JAY SHAHAPURAKAR www.linkedin.com/in/jay-shahapurakar-622661260

System Design for AI-Powered Legal Document Processing

The system will allow lawyers to **upload legal documents**, process them using an **AI model**, and display **extracted insights** in a user-friendly interface.

1. Architecture Overview

- **Frontend:** Web-based UI for document upload & insights visualization
- **Backend:** API to handle document processing & AI model inference
- **AI Model:** NLP-based clause extraction model (Legal-BERT / custom trained)
- **Database:** Stores uploaded documents & extracted insights
- **Cloud Storage:** Stores large document files
- **Security:** Authentication, access control, and encrypted storage

2. Technologies & APIs

Component	**Technology / API**
Frontend	Flask with Jinja (for UI), Bootstrap / Tailwind CSS (for styling)
Backend API	Flask (Python-based web framework)
AI Model	Hugging Face Transformers (Legal-BERT / Custom NLP)
Storage	AWS S3 (for document storage)
Database	PostgreSQL (relational data), MongoDB (if storing raw texts)
Authentication	OAuth (Google Auth / Auth0)
Deployment	Docker + Kubernetes (for scalability)
Cloud / Hosting	AWS, GCP, or Azure

3. System Workflow

Step 1: Lawyer Uploads Document

Frontend (Flask with Jinja Templates)

- Lawyer logs in
- Uploads PDF/DOCX file
- Clicks **"Process"** button

Backend API (Flask)

- Saves the document to **AWS S3**

- Stores metadata (like document name, upload time) in **PostgreSQL**
- Triggers **AI processing** once the document is uploaded

Step 2: AI Model Extracts Insights

AI Processing (Legal NLP Model)

- Converts the uploaded document to text using **pdfplumber** (for PDFs) or **python-docx** (for DOCX files)
- Preprocesses the text (tokenization, lemmatization, etc.)
- Runs the **Legal-BERT** / **Custom NLP Model** to extract clauses like "Termination", "Indemnification", etc.
- Stores extracted insights in **MongoDB** or **PostgreSQL** (for relational data)

Step 3: Display Insights in UI

Frontend (Flask with Jinja Templates)

- Fetches extracted insights from the backend API
- Displays key clauses in an interactive UI
- Allows the lawyer to download a structured report (PDF or CSV)

4. API Design

-Sample_Python-Code:

```
@app.route("/upload", methods=["POST"])
def upload_file():
    """Handles document upload & triggers processing."""
    file = request.files['file']  # Fetch file from form submission
    file_path = save_to_s3(file)  # Save file to AWS S3
    process_document(file_path)  # Trigger AI model processing
    return jsonify({"message": "File uploaded successfully"})
```

-Get Extracted Insights API:

```
@app.route("/insights/<doc_id>", methods=["GET"])
def get_insights(doc_id):
    """Fetches extracted legal clauses from DB."""
    insights = fetch_from_db(doc_id)  # Query database for extracted insights
    return jsonify({"document_id": doc_id, "insights": insights})
```

5. Security Considerations

-User Authentication → **OAuth-based login** (Google Auth / Auth0)

-Role-Based Access Control → Only authorized users (e.g., admins, lawyers) can upload and process documents

-Data Encryption → Documents securely stored in **AWS S3** with encryption (for both storage and transit)

-Logging & Monitoring → Use **Prometheus** and **Grafana** to monitor API usage, track errors, and scale effectively

6. Scalability & Deployment

-Containerization → **Dockerize** the Flask API & AI model for easier deployment and isolation of services

-Orchestration → Use **Kubernetes** to manage scaling and load balancing of the Flask app and AI services

-CDN Caching → Optimize API responses and speed up document downloads using **Redis** for caching

Final Thoughts

This design integrates AI-driven legal document processing with a **Flask-based frontend** and backend. It's designed to be **secure, scalable, and efficient**, leveraging modern cloud technologies like **AWS**, **Docker**, **Kubernetes**, and **OAuth** for authentication.