Legal Document Management Interface

Project Overview

Develop a responsive Legal Document Management Interface using ReactJS that allows users to upload, view, and manage legal documents (PDFs). The application should dynamically adjust its layout based on screen size and provide intuitive modals for uploading and viewing documents with mock data extractions. Optionally, implement a NodeJS backend to handle file uploads and return mock extraction data.

Technical Stack

- **Frontend:**
  - ReactJS
  - CSS Flexbox for responsive design
  - React Modal or similar library for modals
  - PDF.js or react-pdf for PDF previews
- **Backend (Optional):**
  - NodeJS with Express

UI/UX Design

General Layout

- **Page 1: Document Dashboard**
  - **Header:** Icon in the top-left corner.
  - **Content Area:** Nine boxes labeled "Legal Document 1" through "Legal Document 9".
  - **Responsive Design:** Use Flexbox to ensure dynamic positioning based on screen size.
  - Display of upload date and file name upon successful upload of a document.
- **Page 2: Upload Modal**
  - Popup modal triggered by clicking on a document box.
  - File upload interface accepting only PDF files.
- **Page 3: Document Details Modal**
  - Split-screen modal:
    - **Left:** PDF preview.
    - **Right:** Mock extractions with page numbers and "Go To Page" buttons.
  - Header displaying the name of the selected document.
  - Close (X) button at the top-right corner.

Functional Specifications

Page 1: Document Dashboard

- **Icon:**
    - Place an icon in the top-left corner (e.g., a folder or document icon).
- **Document Boxes:**
    - Create nine boxes labeled from "Legal Document 1" to "Legal Document 9".
    - Each box should be a flex item, adjusting based on screen size.
    - **On Click:**
        - **If no document uploaded:** Open Upload Modal (Page 2).
        - **If document exists:** Open Document Details Modal (Page 3).
- **Displaying Uploaded Information:**
    - After a successful upload, each box should display:
        - File Name
        - Upload Date

Page 2: Upload Modal

- **Trigger:** Clicking on a document box without an uploaded file.
- **Features:**
    - File input accepting only PDF files.
    - Submit button to upload the file.
    - Validation to ensure only PDFs are uploaded.
    - Upon successful upload:
        - Capture and store the file name.
        - Capture and store the upload date.
        - Display the above information on the respective document box.
        - Close the modal.
- **Error Handling:**
    - Display error messages for invalid file types or upload failures.

Page 3: Document Details Modal

- **Trigger:** Clicking on a document box with an uploaded file.
- **Layout:**
    - **Header:** Title displaying the selected document's name and an X button to close the modal.
    - **Left Panel:** PDF preview of the uploaded document.
    - **Right Panel:**
        - List of mock extractions (e.g., Extraction 1, Extraction 2).
        - Each extraction displays the page number it appears on.
        - "Go To Page" button next to each extraction to navigate to the respective page in the PDF preview.

- **Features:**
  - PDF preview should be scrollable and support page navigation.
  - "Go To Page" buttons should smoothly scroll the PDF preview to the specified page.
- **Mock Extractions:**
  - Generate random page numbers for each extraction using a randomizer.
  - Example:
    - Extraction 1 - Page 3
    - Extraction 2 - Page 7

Backend Specifications (Optional)

Note: This section is optional and intended for those who wish to extend the project with a backend component.

- **Server Setup:**
  - Use NodeJS with Express framework.
  - Implement CORS to allow frontend communication.
- **Endpoints:**
  - `POST /upload`: Handle PDF file uploads.
    - Accept only PDF files.
    - Store files temporarily (no persistent storage required).
  - `GET /extractions/:documentId`: Return mock extraction data for a given document.
    - Generate and return a list of extractions with random page numbers.
- **Data Handling:**
  - No need for a database; use in-memory storage or local storage on the frontend.
- **Integration:**
  - Frontend should send uploaded files to the backend.
  - Fetch mock extraction data from the backend to display in the Document Details Modal.

Additional Notes

- **Responsive Design:**
  - Utilize CSS Flexbox to ensure that the document boxes rearrange gracefully on different screen sizes.
  - Test the application on various devices to ensure usability.
- **State Management:**
  - Use React's `useState` and `useEffect` hooks for managing component states and side effects.
  - Consider using Context API or Redux if the state becomes complex.
- **File Handling:**

    - Since there's no persistent storage, uploaded files and their metadata will only persist during the session.
    - Utilize Local Storage to retain data across page reloads if necessary.
- **PDF Handling:**
    - Use libraries like `react-pdf` for rendering PDF previews within the application.
    - Ensure that the "Go To Page" functionality interacts correctly with the PDF viewer to navigate to the specified page.
- **Error Handling & Validation:**
    - Implement robust error handling for file uploads and API interactions.
    - Provide user-friendly error messages and feedback.
- **Code Quality:**
    - Maintain clean and readable code with proper commenting.
    - Follow best practices for React and NodeJS development.
- **Testing:**
    - Perform manual testing to ensure all functionalities work as expected.
    - Optionally, write unit tests for critical components and functionalities.