GenAI Developer - Technical Assessment

Overview

This is a practical coding assignment designed to evaluate your hands-on expertise with **LangChain v1** and **LangGraph v1** in building production-grade Generative AI applications.

Problem Statement: Multi-Document RAG System with Agentic Workflow

Build an **intelligent document analysis system** that can:

1. Ingest multiple documents (PDF, TXT, DOCX)
2. Process and store them in a vector database
3. Use a multi-agent architecture to answer complex queries that require reasoning, retrieval, and verification

---

Requirements

1. Document Ingestion Pipeline

What to build:

- Accept multiple document uploads (at least PDF and TXT support required)
- Implement intelligent chunking strategy with overlap
- Generate embeddings and store in a vector database
- Support metadata tagging (document name, upload date, document type)

Technology constraints:

- Must use LangChain v1 document loaders
- Vector DB: Qdrant
- Embeddings: Use OpenAI embeddings
- Chunking: Implement a suitable strategy

Expected output:

- API endpoint: `POST /ingest` accepting file uploads
- Should return: document IDs, chunk count, ingestion status

---

2. RAG Query System

What to build:

- Implement hybrid retrieval (semantic search + keyword search)
- Add reranking mechanism to improve retrieval quality
- Return sources with citations for answers

Technology constraints:

- Use LangChain v1 retrievers
- Implement reranking (use Cohere reranker)
- Must show source documents with relevance scores

Expected output:

- API endpoint: `POST /query` accepting natural language questions
- Response format:

```
{
  "answer": "string",
  "sources": [
    {
      "document": "filename",
      "chunk_id": "string",
      "relevance_score": 0.95,
      "content_snippet": "string"
    }
  ]
}
```

---

3. Multi-Agent Architecture using LangGraph

What to build:

Create a **LangGraph workflow** with at least 3 specialized agents:

Agent 1: Query Analyzer

- Analyzes user query to determine intent
- Classifies query type (factual, comparative, summarization, calculation)
- Extracts key entities and required information

Agent 2: Retrieval & Reasoning Agent

- Performs vector search based on analyzed query
- Retrieves relevant chunks
- Performs multi-hop reasoning if needed (e.g., "Compare X and Y" requires retrieving info about both)

Agent 3: Answer Synthesizer & Verifier

- Synthesizes final answer from retrieved context
- Verifies answer quality and hallucination check
- Adds citations and confidence scores

Technology constraints:

- **Must use LangGraph v1** for agent orchestration
- Implement proper state management between agents
- Use conditional edges based on query classification
- Add at least one tool/function calling capability (e.g., calculator for numerical queries)

Expected output:

- The `/query` endpoint should route through this graph
- Visual representation of your graph structure (can be a diagram or mermaid chart in docs)

---

4. Prompt Engineering & Safety

What to implement:

- Custom system prompts for each agent with clear instructions
- Input validation to prevent prompt injection
- Output guardrails to detect and flag hallucinations
- Token optimization strategy (mention in documentation)

Expected output:

- Document your prompting strategy
- Explain your safety mechanisms in documentation

---

5. Evaluation & Testing

What to provide:

- At least 5 test queries covering different complexity levels:
    i. Simple factual query
    ii. Comparative query (requires multi-document reasoning)
    iii. Summarization query
    iv. Query requiring numerical calculation

    v. Query designed to trigger hallucination checks
- Provide sample documents (at least 3 different PDFs/TXTs) for testing
- Include evaluation metrics:
    - Retrieval accuracy (relevance of retrieved chunks)
    - Answer quality assessment
    - Response time benchmarks

Technical Stack (Required)

Core Framework

- **LangChain v1.x** (latest version)
- **LangGraph v1.x** (latest version)
- **Python 3.10+**

LLM Provider

- OpenAI (GPT-5 models)

Vector Database

- Qdrant

Backend

- FastAPI (required for REST API)
- Pydantic for data validation

Additional Tools

- Docker (containerization required)
- Environment management (.env for API keys)

Deliverables

1. Code Repository

Structure:(this is a reference. you can change as required)

```
project-root/
├── src/
│   ├── agents/          # LangGraph agent definitions
│   ├── chains/          # LangChain chains
│   ├── retrieval/       # RAG pipeline
│   ├── ingestion/       # Document processing
│   ├── api/             # FastAPI routes
│   └── utils/           # Helper functions
├── docs/
│   ├── ARCHITECTURE.md  # System design explanation
│   ├── PROMPTS.md       # Prompt engineering docs
│   └── SETUP.md         # Installation instructions
├── sample_docs/         # Test documents
├── docker-compose.yml
├── Dockerfile
├── requirements.txt
├── .env.example
└── README.md
```

2. Documentation

README.md must include:

- Project overview
- Setup instructions (should work in < 10 minutes)
- API endpoint documentation
- Sample curl commands or Postman collection

ARCHITECTURE.md must explain:

- Your LangGraph workflow design (include diagram)
- Agent responsibilities and interactions

- Vector DB schema design
- Chunking strategy rationale
- Reranking approach
- Design decisions and trade-offs

PROMPTS.md must include:

- All system prompts used
- Prompt engineering techniques applied
- How you handle context window limitations

3. Docker Setup

- Provide `docker-compose.yml` that spins up:
  - Your FastAPI application
  - Vector database
- Should be runnable with: `docker-compose up`

4. Demo Video

- A brief walkthrough showing:
  - Document ingestion
  - Running test queries
  - Explaining agent workflow for one complex query

Evaluation Criteria

Criteria	Weight	What We Look For
LangGraph Implementation	30%	Correct use of StateGraph, conditional edges, proper state management, agent orchestration
RAG Quality	25%	Retrieval accuracy, reranking effectiveness, citation quality, chunking strategy

Criteria	Weight	What We Look For
Code Quality	20%	Clean architecture, modularity, error handling, type hints, docstrings
Documentation	15%	Clarity, completeness, architecture explanation, setup ease
Production Readiness	10%	Docker setup, API design, input validation, safety mechanisms, performance considerations

---

Submission Guidelines

What to Submit:

1. **GitHub repository link** (must be public or share access)
2. **README.md** with clear setup instructions
3. **Sample documents** included in repo
4. **.env.example** with all required environment variables listed
5. **Test results** from your 5 test queries (can be in docs/TESTING.md)

Submission Format:

- Email your repository link to: finny.abraham@focaloid.com
- Subject: "GenAI Developer Assessment - [Your Name]"
- Include a brief cover note (< 200 words) highlighting your design decisions

Deadline:

- As mentioned in the email

Important Notes

What We're NOT Looking For:

- Over-engineered solutions with unnecessary complexity
- UI/Frontend (focus on backend + agents)
- Fine-tuning or training custom models
- Extensive data preprocessing beyond basic document handling

What We ARE Looking For:

- **Deep understanding** of LangChain v1 and LangGraph v1 patterns
- **Practical implementation** of multi-agent systems
- **Production thinking**: error handling, logging, API design
- **Clear documentation** that shows your thought process
- **Working code** that runs successfully with provided setup instructions

Allowed Resources:

- Official LangChain/LangGraph documentation
- Stack Overflow, GitHub discussions
- LLM assistance for boilerplate code (but you must understand everything you submit)

Prohibited:

- Copying existing complete solutions from GitHub
- Using outdated LangChain 0.1.x patterns
- Submitting non-working code

Bonus Points (Optional)

If you complete the core requirements and have time, consider adding:

1. **Streaming responses** for real-time answer generation

2. **Conversation memory** to handle follow-up questions
3. **Observability**: Integration with LangSmith or custom tracing
4. **Benchmark comparison** between different chunking strategies
5. **Multi-modal support**: Add image document processing (OCR)
