# Front-End Engineering Interview Challenge: Intelligent Document Viewer & Signing Flow

## Overview

The goal of this challenge is to assess the candidate's ability to build a scalable, user-friendly, and visually appealing front-end application using React and TypeScript. The project should demonstrate their understanding of modern front-end development practices, state management, mock data handling, and responsiveness. Additionally, we want to see their creativity in designing an intuitive user experience for a complex AI-powered platform.
This project will test your ability to work with React (TypeScript), UI/UX design, API integration, state management, and performance optimization. The backend team handles all data storage and AI processing, so your focus is on building a robust front-end experience with mock API interactions.

## Core Features

Develop a **React-based** Document Management Dashboard using **mock data** that simulates a platform that enables users to:

1. File Upload and List View:
    a. A user can upload a file (PDF or DOCX) via a file upload component.
    b. Uploaded files are displayed in a list view with the following details:
        i. File Name
        ii. Upload Date
        iii. Status (e.g., "Pending Processing," "Processed," "Completed")
        iv. Action buttons (e.g., "View Details," "Mark Complete," "Delete").
2. File Details and Interaction:
    a. When a user clicks "View Details" for a file, they should see a modal or a new page displaying:
        i. A summary of key information (mock data).
        ii. A Q&A section where users can ask questions related to the document's content (mock responses).
3. Signature Flow:
    a. When a user clicks "Sign Document," they should be taken through a signature flow:
        i. A canvas where they can draw their signature or type it.
        ii. A confirmation modal to confirm the signature.
        iii. Once signed, the document status should update to "Signed."
    b. The signature does not need to be saved permanently but should be displayed as part of the document details after signing.
4. User Profile Section:
    a. A dedicated section where users can:
        i. View and Manage Signatures: Display a list of saved signatures (mock data) with options to add, edit, or delete signatures.
        ii. View Pending Signature Requests: Display a list of documents awaiting the user's signature (mock data).
        iii. View Signed Documents: Display a list of all previously signed documents with details like document name, signing date, and status.
5. Responsive Design:
    a. Ensure an excellent UI/UX experience with smooth interactions and modern design principles.
    b. The application should be fully responsive and work seamlessly on desktop, tablet, and mobile devices.

Your primary focus should be **front-end development**, ensuring API interactions are seamless while optimizing for performance and usability.

## Requirements

1. File Upload & Parsing

- Implement a file uploader that supports PDF and text formats.
- Extract and display formatted text from the uploaded document.
- Include basic file validation (e.g., format checks, file size limit).
- Add a progress indicator while processing files.

2. Document Viewer

- Display the document's text in a structured, readable format (e.g., paragraphs, headings, bold text).
- Implement a scrollable interface for better navigation.
- Support text selection and copying.
- Ensure the viewer is responsive across different devices.

3. API Integration (Mocked)

- Implement a request system to send document data to an API.
- Display AI-generated insights received from the backend.
- Ensure smooth API interactions with loading states and error handling.

4. Interactive Q&A Section

- Users should be able to ask questions about the document in a dedicated input field.
- Responses should be displayed in a conversational format.
- Ensure a clear distinction between user questions and AI-generated responses.

5. Simulated Signing Flow

- Allow users to place a signature on a document (e.g., drawing their signature or typing it in a signature field).
- Implement drag-and-drop functionality to position the signature within the document.
- The signature does not need to be saved but should persist within the current session.
- Provide a clear "Sign Document" button that triggers a mock API call and confirms completion.

6. User Profile & Signature Management

- Create a User Profile page where users can:
    - Manage their signatures (upload or create a new one).
    - View pending signature requests (mock list).
    - See previously signed documents (mock list).
- Ensure a clean and intuitive dashboard layout with tabs or sections.

7. UI/UX Enhancements

- Implement a dark mode toggle.
- Add micro-interactions and animations to improve the user experience.
- Ensure mobile responsiveness with a flexible, adaptable layout.
- Provide a floating action button (FAB) for quick access to key features.

8. Mock Data

Since no API endpoints are provided, the candidate should create mock data to simulate the following:

- A list of files with details like id, name, uploadDate, and status.
- Mock summaries for files, including key information and Q&A responses.
- Mock audio files for voice playback (can be placeholder audio files).
- Mock signature data to simulate the signing process.
- Mock user profile data, including saved signatures, pending signature requests, and signed documents.

Example of mock data structure:

```javascript
const mockFiles = [
  {
    id: 1,
    name: "Project Plan",
    uploadDate: "2023-10-01",
    status: "Pending Processing",
    summary: "This document outlines the project timeline...",
    qa: [
      { question: "What is the deadline?", answer: "December 2023" },
      { question: "Who is the project lead?", answer: "John Doe" },
    ],
    audioUrl: "/audio/project_plan_explanation.mp3",
  },
  {
    id: 2,
    name: "Budget Report",
    uploadDate: "2023-10-02",
    status: "Processed",
    summary: "This document details the budget allocation...",
    qa: [
      { question: "What is the total budget?", answer: "$100,000" },
      { question: "Is there a contingency?", answer: "Yes, 10%" },
    ],
    audioUrl: "/audio/budget_report_explanation.mp3",
  },
];

const mockUserProfile = {
  signatures: [
    { id: 1, name: "Signature 1", imageUrl: "/signatures/signature1.png" },
    { id: 2, name: "Signature 2", imageUrl: "/signatures/signature2.png" },
  ],
  pendingRequests: [
    { id: 1, documentName: "Contract A", requestedDate: "2023-10-03" },
    { id: 2, documentName: "Agreement B", requestedDate: "2023-10-04" },
  ],
  signedDocuments: [
    { id: 1, documentName: "NDA", signingDate: "2023-09-15" },
    { id: 2, documentName: "Employment Contract", signingDate: "2023-09-20" },
  ],
};
```

## Technical Requirements

- React with TypeScript: The project must be built using React and TypeScript.
- State Management: Use a state management library like Redux, Recoil, or Context API to manage application state (e.g., file list, selected file, signature status, user profile data).
- Mock Data: Create and manage mock data locally to simulate file uploads, summaries, Q&A interactions, signature workflows, and user profile data.
- Styling: Use a modern CSS framework or library like TailwindCSS, Material-UI, or Styled Components.
- Accessibility: Ensure the application is accessible (e.g., ARIA labels, keyboard navigation).
- Error Handling: Handle errors gracefully and display user-friendly error messages (e.g., if a file upload fails).
- Code Quality: Write clean, modular, and well-documented code. Include unit tests for critical components.

## Creativity and Bonus Features

We encourage candidates to go beyond the core requirements and showcase their creativity. Here are some ideas (optional):

- Dark Mode: Implement a toggle for light/dark mode.
- Language Support: Add a dropdown to switch the file summary and Q&A between different languages.
- Voice Interaction: Implement a voice-based search or navigation feature.
- Animations: Add subtle animations to enhance the user experience (e.g., loading spinners, hover effects).
- Drag-and-Drop Upload: Allow users to drag and drop files for upload.
- Pagination or Search: Add pagination or a search bar to the file list view.

## Deliverables

The candidate should provide the following:

- Source Code: A GitHub repository with the complete project.
- Live Demo: The candidate should be ready to present their work on a video meeting
- README File: A detailed README explaining:
  - How to set up and run the project locally.
  - Key design and technical decisions.
  - Any additional features or creative elements implemented.
- Notes on additional improvements or optimizations made.

## Evaluation Criteria

✅ **Code Quality** – Well-structured, maintainable, and modular.
✅ **UI/UX Design** – Clean, intuitive, and visually appealing.
✅ **Performance** – Smooth interactions and optimized API handling.
✅ **API Integration** – Efficient request handling, error states, and UI feedback.
✅ **Creativity** – Thoughtful design choices, animations, or extra features.

## Final Thoughts

This challenge simulates a **real-world document signing and interaction tool**. While backend processing is mocked, your implementation should demonstrate **strong front-end skills** with attention to detail, performance, and user experience.

We look forward to seeing your work! 🚀