**Documentation & Code Walkthrough**

**1. Comprehensive Documentation**

**1.1. Overview of the System**

This project implements an **Excel-based Q&A Chatbot** using **Streamlit** for the frontend, **LangChain** for document processing, **FAISS** for fast similarity search, and **Google Generative AI** for generating responses based on the provided Excel data. The core functionality allows users to upload an Excel file, process the data, and query it using natural language, with responses generated based on the context retrieved from the document.

**1.2. System Architecture**

The system architecture consists of the following components:

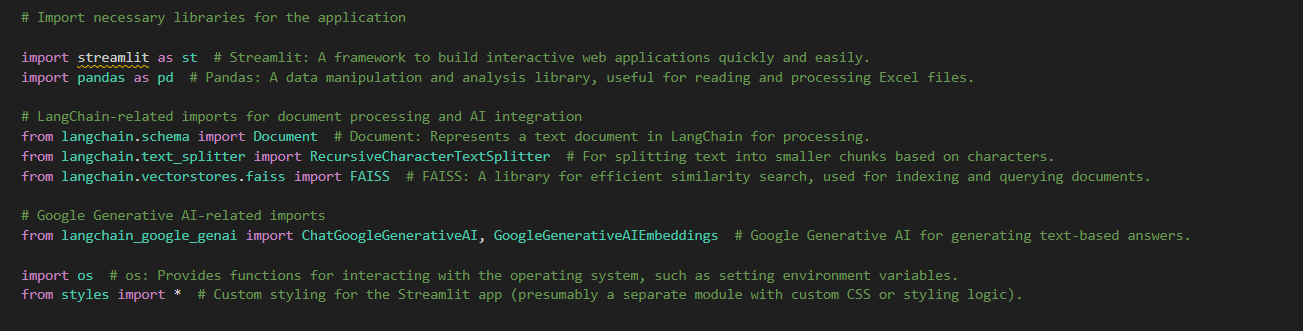
* **Streamlit UI**: Provides a simple and interactive interface for users to upload an Excel file, enter their Google API key, and ask questions.
* **File Processing**: When an Excel file is uploaded, it is processed by the prepare\_excel function. This function reads the data, converts it into a format suitable for document processing, and creates a **FAISS** index for fast retrieval.
* **Text Splitting and Embeddings**: The text from the Excel file is split into smaller chunks using **LangChain's RecursiveCharacterTextSplitter**, and each chunk is embedded using **GoogleGenerativeAIEmbeddings**.
* **Retrieval-Augmented Generation (RAG)**: The rag function performs a similarity search on the FAISS index for the user's query and uses **Google Generative AI** to generate an answer based on the retrieved documents.
* **Session Management**: **Streamlit session state** is used to store the processed FAISS index and chat history.

**1.3. Key Functions and Components**

* **prepare\_excel**: This function reads the Excel file, processes the data, splits it into manageable chunks, and creates a FAISS index.
* **rag**: The function that searches the FAISS index for relevant content and generates an answer using the Google Generative AI model.
* **chatbot\_logic**: The main function that handles Streamlit’s UI, user input, and integrates the file processing and AI model interactions.

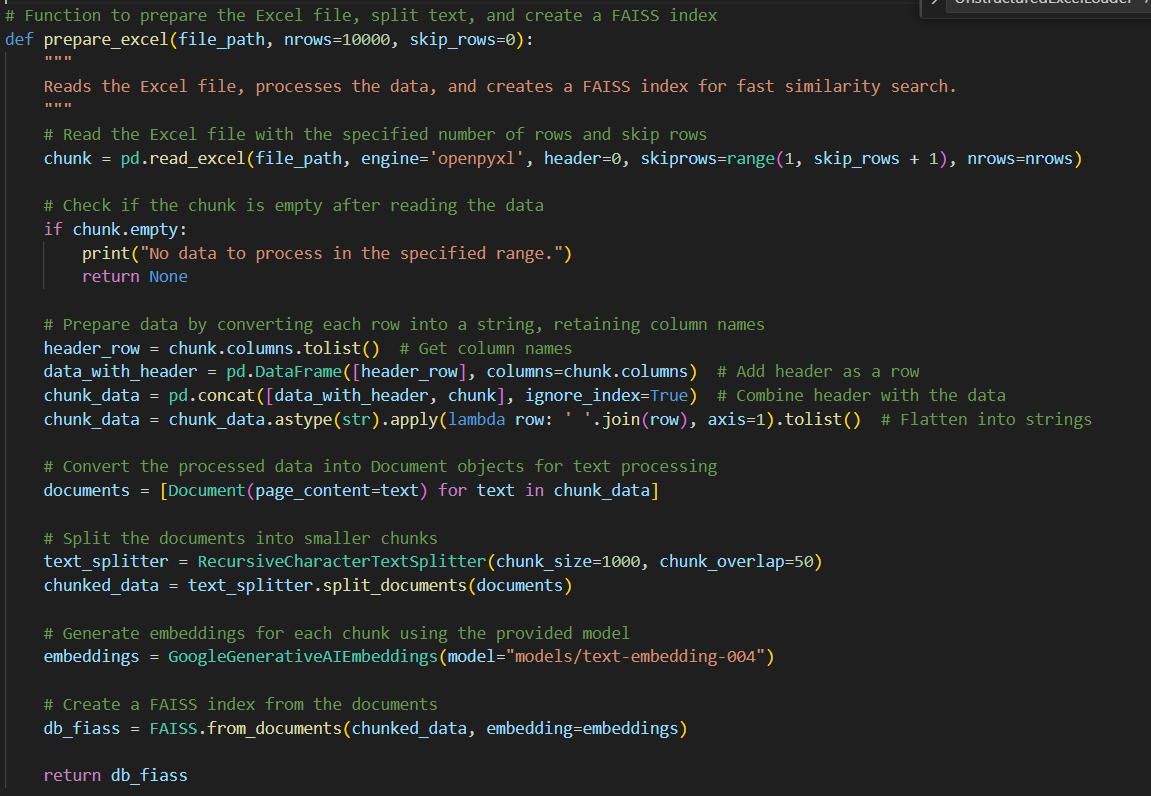
**2. Code Walkthrough**

**2.1. Imports**



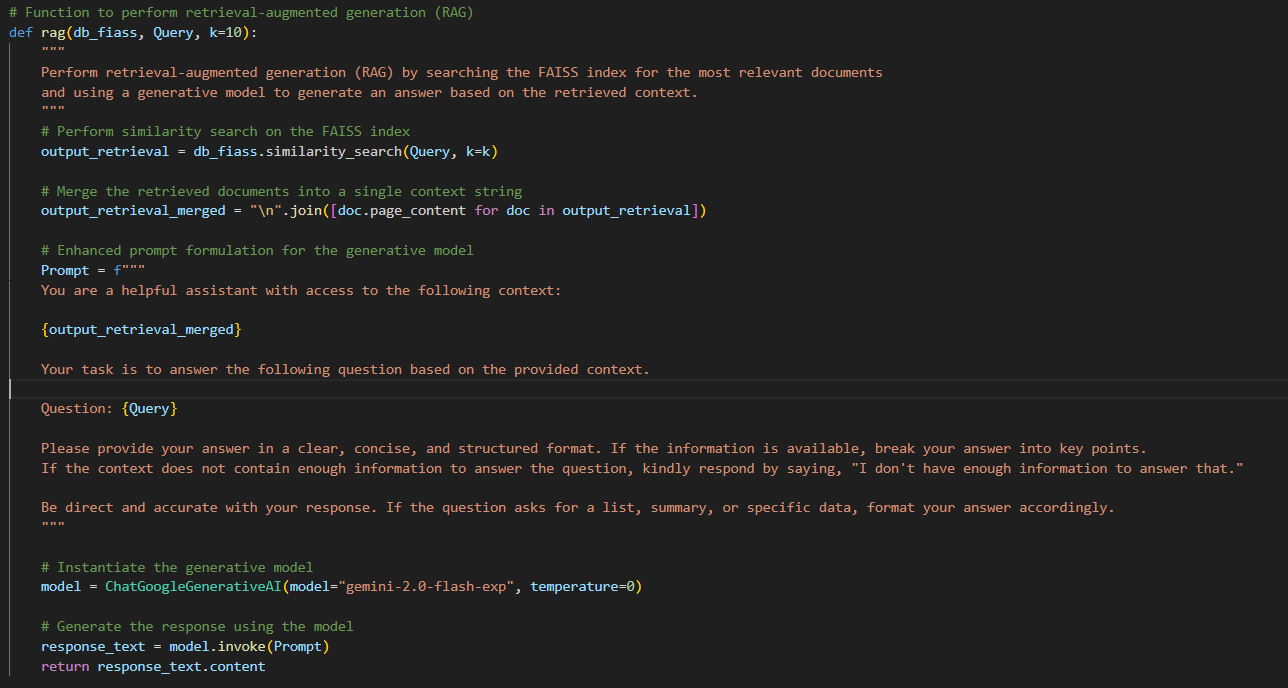
* **streamlit**: Used for building the web-based user interface.
* **pandas**: For reading and processing the Excel file.
* **langchain**: Provides utilities for document processing, text splitting, and interaction with embeddings.
* **faiss**: Used for creating a vector database for efficient similarity searches.
* **os**: Used for setting the environment variable (Google API Key).
* **styles**: Custom styles for the Streamlit UI.

**2.2. prepare\_excel Function**



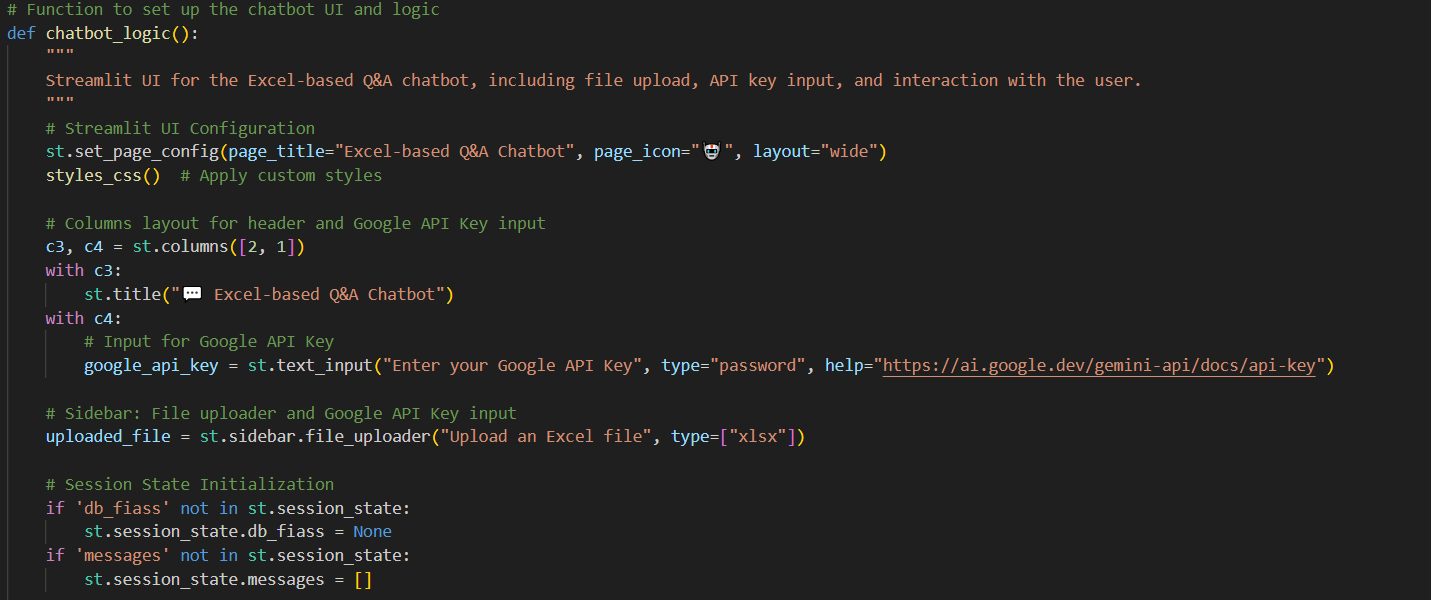
* **File Reading**: The function reads the Excel file using pandas.read\_excel() while supporting row skipping and chunk size.
* **Data Conversion**: Each row is converted into a single string and then into Document objects.
* **Text Splitting**: The data is split into smaller chunks for more efficient processing using RecursiveCharacterTextSplitter.
* **FAISS Indexing**: The chunks are embedded and indexed using FAISS for fast retrieval.

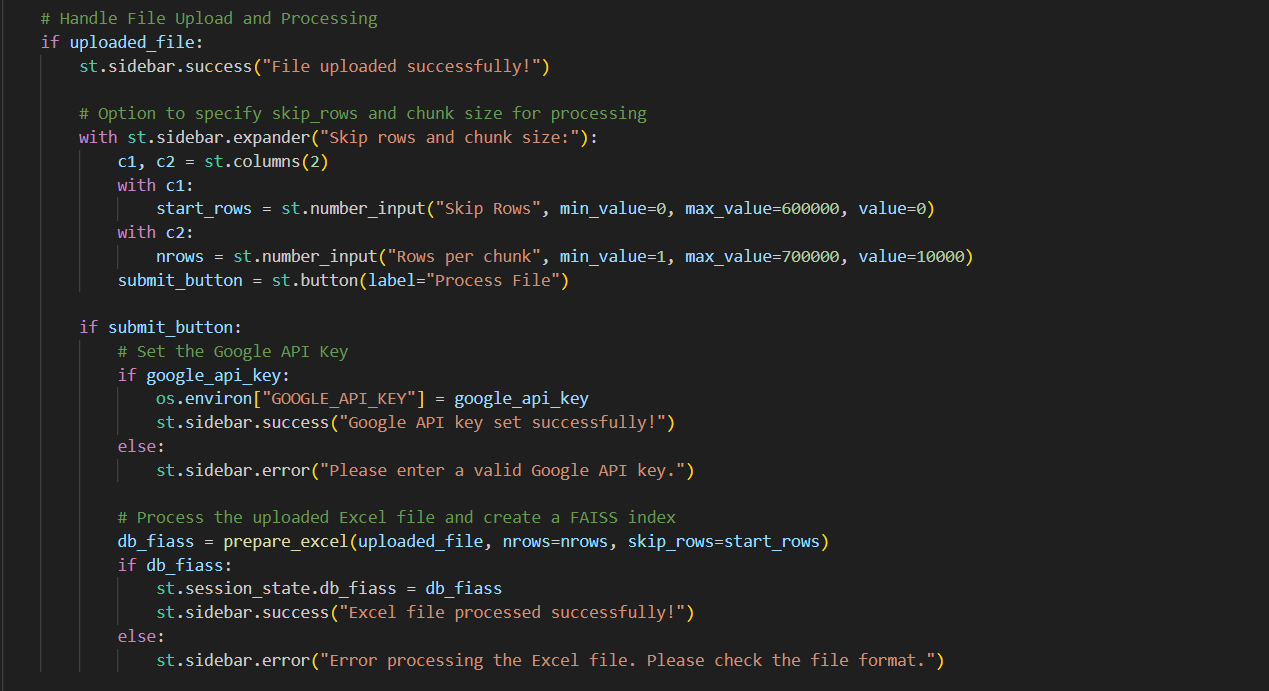
**2.3. rag Function**

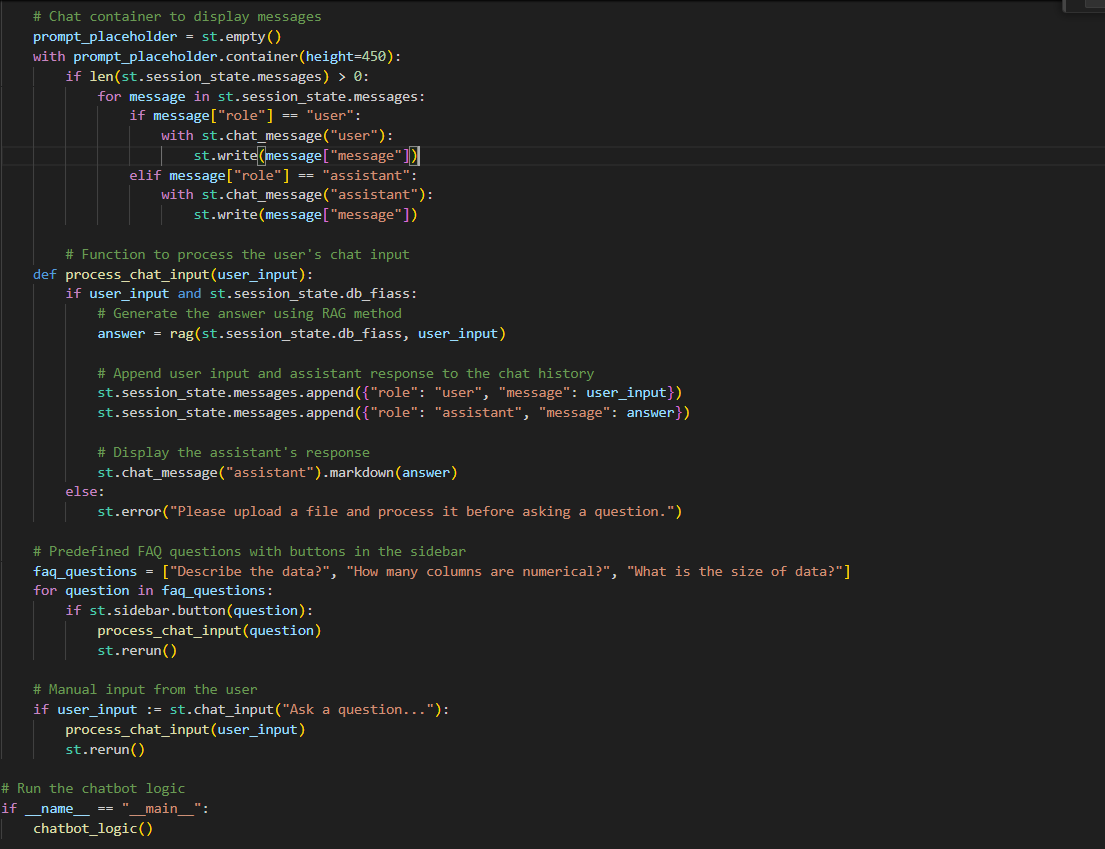


* **Similarity Search**: It uses FAISS to search for the most relevant documents based on the user's query.
* **Context Formation**: The retrieved documents are merged into a context string which is used as input for the generative AI model.
* **Model Interaction**: The ChatGoogleGenerativeAI model is invoked with the constructed prompt to generate a response.

**2.4. chatbot\_logic Function**







* **Streamlit Setup**: Configures the page layout and sets up the sidebar for file upload and API key input.
* **Session Management**: Initializes session states for the FAISS index and chat history.
* **File Upload**: Handles file processing and error checking.
* **Chat Interaction**: Displays user and assistant messages, processes user inputs, and generates answers using the rag function.

**2.5. Challenges Encountered**

* **Handling Large Datasets**: Processing large Excel files could result in performance bottlenecks. This was addressed by splitting the data into smaller chunks and indexing them with FAISS for efficient retrieval.
* **Google API Key Integration**: Ensuring the Google API key was securely handled in the UI was crucial. The solution involves entering the key into a password field and setting it as an environment variable.