Take-Home Coding Challenge: Legal Intel Dashboard

Overview

You're building the foundation for a legal AI platform that allows users to upload folders of legal documents, ask questions across the dataset, and visualize high-level document trends. This challenge is designed to test full-stack capabilities, thoughtful architecture, and adherence to Python and React best practices.

Scope

- Code quality determines passing to the final stage of the interview process
- Please deliver results back within 72 hours upon receipt of the Case Study
- Submission format: GitHub repository with code + short write-up

Objectives

1. Allow users to upload a folder of legal documents (PDF or DOCX)
2. Extract and store metadata and text
3. Enable natural language querying across documents
4. Return structured comparisons (e.g. jurisdiction, agreement type)
5. Display a dashboard of metadata insights (e.g. agreement types, jurisdictions, industries)

Backend Requirements (Python)

1. Document Ingestion

- Endpoint to upload multiple documents at once
- Extract raw text from DOCX/PDF files
- Store text and metadata in SQLite or in-memory storage
- You can assume documents will be small in your code. However, including an explanation on how you would handle bigger docs would be a plus

2. Mass Interrogation API

- Endpoint: POST /query
- Accepts a natural language question as input
- Returns a structured JSON response with rows and columns based on the user's query
    - Example input: "Which agreements are governed by UAE law?"
    - Example output:

```
[
  {"document": "nda_abudhabi.pdf", "governing_law": "UAE"},
  {"document": "supplier_contract_dubai.docx", "governing_law": "UAE"}
```

]

- Simulate LLM behavior using keyword extraction, basic NLP, or mocked responses. Optional: integrate OpenAI or Llama APIs if you have access.

3. Metadata Extraction

Extract the following (either programmatically or via mock):

- **Agreement type** (e.g., NDA, MSA, Franchise Agreement)
- **Governing law / jurisdiction** (e.g., UAE, UK, Delaware)
- **Geography** mentioned (e.g., Middle East, Europe)
- **Industry sector** (e.g., oil & gas, healthcare, technology)

4. Dashboard Data API

- Endpoint: GET /dashboard
- Returns counts and aggregations of metadata fields such as:
- {

```
  "agreement_types": {"NDA": 4, "MSA": 3},
  "jurisdictions": {"UAE": 5, "UK": 2},
  "industries": {"Technology": 3, "Oil & Gas": 4}
}
```

5. Expectations

- Maintain clear project structure
- Maintainable using modern python standards
- Include minimal unit tests
- Log processing steps and errors

Frontend Requirements (React with TypeScript preferred)

1. Upload Page

- Upload multiple legal documents via drag-and-drop interface. Show upload progress and confirmation

2. Mass Interrogation View

- Text input for user questions
- Results table dynamically generated from API response
    - Example: A table comparing jurisdictions or listing all contracts mentioning a specific clause

3. Dashboard View

- Visualize insights from /dashboard API:
    - Bar chart: number of agreements by type
    - Pie chart: governing law breakdown
    - Table: industry and geographic coverage
- Use any charting library

4. Expectations

- Use React functional components and hooks
- Clean file and component organization
- Proper state management
- Typed API interfaces with TypeScript
- User-friendly UI

Deliverables

- GitHub repo with:
    - Full source code (frontend and backend)
    - README.md with:
        - Setup instructions (how to run backend and frontend)
        - Short write-up explaining:
            - Backend architecture and metadata extraction approach
            - Frontend component structure and dashboard rendering
            - How they would scale this to production (e.g., real LLM integration, document indexing)

Evaluation Criteria

Category	**What We're Looking For**
Architecture	Thoughtful separation of concerns and modular structure
Code Quality	Clean, readable, typed code with sensible naming conventions
Full Stack Integration	Working backend + frontend interaction
LLM Handling	Mocked reasoning with extendability for real LLM integration
Dashboard & Query UX	Intuitive upload/query flow with useful visual feedback

Category	What We're Looking For
Documentation	Clarity of setup and rationale in the README

Optional Enhancements (Not Required)

- Search-as-you-type functionality on the question input
- Frontend pagination or filters on results
- Include mock login/auth (JWT-based)
- Export dashboard results as CSV or PDF