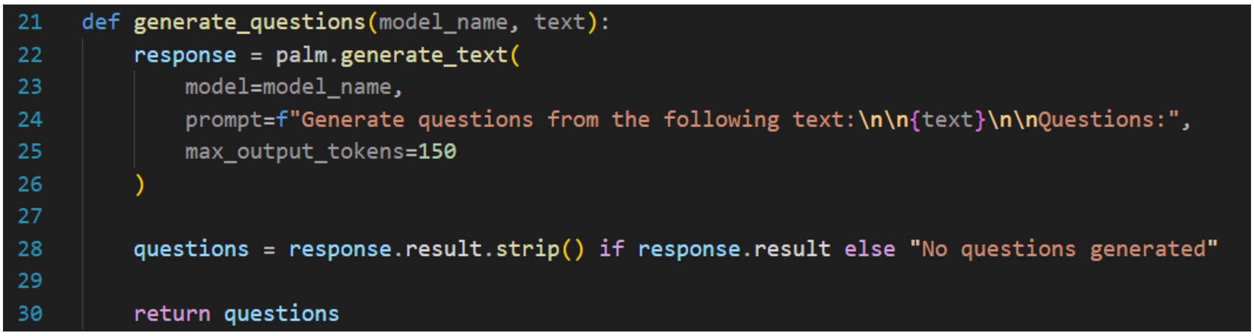
```
translator = Translator()

i = 0
model_list = palm.list_models()
for model in model_list:
    if i == 1:
        model_name = model.name
        break
    i += 1

```

- Listing available models: This line retrieves a list of available models using the list_models function provided by the palm module. It creates a list (models) containing information about each available model.

- Fetching a model name: It extracts the name of the second model from the list of models (models). Note that Python uses 0-based indexing, so models[1] refer to the second model in the list. The name of this model is then stored in the variable model_name.

3. INTERFACING WITH THE PRE-TRAINED MODEL:

Generate questions from text:



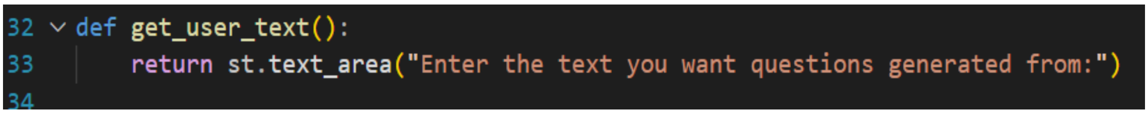
```
21  def generate_questions(model_name, text):
22      response = palm.generate_text(
23          model=model_name,
24          prompt=f"Generate questions from the following text:\n\n{text}\n\nQuestions:",
25          max_output_tokens=150
26      )
27
28      questions = response.result.strip() if response.result else "No questions generated"
29
30      return questions
```

- This function generate_questions takes two parameters: model_name, which specifies the pre-trained language model to be used, and text, which represents the input text from which questions are to be generated.
- It then utilizes the palm.generate_text() method to generate questions based on the input text.
- The prompt parameter provides a prompt for the model, instructing it to generate questions from the given text, and max_output_tokens limits the length of the generated output

- In line 28, This part of the code assigns the generated questions to the variable questions.
- It checks if the response.result attribute exists; if it does, it strips any leading or trailing whitespace from the result and assigns it to questions.
- If response.result is empty or doesn't exist, it assigns the string "No questions generated." to questions.

- Finally, generate_questions function returns the questions variable, containing either the generated questions or the "No questions generated." message.

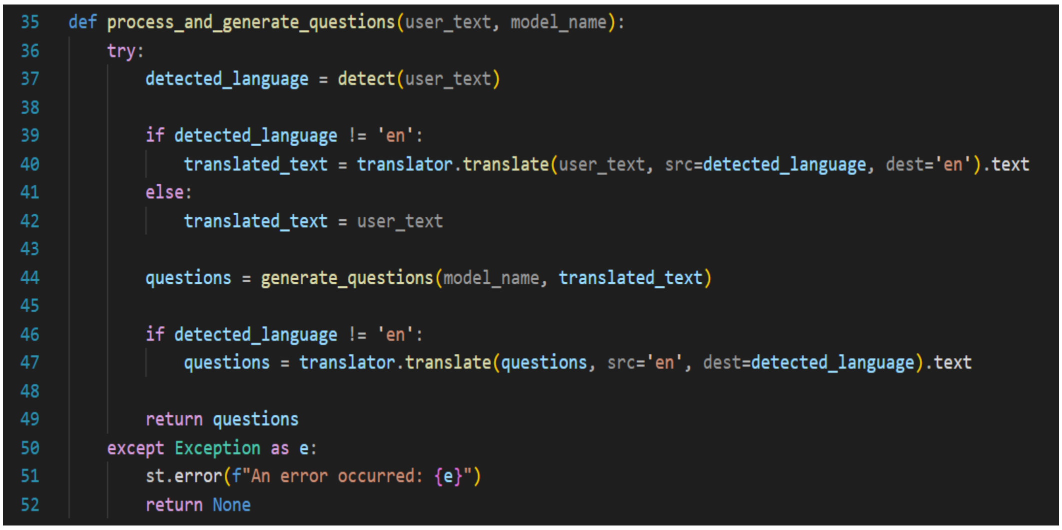
Function to get user text:



```
def get_user_text():
    return st.text_area("Enter the text you want questions generated from:")
```

- It utilizes the st.text_area() function to create a text area where users can enter their desired text

Function for language detection, translation and generating questions:

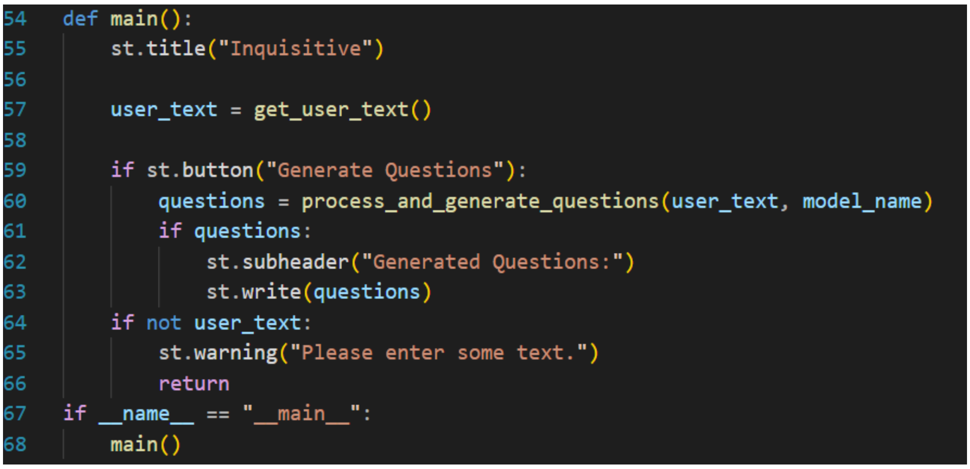


```
def process_and_generate_questions(user_text, model_name):
    try:
        detected_language = detect(user_text)

        if detected_language != 'en':
            translated_text = translator.translate(user_text, src=detected_language, dest='en').text
        else:
            translated_text = user_text

        questions = generate_questions(model_name, translated_text)

        if detected_language != 'en':
            questions = translator.translate(questions, src='en', dest=detected_language).text

        return questions
    except Exception as e:
        st.error(f"An error occurred: {e}")
        return None
```

- The detect method from the langdetect library detects the language of the text given by the user. If the language is not English, the translation of the text is done to English by the translator.translate() function, questions are generated by the generate_questions function (defined before in this project) in English and then translated back to the detected language.
- The translated text is then stored in the variable translated_text. If the detected language is already English, the original user text is retained in the translated_text variable.

- The function returns the questions generated

4. MODEL DEPLOYMENT



```python
def main():
    st.title("Inquisitive")

    user_text = get_user_text()

    if st.button("Generate Questions"):
        questions = process_and_generate_questions(user_text, model_name)
        if questions:
            st.subheader("Generated Questions:")
            st.write(questions)
    if not user_text:
        st.warning("Please enter some text.")
        return
if __name__ == "__main__":
    main()
```

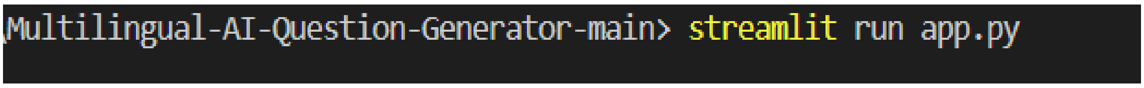
- In the main() function, the get_user_text() function is called to get text as input from the user.

Generate questions button:

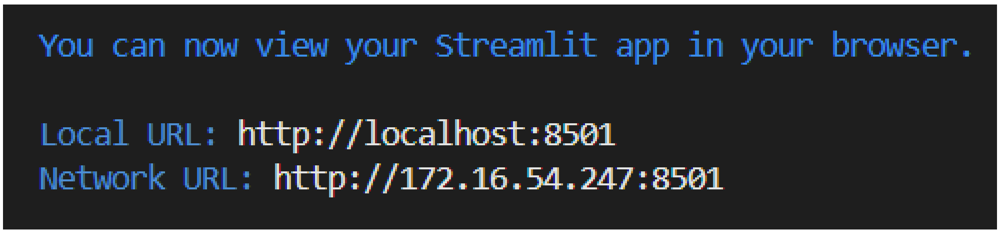
- In this block, process_and_generate_questions function is called. If questions are generated, then a subheader is displayed in the Streamlit app, and the generated questions are displayed in the Streamlit app. If no text is entered by the user, it shows a warning message prompting the user to enter some text.

Run the web application

- In the terminal in the VS code, activate the virtual environment using the command myenv\Scripts\activate (myenv is the name of the virtual environment created), if not already activated
- Now type “streamlit run app.py” command
- Navigate to the localhost where you can view your web page



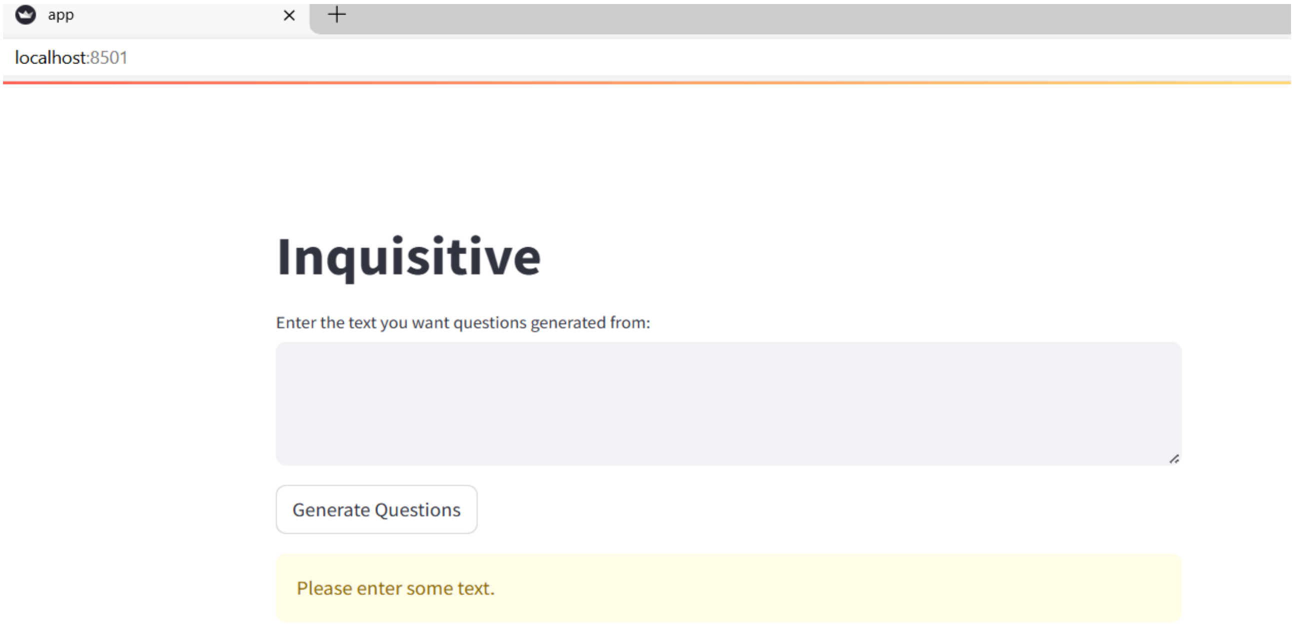
```
Multilingual-AI-Question-Generator-main> streamlit run app.py
```



```
  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://172.16.54.247:8501
```

Now, the application will open in the web browser.

Screenshots:

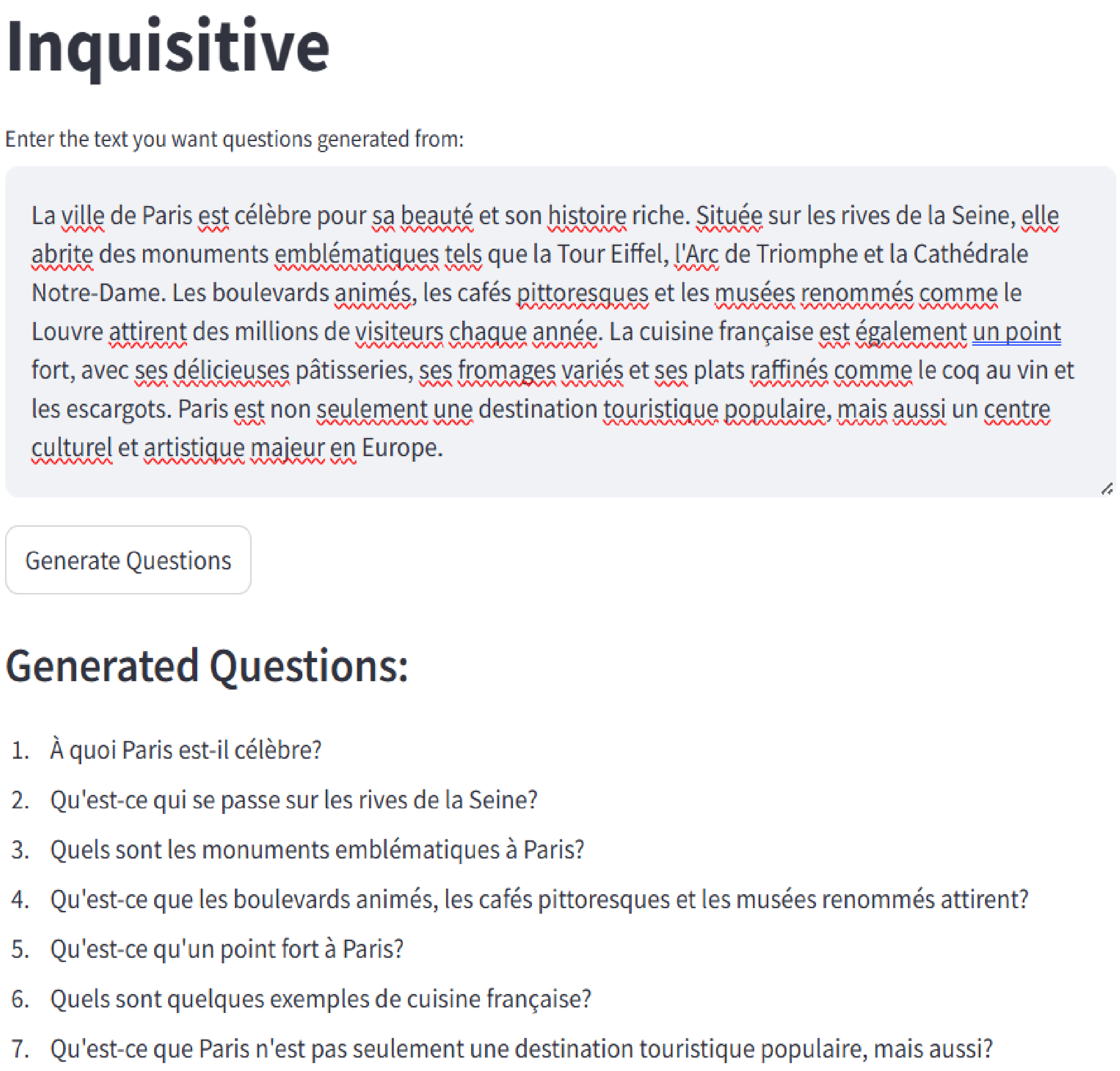


Inquisitive

Enter the text you want questions generated from:

Please enter some text.

Output 1: French Language:



Inquisitive

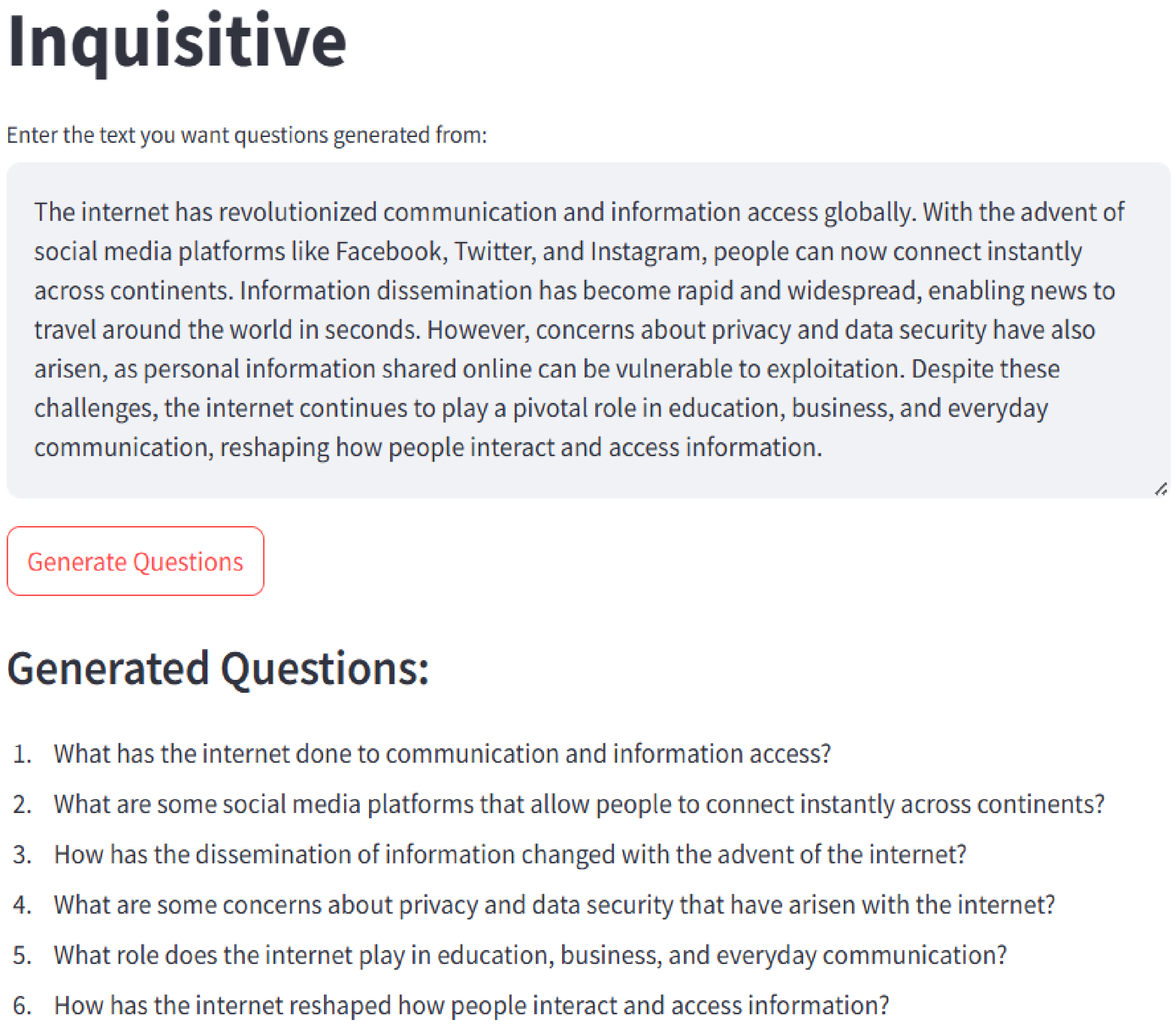
Enter the text you want questions generated from:

La ville de Paris est célèbre pour sa beauté et son histoire riche. Située sur les rives de la Seine, elle abrite des monuments emblématiques tels que la Tour Eiffel, l'Arc de Triomphe et la Cathédrale Notre-Dame. Les boulevards animés, les cafés pittoresques et les musées renommés comme le Louvre attirent des millions de visiteurs chaque année. La cuisine française est également un point fort, avec ses délicieuses pâtisseries, ses fromages variés et ses plats raffinés comme le coq au vin et les escargots. Paris est non seulement une destination touristique populaire, mais aussi un centre culturel et artistique majeur en Europe.

Generated Questions:

1. À quoi Paris est-il célèbre?
2. Qu'est-ce qui se passe sur les rives de la Seine?
3. Quels sont les monuments emblématiques à Paris?
4. Qu'est-ce que les boulevards animés, les cafés pittoresques et les musées renommés attirent?
5. Qu'est-ce qu'un point fort à Paris?
6. Quels sont quelques exemples de cuisine française?
7. Qu'est-ce que Paris n'est pas seulement une destination touristique populaire, mais aussi?

Output 2: English Language:



Inquisitive

Enter the text you want questions generated from:

The internet has revolutionized communication and information access globally. With the advent of social media platforms like Facebook, Twitter, and Instagram, people can now connect instantly across continents. Information dissemination has become rapid and widespread, enabling news to travel around the world in seconds. However, concerns about privacy and data security have also arisen, as personal information shared online can be vulnerable to exploitation. Despite these challenges, the internet continues to play a pivotal role in education, business, and everyday communication, reshaping how people interact and access information.

Generated Questions:

1. What has the internet done to communication and information access?
2. What are some social media platforms that allow people to connect instantly across continents?
3. How has the dissemination of information changed with the advent of the internet?
4. What are some concerns about privacy and data security that have arisen with the internet?
5. What role does the internet play in education, business, and everyday communication?
6. How has the internet reshaped how people interact and access information?

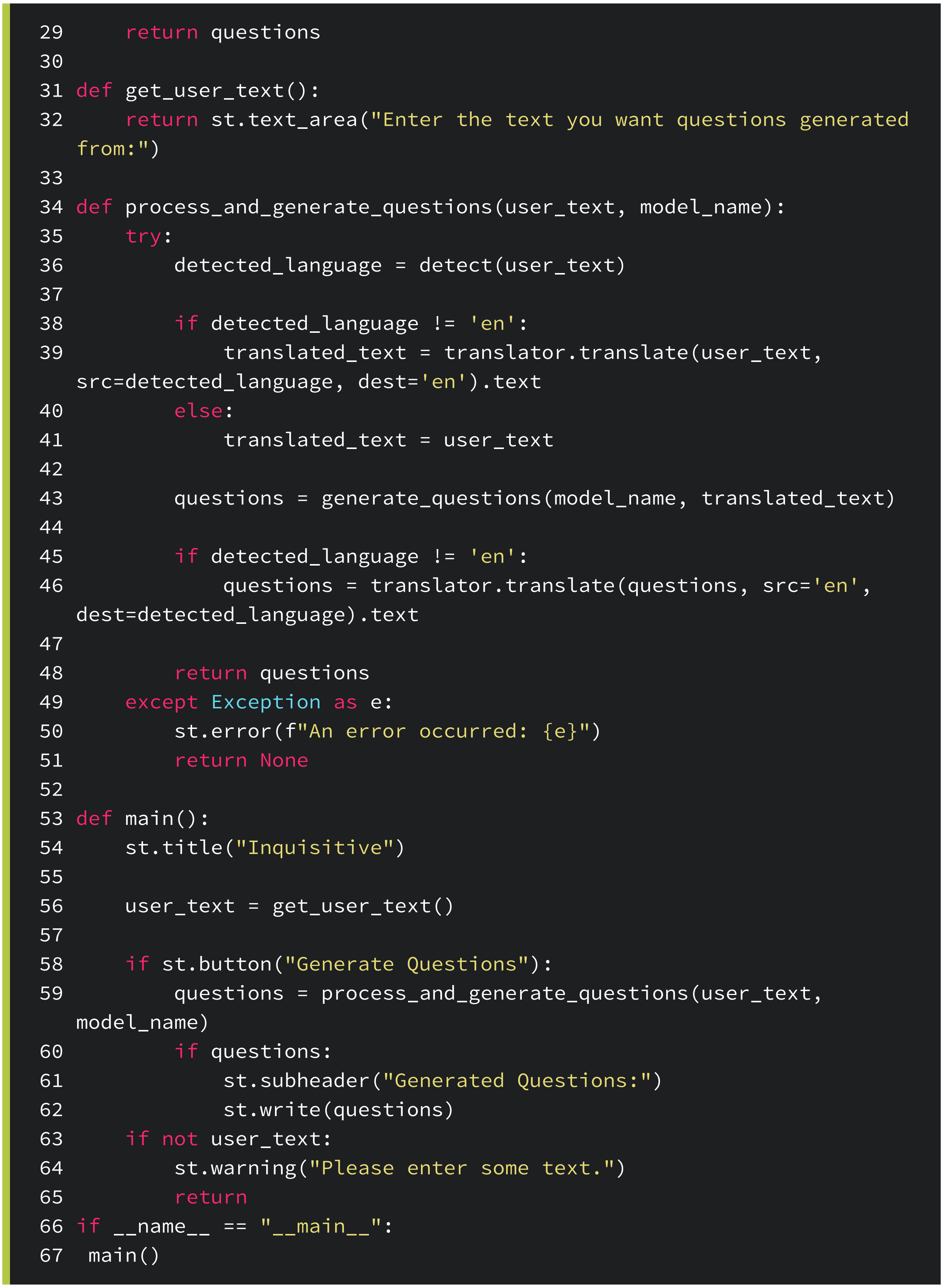
COMPLETE CODE FOR THE PROJECT

File: requirements.txt

```
streamlit
google-generativeai
langdetect
googletrans==4.0.0-rc1
python-dotenv
```

File: app.py

```python
import streamlit as st
import os
from dotenv import load_dotenv
import google.generativeai as palm
from langdetect import detect
from googletrans import Translator, LANGUAGES

load_dotenv()
Api_key = os.getenv("API_KEY")

palm.configure(api_key=Api_key)
translator = Translator()

i = 0
model_list = palm.list_models()
for model in model_list:
    if i == 1:
        model_name = model.name
        break
    i += 1

def generate_questions(model_name, text):
    response = palm.generate_text(
        model=model_name,
        prompt=f"Generate generate follow-up questions which are intriguing and thought-provoking from the provided context text:\n\n{text}\n\nQuestions:",
        max_output_tokens=150
    )
    questions = response.result.strip() if response.result else "No questions generated"
```



```python
    return questions

def get_user_text():
    return st.text_area("Enter the text you want questions generated from:")

def process_and_generate_questions(user_text, model_name):
    try:
        detected_language = detect(user_text)

        if detected_language != 'en':
            translated_text = translator.translate(user_text, src=detected_language, dest='en').text
        else:
            translated_text = user_text

        questions = generate_questions(model_name, translated_text)

        if detected_language != 'en':
            questions = translator.translate(questions, src='en', dest=detected_language).text

        return questions
    except Exception as e:
        st.error(f"An error occurred: {e}")
        return None

def main():
    st.title("Inquisitive")

    user_text = get_user_text()

    if st.button("Generate Questions"):
        questions = process_and_generate_questions(user_text, model_name)
        if questions:
            st.subheader("Generated Questions:")
            st.write(questions)
    if not user_text:
        st.warning("Please enter some text.")
        return
if __name__ == "__main__":
 main()
```