System Design Document: AI Microservice

Executive Summary

Production-grade RAG-based document Q&A microservice with agentic workflow (LangGraph), multi-format ingestion (PDF/DOCX/CSV/JSON/TXT), and async-first FastAPI architecture. Supports 10K+ queries/day with ChromaDB vector search and HuggingFace LLMs.

Architecture Overview

Layered Design: API Gateway · Service Layer · Agent Layer · Data Layer

```
·····················································
·  FastAPI (Routes) · DocumentService + ChatService  ·
·    ·                                                ·
·  LangGraph Agent · Assistant (LLM) · Tools (RAG)   ·
·    ·                                                ·
·  PostgreSQL + ChromaDB + HuggingFace Embeddings    ·
·····················································
```

Core Components:

- **API Gateway:** FastAPI async endpoints (`/upload`, `/chat`, `/documents`)
- **Services:** DocumentService (ingestion pipeline), ChatService (logging, metrics)
- **Agent:** LangGraph state machine with MemorySaver, tool-calling LLM, RAG retrieval
- **Data:** SQLAlchemy (metadata/logs), ChromaDB (vectors), HuggingFace (embeddings)

Data Pipelines:

- *Ingestion:* Upload · Validation · Parse · Chunk · Embed · Store (DB + Vector)
- *Query:* Question · Agent · Semantic Search · LLM Synthesis · Response + Log

---

Key Design Trade-offs

Architecture:

- Async/Await: High I/O concurrency ·· Complex error handling
- LangGraph: State persistence, tools ·· LangChain lock-in

Data:

- ChromaDB: Lightweight, persistent ·· >10M vector limit
- HuggingFace: Cost-effective, private ·· Slower, needs GPU
- File storage: Simple, debuggable ·· 100K doc limit
- SQLite/PostgreSQL: Fast prototyping ·· Migration complexity

Performance: Connection pooling, lazy loading, batch embeddings

Security: Pydantic validation, file whitelist, CORS config, **no auth** (add OAuth2/JWT for prod)

Scaling Roadmap

Current: Single Docker Instance (<1K req/day)

- FastAPI + SQLite + ChromaDB on single VM
- Suitable for prototyping and MVP

Short-term: Horizontal Scaling (<10K req/day)

- Multiple FastAPI instances behind Nginx/ALB
- Shared PostgreSQL (connection pooling)
- Single ChromaDB (<1M vectors)

Medium-term: Distributed Architecture (10K-100K req/day)

```
Load Balancer · [FastAPI Instances] · PostgreSQL (Primary + Replicas)
                      ·                   ·
                ChromaDB/Milvus      Redis Cache
```

- **Bottlenecks:** LLM inference (10-30s/query), vector search, PDF parsing
- **Solutions:** GPU instances (T4/A10G), 4-bit quantization, Celery workers

Long-term: Microservices (>100K req/day)

- Separate services: Document Ingestion, Query Engine, Embedding API
- Event-driven (Kafka/RabbitMQ) for async processing
- Database partitioning (by thread_id/timestamp), read replicas

Cost Optimization:

- Self-hosted GPU: $0.50/1K · $0.05/1K queries
- Spot instances: $50/mo · $15/mo compute
- Managed vector DB free tier

Monitoring & Production Readiness

Current: Structured logging, latency tracking (`latency_ms` column)

Recommended:

- Metrics: Prometheus + Grafana (RPS, P50/P95/P99 latency, errors)
- Tracing: OpenTelemetry (already has dependencies)
- Alerting: PagerDuty for API downtime, DB failures
- Logs: ELK/Loki stack

Future Enhancements:

1. Multi-tenancy (namespace isolation in ChromaDB)
2. WebSocket streaming (progressive answers)
3. Hybrid search (keyword + semantic) + re-ranking
4. Model fine-tuning on user feedback
5. Audit trail for compliance