Take-Home Challenge: “Unstructured → Structured” Extraction Service

Estimated challenge time: 6-8 hours

0) Scenario (what the stakeholder asked)

“We’d like a new feature that converts unstructured documents (up to 100k whitepapers in S3/SharePoint/etc.) into structured JSON. Users pick what metadata to extract — e.g., Author Names, Publish Date, Abstract Summary, Code Snippets — and the service returns normalized JSON for each document. LLMs can help.”

1) Your Deliverables (what we expect back)

1. **Technical PRD (**
    a. Problem statement, goals & non-goals
    b. Target users & personas
    c. Success metrics / SLAs (throughput, latency, accuracy targets, cost bounds)
    d. Functional requirements (multi-tenant, storage connectors, extraction config, result delivery)
    e. Non-functional requirements (security, PII handling, auditability, observability, cost controls)
    f. Acceptance criteria & phased rollout plan
2. **System Design**
    a. High-level architecture (components, queues, stores, LLM providers)
    b. Data model
    c. Sequence diagrams for:
        a) Create job → crawl → extract → deliver results
        b) Retries, backoff, idempotency
    d. Scaling strategy (100k docs, concurrency, backpressure, rate limits, cost caps)
    e. Observability (metrics, logs, traces, dashboards) & ops runbook (SLOs, alerts)

3. **Implementation Plan**
    a. Milestones (MVP → Beta → GA) with cutlines
    b. Risk register (top 5 risks) & mitigations
    c. Effort estimate & roles (who does what)
    d. Build vs buy choices (LLM, OCR, vector store, connectors) and decision criteria
4. **Code (mission-critical component)**
    a. Build the **Extraction Engine Service** (details in §2 below).
    b. A runnable demo.
5. **README + Quickstart**
    a. One-command local run (e.g., docker compose up or make dev)
    b. Example config & sample output
    c. Notes on tradeoffs and what you'd do next

2) Mission-Critical Coding Task (what to build)

Scope & Constraints (so work is focused)

- Treat connectors as **adapters**. Implement **one** of:
    - Local filesystem folder (required), **and optionally** S3 (using LocalStack) or SharePoint (mock).
- Assume documents are **PDF or Markdown/text**. You can skip OCR for scanned PDFs.
- LLM usage is **your choice**: real API, open-source model, or a **mock LLM** (deterministic) for offline repeatability.
- Support **configurable extraction schema** (the user declares fields & hints).
- Input scale target: design for 100k docs; your demo can run on a mininum of 5 files (ex: https://arxiv.org/pdf/2005.04611) .
- Return format: **JSONL** (one JSON per line) and a **job status API**.

Requirements: Extraction Engine Service

A stateless microservice + lightweight job runner that:

1. **Accepts a Job**
2. **Retrieves Documents**
    a. Retrieves files requested from payload in the source adapter (filesystem required; S3 optional).
3. **Extracts Fields**
    a. For each doc:
        i. Chunk if needed; call ExtractorProvider (LLM or mock) with the schema.
        ii. Validate & coerce types (e.g., date parsing).
        iii. Produce a **normalized JSON** record:

4. **Handles Scale & Reliability**
    a. Batching & concurrency (configurable), retry with backoff, idempotency (re-runs don't duplicate).
    b. Backpressure / rate limiting knobs.

Tech guardrails

- Use **Python** or **TypeScript/Node.js**.
- Clean, idiomatic structure (e.g., hex/ports-adapters):

Sample Documents (min 5 for demo)

- Provide sample white papers such as (ex: https://arxiv.org/pdf/2005.04611) :
    - Clear authors, ambiguous dates, abstracts, and papers with and without a few code blocks.

Bonus:

- Provide **unit tests** (core logic), and a tiny **load test** (e.g., 100 docs x 2KB).
- Observability: Basic metrics endpoint (e.g., Prometheus style): processed, succeeded, failed, p50/p95 lat.

3) Evaluation Rubric (100 pts)

- **Technical PRD (20 pts)**
  Clear goals, measurable success, crisp acceptance criteria, compliance awareness.
- **System Design (30 pts)**
  Architecture quality, correctness of data contracts, scaling & reliability, observability, security posture.
- **Implementation Plan (10 pts)**
  Realistic phases, risks/mitigations, resourcing, rationale for buy vs build.
- **Code Quality (30 pts)**
  Elegance, tests, readability, interfaces (providers/adapters), correctness, idempotency.
- **DX & Docs (10 pts)**
  One-command run, example configs, decision log, tradeoffs.

Bonus (up to +10): S3 adapter via LocalStack; simple web UI; basic cost controls (QPS/TTFT caps); schema validation via Pydantic/Zod; SBOM or CI lints.

4) Expected Effort & Timing

- Target **6–8 hours** of effort; OK to stub non-critical pieces.
- If time is not enough for all requirements, make a decision on what to cut, please note clearly what you **cut** and why.

8) Submission

- Git repo (public or zipped) with code, docs, and diagrams (draw.io/Excalidraw ok).
- Include a **Decision Log** (/DECISIONS.md) summarizing key tradeoffs in bullets.

9) What “Great” Looks Like

- A crisp PRD that makes the problem legible and testable.

- A design that’s boring-in-the-right-ways: queues, idempotent workers, observability.
- A neat engine that runs locally, processes a folder, and gives confident, validated JSONL with metrics and retries.
- Thoughtful handling of cost, rate limits, and accuracy/QA (e.g., spot-checks or dual-provider compare).