Let's build a **business-ready chatbot** using **LangChain + Google Gemini** that can handle your **customer support** using your business data. This will include:

- □ RAG (knowledge retrieval from your business docs)
- □ Conversational memory
- □ FastAPI chatbot endpoint
- □ Tool calling (optional for calculations or FAQs)

---

1 □ Install Dependencies

```
pip install -U langchain langchain-google-genai langchain-community faiss-cpu
fastapi uvicorn pydantic python-multipart
```

---

2 □ Project Structure

```
business_chatbot/
│
├── app.py          # FastAPI server
├── chatbot_agent.py # Agent code: Gemini + RAG + Memory
├── docs/           # Your business documents (PDF, TXT, etc.)
│   ├── faq.txt
│   ├── product_manual.pdf
```

---

3 □ Create the Chatbot Agent (`chatbot_agent.py`)

```
import os
from langchain_google_genai import ChatGoogleGenerativeAI,
GoogleGenerativeAIEmbeddings
from langchain.vectorstores import FAISS
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.document_loaders import TextLoader, PyPDFLoader
from langchain.agents import create_tool_calling_agent, AgentExecutor
from langchain.memory import ConversationBufferMemory
from langchain.tools import tool
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder

# -------------------------
# 1. Load Business Data
# -------------------------

def load_documents(folder="docs"):
    docs = []
    for file in os.listdir(folder):
        path = os.path.join(folder, file)
```

```
        if file.endswith(".txt"):
            docs.extend(TextLoader(path).load())
        elif file.endswith(".pdf"):
            docs.extend(PyPDFLoader(path).load())
    return docs

documents = load_documents()

# -------------------------
# 2. Split into Chunks
# -------------------------

splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
docs = splitter.split_documents(documents)

# -------------------------
# 3. Create Embeddings & Vector Store
# -------------------------

embeddings = GoogleGenerativeAIEmbeddings(model="models/embedding-001")
vectorstore = FAISS.from_documents(docs, embeddings)
retriever = vectorstore.as_retriever(search_kwargs={"k": 3})

# -------------------------
# 4. Define Gemini LLM
# -------------------------

llm = ChatGoogleGenerativeAI(model="gemini-1.5-flash", temperature=0.3)

# -------------------------
# 5. Define Tools
# -------------------------

@tool
def knowledge_base(query: str) -> str:
    """Search the business knowledge base."""
    relevant_docs = retriever.get_relevant_documents(query)
    return "\n\n".join([doc.page_content for doc in relevant_docs])

tools = [knowledge_base]

# -------------------------
# 6. Memory for Chat
# -------------------------

memory = ConversationBufferMemory(
    memory_key="chat_history",
    return_messages=True
)

# -------------------------
# 7. Chat Prompt Template
# -------------------------

prompt = ChatPromptTemplate.from_messages(
    [
        ("system", "You are a helpful AI assistant for customer support.
```

```
Answer using business knowledge."),
        MessagesPlaceholder(variable_name="chat_history"),
        ("human", "{input}"),
        MessagesPlaceholder(variable_name="agent_scratchpad"),
    ]
)

# -------------------------
# 8. Create Agent
# -------------------------

agent = create_tool_calling_agent(llm=llm, tools=tools, prompt=prompt)
agent_executor = AgentExecutor(agent=agent, tools=tools, memory=memory,
verbose=False)

# -------------------------
# 9. Function to Ask Agent
# -------------------------

def ask_chatbot(query: str):
    response = agent_executor.invoke({"input": query})
    return response["output"]
```

---

4 FastAPI Chatbot Server (`app.py`)

```
from fastapi import FastAPI
from pydantic import BaseModel
from chatbot_agent import ask_chatbot

app = FastAPI(title="Business Customer Support Chatbot")

class QueryRequest(BaseModel):
    question: str

@app.post("/chat")
def chat_endpoint(request: QueryRequest):
    answer = ask_chatbot(request.question)
    return {"answer": answer}
```

---

5 Run the Chatbot

```
uvicorn app:app --reload
```

Visit:

```
http://127.0.0.1:8000/docs
```

- Use `/chat` endpoint to ask customer questions.
- Example:

```
{
  "question": "How do I reset my password?"
}
```

Gemini will search your business docs and respond based on your knowledge base.

---

6 Tips for a Real Customer Support Bot

1. **Document Types**
   - PDF manuals → `PyPDFLoader`
   - FAQ TXT/CSV → `TextLoader`
   - Emails / support tickets → `CSVLoader`
2. **Memory**
   - Current memory is short-term. For multi-turn support sessions, you can use `ConversationSummaryMemory` or store sessions in a database.
3. **Persistent Vector Store**

```
vectorstore.save_local("faiss_index")
vectorstore = FAISS.load_local("faiss_index", embeddings)
```

4. **Streaming Responses**
   - FastAPI + `StreamingResponse` to send replies token-by-token for faster UI experience.
5. **Multi-Tool Support**
   - You can add tools like:
     - `calculator`
     - `order_lookup(order_id)`
     - `faq_search(keyword)`

 With this setup, you now have a **custom GPT-powered customer support chatbot** using **Gemini** and **LangChain** that can answer queries from your business data.