RAG Document Ingestion & Retrieval — File & Folder Structure Note

1. Purpose

Define a **clean, scalable repository structure** that directly reflects system components and responsibilities.

This structure is designed to: - Prevent tight coupling - Make ownership clear - Support future scaling (services / pipelines) - Be understandable to mentors, PMs, and new engineers

No framework or language is assumed.

---

2. Core Principle (Industry Rule)

> **Folders represent responsibilities, not technologies.**

If a folder name contains a tool name, the design is already leaking.

---

3. High-Level Repository Layout

```
rag-system/
│
├── docs/
│   ├── design-note.md
│   ├── implementation-approach.md
│   ├── tools-component-mapping.md
│   └── file-structure.md
│
├── config/
│   ├── rules/
│   │   ├── parsing-rules.yaml
│   │   ├── promotion-rules.yaml
│   │   └── trust-thresholds.yaml
│   │
│   └── environments/
│       ├── dev.yaml
│       └── prod.yaml
│
├── core/
```

```
│   ├── registry/
│   ├── probe/
│   ├── decision_engine/
│   ├── parsing/
│   ├── quality/
│   ├── promotion/
│   └── orchestration/
│
├── storage/
│   ├── sql/
│   ├── object_store/
│   ├── vector/
│   └── graph/
│
├── retrieval/
│   ├── query_router/
│   ├── vector_search/
│   ├── graph_reasoning/
│   └── response_builder/
│
├── workflows/
│   ├── ingestion_flow/
│   ├── parsing_flow/
│   └── promotion_flow/
│
├── shared/
│   ├── models/
│   ├── schemas/
│   ├── utils/
│   └── logging/
│
├── tests/
│   ├── unit/
│   ├── integration/
│   └── replay/
│
└── README.md
```

---

4. Folder Responsibilities (What Goes Where)

4.1 `/docs`

Design and decision artifacts.

Why: Architecture lives longer than code.

4.2 `/config`

All **rules and thresholds** live here — not hardcoded.

- Parsing strategies
- Promotion thresholds
- Trust policies

Why: Allows tuning without redeploying code.

4.3 `/core`

Contains **business logic**, independent of storage or tools.

Subfolders: - `registry/` → document lifecycle - `probe/` → structure inspection - `decision_engine/` → logical parameters - `parsing/` → deterministic + ML routing - `quality/` → confidence & trust scoring - `promotion/` → routing to vector / graph - `orchestration/` → workflow coordination

Rule: No external system calls directly from here.

4.4 `/storage`

All persistence logic.

- `sql/` → authoritative store
- `object_store/` → PDFs, images
- `vector/` → embeddings
- `graph/` → relationships

Rule: Storage layers are dumb and replaceable.

4.5 `/retrieval`

User-facing query resolution.

- `query_router/` → decides vector vs graph
- `vector_search/` → semantic retrieval
- `graph_reasoning/` → relationship queries
- `response_builder/` → grounded responses

Guarantee: Responses must reference SQL-backed data.

4.6 `/workflows`

Defines **end-to-end flows**.

- Ingestion
- Parsing
- Promotion

Why: Keeps orchestration separate from logic.

4.7 `/shared`

Common definitions.

- Data models
- Schemas
- Utilities
- Logging

Rule: No business decisions here.

4.8 `/tests`

Testing mirrors architecture.

- Unit tests per component
- Integration tests per pipeline
- Replay tests for reprocessing documents

---

5. Why This Structure Works

- Mirrors system architecture exactly
- Prevents cross-layer leakage
- Encourages clean abstractions
- Makes scaling to microservices trivial
- Easy for reviewers to understand

---

6. Migration & Scaling Path

- Start as a modular monolith
- Extract `/core` subfolders into services later
- Storage adapters remain unchanged
- Workflows evolve independently

---

7. One-Sentence Summary

“This repository structure enforces trust, separation of concerns, and long-term maintainability by construction.”

End of file & folder structure note.