Backend Developer Task (FastAPI)

Allowed: Any LLM, frameworks, libraries (explain your choices in the README)
Not allowed: Copy-paste of tutorials without understanding
Submission: Working code (or runnable skeleton) + README with design decisions + short answers to the role-specific questionnaire below.

We are evaluating **thinking and design**, not polish.

Objective

Build a **secure, well-structured API** for document lifecycle management — **without** implementing OCR or LLM logic. This is an API engineering and system design task.

What to Build

1. Authentication (Mandatory)

- **JWT access token** and **refresh token**
- Token expiry handling

Endpoints:

```
POST /auth/login
POST /auth/refresh
```

2. Document APIs

Endpoint	Responsibility
`POST /documents/upload`	Accept PDF/DOCX, store file locally, save metadata to DB, log events. Return e.g. `{ "document_id": "..." }`.
`GET /documents`	Return list: document ID, filename, status, created time.
`POST /documents/{id}/process`	Validate document exists, update status to PROCESSING, run processing **asynchronously**. For this task **simulate** processing (e.g. sleep then set COMPLETED + mock result). No real LLM.
`GET /documents/{id}/status`	Return current status, optional progress/message. Values: UPLOADED, PROCESSING, COMPLETED, FAILED.
`GET /documents/{id}/result`	When COMPLETED, return processed output (mock JSON/text). Otherwise appropriate error.

3. Storage

- **Files:** Local file storage (abstraction so you could swap to S3 later).
- **DB:** Metadata (SQLite OK): document_id, filename, file_path, status, timestamps. Optionally store result (e.g. extracted JSON).
- Clear separation between file storage and metadata.

4. Logging (Important)

- **Structured logs** (e.g. JSON or key-value)
- Request lifecycle: request_id, method, path, status, duration
- Document lifecycle: document_id, event (uploaded, process_started, process_completed, process_failed)
- Errors with context, no secrets in logs

5. CI/CD Preparation

You do **not** need to deploy. Include:

- **Dockerfile** that builds and runs the API
- **Basic CI** (e.g. GitHub Actions / GitLab CI): Lint, Tests (minimal OK), Build

---

Out of Scope

- OCR, LLM, LangGraph
- WebSocket/SSE (polling is enough)
- RBAC

Questionnaire (answer in README)

1. Why should processing be **decoupled** from upload?
2. How would you make `POST /documents/{id}/process` **idempotent**?
3. How would you scale this to **1,000 concurrent uploads**?
4. Where would the LangGraph/ML pipeline fit later?

Deliverables

- [ ] Working FastAPI app (runnable locally and/or via Docker)
- [ ] JWT login + refresh; all five document endpoints above
- [ ] File storage + DB; structured logging; Dockerfile + basic CI
- [ ] README with setup, design decisions, and answers to the four questions above