RAG Agent Technical Assessment

Tech Stack:

- LangChain
- Pinecone
- OpenAI GPT-4.x **or** Google Gemini
- (Optional+) Voice Communication Pipeline (STT → RAG → TTS)

🎯 Objective

Build a **Retrieval-Augmented Generation (RAG)** system that can answer user questions grounded in a provided knowledge base, with clear citations and a reproducible setup.

This assessment evaluates your ability to:

- Architect and implement a complete RAG pipeline
- Work with vector databases (Pinecone)
- Use LangChain for orchestration
- Apply grounding, retrieval, and prompting best practices
- Design clean, modular, and documented code
- (Optional+) Add a realtime voice interface

📋 Core Requirements

1. **Ingestion & Indexing**

- Load and preprocess documents (PDF, HTML, or Markdown).
- Split them into chunks with configurable size and overlap.
- Generate embeddings and upsert them into **Pinecone** with metadata.

2. **Retrieval & Generation**
    - Use LangChain retrievers to fetch relevant context.
    - Pass context to **GPT-4 or Gemini** for grounded generation.
    - Return results that include:
        - Answer text
        - Citations (source ID and short snippet)
        - Confidence score
3. **Interface**
    - Provide either:
        - A simple **API endpoint** (`/rag/ask`) using FastAPI or Flask, **OR**
        - A minimal **UI** (Streamlit, Next.js, etc.) with an input field and displayed results.
4. **Evaluation**
    - Include a basic evaluation script (e.g., RAGAS or simple accuracy checks).
    - Provide metrics or comments on answer quality.
5. **Documentation**
    - Include a `README.md` explaining setup, architecture, and instructions to run the app.
    - Provide `.env.example` for all required environment variables.

🎧 Optional (Bonus Task – Voice Interaction)

Build a **realtime voice communication** pipeline:

- Convert speech → text using STT (Whisper, Google Speech, or similar).
- Process the text via your RAG pipeline.
- Convert the agent's response back to speech (TTS) and play it in near realtime.
- UX goal: <2.5 seconds response latency.

(This part is optional but highly valued.)

🧱 Suggested Structure

```
.
├─ data/
├─ ingestion/
│  ├─ loader.py
│  ├─ chunker.py
│  ├─ embed_and_upsert.py
├─ rag/
│  ├─ retriever.py
│  ├─ chain.py
│  ├─ prompts.py
├─ api/
│  ├─ app.py
│  └─ schemas.py
├─ eval/
│  ├─ eval_ragas.py
│  ├─ qa_gold.jsonl
├─ voice/  (optional)
│  ├─ stt.py
│  ├─ tts.py
│  ├─ server.py
├─ tests/
│  ├─ test_ingestion.py
```

```
│   ├── test_retrieval.py
│   ├── test_citations.py
│   ├── test_chain.py
│   ├── test_regression.py
├── config.yaml
├── .env.example
└── README.md
```

🗒️ Deliverables

Please submit:

- GitHub repo (or zipped project folder)
- README with setup and design explanation
- Example queries and sample outputs (with citations)
- Evaluation results or summary
- (Optional) Short demo video (≤5 min)
- (Optional+) Voice interaction demo

🧮 Evaluation Criteria (100 Points)

Area	Point	What We're Looking For
Architecture & Code Quality	15	Clean, modular, maintainable code
Ingestion & Indexing	15	Smart chunking, correct embeddings, metadata
Retrieval & Grounding	15	Relevant context, no hallucination
Prompting & Generation	15	Clear answers with citations
Answer Quality	15	Accuracy, clarity, completeness

Performance & Efficiency	8	Latency, cost awareness
Testing & Evaluation	10	Automated tests or eval scripts
Documentation	7	Setup clarity, decision reasoning
Optional Voice Agent	+10	Smooth STT→RAG→TTS pipeline

Passing Score: 70 +
Outstanding Score: 85 + (voice module or advanced features)