Mini Project: AI-Powered File Processing Microservice System

Goal

Build a **three-microservice system** where users upload documents, the system stores them, extracts text, and sends that text to another AI microservice (Gemini API) for summarization/analysis. The system is intentionally simple but covers all foundational SDE + AI + microservice concepts.

System Overview

Microservice 1: File Service (FastAPI + DB + File Handling)

Handles uploading, storing files, and retrieving metadata.

Responsibilities:

- Upload file (PDF, text, docs)
- Generate a unique File ID
- Store file in a storage directory
- Save metadata (filename, file_id, path, upload_time) in DB
  - SQLite or MongoDB
- Endpoint to retrieve file metadata
- Endpoint to download the file
- Endpoint to trigger "Process File" → communicates with Microservice 2

Endpoints:

- `POST /upload` → returns `{file_id}`
- `GET /file/{file_id}` → metadata
- `GET /download/{file_id}` → return file content
- `POST /process/{file_id}` → call Text Extractor service

Microservice 2: Text Extractor Service (FastAPI + PDF parser)

Extracts raw text and optionally stores embeddings.

Responsibilities:

- Given a `file_id`, fetch file from Microservice 1

- Read PDF/text using `pypdf`, `python-docx`, etc.
- Clean and segment text

Endpoints:

- `POST /extract-text` (body: `{ file_id }`) → returns `{ text, chunks }`

Microservice 3: AI Processing Service (FastAPI + Gemini API)

Receives extracted text and performs AI operations.

Responsibilities:

- Call **Gemini API** with extracted text
- Provide:
  - summary
  - insights
  - topic extraction
  - sentiment
- Return structured JSON

Endpoints:

- `POST /summarize`
- `POST /analyze`

Payload:

```
{
  "file_id": "123",
  "text": "Extracted text here..."
}
```

System Flow

1. User uploads a PDF → **File Service**
2. File Service stores file + metadata in DB → returns File ID
3. User (or UI) calls "Process File" → File Service → Text Extractor
4. Text Extractor extracts text → returns text
5. File Service calls AI Service → sends text
6. AI Service returns summary/analysis
7. File Service stores final output in DB

8. User fetches results via FastAPI endpoint

Technologies Covered

Backend Fundamentals

FastAPI CRUD Path params Pydantic Background tasks Uvicorn Modular project structure

Databases

SQLite or MongoDB ORM (SQLAlchemy / Beanie)

AI / Agentic

Gemini API Prompt engineering RAG basics (if embedding storage is used) MCP architecture (microservice separation)

Dev Tools

Postman for API testing Streamlit optional UI Git/GitHub version control

---

Stretch Features (Optional for extra practice)

1. Add Authentication

- JWT login + role-based access

2. Add a Streamlit UI

- Upload file
- View results
- Display visual summary

3. Add Pub/Sub (simple version)

- Use Redis as a background queue for long PDF processing

4. Add a chatbot over document

- Chat endpoint calling Gemini API with context
- Use embeddings from Vector DB

5. Add retry mechanism

- If Gemini API fails
- If file extraction fails

Deliverables

1. **3 separate FastAPI apps**
2. README with system flow
3. Postman collection
4. Folder structure + environment variables
5. Optional Streamlit UI