Case Study for AI/Software Engineer

Title: Building the Conversational Backend for BOT GPT

Company: BOT Consulting
Role: AI/Software Engineer
Domain: Conversational AI, Large Language Model Integration, Scalable Backend Systems

1. Context

BOT Consulting is building **"BOT GPT"** - a production-grade conversational AI platform.
It will power internal assistants, customer-facing chatbots, and multi-turn enterprise workflows.

We are evaluating your ability to design:

- A clean backend architecture
- A well-structured API layer
- Conversation data persistence
- Integration with external LLMs (OpenAI, Claude, Llama 3, etc.)
- Scalability & cost-aware context management

This is NOT an ML problem.
We are evaluating engineering thinking, API fluency, data modeling and clarity of design.

2. Requirements & Constraints

Your backend design must support:

A) Core Conversation Flow

The backend must support a continuous, turn-based chat experience for the user, with two primary modes:

1. **Open Chat Mode**
    - User chats with the LLM on general topics (no additional context).
    - Backend forwards the conversation history directly to the external LLM and returns the reply.

2. **Grounded Chat / RAG Mode (Retrieval-Augmented Generation)**

    - Users can start a conversation that is *grounded in one or more documents* (e.g., PDF, knowledge articles).

- The system must:
    - Allow a user to associate one or more documents with a conversation (e.g., “chat with this PDF”).
    - Retrieve the most relevant chunks/passages from those documents for each new user message.
    - Construct the LLM request using the **current user message + relevant retrieved context** (RAG) instead of sending the entire raw document.

In both modes, the backend is responsible for:

- Managing conversation history.
- Calling the external LLM.
- Persisting both user and assistant messages for future continuation.

B) REST API (CRUD)

Must support:

Operation	**Description**
Create	Start new conversation + first message
Read (list)	Get all past conversations for a user
Read (detail)	Get full conversation history by ID
Update	Add a new message to an existing conversation
Delete	Delete a conversation & its messages

C) LLM Integration

Backend must:

- Call external LLM via API
- Pass relevant conversation history
- Manage token context length
- Parse & store responses

Retrieval-Augmented Generation (RAG) Strategy (Optional but Strongly Preferred)

- If you choose to support grounded conversations over documents, briefly describe:
    - How documents are stored and/or indexed (e.g., chunking, embeddings or simple text search).

    - How you will retrieve relevant pieces of information for each user message.
    - How the retrieved context is combined with the user’s message when calling the LLM.

- You do **not** need to implement actual embedding or vector DB code – we are evaluating your system design and data flow.

D) Data Persistence

Must persist:

- Users
- Conversations
- Messages
- Metadata (timestamps, roles, tokens, etc.)

Allow:

- Listing prior chats
- Resuming existing chats
- Deleting chats

Conversation history and user-specific metadata must be reliably stored in a chosen database.
In addition, your design may optionally include a **Document / Knowledge Base** entity to store or reference:

- Uploaded documents (e.g., PDFs)
- Pre-processed chunks or embeddings for retrieval
- Links between documents and conversations (for “chat with this PDF” style experiences)

3. Your Deliverables

You must submit a **concise design document or slide deck** (PDF preferred) addressing:

A) Architecture & Design

1. **High-level Architecture Diagram**

    - API layer
    - Service layer
    - DB / Persistence Layer
    - LLM integration component

2. **Tech Stack Justification**
   Example:

    - FastAPI + Postgres + Redis
    - OR Node.js + MongoDB + OpenAI API
    - etc.

Explain why.

B) Data & Storage Design

Answer:

- What database would you choose and why?
- How do you store conversation history efficiently?
- How do you maintain message ordering?
- What's your schema or document model?

Expected Entities:

- User
- Conversation
- Message

Bonus:

- Token usage
- Summaries
- Conversation state flags

C) REST API Design

Provide endpoint definitions:

Example format:

```
POST /conversations
Payload: { first_message: "…" }
Response: { conversation_id: "…" }
```

Clarify:

- URL structure
- Method choice (POST vs PUT)
- Pagination (for lists)
- Errors & HTTP codes

D) LLM Context & Cost Management

Must explain:

- How context is constructed
- What happens when history exceeds model limits
- Strategies to reduce cost (tokens = money)

Suggested:

- Sliding windows
- Summarization
- Caching
- System prompts

E) Error Handling & Scalability

Identify **failure points**:

Examples:

- LLM API timeout
- DB write failure
- Token limit breach

Show:

- Fail-safe responses
- Retry strategy
- Logging & monitoring

Scalability:

- Which layer becomes a bottleneck at 1M users?
- How would you scale it? (horizontal, queues, sharding etc.)

F) Deployment & DevOps Skills

- Public GitHub repo
- Dockerfile
- CI pipeline (GitHub Actions)
- At least 2–3 unit tests

We are checking:

- Code hygiene

- CICD familiarity
- Comfort with real software practices

4. Expectations

We are **NOT looking for**:
❌ ML training
❌ Fine-tuning logic
❌ Research papers
❌ HuggingFace models re-implementation

We **ARE looking for**:
✔ Clean backend thinking
✔ REST API maturity
✔ Data modeling skill
✔ Ability to integrate LLMs via API
✔ Cost-aware design
✔ Scalable architecture
✔ Code ownership attitude
✔ Python/Node + cloud readiness
✔ Awareness of real-world constraints

5. Submission Format

Deliverable options:

- PDF document (3–6 pages) or Slide deck (5–10 slides)
- Public GitHub + Markdown README

Must include:

- Diagrams
- API spec
- Data schema
- Design rationale

Optional: submit working prototype (even partial).

BOT GPT Case Study – Candidate FAQ

1. Do I need to write actual code or only submit a design document?

Yes, actual code is required.
This role needs strong software engineering skills, so your submission must include:

- A GitHub repository
- Backend code (Python/Node or your preferred language)
- Basic project structure (folders, modules, environment setup)
- At least 2–3 routes implemented
- API calling logic to ANY free/accessible LLM provider

However, your design document/slide deck is also required.

Both are mandatory.

2. Do I need to build the entire backend fully?

No.
We are evaluating **coding quality + architecture thinking**, not production completeness.

Minimum code expectations:

- Project setup
- Key routes implemented (start conversation, continue conversation, list conversations, delete conversation)
- Integration with an LLM provider
- Basic persistence (can be DB or even local JSON file for simplicity)
- Clear README to run your service

Bonus (Good To Have):

- Proper DB schema
- Dockerfile
- CI/CD pipeline
- Unit tests

3. Do I need to host or deploy the backend?

Hosting is optional.
Running locally is perfectly fine.

4. Which LLM API should I use?

You MUST integrate with a real LLM API.
To avoid cost issues, you can use **any free-tier provider**, such as:

Recommended Free Options

- **Groq API (free)** – Llama models
- **HuggingFace Inference API (free models)**
- **Gemini 1.0 / 1.5 Flash (free tier)** if available
- **Ollama (local models)** if you prefer running locally

You do NOT need GPT-5 or any paid model.

5. Do I need to implement RAG fully?

No, full RAG pipelines (embedding + vector DB) are NOT required.

What we expect:

- The candidate must show **how** RAG would fit into the system
- Optionally simulate retrieval using:
    - Hardcoded chunks
    - JSON/text lookup
    - Simple in-memory search
- Clear architecture design of retrieval flow

We do **not** expect (Good to have though):

- FAISS
- Pinecone
- Chroma DB
- Embedding tuning
- Full vector search implementation

6. How much detail do you expect in the design document?

5–10 slides or a 3–6 page PDF is enough.

Must include:

- Architecture diagram
- API design
- Data model
- LLM call flow
- Context management strategy
- RAG flow
- Scalability & error handling notes

Don’t write a long thesis.
Clarity > length.

7. What database should I use?

Anything is fine:

- SQLite (simple)
- JSON/Local file (acceptable)
- MongoDB Atlas free tier
- PostgreSQL local or in Docker

We are evaluating:

- How you structure data
- How you design your schema
- How you maintain message ordering

NOT advanced DB mastery.

8. Is frontend/UI required?

No.
Use Postman / ThunderClient / cURL / Swagger.
If you want to add a minimal UI (React, HTML, Streamlit), that’s purely optional.

9. What if I cannot finish everything in 48 hours?

You are not expected to finish everything.

We expect:

- A strong attempt
- Clean coding style
- Clear architecture thinking
- Working API endpoints for the core flows
- Correct LLM integration
- Ability to explain your choices during the interview

Your code does NOT need to be production-ready.

10. Is plagiarism checked?

Yes.
We expect original design and code.

Using AI tools (ChatGPT, Copilot) is allowed **for assistance**, but:

- You must understand everything you submit
- You must be able to explain it clearly in the interview
- Direct copy-paste from GitHub or templates will be flagged

11. What is the evaluation criteria?

We look for:

Core Engineering Skills (Most Important):

- Clean, modular code
- API design clarity
- Data modeling quality
- Error handling

GenAI Integration:

- Ability to call an LLM
- Correct context flow
- Basic RAG thinking

Ownership & Communication:

- README clarity
- Explanation during interview
- Approach and reasoning

Bonus points:

- Dockerfile
- CI/CD
- Unit tests
- Structured logging

12. Where should I submit the assignment?

Reply to the email with:

- GitHub repository link
- PDF / slide deck
- Any additional notes (if needed)

13. Can I use frameworks like Django, FastAPI, Express, etc.?

Absolutely.
Choose whatever you're most comfortable with.

Recommended:

- **FastAPI** (Python)
- **Express.js** (Node.js)
- **Flask** (Python)
- **Django Rest Framework** (Python)

14. Is using AI to generate code allowed?

Allowed only if:

- You understand the generated code
- You refine it and adapt it
- You can debug it when asked

Not allowed:

- Blind copy-paste
- Submitting code you cannot explain

We train engineers — we need people who can *think*.

15. Will I be asked to run or demonstrate the code during the interview?

Yes.
Be ready to:

- Walk through your code
- Run your API locally
- Explain your architecture
- Show your LLM call flow
- Discuss improvements