NeuroChain Agent Suite — Implementation Blueprint

A complete, interview-ready implementation plan: folder structure, ERD, architecture diagrams, API schemas, LangGraph node templates, Next.js component architecture, deployment notes, branding, and a pitch-deck outline.

Table of Contents

---

1. Project overview & goals

NeuroChain Agent Suite is a production-grade multi-agent platform that provides: document intelligence, automated workflows, tool-using agents, hybrid retrieval (vector + knowledge graph), secure FastAPI microservices, and a Next.js admin console with real-time streaming. The primary hiring signal this project demonstrates: system design for agents, RAG expertise, tool orchestration, secure APIs, and polished UX.

MVP goals (8 weeks): - Core RAG ingestion & retrieval - One end-to-end agentic workflow (Document Q&A → Action) - FastAPI endpoints + authentication - Next.js console with streaming chat UI - LangGraph orchestration with 3 agent types

---

2. High-level architecture

- **Frontend (Next.js)**: user console, document upload, chat interface, workflow monitor, analytics.
- **API (FastAPI)**: authentication, agent endpoints, ingestion endpoints, webhooks, admin APIs.
- **LangGraph (or LangChain orchestration layer)**: agent definitions, workflows, state machine, tool bindings.
- **RAG infra**: PDF extractor, text chunking, embeddings pipeline, vector store (Milvus/Weaviate/Pinecone), metadata store (Postgres)
- **Knowledge Graph**: optional Neo4j/ArangoDB for relationship queries and entity linking

- **Storage**: S3-compatible for raw files; Redis for cache & ephemeral convo state
- **Observability**: OpenTelemetry traces, Prometheus metrics, Grafana dashboards

---

3. Folder structure (monorepo)

```
neurochain/
├─ infra/                      # IaC (Terraform / Pulumi) and deployment
scripts
├─ services/
│  ├─ api/                      # FastAPI service
│  │  ├─ app/
│  │  │  ├─ main.py
│  │  │  ├─ api/
│  │  │  ├─ models/
│  │  │  ├─ services/
│  │  │  └─ workers/
│  │  └─ Dockerfile
│  ├─ ingestion/                # ingestion + embedding workers
│  ├─ agents/                   # LangGraph/agent node definitions
│  └─ knowledge-graph/          # optional graph DB integration
├─ web/                        # Next.js app
│  ├─ app/
│  └─ components/
├─ scripts/                    # helper scripts (db migrate, seed)
├─ tests/
└─ README.md
```

Notes: - Keep LangGraph code in `services/agents/` so agent lifecycle is colocated. - Use Dockerfiles per service and a docker-compose for local dev.

---

4. ERD & data model

Key entities: - **users** (id, org_id, email, role, hashed_password, created_at) - **organizations** (id, name, billing_info) - **threads** (id, org_id, user_id, title, created_at) - **messages** (id, thread_id, role, content, metadata, created_at) - **documents** (id, org_id, uploader_id, s3_path, mime, created_at) - **chunks** (id, document_id, text, embedding_id, vector_index, metadata) - **workflows** (id, name, state, definition) - **agents** (id, name, type, config)

Foreign keys: documents -> orgs, messages -> threads -> orgs Store embeddings metadata separately for easy reindexing.

---

5. FastAPI — API surface & schemas

Authentication

- `/auth/register` (POST) → returns access token
- `/auth/login` (POST)
- JWT + refresh tokens, role-based scopes

Agent endpoints

- `POST /agents/execute` → { agent_name, input, config } → returns job_id
- `GET /agents/status/{job_id}`
- `GET /agents/result/{job_id}`

RAG & ingestion

- `POST /ingest/document` (multipart) → schedules embedding job
- `GET /documents/{id}/summary`
- `POST /rag/query` → {query, top_k, filters}

Workflows

- `POST /workflows/start` → starts LangGraph workflow
- `GET /workflows/{id}/trace` → returns execution trace/logs

Webhooks & tools

- `POST /tools/webhook/{tool_name}`

Schemas (Pydantic examples)

```
# app/schemas.py
class RagQuery(BaseModel):
    query: str
    top_k: int = 5
    filters: Optional[Dict[str, Any]] = None

class AgentExecuteRequest(BaseModel):
    agent_name: str
    input: Dict[str, Any]
    config: Optional[Dict[str, Any]]
```

---

6. LangGraph / LangChain — node templates & orchestration patterns

Node pattern: `Retriever -> Reasoner -> ToolExecutor -> Validator`

Provide a Python template for each node type. Example: `agent_tasks/document_qna.py`

```python
# agents/document_qna.py
from langchain_core.messages import HumanMessage, AIMessage

class DocumentQnAAgent:
    def __init__(self, retriever, llm, tools):
        self.retriever = retriever
        self.llm = llm
        self.tools = tools

    def run(self, query, context={}):
        docs = self.retriever.get_relevant_documents(query)
        prompt = build_prompt(query, docs)
        response = self.llm.generate(prompt)
        # optional: call tools or validators
        return response
```

LangGraph orchestration

- Represent workflows as DAGs with guarded transitions.
- Provide retry and compensating actions in nodes.
- Logging hooks and observability (trace IDs passed via metadata).

---

7. RAG pipeline & vector store strategy

- **Embeddings**: OpenAI / Cohere / local (Mosaic, SentenceTransformers)
- **Vector store**: Pinecone (managed) / Milvus (self-host) / Weaviate
- **Chunking**: smart chunking by semantic boundaries (sentences, sections)
- **Metadata**: store source, doc_id, page_no, offsets
- **Reindex strategy**: background jobs for updates, ensure idempotency

Advanced: hybrid retrieval — use vector search to get nodes, then run a graph traversal on knowledge graph to expand relevant entities.

---

8. Next.js frontend — pages & components

Pages

- `/` → Dashboard
- `/console` → Command console (chat + agents)
- `/documents` → upload & document library
- `/workflows` → run & monitor workflows
- `/analytics` → agent usage, latency, cost

Key components

- `ChatStream` — SSE/WS streaming chat component
- `Uploader` — multi-file uploader with progress

- `WorkflowVisualizer` — visualize LangGraph state transitions
- `AgentPlayground` — test-run agents and inspect trace
- `Auth` components — login/register

Streaming approach: use SSE or WebSocket (FastAPI `EventSourceResponse` or `websockets`) to deliver token-by-token output.

---

9. Dev tooling, CI/CD, security & observability

- **CI**: GitHub Actions — lint, tests, build images
- **CD**: Vercel for Next.js; AWS ECS/EKS or App Runner for FastAPI; image registry on ECR
- **Secrets**: AWS Secrets Manager / HashiVault
- **Observability**: OpenTelemetry traces (instrument LLM calls + LangGraph nodes), Prometheus + Grafana for metrics
- **Security**: Input validation, rate limits, per-org quotas, encryption at rest (S3), TLS
- **Testing**: unit tests + integration tests for agent workflows using recorded traces (playback)

---

10. Deployment & cost estimate (AWS/Vercel)

MVP stack (small scale): - Vercel: Next.js (Hobby/Pro) - AWS: ECS Fargate for FastAPI, RDS Postgres (db.t3.medium), S3, ElastiCache Redis - Vector store: Pinecone Starter (~$50–100/month) or Milvus self-hosted on t3.xlarge - Approx monthly cost (MVP): $200–800 depending on vector store and LLM usage

For production, include autoscaling, multiple worker types, and observability - costs rise accordingly.

---

11. Branding, README, and pitch deck outline

Project name: NeuroChain Agent Suite **One-liner:** "An enterprise-grade agent fabric that converts documents into actionable workflows using hybrid retrieval and deterministic agent orchestration."

README should include:

- Project summary
- Architecture diagram
- Quickstart (local dev with docker-compose)
- How to run ingestion, how to run agents
- Environment variables
- How to run tests

Pitch Deck (6 slides):

1. Problem & Opportunity
2. Solution: NeuroChain (pic of architecture)
3. Demo flow (document -> agent -> action)
4. Tech Stack & differentiators
5. Business impact & use-cases
6. Roadmap & ask (for interviews: what you built)

---

12. Next steps & sprint plan (MVP → v1)

Sprint 0 (1 week): Repo scaffold, docker-compose, basic FastAPI, Next.js shell, auth **Sprint 1 (2 weeks)**: RAG ingestion pipeline, vector store, embedding worker **Sprint 2 (2 weeks)**: LangGraph agents: DocumentQnA + ToolExecutor, end-to-end demo **Sprint 3 (2 weeks)**: Frontend streaming chat, workflow monitor, tests **Sprint 4 (1 week)**: Polish, docs, pitch deck, deployment

---

Deliverables I can provide right now (pick any):

- Full repo scaffold (Docker + compose + basic endpoints)
- LangGraph node templates for the 3 core agents
- FastAPI agent executor example with SSE streaming
- Next.js streaming chat component
- Pitch deck markdown + images (wireframes)

Tell me which deliverable you want first and I will generate it.