DATA SCIENTIST TECHNICAL CASE STUDY

RAG-Based Chatbot Development for Government Technology

POSITION: Data Scientist-IV
EXPERIENCE REQUIRED: 7+ years AI/ML, 2+ years Generative AI
DURATION: 3-4 hours of efforts, however the candidate can take up to 24 hours for submission
EVALUATION: Technical interview follows implementation

EXECUTIVE SUMMARY

You are tasked with developing a Retrieval-Augmented Generation (RAG) chatbot system for Granicus to provide intelligent customer support. This case study simulates a real-world government technology challenge requiring expertise in document processing, vector databases, language models, and enterprise software architecture.

KEY OBJECTIVES

- Build a production-ready RAG system that processes multi-format documents
- Implement strict context grounding to prevent hallucination
- Demonstrate scalable architecture suitable for government applications
- Show mastery of AI/ML engineering best practices

PROBLEM STATEMENT

Business Context

We need an intelligent chatbot that can answer questions about Granicus products and services. The system must handle diverse document formats, provide accurate information, and maintain strict adherence to the provided knowledge base.

Technical Challenge

Develop a RAG system that:

1. **INGESTS MULTI-FORMAT DOCUMENTS**
    - PDF files with complex layouts and tables
    - CSV files with structured pricing and feature data
    - TXT files with FAQ content and technical specifications
2. **PROVIDES ACCURATE, GROUNDED RESPONSES**
    - Answer questions based solely on provided documents

    - Refuse to respond to queries outside the knowledge base
    - Cite sources and provide confidence scores
3. **DEMONSTRATES ENTERPRISE-GRADE ARCHITECTURE**
    - Scalable design for increasing data volumes
    - Proper error handling and edge case management
    - Security considerations for government applications
4. **INCLUDES COMPREHENSIVE TESTING**
    - Unit tests for all components
    - Integration tests for the full pipeline
    - Performance benchmarks and validation

DATASET OVERVIEW

You are provided with **several realistic data files** containing Granicus product information such as shown below. Ingest all files in the vectordb for building Q&A Chatbot.

DOCUMENT FILES

File	Format	Content	Size	Complexity
granicus_products.pdf or .md file	PDF	Complete product catalog, specifications	~15KB	High - Tables, sections
pricing_matrix.csv	CSV	Detailed pricing across all tiers	~3KB	Medium - Structured data
feature_comparison.csv	CSV	Feature matrix by product tier	~2KB	Medium - Comparison table
customer_segments.csv	CSV	Customer types and use cases	~4KB	Medium - Segmentation data
faq_content.txt	TXT	Comprehensive FAQ content	~12KB	Low - Q&A format
technical_specs.txt	TXT	Technical requirements, APIs	~8KB	High - Technical details
release_notes.txt	TXT	Product updates and changes	~6KB	Medium - Chronological

DATA CHARACTERISTICS

- **Total Content**: ~50KB of realistic government technology data
- **Document Types**: Product specs, pricing, features, FAQs, technical docs

- **Data Formats**: Unstructured text, structured tables, mixed content
- **Complexity**: Varying from simple Q&A to complex technical specifications

TECHNICAL REQUIREMENTS

CORE IMPLEMENTATION COMPONENTS

1. DOCUMENT INGESTION PIPELINE

Requirements: - Process PDF, CSV, and TXT files automatically - Extract text while preserving structure and metadata - Implement intelligent chunking strategies - Handle errors gracefully (corrupted files, encoding issues)

Key Challenges: - PDF tables and complex layouts - CSV files with different structures - Maintaining document relationships and context

2. VECTOR STORAGE & RETRIEVAL

Requirements: - Choose and implement appropriate vector database - Generate high-quality embeddings for document chunks - Implement efficient similarity search - Support metadata filtering and hybrid search

Technology Options: - ChromaDB, FAISS, Pinecone, Weaviate - Sentence-transformers, OpenAI embeddings - Cosine similarity, semantic search optimization

3. LANGUAGE MODEL INTEGRATION

Requirements: - Integrate with LLM service (OpenAI, Hugging Face, Anthropic, or local) - Design effective prompts for government/product queries - Implement response validation and grounding checks - Add confidence scoring and uncertainty handling. No API key will be provided by Granicus.

Key Considerations: - Prompt engineering for accurate responses - Preventing hallucination and off-topic answers - Handling ambiguous or incomplete information

4. API & USER INTERFACE

Requirements: - API with proper HTTP methods - Input validation and sanitization - Error handling and status codes - Optional: Simple web interface or CLI

Endpoints to Implement: - `POST /chat` - Main query endpoint - `GET /health` - System health check - `GET /stats` - System statistics - `GET /docs` - API documentation

IMPLEMENTATION CONSTRAINTS

TECHNOLOGY STACK

- **Primary Language**: Python 3.9+
- **Web Framework**: FastAPI (recommended) or Flask
- **Dependencies**: Provided in requirements.txt
- **Optional**: Docker for containerization

QUALITY STANDARDS

- **Code Quality**: Clean, readable, well-documented code
- **Testing**: Minimum 70% test coverage
- **Documentation**: Clear setup and usage instructions
- **Error Handling**: Comprehensive error management
- **Security**: Basic input validation and API security

PERFORMANCE TARGETS

- **Response Time**: <3 seconds for simple queries, <10 seconds for complex
- **Concurrent Users**: Handle at least 10 simultaneous requests
- **Memory Usage**: Efficient resource management
- **Accuracy**: >90% correct answers for in-scope questions

SAMPLE QUESTIONS FOR TESTING

Your system will be evaluated using questions like these:

IN-SCOPE QUESTIONS (Should Answer Correctly)

1. **Product Information**
    - "What are the key features of GovDelivery Communications Cloud?"
    - "What products does Granicus offer for government organizations?"
2. **Pricing Queries**
    - "How much does the Enterprise plan cost for 100,000 subscribers?"
    - "What's included in the Professional tier pricing?"
3. **Technical Specifications**
    - "What are the API rate limits for different plan tiers?"
    - "What integrations are available with Granicus products?"
4. **Feature Comparisons**
    - "What's the difference between Starter and Professional plans?"
    - "Which plan includes advanced analytics features?"

OUT-OF-SCOPE QUESTIONS (Should Refuse)

1. **External Information**
    - "What's the weather like today?"
    - "How does Granicus compare to Mailchimp?"
2. **Personal/Confidential**
    - "What's the CEO's personal phone number?"
    - "Can you provide customer contact information?"
3. **Opinion-Based**
    - "Which Granicus product is the best?"
    - "Should I choose Granicus over competitors?"

DELIVERABLES CHECKLIST

CODE DELIVERABLES

- ☐ **Complete RAG Implementation** in */src/* directory
- ☐ **Comprehensive Test Suite** in */tests/* directory
- ☐ **Configuration Files** (requirements.txt, .env, etc.)
- ☐ **Docker Configuration** (optional but recommended)

DOCUMENTATION DELIVERABLES

- ☐ **README.md** with setup and usage instructions
- ☐ **Architecture Overview** explaining design decisions
- ☐ **API Documentation** with endpoint descriptions
- ☐ **Performance Analysis** with benchmarks and metrics

DEMONSTRATION DELIVERABLES

- ☐ **Working Demo** showing key functionality
- ☐ **Test Results** proving system reliability
- ☐ **Sample Interactions** demonstrating capabilities
- ☐ **Performance Metrics** showing response times and accuracy

SUCCESS STRATEGIES

MINIMUM VIABLE SOLUTION (70+ Points)

Focus on these core elements first: 1. **Basic document processing** for all three file formats 2. **Simple vector storage** with similarity search 3. **LLM integration** with context-grounded responses 4. **API endpoints** with basic error handling 5. **Essential tests** covering main functionality

EXCELLENCE INDICATORS (90+ Points)

Demonstrate advanced capabilities: 1. **Sophisticated RAG techniques** (hybrid search, re-ranking) 2. **Production-ready architecture** with proper scaling 3. **Advanced error handling** and edge case management 4. **Comprehensive testing** including performance benchmarks 5. **Security features** and monitoring capabilities

IMPLEMENTATION TIPS

- **Start Simple**: Get basic functionality working before adding complexity
- **Test Early**: Implement tests alongside features for better reliability
- **Document Decisions**: Explain your technology choices and trade-offs
- **Handle Errors**: Government applications require robust error handling
- **Think Production**: Consider scalability and deployment requirements

COMMON PITFALLS TO AVOID

- **Over-engineering**: Don't spend too much time on advanced features at the expense of core functionality
- **Ignoring Error Handling**: Government applications require robust error management
- **Poor Testing**: Inadequate tests will significantly impact your score
- **Unclear Documentation**: Evaluators need to understand and run your code
- **Scope Creep**: Focus on the specific requirements rather than adding unnecessary features

GOOD LUCK!

This case study is designed to assess your real-world AI engineering capabilities in a government technology context. Focus on demonstrating your expertise in RAG systems, clean code practices, and enterprise software development.

We look forward to seeing your innovative approach to building reliable, scalable AI systems for public sector applications!