**📄 Gemini-Powered RAG Web QA System**

**🧭 Project Overview:**

This project implements a **Retrieval-Augmented Generation (RAG)** pipeline that allows users to:

* Index content from public websites
* Ask natural language questions about that content
* Receive grounded, citation-rich answers generated by Gemini AI

The system is modular, open-source, and designed for extensibility. It uses:

* **FastAPI** for backend services
* **Streamlit** for user interface
* **FAISS** for vector-based retrieval
* **SentenceTransformers** for semantic embeddings
* **Gemini AI (Google)** for answer generation

**🧱 System Architecture:**

The pipeline follows a classic RAG structure:

1. **Document Ingestion**  
   Web pages are scraped and cleaned to extract readable text.
2. **Chunking and Embedding**  
   Text is split into manageable chunks and converted into vector embeddings using a pre-trained model.
3. **Indexing**  
   Embeddings are stored in a FAISS index for fast similarity search.
4. **Retrieval**  
   When a user submits a query, the system retrieves the most relevant chunks from the index.
5. **Prompt Construction**  
   Retrieved chunks are formatted into a prompt with citations.
6. **Answer Generation**  
   The prompt is sent to Gemini AI via the Google Generative AI SDK, which returns a grounded answer.
7. **User Interface**  
   The answer is displayed in Streamlit, along with status codes and raw response data.

**📂 Project Structure according to the files:**

app.py -- Streamlit frontend

main.py -- FastAPI backend with /index and /chat

rag\_generator.py -- Gemini AI generation logic

rag\_retriever.py -- FAISS-based retrieval

utils\_embedding.py -- Chunking + embedding

utils\_scraper.py -- Web scraping

.env -- Gemini API key

**🔐 API Integration:**

The system uses Gemini AI via the **Google Generative AI SDK**. You must:

* Create a free API key from [Google AI Studio](https://makersuite.google.com/app)
* Store the key in a .env file
* Load and configure the SDK using genai.configure(api\_key=...)

**The model used is;**

‘models/gemini-pro-latest’. This model supports text-only generation via generate\_content().

**🧠 Model Components:**

* **Embedding Model**:  
  A lightweight transformer model (all-MiniLM-L6-v2) is used to convert text chunks and queries into semantic vectors.
* **Vector Store**:  
  FAISS is used for efficient similarity search over embedded chunks.
* **Generation Model**:  
  Gemini Pro generates answers based on retrieved context. Responses include citation markers like [1], [2], etc.

**🖥️ User Interface - Streamlit app:**

* A section to index new URLs
* A section to ask questions
* Display of status codes and raw response text
* Markdown rendering of the final answer

**🔄 Workflow Summary:**

🔹 Indexing:

* User enters one or more URLs into the Streamlit “Index New URLs” box.
* The backend scrapes and processes each page.
* Text is chunked and embedded.
* Embeddings are stored in FAISS for retrieval.

🔹 Question Answering:

* User enters a question in the “Ask a question…” box.
* The query is embedded and compared against the indexed chunks.
* Top-k relevant chunks are selected.
* These chunks are injected into a prompt with citation markers.
* The prompt is sent to Gemini AI via SDK.
* The generated answer is returned and displayed.

**🧪 Sample Inputs:**

URLs to Index:

* Huyen Chip’s GenAI platform post  
  https://huyenchip.com/2024/07/25/genai-platform.html
* Jina AI’s ColBERT search explanation  
  https://jina.ai/news/what-is-colbert-and-late-interaction-and-why-they-matter-in-search/
* Lilian Weng’s hallucination article  
  https://lilianweng.github.io/posts/2024-07-07-hallucinatio
* Quora Engineering’s embedding search write-up  
  https://quoraengineering.quora.com/Building-Embedding-Search-at-Quora

Questions to Ask:

* “What is Build Time in GenAI platforms?”
* “How does ColBERT differ from dense retrieval?”
* “What causes hallucinations in LLMs?”
* “How does Quora use embeddings for multilingual search?”

**📈 Output Format:**

Answers are returned in natural language, grounded in retrieved context, and include citation markers. Example:“Build Time refers to the latency between prompt submission and model readiness. It is distinct from Time to First Token (TFT) and Total Latency [1].”

**🛠️ Error Handling:**

* If no documents are indexed, the system returns a clear error.
* Gemini API errors (e.g., invalid model name or quota limits) are caught and displayed.
* If the answer cannot be generated, the system returns a descriptive fallback message.