**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.

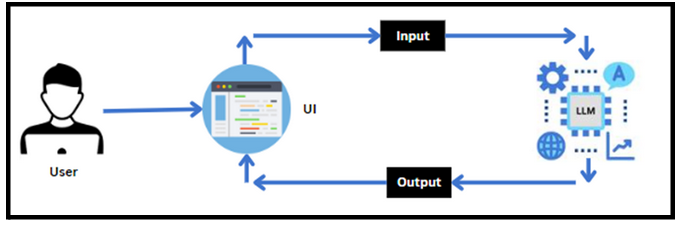
**Scenario 2: Content Creation for Marketing**

**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.

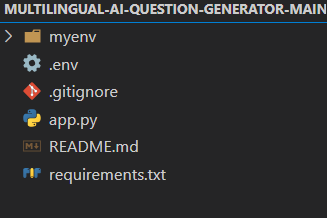
Step 5: Users can view the dynamically generated questions and interact with them to gain deeper insights into the content.

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:



**app.py**: Main application file containing the code for Inquisitive. It serves as

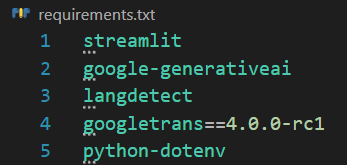
the primary application file housing both the model and Streamlit UI code.

**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:** 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

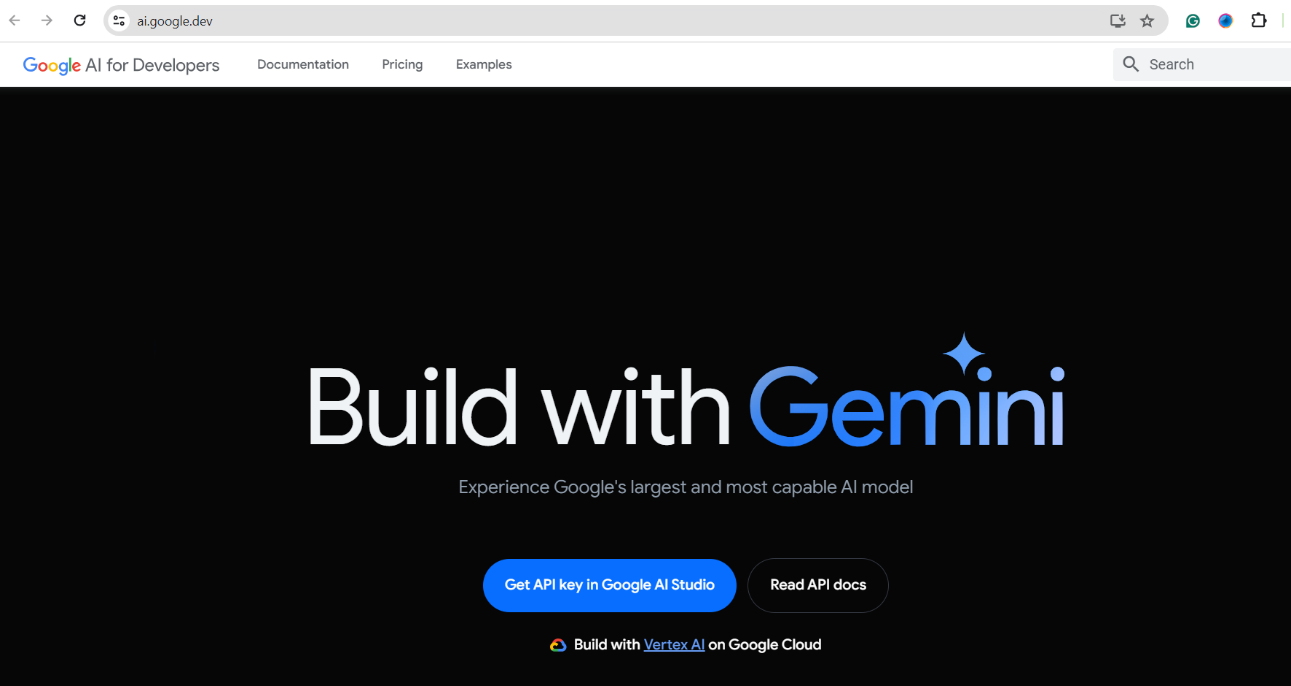


1. **INITIALIZATION OF THE MODEL**

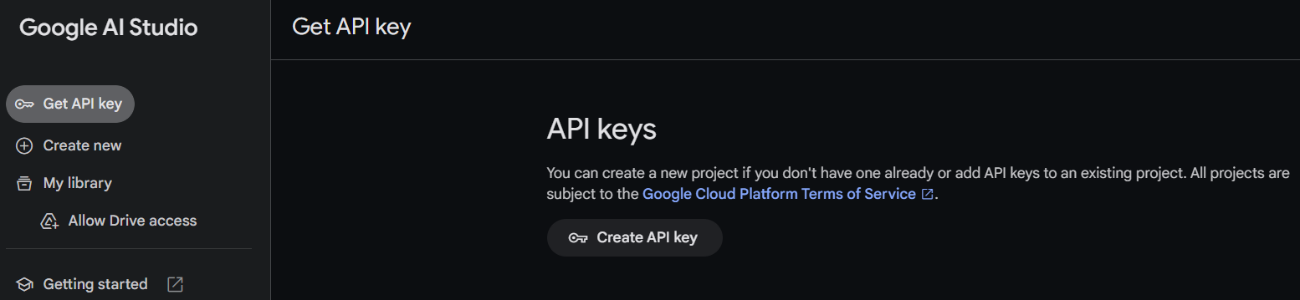
* The Google API key is a secure access token provided by Google, enabling developers to authenticate and interact with various Google APIs.
* It acts as a form of identification, allowing users to access specific Google services and resources.
* This key plays a crucial role in authorizing and securing 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/](https://skillwallet.smartinternz.com/Student/guided_project_workspace/(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.



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.

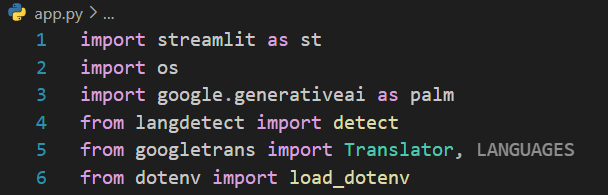


Next, click on 'Create API Key' and choose the generative language client as the project. Then, select 'Create API key in existing project'.

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

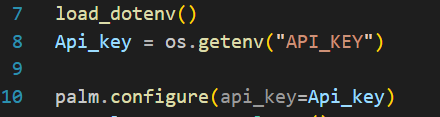
**Initialize the pre-trained model**

**i) Import necessary files:**



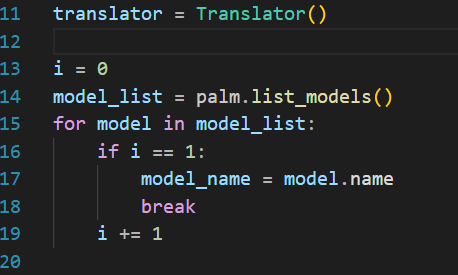
* Streamlit, a popular Python library, is imported as st, enabling the creation of user interfaces directly within the Python script.
* Importing the os module: The os module in Python provides a way of using operating system-dependent functionality such as manipulating files and directories, working 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:**



* 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.

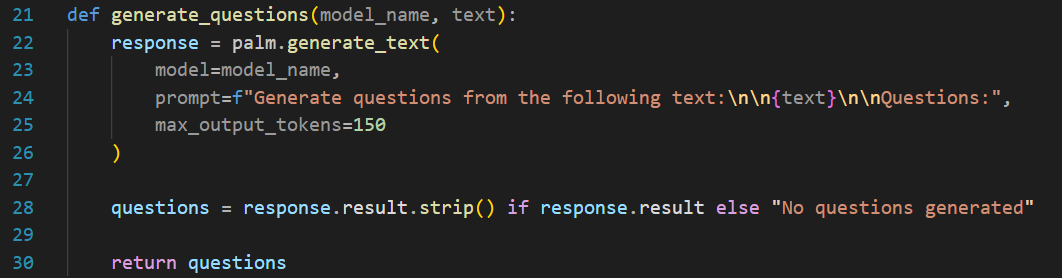
**iii) Creating a translator object, then listing the available models and fetching a model name:**



* 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.

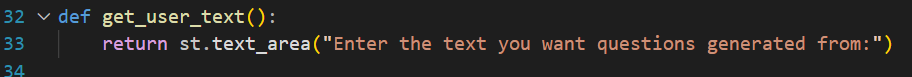
1. **INTERFACING WITH THE PRE-TRAINED MODEL:**

**Generate questions from text****:**



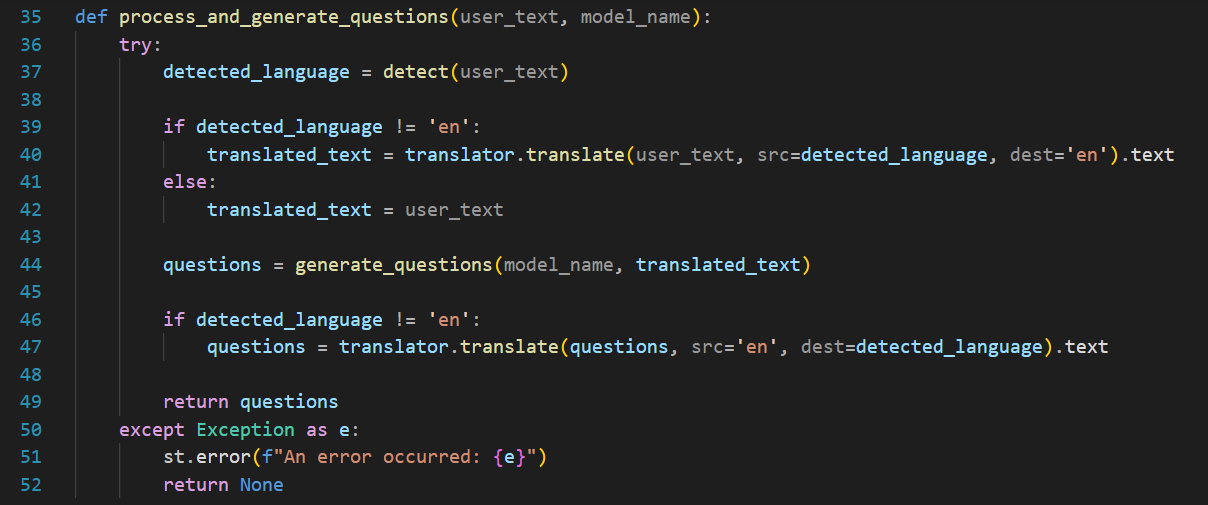
* This function generate\_questions takes two parameters: model\_name, which specifies the pre-trained language model to be used, and text, which represents the input text from which questions are to be generated.
* It then utilizes the palm.generate\_text() method to generate questions based on the 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:**



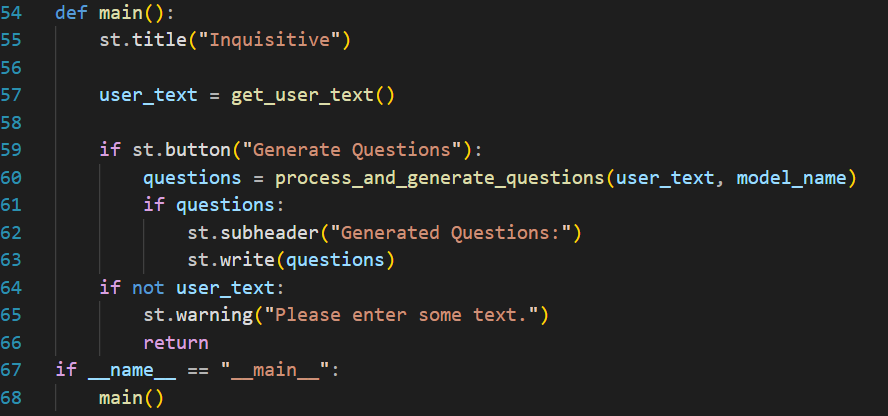
* It utilizes the st.text\_area() function to create a text area where users can enter their desired text

**Function for language detection, translation and generating questions:**



* 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

1. **MODEL DEPLOYMENT**



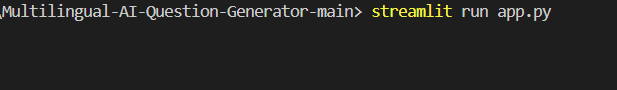
* In the main() function, the get\_user\_text() function is called to get text as input from the user.

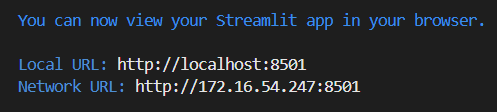
**Generate questions button:**

* 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**

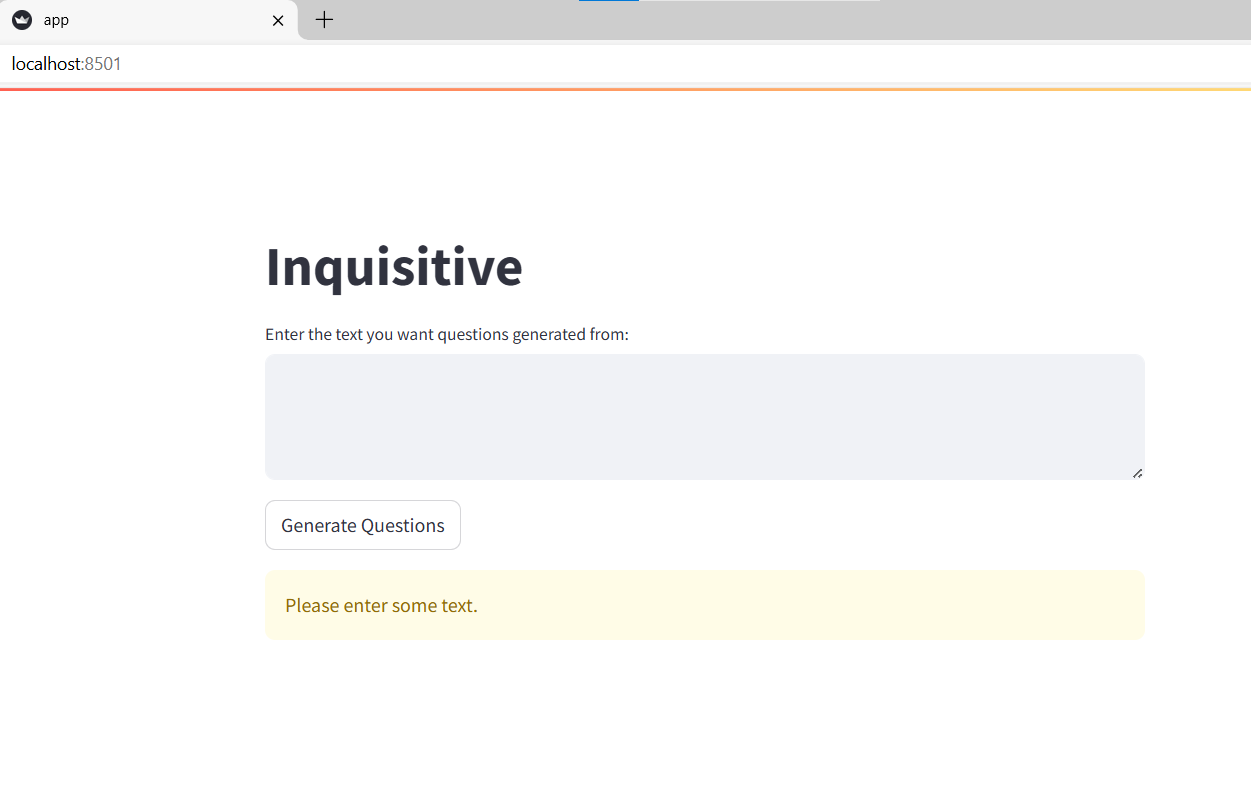
* In the terminal in the VS code, activate the virtual environment using the command myenv\Scripts\activate (myenv is the name of the virtual environment created), if not already activated
* Now type “streamlit run app.py” command
* Navigate to the localhost where you can view your web page



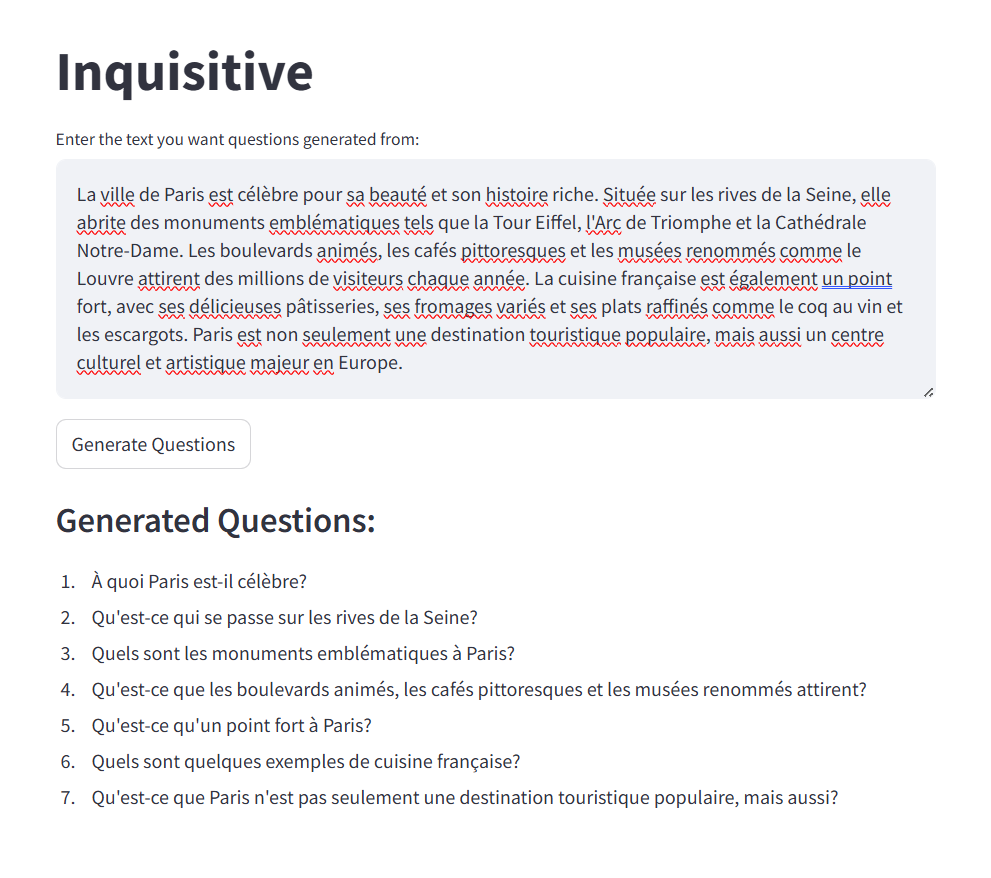


**Now, the application will open in the web browser.**

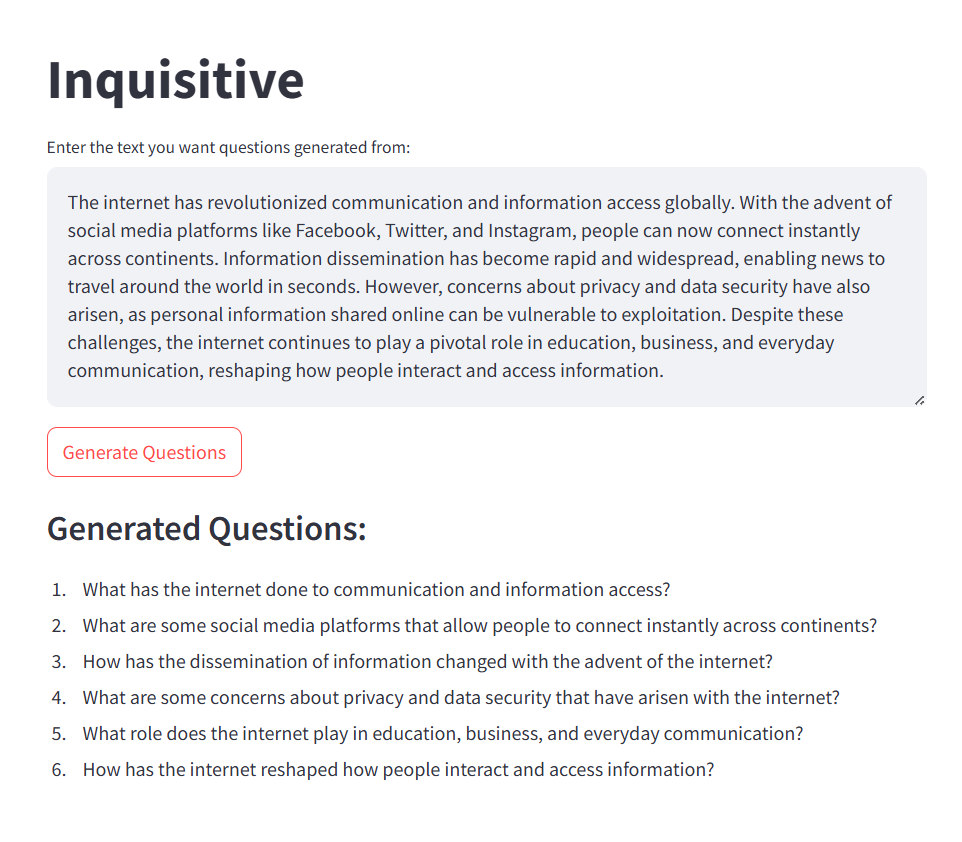
**Screenshots:**



**Output 1: French Language:**



**Output 2: English Language:**



**COMPLETE CODE FOR THE PROJECT**

**File: requirements.txt**

1. streamlit
2. google-generativeai
3. langdetect
4. googletrans==4.0.0-rc1
5. python-dotenv

**File: app.py**

1. import streamlit as st
2. import os
3. from dotenv import load\_dotenv
4. import google.generativeai as palm
5. from langdetect import detect
6. from googletrans import Translator, LANGUAGES
7. load\_dotenv()
8. Api\_key = os.getenv("API\_KEY")
9. palm.configure(api\_key=Api\_key)
10. translator = Translator()
11. i = 0
12. model\_list = palm.list\_models()
13. for model in model\_list:
14. if i == 1:
15. model\_name = model.name
16. break
17. i += 1
18. def generate\_questions(model\_name, text):
19. response = palm.generate\_text(
20. model=model\_name,
21. prompt=f"Generate generate follow-up questions which are intriguing and thought-provoking from the provided context text:\n\n{text}\n\nQuestions:",
22. max\_output\_tokens=150
23. )
24. questions = response.result.strip() if response.result else "No questions generated"
25. return questions
26. def get\_user\_text():
27. return st.text\_area("Enter the text you want questions generated from:")
28. def process\_and\_generate\_questions(user\_text, model\_name):
29. try:
30. detected\_language = detect(user\_text)
31. if detected\_language != 'en':
32. translated\_text = translator.translate(user\_text, src=detected\_language, dest='en').text
33. else:
34. translated\_text = user\_text
35. questions = generate\_questions(model\_name, translated\_text)
36. if detected\_language != 'en':
37. questions = translator.translate(questions, src='en', dest=detected\_language).text
38. return questions
39. except Exception as e:
40. st.error(f"An error occurred: {e}")
41. return None
42. def main():
43. st.title("Inquisitive")
44. user\_text = get\_user\_text()
45. if st.button("Generate Questions"):
46. questions = process\_and\_generate\_questions(user\_text, model\_name)
47. if questions:
48. st.subheader("Generated Questions:")
49. st.write(questions)
50. if not user\_text:
51. st.warning("Please enter some text.")
52. return
53. if \_\_name\_\_ == "\_\_main\_\_":
54. main()