### Project Report

|  |  |
| --- | --- |
| Date of Submission | 20/07/2024 |
| Team ID | SWTID1720459891 |
| Project Name | Inquisitive: A Multilingual AI Question Generator Using PaLM’s Text-Bison-001 |

### TEAM MEMBERS

|  |  |
| --- | --- |
| **Name** | **Student Mail ID** |
| Siddarth SR Pillai | siddarth.srpillai2021@vitstudent.ac.in |
| Akshaj Anil | akshaj.anil2021@vitstudent.ac.in |
| Chirag Mahesh | chirag.mahesh2021@vitstudent.ac.in |
| Akshay R | akshay.r2022@vitstudent.ac.in |

### Inquisitive:A Multilingual AI Question Generator Using Palm’s Text-Bison-001

Inquisitive is an application that 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.

**Scenario 1: Corporate Training Programs**

Enhance corporate training programs by incorporating Inquisitive: The GenAI Question Generator. 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.

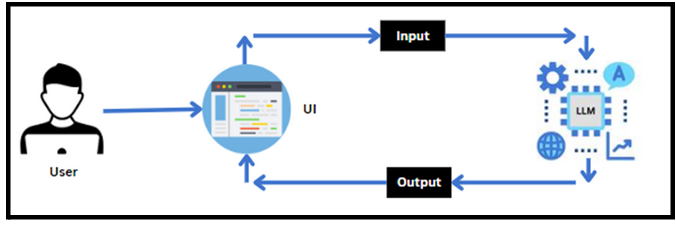
**Scenario 2: Content Creation for Marketing**

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**

**Project Flow**

1. Users input text into the UI of Inquisitive.
2. The input text is then processed and analyzed by the PALM architecture, which is integrated into the backend.
3. PALM autonomously generates questions based on the input text.
4. The generated questions are sent back to the frontend for display on the UI.
5. Users can view the dynamically generated questions and interact with them to gain deeper insights into the content.

**To accomplish this, we have to complete all the activities listed below,**

* **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

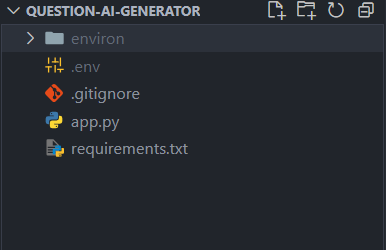
**Prior Knowledge**

You must have prior knowledge of the following topics to complete this project.

* 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 Structure**

Create the Project folder which contains files as shown below:

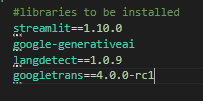


* app.py: It serves as the primary application file housing both the model and Streamlit UI code.
* requirements.txt: It enumerates the libraries necessary for installation to ensure proper functioning.
* .gitignore : This file is used in Git to specify which files and directories should be ignored and not tracked by version control, helping to keep the repository clean from unnecessary files such as build artifacts, temporary files, and sensitive information.
* .env : This file stores all the environment variables required for your application. In this case, it is the google api key
* environ : This is a virtual environment under which the application is written so as to isolate the libraries installed in the application from the system.

**Milestone 1: Requirements Specification**

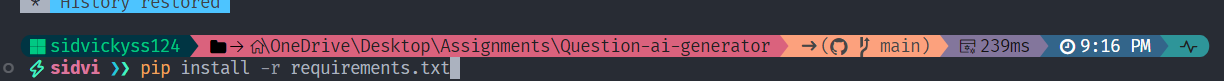
Specifying the required libraries in the requirements.txt file ensures seamless setup and reproducibility of the project environment, making it easier for others to replicate the development environment.

**Step 1: Create a requirements.txt file 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==1.0.9** : 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 allows you to manage environment variables stored in a .env file for your Python projects.

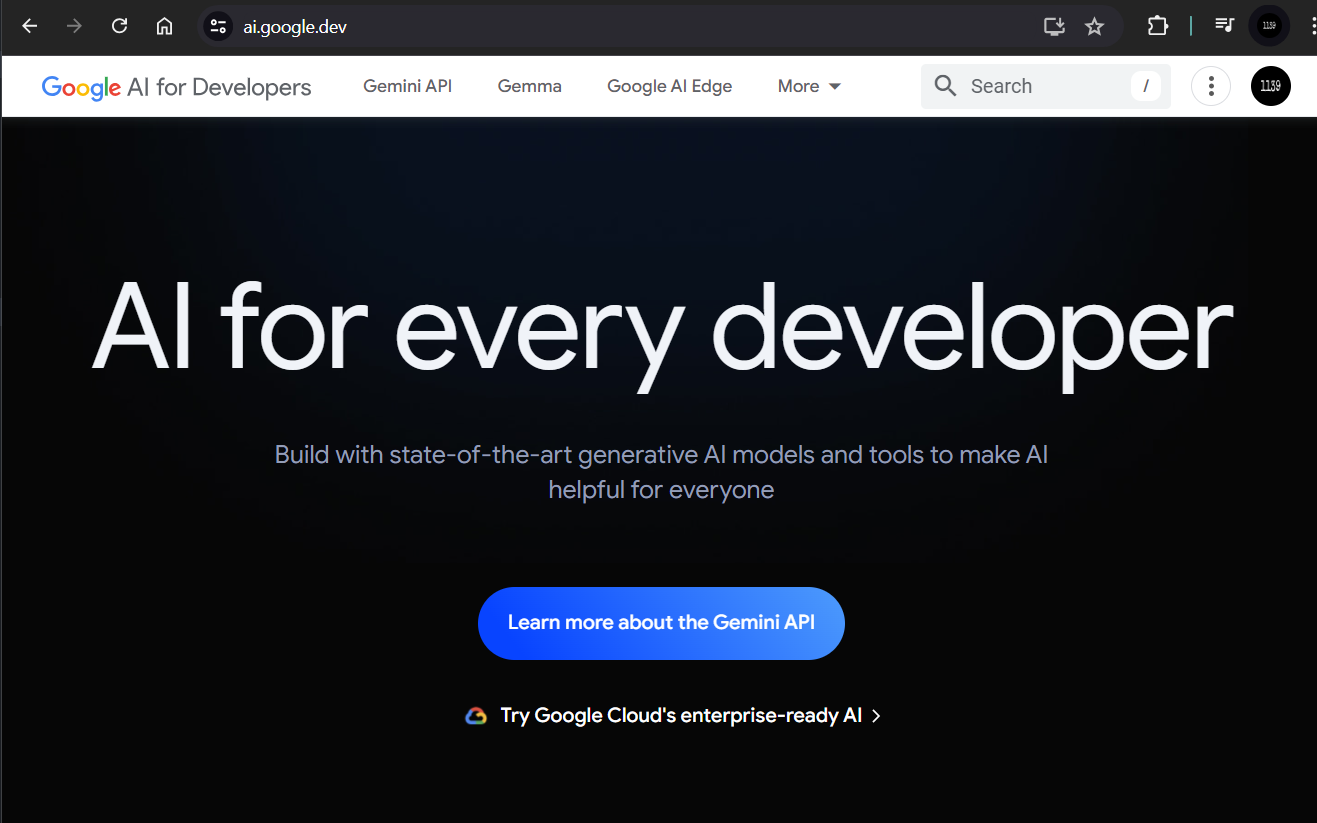
**Step 2 : Install the Required Libraries**



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

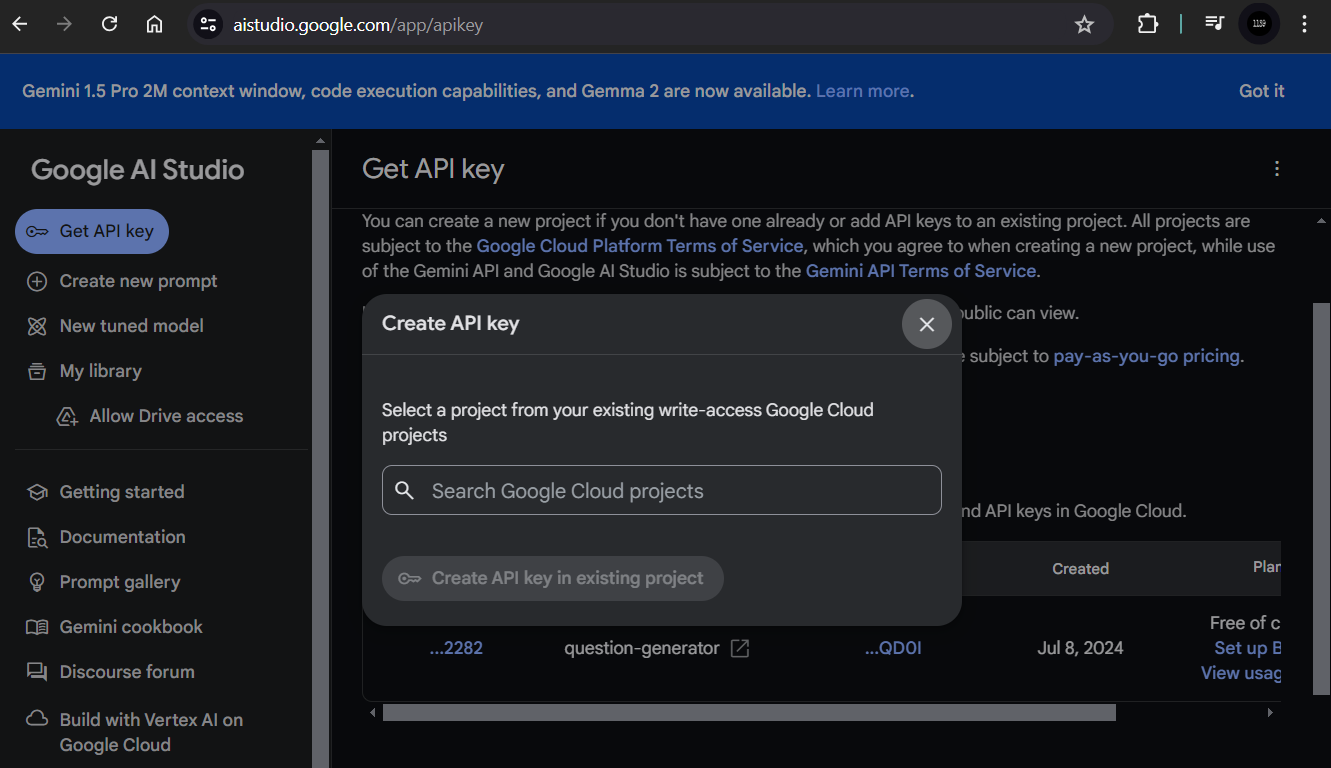
**Milestone 2 : Initialization The Model**

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.

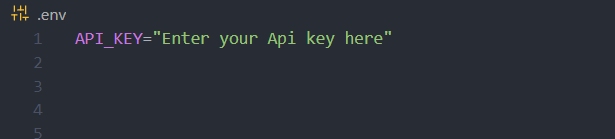


**Step 1 : Generate PALM API key**

* Click on the link [(https://developers.generativeai.google/](https://skillwallet.smartinternz.com/Student/guided_project_info/(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”.
* Copy the API key.



**Step 2 : Initialize Google API Key**



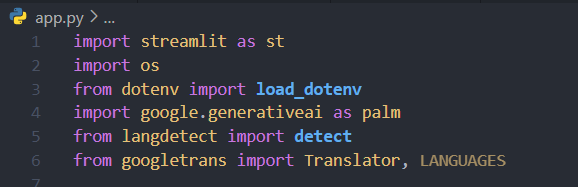
● Create a ***.env*** file and define a variable named ***API\_KEY***.

● Assign the copied Google API key to this variable.

● Paste the API key obtained from the previous steps here.

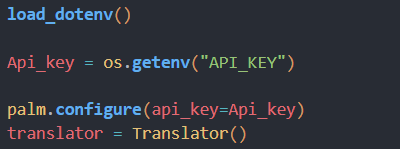
**Milestone 3 : Initialize The Pre-Trained Model**

**Step 1 : Import necessary files**



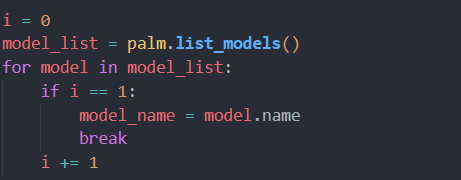
* Imports the **Streamlit** library, which is used for creating web applications.
* Imports the **OS** module, which provides functions for interacting with the operating system.
* Imports the **load\_dotenv** function from the **dotenv** library, used to load environment variables from a .env file.
* Imports the **Google Generative AI** library, aliasing it as palm, for accessing generative AI functions provided by Google.
* Imports the **detect** function from the **langdetect** library, which is used to detect the language of a given text.
* Imports the **Translator** class and **LANGUAGES** constant from the googletrans library, used for translating text between languages and accessing a list of supported languages, respectively.

**Step 2 : Configuration of the PALM API with the API key and initialize translator**



* The script starts by loading environment variables from a ***.env*** file using the ***load\_dotenv()*** function from the dotenv package
* Configuring the API key: The ***configure*** function is used to set up Google Generative AI API with the API key retrieved from the environment variables, ensuring secure and authorized access to the AI services.
* The ***Translator*** class facilitates language translation capabilities within the application.

**Step 3 : List Available models and select one**

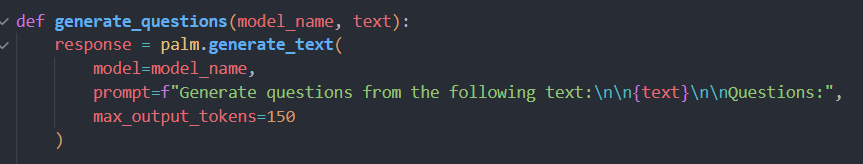


1. **Model List Retrieval and Iteration:**
   * The code retrieves a list of available models using ***palm.list\_models()*** and iterates through this list with a for loop.
2. **Second Model Selection:**
   * It selects the name of the second model which is text-bison-001 in the list (i == 1) and assigns it to the variable ***model\_name***, then exits the loop.

**Milestone 3 : Interfacing With Pre-Trained Model**

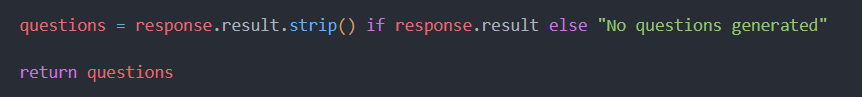
In this milestone, we will build a prompt template for summarizing the input content.

**Step 1 : 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.

**Step 2 : Extract Generated Text from the Result Attribute**

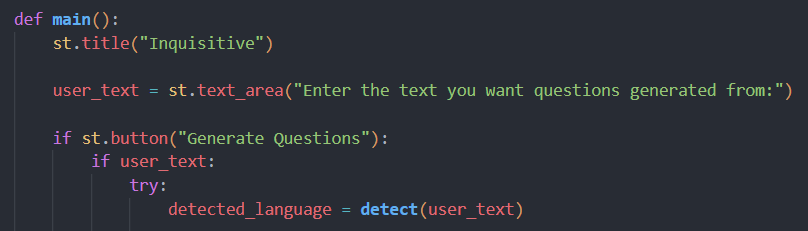


* This part of the code assigns the generated questions to the variable ***questions***.
* It checks if the ***response.result*** attribute exists; if it does, it strips any leading or trailing whitespace from the result and assigns it to ***questions***.
* If ***response.result*** is empty or doesn't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

**Milestone 4 : Model Deployment**

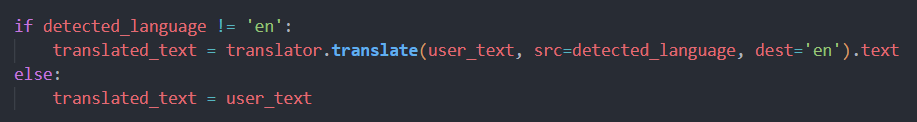
In this milestone, we are deploying the created model using ***streamlit***. Model deployment using ***Streamlit*** involves creating a user-friendly web interface for deploying machine learning models, enabling users to interact with the model through a browser. ***Streamlit*** provides easy-to-use tools for developing and deploying data-driven applications, allowing for seamless integration of machine learning models into web-based applications.

**Step 1 : Take Input From the User And Detect The Language**



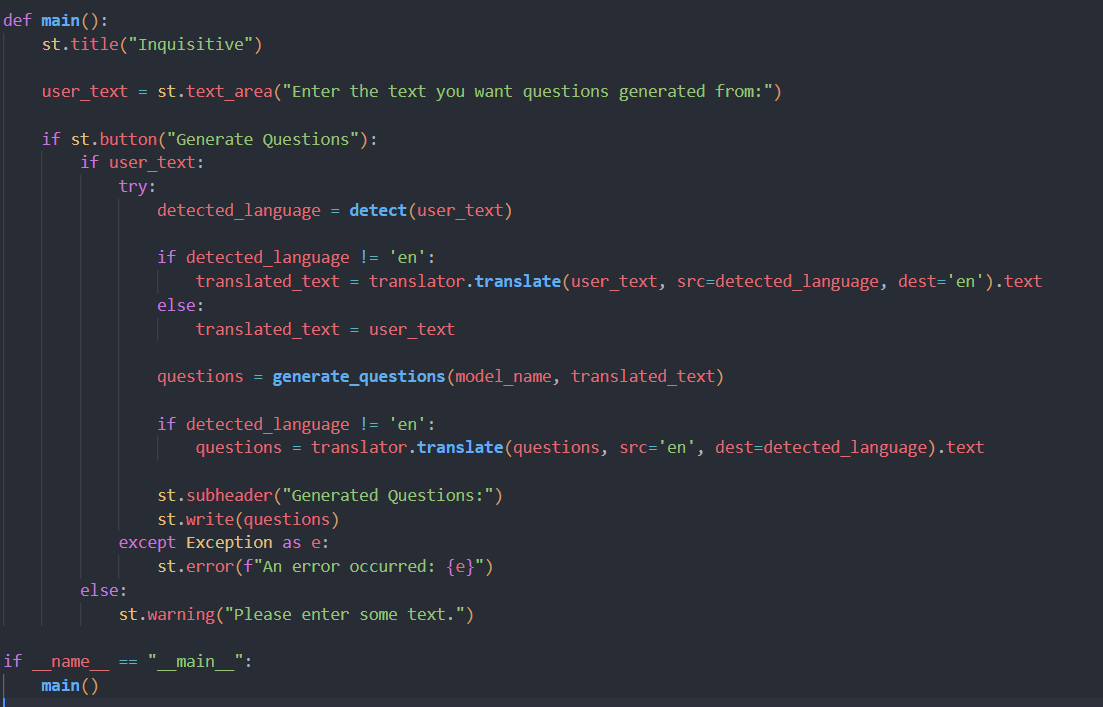
* The ***main*** function is defined, which contains the entire logic of the Streamlit application.
* ***st.title("Inquisitive")*** sets the title of the Streamlit app to "Inquisitive".
* ***user\_text = st.text\_area("Enter the text you want questions generated from:")*** creates a text area for the user to input the text from which they want questions generated.
* ***if st.button("Generate Questions"):*** checks if the "Generate Questions" button is pressed.
* ***if user\_text:*** checks if the user has entered any text.
* ***detected\_language = detect(user\_text)*** detects the language of the user input text using the langdetect library.

**Step 2 : Translate the Text To English If In Any Other Language**



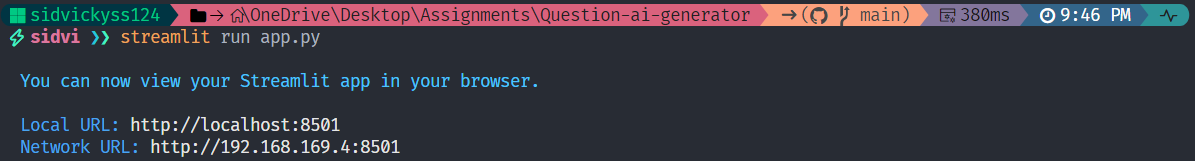
* This segment of code checks if the detected language of the user's input text is not English.
* If the language is not English, it uses the ***translator.translate()*** method to translate the text to English, specifying the source language as the detected language and the destination language as English.
* 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.

**Step 3 : Generate Questions Button**



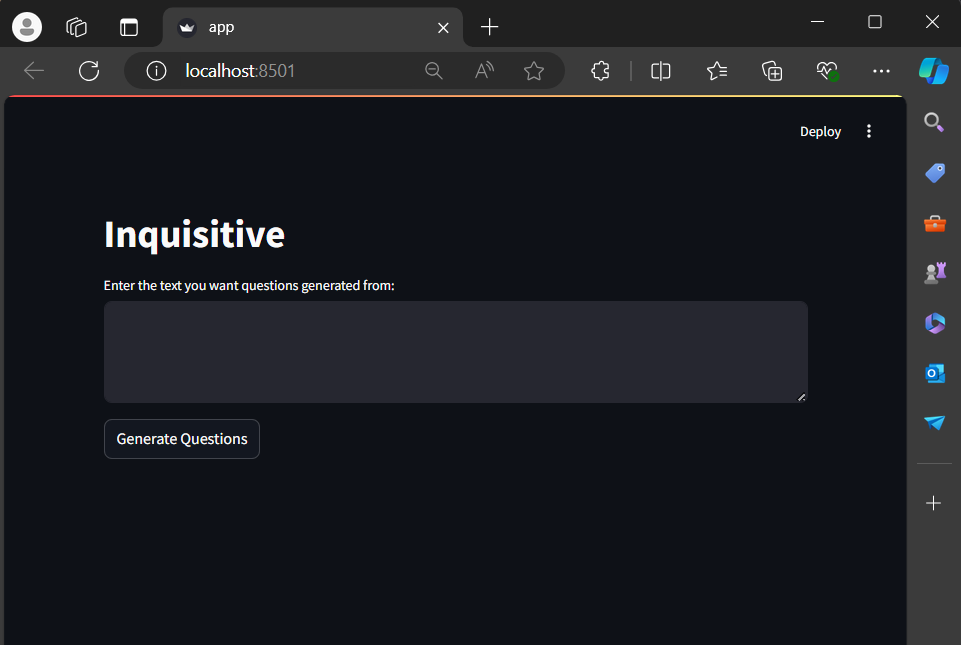
* This block of code executes when the user clicks the "Generate Questions" button. It first checks if the user has entered some text. If text is present, it calls the ***generate\_questions()*** function to generate questions based on the input text.
* If the detected language of the input text is not English, it translates the generated questions back to the original language using the ***translator.translate()*** method.
* Finally, it displays the generated questions under the subheader "Generated Questions." If no text is entered by the user, it shows a warning message prompting the user to enter some text.
* The code is wrapped in a try-except block to catch and display any errors that occur during execution using ***st.error(f"An error occurred: {e}")***.

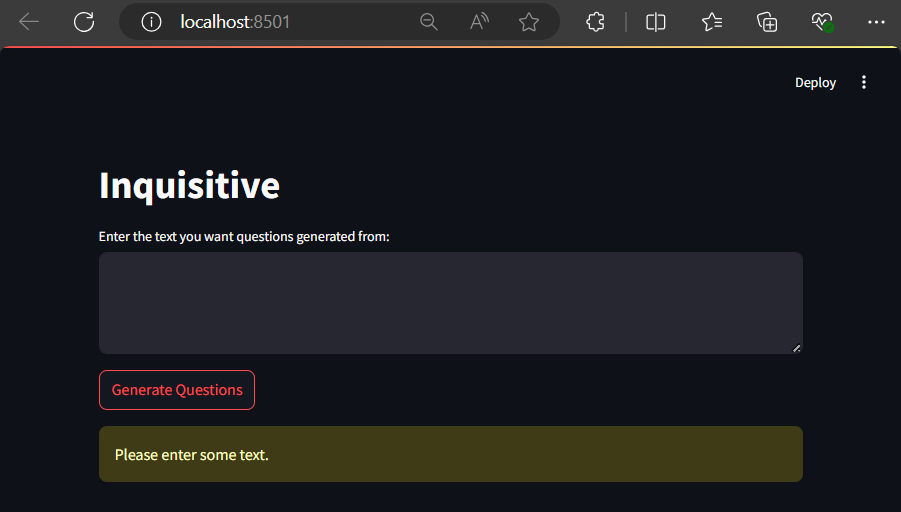
**Step 4 : Run the Web Application**



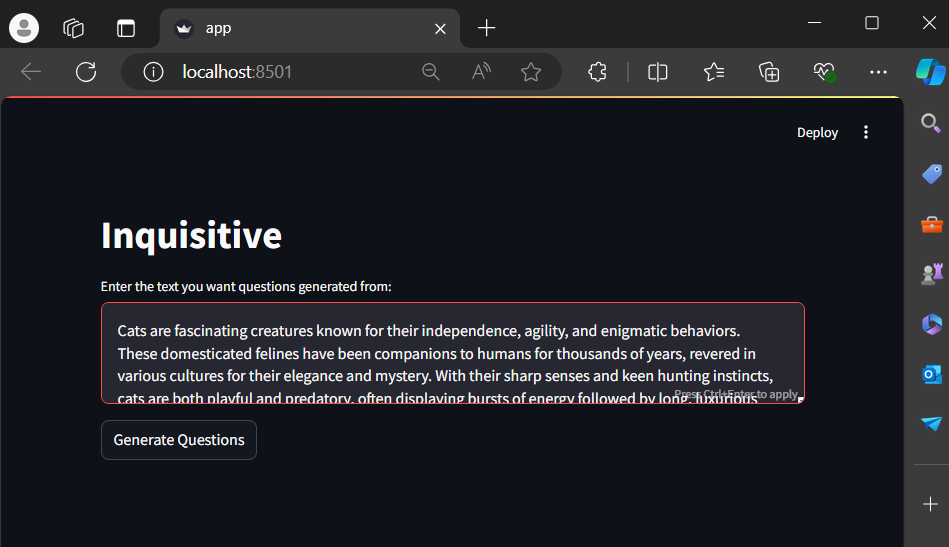
* Navigate to the folder where your Python script is.
* Open the terminal.
* Now type “streamlit run app.py” command.
* Navigate to the localhost where you can view your web page.

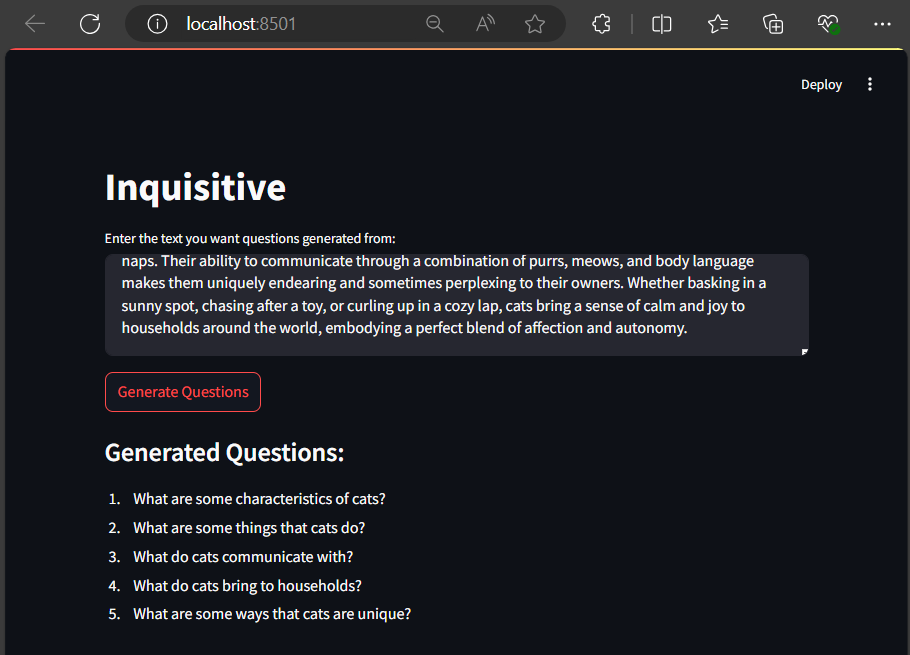
Now, the application will open in the web browser.



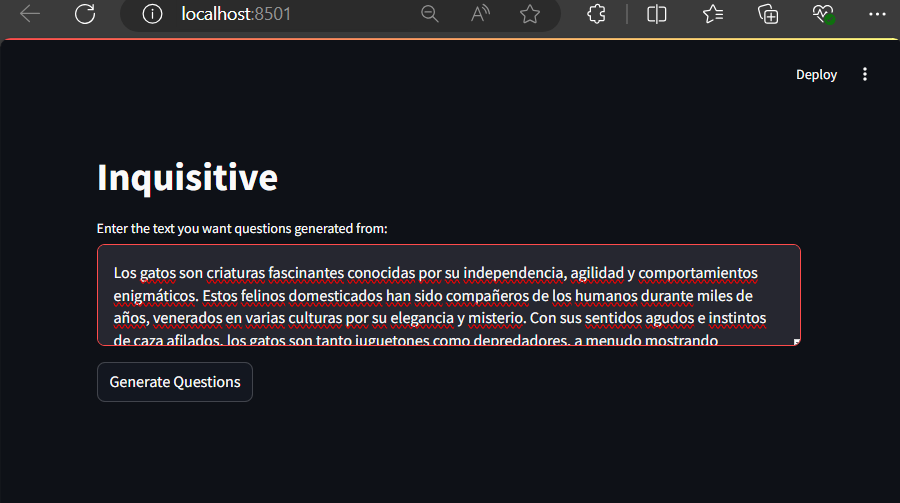


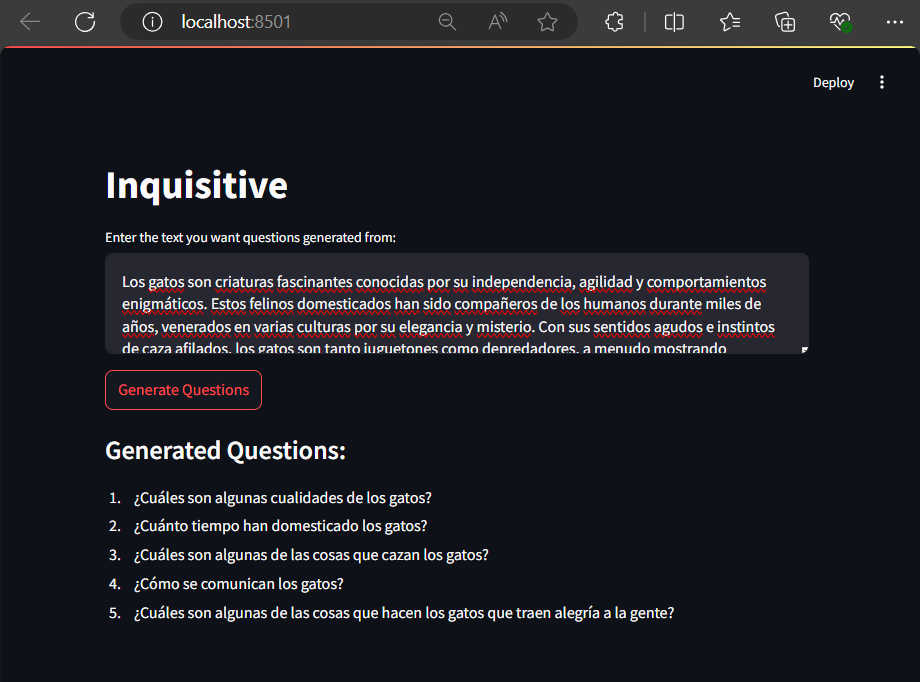
**Test case 1: English Language**



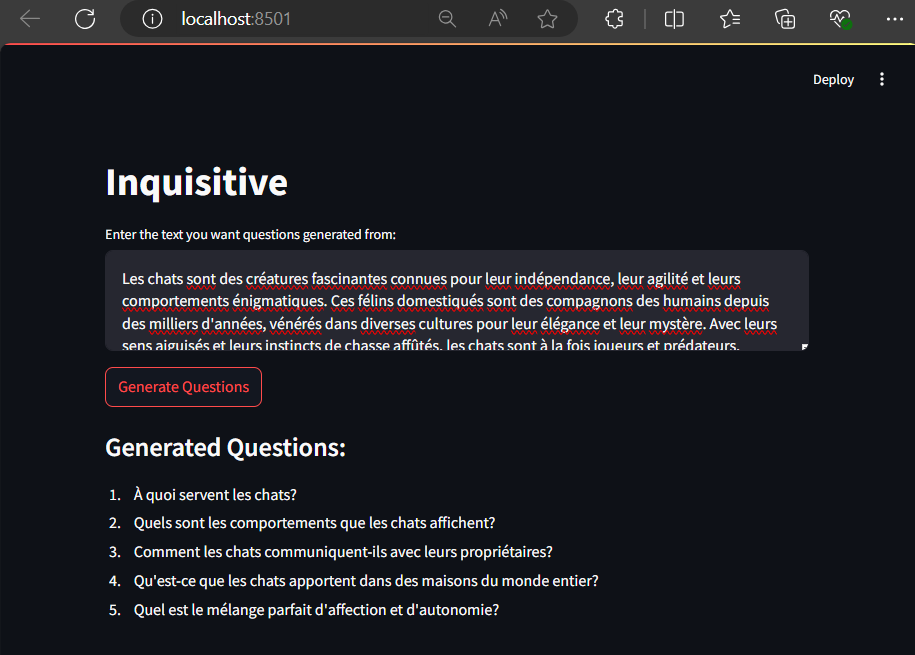


**Test case 2: Spanish Language**





**Test case 3: French Language**



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*