Interns Assigned:

J. Indu

AI-Powered Contract Review and Clause Extraction Platform

- **Objecti\ve:** To build a secure web application that assists lawyers and paralegals by automatically analyzing uploaded legal contracts, extracting key data points, and identifying critical clauses. This reduces manual review time and minimizes the risk of overlooking important terms.
- **Problem Statement:** Legal professionals spend countless hours manually reading through lengthy contracts to find specific information like effective dates, party names, liability limits, termination conditions, and governing law. This process is tedious, expensive, and prone to human error. You are tasked with building a full-stack application that automates this process. A user should be able to upload a contract (e.g., a PDF or DOCX file). The system's backend will then use an AI/NLP model to parse the document, identify and extract predefined legal clauses and data points, and store them. The frontend will present this extracted information in a clean, structured, and easily digestible dashboard, allowing the user to quickly grasp the contract's key terms.
- **Key Learning Areas & Tech Stack:**
    - **Skills:** NLP Model Integration, Secure File Upload/Storage, Document Parsing (PDF/DOCX), API Design (REST), Backend Job Queues (for processing large documents), Database Schema Design, Frontend State Management, Role-Based Access Control.
    - **Tech Stack:**
        - **Frontend: React** or **Vue.js**, a PDF rendering library (like react-pdf), a UI component library (like Material-UI or Ant Design), and a state management tool.
        - **Backend: Python** (with **Django** or **Flask**) is highly recommended due to its mature NLP ecosystem.Key libraries would include PyMuPDF/python-docx for parsing, and spaCy or **Hugging Face Transformers** for the NLP tasks. A task queue like **Celery** with **Redis** would be needed for asynchronous processing.
        - **Database: PostgreSQL** is an excellent choice for its robust support for structured data (users, documents) and semi-

structured data (using JSONB fields to store the extracted clause information).
        - **Deployment: Docker**, **NGINX**, and a cloud provider like **AWS** (using S3 for secure file storage) or **Google Cloud**.
- **Phase-wise Plan:**
    - **Phase 1 (R&D and Backend Foundation):** Research NLP techniques for **Named Entity Recognition (NER)** and **Text Classification** tailored to legal text. The core R&D is determining how to accurately identify legal-specific entities that pre-trained models don't know (e.g., "Indemnification Clause"). Set up the Python backend with secure user authentication and API endpoints for file upload. Implement the file parsing logic to convert various document formats into clean text.
    - **Phase 2 (The AI Extraction Engine):** This is the heart of the project. Build the NLP service.
        - **Approach A (Simpler):** Use **spaCy's** rule-based EntityRuler to define patterns for key data points (e.g., finding dates near the word "Effective Date," or recognizing company names after "between").
        - **Approach B (More Advanced):** Fine-tune a pre-trained **transformer model** (like one from the BERT family) on a small, custom-labeled dataset of legal sentences to classify which sentences belong to which clause type (e.g., Termination, Liability).
        - Integrate this NLP module into the backend, triggered asynchronously by a task queue (Celery) whenever a new document is uploaded.
    - **Phase 3 (Frontend Dashboard and Integration):** Develop the React frontend. Create a secure portal where a user can log in, upload a contract, and view a list of their documents. Build the main "Analysis Dashboard" which, for a selected document, displays the extracted key-value pairs (e.g., Governing Law: "State of Delaware") on one side and the full text of the extracted clauses (e.g., the entire Limitation of Liability section) on the other.
    - **Phase 4 (Advanced Features and Deployment):** Enhance the user experience. Implement a feature that highlights the extracted clauses directly within a rendered view of the original PDF document in the browser. Add user roles (e.g., an "Admin" who can manage users vs. a "Lawyer" who can only upload). Containerize the entire application (frontend, backend, database, queue) using Docker and write deployment scripts.
- **Expected Deliverable:**

- A link to a live, deployed web application.
- A link to the GitHub repository containing all source code.
- A detailed README.md file explaining the system architecture, the chosen NLP approach, and instructions on how to run the project locally.