04 LDIP 보고서 B — 모듈별 상세 사양

진행 상태	읽기 전

프로젝트: Project 4 · 기반: Flask(Python) · 작성: 2025-09-15

본 문서는 코어 데이터 경로(업로드→파싱→탐지→컨텍스트→마스킹/암호화)의 **목적·근거·설계 원칙·의존 라이브러리·실행 코드 예시·API/DB/시퀀스·품질 기준**을 상세화함. 거버넌스/정책/키관리/DevOps는 **D2** 문서 참조.

---

0) 공통 개요

0.1 목적

- 민감정보(PII) 유출 최소화 및 합법적 활용을 위한 **안전한 문서 처리 파이프라인** 구축
- 파일 정본은 **암호문 우선** 원칙, 검토자에게는 **최소 필요 컨텍스트** 제공
- 작업은 비동기 큐로 분산 처리하여 **성능·안정성** 확보

0.2 설계 근거

- Python/Flask 생태계의 광범위한 라이브러리(파싱/암호화/큐) 및 운영성
- AES-256-GCM(무결성 보장), AWS KMS(키 수명주기·감사), Celery/Redis(표준 비동기)
- 텍스트형 우선(MVP) → 변환/OCR 확장(단계적 리스크 관리)

0.3 공통 라이브러리·런타임

- 웹/API: **Flask**, **Gunicorn**
- 큐/워커: **Celery**, 브로커/백엔드 **Redis**
- 암호화: **cryptography**(AESGCM), **boto3**(KMS)
- 파싱: **pdfminer.six**, **pypdf**, **python-docx**, **openpyxl**
- 유틸: **pydantic**(스키마 검증), **sqlalchemy**(ORM), **alembic**(마이그레이션)

0.4 공통 DB 테이블(요약)

- `documents(id, owner_id, filename, mime, size, status, created_at)`
- `document_versions(id, doc_id, v_no, is_encrypted, is_masked, enc_algo, enc_iv_b64, created_at)`
- `pii_findings(id, doc_id, v_no, category, start, end, rule_version, context, value_hash)`
- `doc_keys(id, doc_id, v_no, mode, kms_ciphertext_b64, aad_json, created_at)`
- `audit_log(id, actor_id, action, resource, detail_json, ts)`

0.5 환경 변수(.env)

```
DATABASE_URL=postgresql+psycopg2://app:app_pw@db:5432/appdb
REDIS_URL=redis://redis:6379/0
UPLOAD_ROOT=/data/uploads
AWS_REGION=ap-northeast-2
KMS_KEY_ID=alias/ldip-data
ENCRYPTION_MODE=KMS
FLASK_ENV=production
```

---

1) Q — 작업큐/워커 모듈

목적: 오래 걸리는 작업(파싱/암호화/마스킹)을 비동기로 처리하여 API 응답 성능과 안정성 확보

1.1 포함 내용(체크리스트)

- 큐/워커 토폴로지 정의, 우선순위 큐(`enc`, `parse`, `mask`)
- 재시도/백오프/타임아웃/사망 큐(DLQ) 정책
- Idempotency 키, 작업 상태 조회 API, 감사 로깅

1.2 의존 라이브러리

- `celery`, `redis`

1.3 최소 실행 코드

```
# app/tasks/__init__.py
from celery import Celery
import os
```

```
celery = Celery(
    __name__,
    broker=os.environ.get("REDIS_URL"),
    backend=os.environ.get("REDIS_URL"),
)
celery.conf.task_queues = {
    "enc": {},
    "parse": {},
    "mask": {},
}
celery.conf.task_default_queue = "parse"
celery.conf.task_routes = {
    "encrypt_document_task": {"queue": "enc"},
    "parse_document_task": {"queue": "parse"},
    "mask_document_task": {"queue": "mask"},
}
```

```
# app/api/jobs.py
from flask import Blueprint, jsonify
bp = Blueprint("jobs", __name__)

@bp.get("/jobs/<job_id>")
def job_status(job_id):
    # Redis에서 상태 조회(예: celery AsyncResult)
    from celery.result import AsyncResult
    from app.tasks import celery
    r = AsyncResult(job_id, app=celery)
    return jsonify({"status": r.status, "result": r.result if r.successful() else No
ne})
```

1.4 시퀀스

- API → Redis 큐잉 → 워커 처리 → 상태 업데이트 → 감사 로그 기록

1.5 품질 기준

- 실패 3회 재시도 후 `FAILED` 기록, DLQ 보존 ≥ 7일
- 동일 Idempotency 키 중복 큐잉 방지

---

2) PAR — 문서 파싱 모듈

목적: 문서에서 텍스트·메타(페이지/단락/셀)를 추출하여 **탐지(DET)** 가 분석 가능하도록 표준화

2.1 포함 내용(체크리스트)

- 지원 타입: **PDF(텍스트), DOCX, XLSX, TXT**
- 비지원(초기): 스캔PDF/이미지, HWP, PPT/PPTX → D2의 convert/OCR로 확장
- 정규화: UTF-8, NFC, 공백 정리, 페이지·단락·셀 locator 태깅

2.2 의존 라이브러리

- `pdfminer.six`, `pypdf`, `python-docx`, `openpyxl`

2.3 최소 실행 코드

```
# app/services/parser.py
from pdfminer.high_level import extract_text as pdf_text
from docx import Document as Docx
from openpyxl import load_workbook
from pathlib import Path

class ParseResult:
    def __init__(self, text: str, meta: dict):
        self.text, self.meta = text, meta

class Parser:
    def parse(self, path: Path, mime: str) → ParseResult:
        if mime == "application/pdf":
            t = pdf_text(str(path))
            meta = {"pages": _count_pdf_pages(path)}
            return ParseResult(_normalize(t), meta)
        if mime == "application/vnd.openxmlformats-officedocument.wordpro
cessingml.document":
            d = Docx(str(path))
            paras = [p.text for p in d.paragraphs]
            t = "\n".join(paras)
            meta = {"paragraphs": len(paras)}
            return ParseResult(_normalize(t), meta)
```

```
        if mime == "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet":
            wb = load_workbook(str(path), data_only=True)
            cells = []
            for ws in wb.worksheets:
                for row in ws.iter_rows(values_only=True):
                    for v in row:
                        if isinstance(v, str):
                            cells.append(v)
            return ParseResult(_normalize("\n".join(cells)), {"sheets": len(wb.worksheets)})
        if mime.startswith("text/"):
            t = Path(path).read_text(encoding="utf-8", errors="ignore")
            return ParseResult(_normalize(t), {})
        raise ValueError("PARSE_UNSUPPORTED")

def _normalize(text: str) -> str:
    # 간단 정규화(필요 시 unicodedata.normalize("NFC", text))
    return "\n".join(line.strip() for line in text.splitlines())

def _count_pdf_pages(path: Path) -> int:
    from pypdf import PdfReader
    return len(PdfReader(str(path)).pages)
```

2.4 API 예시

```
# app/api/parse.py
from flask import Blueprint, request, jsonify
from app.tasks.parse import parse_document_task
bp = Blueprint("parse", __name__)

@bp.post("/documents/<int:doc_id>/parse")
def parse_trigger(doc_id):
    version = (request.json or {}).get("version", 1)
    job = parse_document_task.delay(doc_id, version)
    return jsonify({"job_id": job.id}), 202
```

2.5 품질 기준

- 텍스트 PDF 5건 100% 성공, DOCX 한글깨짐 0건, XLSX 문자 셀 ≥ 90% 추출
- 파싱 실패는 `audit_log` 에 원인·파일메타 기록

---

3) DET — PII 탐지 모듈

목적: 표준화된 텍스트에서 PII 구간(span)을 **정확·재현 가능**하게 탐지하여 후속 단계(CTX/마스킹)에 입력 제공

3.1 포함 내용(체크리스트)

- 카테고리: **RRN, PHONE, EMAIL, ADDRESS, NAME, ACCOUNT, PASSPORT, DRIVER_LICENSE, EMPLOYEE_ID**
- 규칙 버전 관리, 오탐/미탐 테스트 세트, `value_hash`(HMAC-SHA256+salt) 옵션

3.2 의존 라이브러리

- `re`(표준 정규식), `hashlib`, `hmac`

3.3 최소 실행 코드

```
# app/services/detector.py
import re, hmac, hashlib, os

RULES = {
    "EMAIL": re.compile(r"(?i)[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}"),
    "PHONE": re.compile(r"\b01[016789]-?\d{3,4}-?\d{4}\b"),
    # 간단 예시. 실제로는 더 정교한 패턴 사용 권장
}

SALT = os.environ.get("PII_HASH_SALT", "ldip-demo-salt").encode()

def detect(text: str) -> list[dict]:
    findings = []
    for cat, pat in RULES.items():
        for m in pat.finditer(text):
            span = {"category": cat, "start": m.start(), "end": m.end()}
            val = text[m.start():m.end()].encode()
            span["value_hash"] = hmac.new(SALT, val, hashlib.sha256).hexdige
```

```
st()
            findings.append(span)
    return sorted(findings, key=lambda x: x["start"])
```

3.4 API 예시

```
# app/api/pii.py
from flask import Blueprint, jsonify
from app.services.detector import detect
bp = Blueprint("pii", __name__)

@bp.get("/documents/<int:doc_id>/pii")
def get_pii(doc_id):
    # 저장소에서 텍스트 로드(또는 파싱 직후 메모리 전달)
    text = _load_plaintext(doc_id)
    return jsonify({"findings": detect(text)})
```

3.5 품질 기준

- 카테고리별 Positive ≥ 3, 오탐률 < 5%, 재현성 100%
- 룰 변경 시 테스트 세트 전수 재검

---

4) CTX — 컨텍스트 필터 모듈

목적: 검토자에게 민감정보의 **주변 최소 맥락**만 제공하여 과다노출 위험을 줄임

4.1 포함 내용(체크리스트)

- 스니펫 윈도우 N(기본 20자), 문장 경계 보정, 화이트/블랙리스트 토큰 필터
- 파일 포맷별 locator 표준: PDF(page, offset), DOCX(paragraph, char), XLSX(sheet, cell)

4.2 최소 실행 코드

```
# app/services/context.py
from dataclasses import dataclass

@dataclass
```

```
class Snippet:
    start: int; end: int; text: str; category: str

def make_snippets(text: str, findings: list[dict], window: int = 20) → list[Snip
pet]:
    out = []
    n = len(text)
    for f in findings:
        s = max(0, f["start"] - window)
        e = min(n, f["end"] + window)
        t = text[s:e]
        # 민감값 자체는 ** 로 마스킹
        masked = t.replace(text[f["start"]:f["end"]], "**")
        out.append(Snippet(s, e, masked, f["category"]))
    return out
```

4.3 API 예시

```
# app/api/context.py
from flask import Blueprint, request, jsonify
from app.services.context import make_snippets
bp = Blueprint("context", __name__)

@bp.post("/documents/<int:doc_id>/context")
def context_snippets(doc_id):
    body = request.get_json(force=True)
    window = int(body.get("window", 20))
    text, findings = _load_text_and_findings(doc_id)
    snippets = make_snippets(text, findings, window)
    return jsonify({"snippets": [s.__dict__ for s in snippets]})
```

4.4 품질 기준

- 스니펫에 민감값 원문 미노출(마스킹 적용), window 준수, locator 일관성

---

5) ENC — 암호화 모듈

목적: 업로드 문서의 정본을 AES-256-GCM으로 암호화하여 저장하고, 복호는 최소 권한으로 통제

5.1 포함 내용(체크리스트)

- AWS KMS `GenerateDataKey/Decrypt` 연동, 파일단위 **유일 IV**(12B) + **AAD**
- `doc_keys` 메타 저장, 평문 DEK 메모리 즉시 파기, 감사 로깅

5.2 의존 라이브러리

- `boto3`, `cryptography`

5.3 최소 실행 코드

```
# app/services/crypto.py
import os, json
from cryptography.hazmat.primitives.ciphers.aead import AESGCM
import boto3, base64

kms = boto3.client("kms", region_name=os.getenv("AWS_REGION"))
KMS_KEY_ID = os.getenv("KMS_KEY_ID")

def encrypt_file(path_in: str, path_out: str, aad: dict) -> dict:
    # 1) 데이터키 발급
    resp = kms.generate_data_key(KeyId=KMS_KEY_ID, KeySpec="AES_25
6")
    dek_plain = resp["Plaintext"]
    dek_wrapped = resp["CiphertextBlob"]
    # 2) 파일 암호화
    aes = AESGCM(dek_plain)
    iv = os.urandom(12)
    aad_bytes = json.dumps(aad, separators=(",", ":")).encode()
    with open(path_in, "rb") as f:
        pt = f.read()
    ct = aes.encrypt(iv, pt, aad_bytes)  # ct = ciphertext||tag
    with open(path_out, "wb") as f:
        f.write(ct)
    # 3) 평문 DEK 폐기
    del dek_plain
    return {
```

```
        "kms_key_id": KMS_KEY_ID,
        "wdek_b64": base64.b64encode(dek_wrapped).decode(),
        "iv_b64": base64.b64encode(iv).decode(),
        "aad_json": aad,
        "alg": "AES-256-GCM",
    }

def decrypt_file(path_in: str, path_out: str, meta: dict):
    # 1) DEK 복원
    wdek = base64.b64decode(meta["wdek_b64"]) if isinstance(meta["wde
k_b64"], str) else meta["wdek_b64"]
    dek_plain = kms.decrypt(CiphertextBlob=wdek)["Plaintext"]
    # 2) 복호화
    aes = AESGCM(dek_plain)
    iv = base64.b64decode(meta["iv_b64"]) if isinstance(meta["iv_b64"], st
r) else meta["iv_b64"]
    aad_bytes = json.dumps(meta["aad_json"], separators=(",", ":")).encode
()
    with open(path_in, "rb") as f:
        ct = f.read()
    pt = aes.decrypt(iv, ct, aad_bytes)
    with open(path_out, "wb") as f:
        f.write(pt)
    del dek_plain
```

5.4 API 예시

```
# app/api/enc.py
from flask import Blueprint, request, send_file, jsonify
from app.services.crypto import encrypt_file, decrypt_file
from datetime import datetime
import os

bp = Blueprint("enc", __name__)

@bp.post("/documents/<int:doc_id>/encrypt")
def enc(doc_id):
    # 경로/메타 조회 생략
```

```
    meta = encrypt_file(f"/data/uploads/{doc_id}.bin", f"/data/uploads/{doc_id}.bin.enc", {
        "doc_id": doc_id,
        "ts": datetime.utcnow().isoformat()+"Z"
    })
    _save_doc_keys(doc_id, meta)
    return jsonify({"doc_id": doc_id, "version": 2, "mode": "KMS"})

@bp.post("/documents/<int:doc_id>/decrypt")
def dec(doc_id):
    meta = _load_doc_keys(doc_id)
    decrypt_file(f"/data/uploads/{doc_id}.bin.enc", f"/data/uploads/{doc_id}.bin.dec", meta)
    return send_file(f"/data/uploads/{doc_id}.bin.dec", as_attachment=True)
```

5.5 품질 기준

- 10MB 파일 암호화 ≤ 2s(p95), 복호 실패율(KMS/AAD) < 0.1%
- 비소유자 복호 요청 100% 차단(403)

---

6) 코어 파이프라인 연계(샘플)

```
# app/tasks/parse.py
from app.tasks import celery
from app.services.parser import Parser
from app.services.detector import detect
from app.services.context import make_snippets

@celery.task(name="parse_document_task", autoretry_for=(Exception,), retry_backoff=True, retry_kwargs={"max_retries":3})
def parse_document_task(doc_id: int, version: int):
    path, mime = _resolve_path_and_mime(doc_id, version)
    pr = Parser().parse(path, mime)
    findings = detect(pr.text)
    snippets = make_snippets(pr.text, findings, window=20)
    _store_findings(doc_id, version, findings)
```

```
    _store_context(doc_id, version, [s.__dict__ for s in snippets])
    return {"findings": len(findings), "snippets": len(snippets)}
```

---

7) 보안·품질·운영 기준(코어 공통)

- **보안**: 정본은 암호문 저장, 평문은 메모리 상주 최소화, AAD 사용, 최소 권한 IAM, 모든 접근 감사
- **성능**: 큐 대기 p95 ≤ 10s, 파싱 p95 ≤ 60s, 컨텍스트 생성 p95 ≤ 2s
- **신뢰성**: 재시도 3회, DLQ 보존 ≥ 7일, 실패 원인 분류율 ≥ 95%
- **가시성**: `X-Request-Id` 트레이싱, `/metrics` (Prometheus) 옵션, 감사 대시보드
- **테스트**: 단위≥60% 커버리지, 카테고리별 테스트 세트 상시 유지, E2E 스모크(업→암→파→탐→컨)

---

8) 붙임: 엔드포인트 요약(OpenAPI는 D2 부록 AE 전문 참조)

- `POST /documents/{id}/encrypt`, `POST /documents/{id}/decrypt`
- `POST /documents/{id}/parse`, `GET /parse/jobs/{job_id}`
- `GET /documents/{id}/pii`, `POST /documents/{id}/context`
- `GET /jobs/{job_id}`

> 본 문서의 코드 예시는 학습·통합 가이드용 최소 구현임. 실제 운영에서는 예외 처리, 권한 검증, 입력 스키마 검증(pydantic), 트랜잭션, 로깅을 강화함.

---

9) 제작 가이드 — 어떻게 만들고 실행하는가

목적: 본 장은 코어 모듈(Q/PAR/DET/CTX/ENC)을 **제로부터 실행** 가능한 수준으로 구축하는 절차를 제시함. Ubuntu 22.04 + Docker/Compose 기준. 로컬(개발)과 컨테이너(운영) 흐름 모두 포함함.

9.1 사전 요구사항

- OS: Ubuntu 22.04 LTS
- 필수 도구: `git`, `docker`, `docker compose`, `make`, `curl`
- Python: 3.11 (로컬 실행 시)

- AWS 자격: `ap-northeast-2` 리전, KMS 키(예: `alias/ldip-data`), 권한 `kms:GenerateDataKey`, `kms:Decrypt`

9.2 프로젝트 스캐폴드(초안)

```
ldip/
  ├─ app/
  │   ├─ api/            # Flask Blueprints
  │   │   ├─ __init__.py   # app factory에서 등록
  │   │   ├─ enc.py        # ENC API
  │   │   ├─ parse.py      # PAR API
  │   │   ├─ pii.py        # DET API
  │   │   ├─ context.py    # CTX API
  │   │   └─ jobs.py       # Q API
  │   ├─ services/
  │   │   ├─ parser.py     # PAR 서비스
  │   │   ├─ detector.py   # DET 서비스
  │   │   ├─ context.py    # CTX 서비스
  │   │   └─ crypto.py     # ENC 서비스
  │   ├─ tasks/
  │   │   ├─ __init__.py   # Celery 인스턴스
  │   │   ├─ parse.py      # parse_document_task
  │   │   ├─ encrypt.py    # encrypt_document_task
  │   │   └─ mask.py       # mask_document_task(후행)
  │   ├─ models.py         # SQLAlchemy 모델(요약)
  │   ├─ __init__.py       # Flask app factory
  │   └─ config.py         # 설정 로딩(.env 포함)
  ├─ migrations/           # Alembic 마이그레이션
  ├─ samples/              # 샘플 파일/테스트 데이터
  ├─ requirements.txt
  ├─ docker-compose.yml
  ├─ Dockerfile
  ├─ Makefile
  └─ .env.example
```

9.3 의존성 설치

requirements.txt(예시)

```
flask==3.0.0
gunicorn==21.2.0
celery==5.3.6
redis==5.0.4
sqlalchemy==2.0.32
psycopg2-binary==2.9.9
alembic==1.13.2
cryptography==42.0.8
boto3==1.34.148
pdfminer.six==20231228
pypdf==4.2.0
python-docx==1.1.2
openpyxl==3.1.3
pydantic==2.8.2
python-dotenv==1.0.1
```

로컬 설치:

```
python3.11 -m venv .venv && source .venv/bin/activate
pip install -U pip && pip install -r requirements.txt
```

9.4 Flask 앱 팩토리(요약)

```
# app/__init__.py
from flask import Flask
from .api.enc import bp as enc_bp
from .api.parse import bp as parse_bp
from .api.pii import bp as pii_bp
from .api.context import bp as ctx_bp
from .api.jobs import bp as jobs_bp

def create_app():
    app = Flask(__name__)
    app.register_blueprint(enc_bp, url_prefix="/api/v1")
    app.register_blueprint(parse_bp, url_prefix="/api/v1")
    app.register_blueprint(pii_bp, url_prefix="/api/v1")
    app.register_blueprint(ctx_bp, url_prefix="/api/v1")
```

```
    app.register_blueprint(jobs_bp, url_prefix="/api/v1")
    @app.get("/healthz")
    def healthz():
        return {"ok": True}
    return app
```

9.5 DB 모델과 마이그레이션

간단 모델(발췌)

```
# app/models.py
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column
from sqlalchemy import Integer, String, Boolean, DateTime, JSON
from datetime import datetime

class Base(DeclarativeBase):
    pass

class Document(Base):
    __tablename__ = "documents"
    id: Mapped[int] = mapped_column(Integer, primary_key=True)
    owner_id: Mapped[int] = mapped_column(Integer)
    filename: Mapped[str] = mapped_column(String(255))
    mime: Mapped[str] = mapped_column(String(128))
    size: Mapped[int] = mapped_column(Integer)
    status: Mapped[str] = mapped_column(String(32), default="NEW")
    created_at: Mapped[datetime] = mapped_column(DateTime, default=dat
etime.utcnow)
```

Alembic 초기화/생성:

```
alembic init migrations
# env.py에서 DATABASE_URL 바인딩, target_metadata=Base.metadata 설정
alembic revision -m "init" --autogenerate
alembic upgrade head
```

9.6 Dockerfile / docker-compose

Dockerfile(예시)

```
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -U pip && pip install -r requirements.txt
COPY app ./app
ENV PYTHONUNBUFFERED=1
EXPOSE 8080
CMD ["gunicorn","-w","4","-b","0.0.0.0:8080","app:create_app()"]
```

docker-compose.yml(핵심 발췌)

```
services:
  api:
    build: .
    environment:
      - DATABASE_URL=${DATABASE_URL}
      - REDIS_URL=${REDIS_URL}
      - UPLOAD_ROOT=/data/uploads
      - AWS_REGION=${AWS_REGION}
      - KMS_KEY_ID=${KMS_KEY_ID}
    volumes:
      - ./data/uploads:/data/uploads
    ports: ["8080:8080"]
    depends_on: [db, redis]
  worker:
    build: .
    command: ["celery","-A","app.tasks.celery","worker","--loglevel","INF
O","-Q","enc,parse,mask"]
    environment: ["DATABASE_URL=${DATABASE_URL}","REDIS_URL=${RED
IS_URL}","AWS_REGION=${AWS_REGION}","KMS_KEY_ID=${KMS_KEY_I
D}"]
    depends_on: [api, redis]
  db:
    image: postgres:16
    environment: ["POSTGRES_USER=app","POSTGRES_PASSWORD=app_p
w","POSTGRES_DB=appdb"]
    ports: ["5433:5432"]
  redis:
```

```
    image: redis:7-alpine
    ports: ["6379:6379"]
```

9.7 Makefile(개발 편의)

```
install:
    python -m venv .venv && . .venv/bin/activate && pip install -U pip && pip i
nstall -r requirements.txt
run:
    gunicorn -w 4 -b 0.0.0.0:8080 'app:create_app()'
worker:
    celery -A app.tasks.celery worker --loglevel INFO -Q enc,parse,mask
migrate:
    alembic upgrade head
dbgen:
    alembic revision -m "auto" --autogenerate
test:
    pytest -q --maxfail=1 --disable-warnings
fmt:
    ruff check . && black .
```

9.8 모듈별 제작 순서(Recipes)

Q(작업큐)

1. `app/tasks/__init__.py` 에 Celery 인스턴스/큐 라우팅 정의
2. 각 태스크 파일 생성(parse/encrypt/mask) → `@celery.task` 데코레이터 적용
3. `api/jobs.py` 에 상태조회 엔드포인트 생성(AsyncResult)

PAR(파싱)

1. `services/parser.py` 클래스 작성(파일타입 분기, 정규화)
2. `api/parse.py` 에 트리거 엔드포인트 작성 → Celery 태스크 큐잉
3. 샘플 파일로 유닛테스트 작성(pdf/docx/xlsx/txt)

DET(탐지)

1. `services/detector.py` 에 카테고리별 정규식/룰 버전 정의
2. `api/pii.py` 에 결과 반환 엔드포인트 작성

3. 합성 테스트 데이터(JSONL)로 오탐/미탐 측정

CTX(컨텍스트)

1. `services/context.py` 에 스니펫 생성 로직 작성(민감값 `*` 치환)
2. `api/context.py` 에 윈도우 파라미터/locator 설계 반영
3. window 경계/다중 겹침 케이스 테스트

ENC(암호화)

1. `services/crypto.py` 에 `encrypt_file/decrypt_file` 구현(AESGCM+KMS)
2. `api/enc.py` 에 암·복호 엔드포인트, AAD 설계 반영
3. 10MB 파일 기준 성능/권한 테스트, 감사로그 점검

9.9 로컬 실행 절차(퀵 스타트)

```
cp .env.example .env  # 환경값 설정
docker compose up --build -d
# DB 마이그레이션
docker compose exec api alembic upgrade head
# 샘플 업로드
curl -F file=@samples/sample.pdf http://localhost:8080/api/v1/documents/
upload
# 파싱 트리거
curl -X POST http://localhost:8080/api/v1/documents/1/parse -H 'Content-
Type: application/json' -d '{"version":1}'
# PII 조회
curl http://localhost:8080/api/v1/documents/1/pii
# 스니펫 생성
curl -X POST http://localhost:8080/api/v1/documents/1/context -H 'Content
-Type: application/json' -d '{"version":1,"window":20}'
# 암호화
curl -X POST http://localhost:8080/api/v1/documents/1/encrypt
```

9.10 품질 게이트(출고 전 체크)

- **보안**: 평문 파일 잔존 X, AAD 검증, IAM 최소권한, 비밀은 .env/Secret Manager 관리
- **성능**: p95 업로드→암호화 ≤ 2s(10MB), 파싱 ≤ 60s(텍스트 PDF)

- **신뢰성**: 재시도/백오프, DLQ 모니터링, 장애 복구 리허설(Decrypt 테스트 포함)
- **가시성**: `X-Request-Id` 전파, 에러 코드 표준, 감사 로그 샘플 검증
- **테스트**: 단위≥60% 커버리지, E2E 스모크(업→암→파→탐→컨) 통과

9.11 근거·기준 요약

- 암호화: AES-256-GCM(무결성 태그), 데이터키는 KMS로 발급/복호, IV 재사용 금지, AAD 적용
- 파싱: 텍스트형 우선, 변환/OCR은 별도 컨테이너(D2 참조)
- 탐지: 카테고리/룰 버전 관리, HMAC-SHA256 해시(옵션)로 교차문서 링크
- 컨텍스트: 최소 필요 스니펫, locator로 선택 마스킹/프리뷰 연결
- 큐: Celery/Redis 표준, 지수 백오프·DLQ로 안정성 확보