Python Candidate Test – RAG Pipeline with Streamlit + LangChain

🎯 Objective

Build a Retrieval-Augmented Generation (RAG) chatbot using Python that:

Ingests custom .txt content (e.g., blog posts)

Embeds and indexes it using FAISS

Uses LangChain to retrieve relevant chunks

Uses a Hugging Face model (GPT2 or Mistral) to answer queries

Includes a basic Streamlit UI for interaction

✅ Deliverables

A Colab-compatible .ipynb notebook

A Streamlit app script (app.py) to run the chatbot locally

Sample output from 2 questions

Link to GitHub repo or Google Drive (if required)

📁 Project Structure

```
rag_test/
├── blogs/
│   ├── blog1.txt
│   └── blog2.txt
├── rag_pipeline.ipynb   ✅ (main notebook for logic)
├── app.py               ✅ (streamlit UI)
└── requirements.txt     ✅ (dependencies)
```

🧠 Tools to Use

LangChain (for pipeline orchestration)

sentence-transformers (all-MiniLM-L6-v2 or bge-small-en)

faiss (vector store)

transformers (for LLM)

Streamlit (for frontend UI)

📌 Task Breakdown

✅ Step 1: Data Ingestion & Preprocessing

Load text from files in the blogs/ folder.

Chunk text into ~200-word sections with titles preserved.

Clean basic punctuation and newlines.

✅ Step 2: Embedding + Vector Store
Use sentence-transformers to generate embeddings.

Store in FAISS with document metadata.

Save and reload FAISS index for persistence.

✅ Step 3: LangChain Integration
Create a FAISS Retriever with LangChain

Use ConversationalRetrievalChain with:

A basic prompt template

A LLM like GPT2 or any Hugging Face small model

Allow queries like:

“What is energy mastery?”

“How do high performers avoid burnout?”

✅ Step 4: Streamlit UI
Create a simple UI with:
Title: “Your Personal Knowledge Chatbot”

Textbox for user input

Chat history below

Call backend via LangChain pipeline

Display model’s answers cleanly

```
import streamlit as st

st.set_page_config(page_title="Knowledge Chatbot")

st.title("📚 Your Personal Knowledge Chatbot")

if "history" not in st.session_state:
    st.session_state.history = []
```

```
query = st.text_input("Ask something from your content:")

if query:
    from backend import get_answer  # optional helper

    response = get_answer(query)
    st.session_state.history.append((query, response))

for q, r in reversed(st.session_state.history):
    st.markdown(f"**You:** {q}")
    st.markdown(f"**Bot:** {r}")
```

🔄 Sample Prompt Template (LangChain)

```
template = """You are a helpful assistant. Use the context below to answer the
user's question.

Context:
{context}

Question:
{question}

Answer:"""
```

💾 requirements.txt

```
transformers
sentence-transformers
faiss-cpu
langchain
streamlit
```

▫ Sample Test Data: blog1.txt
link

✅ Evaluation Criteria

Area	Points
Clean Python structure & documentation	10
FAISS + embedding setup	10
LangChain integration	10
Working Streamlit UI	10
Answer relevance from model	10
Total	50

🎁 Bonus (Optional)
Add conversation memory
Replace GPT2 with Mistral (via Ollama) or Mixtral (if Colab supports it)
Use ChromaDB instead of FAISS