RAG Chatbot Integration: Two Architectural Approaches

✅ Option 1: Session Management via FastAPI (Recommended Clean Architecture)

🔧 Responsibilities:

FastAPI:

- Handles session state, message history, and API requests
- Manages Redis (cache) + MongoDB (persistent store)
- Delegates only the LLM/vector processing to LangChain

LangChain RAG Module:

- Accepts session ID, question, and chat history from FastAPI
- Returns generated answer and updated chat messages

📦 Folder Structure

```
bash

project/
│
├── app/
│   ├── main.py            # FastAPI entry point
│   ├── routes/
│   │   └── chat.py        # /chat endpoint
│   ├── services/
│   │   ├── memory.py      # Redis + Mongo logic
│   │   └── rag.py         # Wraps LangChain chain.invoke
│   ├── models/
│   │   └── message.py     # Pydantic models
│   └── config.py          # Env + client setup
├── requirements.txt
├── .env
└── README.md
```

📡 Flow:

Frontend/user sends a message → POST /chat

FastAPI:

- Loads chat history from Redis → fallback to MongoDB
- Passes session_id, question, chat_history to RAG module

- Stores response (AI + user messages) back in Redis + Mongo
- Returns answer to the frontend

✅ Pros:

- Modular, clean separation of concerns
- Centralized control of history and state
- Easier to scale or swap components (e.g., vector DB or LLM)
- Suitable for multi-user, production-ready systems

❌ Cons:

- Slightly more dev effort up front
- Requires coordination between services (FastAPI + RAG)

⚙ Option 2: Full Backend in LangChain Script (All-in-One RAG Pipeline)

🔧 Responsibilities:

LangChain script handles everything:

- User input → fetch history → pass to LLM → return + save answer
- Redis/Mongo are connected inside the RAG code
- No dedicated FastAPI layer

📄 Structure

```bash
rag_pipeline/
├── main.py            # CLI or script entry
├── utils/
│   ├── memory.py      # Redis + Mongo logic
│   └── rag_chain.py   # RAG processing logic
├── .env
└── requirements.txt
```

🧩 Flow:

- User runs script or hits a CLI/webhook
- Script fetches history → runs RAG chain → saves messages

✅ Pros:

- Simpler to start

- Fewer moving parts
- Easier for experimentation or internal tools

❌ Cons:

- No clean API layer (hard to integrate with UI/frontend)
- Not easily scalable (single-user or dev environments only)
- Risk of tight coupling and spaghetti code
- Poor maintainability for large teams or long-term projects

🧭 Summary Table

Feature	Option 1: FastAPI + LangChain	Option 2: All-in-One RAG
Session Management	FastAPI (Redis + Mongo)	Inside RAG script
Frontend Integration	Easy (REST API)	Hard (no direct API)
Modularity	High	Low
Best for	Multi-user production systems	Prototypes / Demos
Scalability	Easy to scale	Hard to scale
Dev Time	Medium	Low

RAG Chatbot Integration Guide - Generated on July 21, 2025