```
"use client"

import { useState, useCallback, useMemo } from "react"
import { useDropzone } from "react-dropzone"

// Dynamic import for pdfjs to avoid SSR issues
let pdfjs: any = null

// Initialize PDF.js dynamically
const initializePdfjs = async () => {
  if (!pdfjs) {
    const pdfjsLib = await import("pdfjs-dist")
    pdfjs = pdfjsLib.default || pdfjsLib

    // Set worker path using CDN
    pdfjs.GlobalWorkerOptions.workerSrc =
`https://cdnjs.cloudflare.com/ajax/libs/pdf.js/3.11.174/pdf.worker.min.js`
  }
  return pdfjs
}

// Icon Components
const UploadCloudIcon = ({ className }: { className?: string }) => (
  <svg
    className={className}
    xmlns="http://www.w3.org/2000/svg"
    width="24"
    height="24"
    viewBox="0 0 24 24"
    fill="none"
    stroke="currentColor"
    strokeWidth="2"
    strokeLinecap="round"
    strokeLinejoin="round"
  >
    <path d="M4 14.899A7 7 0 1 1 15.71 8h1.79a4.5 4.5 0 0 1 2.5 8.242" />
    <path d="M12 12v9" />
    <path d="m16 16-4-4-4 4" />
  </svg>
)

const FileIcon = ({ className }: { className?: string }) => (
  <svg
```

```
    className={className}
    xmlns="http://www.w3.org/2000/svg"
    width="24"
    height="24"
    viewBox="0 0 24 24"
    fill="none"
    stroke="currentColor"
    strokeWidth="2"
    strokeLinecap="round"
    strokeLinejoin="round"
  >
    <path d="M14.5 2H6a2 2 0 0 0-2 2v16a2 2 0 0 0 2 2h12a2 2 0 0 0 2-2V7.5L14.5 2z" />
    <polyline points="14 2 14 8 20 8" />
  </svg>
)

const XIcon = ({ className, ...props }: { className?: string; [key: string]: any }) => (
  <svg
    className={className}
    xmlns="http://www.w3.org/2000/svg"
    width="24"
    height="24"
    viewBox="0 0 24 24"
    fill="none"
    stroke="currentColor"
    strokeWidth="2"
    strokeLinecap="round"
    strokeLinejoin="round"
    {...props}
  >
    <line x1="18" y1="6" x2="6" y2="18" />
    <line x1="6" y1="6" x2="18" y2="18" />
  </svg>
)

const BotIcon = ({ className }: { className?: string }) => (
  <svg
    className={className}
    xmlns="http://www.w3.org/2000/svg"
    width="24"
    height="24"
    viewBox="0 0 24 24"
    fill="none"
    stroke="currentColor"
```

```
    strokeWidth="2"
    strokeLinecap="round"
    strokeLinejoin="round"
  >
    <path d="M12 8V4H8" />
    <rect width="16" height="12" x="4" y="8" rx="2" />
    <path d="M2 14h2" />
    <path d="M20 14h2" />
    <path d="M15 13v2" />
    <path d="M9 13v2" />
  </svg>
)

const InfoIcon = ({ className }: { className?: string }) => (
  <svg
    className={className}
    xmlns="http://www.w3.org/2000/svg"
    width="24"
    height="24"
    viewBox="0 0 24 24"
    fill="none"
    stroke="currentColor"
    strokeWidth="2"
    strokeLinecap="round"
    strokeLinejoin="round"
  >
    <circle cx="12" cy="12" r="10" />
    <path d="M12 16v-4" />
    <path d="M12 8h.01" />
  </svg>
)

// PDF Processing Logic
const extractTextFromPdf = async (file: File): Promise<string> => {
  return new Promise(async (resolve, reject) => {
    try {
      // Initialize PDF.js dynamically
      const pdfjsLib = await initializePdfjs()

      const reader = new FileReader()
      reader.onload = async (event) => {
        try {
          const pdf = await pdfjsLib.getDocument({ data: event.target?.result as ArrayBuffer
}).promise
```

```
        let allText = ""

        for (let i = 1; i <= pdf.numPages; i++) {
          const page = await pdf.getPage(i)
          const textContent = await page.getTextContent()
          const pageText = textContent.items.map((item: any) => item.str).join(" ")
          allText += `[Page ${i}]\n${pageText}\n\n`
        }
        resolve(allText)
      } catch (error) {
        console.error(`Error processing PDF (${file.name}):`, error)
        reject(`Failed to process ${file.name}. It might be corrupted or not a valid PDF.`)
      }
    }
    reader.onerror = (error) => reject(error)
    reader.readAsArrayBuffer(file)
  } catch (error) {
    console.error("Error initializing PDF.js:", error)
    reject("Failed to initialize PDF processing library.")
  }
 })
}

// Types
interface ExtractedSection {
  document_name: string
  section_title: string
  page_numbers: number[]
  relevance_score: number
  summary: string
  importance_rank: number
}

interface SubSectionAnalysis {
  section_title: string
  sub_section_title: string
  specific_details: string
  relevance_to_job: string
  importance_rank: number
}

interface AnalysisResults {
  metadata: {
    input_documents: string[]
```

```
    persona: string
    job_to_be_done: string
    processing_timestamp: string
  }
  extracted_sections: ExtractedSection[]
  sub_section_analysis: SubSectionAnalysis[]
}

export default function App() {
  // State Management
  const [persona, setPersona] = useState("PhD Researcher in Computational Biology")
  const [job, setJob] = useState(
    "Prepare a comprehensive literature review focusing on methodologies, datasets, and
performance benchmarks.",
  )
  const [files, setFiles] = useState<File[]>([])
  const [isLoading, setIsLoading] = useState(false)
  const [analysisStep, setAnalysisStep] = useState("")
  const [error, setError] = useState<string | null>(null)
  const [results, setResults] = useState<AnalysisResults | null>(null)
  const [activeTab, setActiveTab] = useState("metadata")
  const [showExplanation, setShowExplanation] = useState(false)

  // File Dropzone Handler
  const onDrop = useCallback((acceptedFiles: File[]) => {
    setFiles((prevFiles) => {
      const newFiles = acceptedFiles.filter((newFile) => !prevFiles.some((prevFile) =>
prevFile.name === newFile.name))
      return [...prevFiles, ...newFiles].slice(0, 10)
    })
    setError(null)
  }, [])

  const { getRootProps, getInputProps, isDragActive } = useDropzone({
    onDrop,
    accept: { "application/pdf": [".pdf"] },
    maxFiles: 10,
  })

  const removeFile = useCallback((fileName: string) => {
    setFiles((files) => files.filter((file) => file.name !== fileName))
  }, [])

  // Core Analysis Logic
```

```
  const handleAnalyze = async () => {
    if (files.length < 1 || files.length > 10) {
      setError("Please upload between 1 and 10 PDF documents.")
      return
    }
    if (!persona.trim() || !job.trim()) {
      setError("Please define both a Persona and a Job-to-be-Done.")
      return
    }

    setIsLoading(true)
    setError(null)
    setResults(null)

    try {
      // 1. Extract text from all PDFs
      setAnalysisStep("Extracting text from PDFs...")
      console.log("[ANALYSIS_STEP] 1. Starting PDF text extraction.")

      const documentContents = await Promise.all(
        files.map(async (file) => {
          console.log(`  - Processing ${file.name}`)
          const content = await extractTextFromPdf(file)
          console.log(`  - Finished ${file.name}`)
          return { name: file.name, content }
        }),
      )

      console.log("[ANALYSIS_STEP] 1. PDF text extraction complete.")

      // 2. Construct the detailed prompt
      setAnalysisStep("Constructing analysis prompt...")
      console.log("[ANALYSIS_STEP] 2. Constructing prompt for AI.")

      const fileNames = files.map((f) => f.name)
      const documentsString = documentContents
        .map((doc) => `--- Document: ${doc.name} ---\n${doc.content}`)
        .join("\n\n")

      const prompt = `You are an intelligent document analyst. Your task is to analyze the
following documents based on a given persona and job-to-be-done.

You must extract and prioritize the most relevant sections and sub-sections.
```

Persona: ${persona}

Job-to-be-Done: ${job}

Document Contents:
${documentsString}

Instructions:
1. Analyze all the provided document text.
2. Identify sections and sub-sections that are most relevant to the persona's job.
3. Rank the findings by importance. A lower rank (e.g., 1) is more important.
4. Generate a JSON object that strictly adheres to the specified schema. Do not add any extra text, explanations, or markdown formatting around the JSON output.`

```
    console.log("[ANALYSIS_STEP] 2. Prompt constructed.")

    // Define JSON schema
    const schema = {
      type: "OBJECT",
      properties: {
        metadata: {
          type: "OBJECT",
          properties: {
            input_documents: { type: "ARRAY", items: { type: "STRING" } },
            persona: { type: "STRING" },
            job_to_be_done: { type: "STRING" },
            processing_timestamp: { type: "STRING", format: "date-time" },
          },
          required: ["input_documents", "persona", "job_to-be-done", "processing_timestamp"],
        },
        extracted_sections: {
          type: "ARRAY",
          items: {
            type: "OBJECT",
            properties: {
              document_name: { type: "STRING" },
              section_title: { type: "STRING" },
              page_numbers: { type: "ARRAY", items: { type: "NUMBER" } },
              relevance_score: { type: "NUMBER" },
              summary: { type: "STRING" },
              importance_rank: { type: "NUMBER" },
            },
            required: [
              "document_name",
```

```
          "section_title",
          "page_numbers",
          "relevance_score",
          "summary",
          "importance_rank",
        ],
      },
    },
    sub_section_analysis: {
      type: "ARRAY",
      items: {
        type: "OBJECT",
        properties: {
          section_title: { type: "STRING" },
          sub_section_title: { type: "STRING" },
          specific_details: { type: "STRING" },
          relevance_to_job: { type: "STRING" },
          importance_rank: { type: "NUMBER" },
        },
        required: [
          "section_title",
          "sub_section_title",
          "specific_details",
          "relevance_to_job",
          "importance_rank",
        ],
      },
    },
  },
  required: ["metadata", "extracted_sections", "sub_section_analysis"],
}

// 3. Prepare and send the request to the Gemini API
setAnalysisStep("Analyzing with AI...")
console.log("[ANALYSIS_STEP] 3. Calling Gemini API.")

const payload = {
  contents: [{ role: "user", parts: [{ text: prompt }] }],
  generationConfig: {
    responseMimeType: "application/json",
    responseSchema: schema,
  },
}
```

```
      // For demo purposes, we'll simulate the API response
      // In production, you would use your actual Gemini API key
      const simulateApiResponse = () => {
        return new Promise<AnalysisResults>((resolve) => {
          setTimeout(() => {
            const mockResults: AnalysisResults = {
              metadata: {
                input_documents: fileNames,
                persona: persona,
                job_to_be_done: job,
                processing_timestamp: new Date().toISOString(),
              },
              extracted_sections: [
                {
                  document_name: fileNames[0] || "sample.pdf",
                  section_title: "Methodology Overview",
                  page_numbers: [1, 2, 3],
                  relevance_score: 95,
                  summary:
                    "Comprehensive overview of computational methods used in biological research,
including machine learning approaches and statistical analysis techniques.",
                  importance_rank: 1,
                },
                {
                  document_name: fileNames[0] || "sample.pdf",
                  section_title: "Dataset Description",
                  page_numbers: [4, 5],
                  relevance_score: 88,
                  summary:
                    "Detailed description of the datasets used, including sample sizes, data collection
methods, and preprocessing steps.",
                  importance_rank: 2,
                },
                {
                  document_name: fileNames[1] || "sample2.pdf",
                  section_title: "Performance Benchmarks",
                  page_numbers: [6, 7, 8],
                  relevance_score: 92,
                  summary:
                    "Comparative analysis of different algorithms showing accuracy, precision, recall,
and computational efficiency metrics.",
                  importance_rank: 3,
                },
              ],
```

```
          sub_section_analysis: [
            {
              section_title: "Methodology Overview",
              sub_section_title: "Machine Learning Algorithms",
              specific_details:
                "Implementation of Random Forest, SVM, and Neural Network approaches with
hyperparameter optimization using grid search.",
              relevance_to_job:
                "Directly relevant for literature review as it provides specific algorithmic approaches
that can be compared across studies.",
              importance_rank: 1,
            },
            {
              section_title: "Dataset Description",
              sub_section_title: "Data Preprocessing",
              specific_details:
                "Normalization techniques, outlier detection using IQR method, and feature
selection using mutual information.",
              relevance_to_job:
                "Essential for understanding data quality and preprocessing standards in
computational biology research.",
              importance_rank: 2,
            },
            {
              section_title: "Performance Benchmarks",
              sub_section_title: "Cross-validation Results",
              specific_details:
                "10-fold cross-validation showing mean accuracy of 94.2% ± 2.1% with statistical
significance testing using t-tests.",
              relevance_to_job:
                "Critical for comparing performance across different studies and establishing
benchmark standards.",
              importance_rank: 3,
            },
          ],
        }
        resolve(mockResults)
      }, 3000)
    })
  }

  // Simulate API call (replace with actual API call in production)
  const mockResults = await simulateApiResponse()
  setResults(mockResults)
```

```
      console.log("[ANALYSIS_STEP] 4. Analysis complete and results set.")
    } catch (err: any) {
      console.error("Detailed error in handleAnalyze:", err)
      let friendlyError = "An unknown error occurred during analysis."

      if (err.message?.includes("Failed to fetch")) {
        friendlyError = "Network error. Please check your connection or if an ad-blocker is
interfering."
      } else if (err.message?.includes("API call failed")) {
        friendlyError = `The analysis service failed. Please try again later.`
      } else if (err.message?.includes("Failed to process")) {
        friendlyError = err.message
      } else if (err.message?.includes("JSON.parse") || err.message?.includes("invalid JSON
format")) {
        friendlyError =
          "The analysis service returned an invalid format. This might be due to an incorrect API
response."
      } else if (err.message?.includes("API Key is not configured")) {
        friendlyError = err.message
      }

      setError(friendlyError)
    } finally {
      setIsLoading(false)
      setAnalysisStep("")
    }
  }

  // Memoized File List for Performance
  const fileList = useMemo(
    () =>
      files.map((file) => (
        <div
          key={file.name}
          className="bg-slate-100 dark:bg-slate-700 p-2 rounded-md flex items-center
justify-between text-sm"
        >
          <div className="flex items-center gap-2 overflow-hidden">
            <FileIcon className="h-5 w-5 text-blue-500 flex-shrink-0" />
            <span className="truncate">
              {file.name} - {(file.size / 1024).toFixed(2)} KB
            </span>
          </div>
          <button
```

```
          onClick={() => removeFile(file.name)}
          className="p-1 rounded-full hover:bg-slate-200 dark:hover:bg-slate-600"
        >
          <XIcon className="h-4 w-4 text-slate-500 dark:text-slate-400" />
        </button>
      </div>
    )),
  [files, removeFile],
 )

 return (
  <div className="bg-slate-50 dark:bg-slate-900 min-h-screen font-sans text-slate-800
dark:text-slate-200 p-4 sm:p-6 lg:p-8">
    <div className="max-w-7xl mx-auto">
     {/* Header */}
     <header className="mb-8 flex justify-between items-center">
      <div className="flex items-center gap-3">
       <BotIcon className="h-8 w-8 text-blue-600 dark:text-blue-500" />
       <div>
        <h1 className="text-2xl sm:text-3xl font-bold text-slate-900 dark:text-white">
         Persona-Driven Document Intelligence
        </h1>
        <p className="text-sm text-slate-500 dark:text-slate-400">
         Extract what matters, for the user who matters.
        </p>
       </div>
      </div>
      <button
       onClick={() => setShowExplanation(true)}
       className="flex items-center gap-2 px-4 py-2 text-sm font-medium text-blue-600
dark:text-blue-400 bg-blue-100 dark:bg-blue-900/50 rounded-lg hover:bg-blue-200
dark:hover:bg-blue-900 transition-colors"
      >
       <InfoIcon className="h-4 w-4" />
       Approach
      </button>
     </header>

     {/* Main Content Grid */}
     <main className="grid grid-cols-1 lg:grid-cols-2 gap-8">
      {/* Input Section */}
      <div className="space-y-6">
       {/* Persona Input */}
       <div>
```

```
            <label htmlFor="persona" className="block text-sm font-medium text-slate-700
dark:text-slate-300 mb-1">
              1. Define Persona
            </label>
            <input
              type="text"
              id="persona"
              value={persona}
              onChange={(e) => setPersona(e.target.value)}
              className="w-full p-3 bg-white dark:bg-slate-800 border border-slate-300
dark:border-slate-600 rounded-lg focus:ring-2 focus:ring-blue-500 focus:border-blue-500
transition"
              placeholder="e.g., Investment Analyst"
            />
          </div>

          {/* Job Input */}
          <div>
            <label htmlFor="job" className="block text-sm font-medium text-slate-700
dark:text-slate-300 mb-1">
              2. Define Job-to-be-Done
            </label>
            <textarea
              id="job"
              value={job}
              onChange={(e) => setJob(e.target.value)}
              rows={3}
              className="w-full p-3 bg-white dark:bg-slate-800 border border-slate-300
dark:border-slate-600 rounded-lg focus:ring-2 focus:ring-blue-500 focus:border-blue-500
transition"
              placeholder="e.g., Analyze revenue trends and R&D investments"
            />
          </div>

          {/* File Upload */}
          <div>
            <label className="block text-sm font-medium text-slate-700 dark:text-slate-300
mb-1">
              3. Upload Documents (1-10 PDFs)
            </label>
            <div
              {...getRootProps()}
              className={`p-6 border-2 border-dashed rounded-lg cursor-pointer transition-colors
text-center ${
```

```
              isDragActive
                ? "border-blue-500 bg-blue-50 dark:bg-blue-900/20"
                : "border-slate-300 dark:border-slate-600 hover:border-blue-400"
            }`}
          >
            <input {...getInputProps()} />
            <UploadCloudIcon className="h-10 w-10 mx-auto text-slate-400 dark:text-slate-500 mb-2" />
            {isDragActive ? <p>Drop the files here ...</p> : <p>Drag & drop PDFs here, or click to select</p>}
            <p className="text-xs text-slate-500 dark:text-slate-400 mt-1">Max 10 files</p>
          </div>
          {files.length > 0 && <div className="mt-4 space-y-2 max-h-48 overflow-y-auto pr-2">{fileList}</div>}
        </div>

        {/* Action Button */}
        <div className="pt-2">
          <button
            onClick={handleAnalyze}
            disabled={isLoading || files.length === 0}
            className="w-full flex items-center justify-center gap-3 py-3 px-4 bg-blue-600 text-white font-semibold rounded-lg shadow-md hover:bg-blue-700 disabled:bg-slate-400 disabled:cursor-not-allowed transition-all"
          >
            {isLoading ? (
              <>
                <svg
                  className="animate-spin -ml-1 mr-3 h-5 w-5 text-white"
                  xmlns="http://www.w3.org/2000/svg"
                  fill="none"
                  viewBox="0 0 24 24"
                >
                  <circle
                    className="opacity-25"
                    cx="12"
                    cy="12"
                    r="10"
                    stroke="currentColor"
                    strokeWidth="4"
                  ></circle>
                  <path
                    className="opacity-75"
                    fill="currentColor"
```

```
                d="M4 12a8 8 0 018-8V0C5.373 0 0 5.373 0 12h4zm2 5.291A7.962 7.962 0 014
12H0c0 3.042 1.135 5.824 3 7.938l3-2.647z"
              ></path>
            </svg>
            Analyzing...
          </>
        ) : (
          "Analyze Documents"
        )}
      </button>
      {error && <p className="text-red-500 text-sm mt-3 text-center">{error}</p>}
    </div>
  </div>

  {/* Output Section */}
  <div className="bg-white dark:bg-slate-800/50 rounded-lg shadow-sm border
border-slate-200 dark:border-slate-700 p-1">
    <div className="bg-slate-100 dark:bg-slate-800 rounded-t-md p-4 h-full">
      <h2 className="text-lg font-semibold text-slate-800 dark:text-slate-100
mb-4">Analysis Results</h2>

      {isLoading && (
        <div className="flex flex-col items-center justify-center h-full text-slate-500
dark:text-slate-400">
          <BotIcon className="h-12 w-12 mb-4 animate-pulse" />
          <p className="font-semibold">Processing...</p>
          <p className="text-sm mt-1">{analysisStep}</p>
        </div>
      )}

      {!isLoading && !results && (
        <div className="flex flex-col items-center justify-center h-full text-slate-500
dark:text-slate-400 text-center">
          <InfoIcon className="h-12 w-12 mb-4" />
          <p>Your analysis results will appear here.</p>
          <p className="text-sm">Fill out the form and click "Analyze".</p>
        </div>
      )}

      {results && (
        <div>
          {/* Tabs */}
          <div className="border-b border-slate-300 dark:border-slate-600 mb-4">
            <nav className="-mb-px flex space-x-6" aria-label="Tabs">
```

```
                <button
                  onClick={() => setActiveTab("metadata")}
                  className={`${
                    activeTab === "metadata"
                      ? "border-blue-500 text-blue-600"
                      : "border-transparent text-slate-500 hover:text-slate-700
hover:border-slate-300 dark:text-slate-400 dark:hover:text-slate-200"
                  } whitespace-nowrap py-3 px-1 border-b-2 font-medium text-sm`}
                >
                  Metadata
                </button>
                <button
                  onClick={() => setActiveTab("sections")}
                  className={`${
                    activeTab === "sections"
                      ? "border-blue-500 text-blue-600"
                      : "border-transparent text-slate-500 hover:text-slate-700
hover:border-slate-300 dark:text-slate-400 dark:hover:text-slate-200"
                  } whitespace-nowrap py-3 px-1 border-b-2 font-medium text-sm`}
                >
                  Extracted Sections
                </button>
                <button
                  onClick={() => setActiveTab("subsections")}
                  className={`${
                    activeTab === "subsections"
                      ? "border-blue-500 text-blue-600"
                      : "border-transparent text-slate-500 hover:text-slate-700
hover:border-slate-300 dark:text-slate-400 dark:hover:text-slate-200"
                  } whitespace-nowrap py-3 px-1 border-b-2 font-medium text-sm`}
                >
                  Sub-section Analysis
                </button>
              </nav>
            </div>

            {/* Tab Content */}
            <div className="max-h-[600px] overflow-y-auto pr-2">
              {activeTab === "metadata" && (
                <div className="space-y-2 text-sm">
                  <p>
                    <strong>Persona:</strong> {results.metadata.persona}
                  </p>
                  <p>
```

```
                  <strong>Job-to-be-Done:</strong> {results.metadata.job_to_be_done}
                </p>
                <p>
                  <strong>Input Documents:</strong> {results.metadata.input_documents.join(",
")}
                </p>
                <p>
                  <strong>Processed At:</strong>{" "}
                  {new Date(results.metadata.processing_timestamp).toLocaleString()}
                </p>
              </div>
            )}

            {activeTab === "sections" && (
              <div className="space-y-4">
                {results.extracted_sections.length > 0 ? (
                  results.extracted_sections
                    .sort((a, b) => a.importance_rank - b.importance_rank)
                    .map((section, index) => (
                      <div
                        key={index}
                        className="bg-slate-50 dark:bg-slate-700 p-3 rounded-md border
border-slate-200 dark:border-slate-600"
                      >
                        <h3 className="font-semibold text-blue-600 dark:text-blue-300">
                          {section.section_title} (Rank: {section.importance_rank})
                        </h3>
                        <p className="text-sm text-slate-600 dark:text-slate-300
mt-1">{section.summary}</p>
                        <p className="text-xs text-slate-500 dark:text-slate-400 mt-1">
                          Document: {section.document_name} | Pages:
{section.page_numbers.join(", ")} |
                          Relevance: {section.relevance_score}%
                        </p>
                      </div>
                    ))
                ) : (
                  <p className="text-center text-slate-500 dark:text-slate-400">No sections
extracted.</p>
                )}
              </div>
            )}

            {activeTab === "subsections" && (
```

```
                <div className="space-y-4">
                  {results.sub_section_analysis.length > 0 ? (
                    results.sub_section_analysis
                      .sort((a, b) => a.importance_rank - b.importance_rank)
                      .map((sub, index) => (
                        <div
                          key={index}
                          className="bg-slate-50 dark:bg-slate-700 p-3 rounded-md border
border-slate-200 dark:border-slate-600"
                        >
                          <h3 className="font-semibold text-green-600 dark:text-green-300">
                            {sub.sub_section_title} (from {sub.section_title}) (Rank:
{sub.importance_rank})
                          </h3>
                          <p className="text-sm text-slate-600 dark:text-slate-300 mt-1">
                            <strong>Details:</strong> {sub.specific_details}
                          </p>
                          <p className="text-xs text-slate-500 dark:text-slate-400 mt-1">
                            <strong>Relevance to Job:</strong> {sub.relevance_to_job}
                          </p>
                        </div>
                      ))
                  ) : (
                    <p className="text-center text-slate-500 dark:text-slate-400">No sub-sections
analyzed.</p>
                  )}
                </div>
              )}
            </div>
          )}
        </div>
      </main>

      {/* Explanation Modal */}
      {showExplanation && (
        <div className="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center p-4
z-50">
          <div className="bg-white dark:bg-slate-800 rounded-lg shadow-xl max-w-lg w-full p-6
relative">
            <button
              onClick={() => setShowExplanation(false)}
```

```
            className="absolute top-3 right-3 text-slate-500 hover:text-slate-700
dark:text-slate-400 dark:hover:text-slate-200"
          >
            <XIcon className="h-6 w-6" />
          </button>
          <h2 className="text-xl font-bold mb-4 text-slate-900 dark:text-white">How This App
Works</h2>
          <p className="text-slate-700 dark:text-slate-300 mb-4">
            This application leverages the power of AI to intelligently extract and prioritize
information from PDF
            documents based on a user-defined <strong>Persona</strong> and a specific{" "}
            <strong>Job-to-be-Done</strong>.
          </p>
          <ol className="list-decimal list-inside space-y-2 text-slate-700 dark:text-slate-300">
            <li>
              <strong>PDF Text Extraction:</strong> First, your uploaded PDF documents are
processed locally in your
              browser using{" "}
              <code className="font-mono text-sm bg-slate-200 dark:bg-slate-700 px-1 py-0.5
rounded">pdf.js</code>{" "}
              to extract all readable text content. Page markers are added to maintain context.
            </li>
            <li>
              <strong>Prompt Construction:</strong> The extracted text, along with your defined
Persona and
              Job-to-be-Done, is then used to create a detailed prompt for the AI. This prompt
instructs the AI to
              act as a document analyst.
            </li>
            <li>
              <strong>AI Analysis:</strong> The prompt is sent to an AI API. The API is instructed
to return a
              structured JSON response containing:
              <ul className="list-disc list-inside ml-4 mt-1 text-sm">
                <li>
                  <strong>Metadata:</strong> Information about the analysis (input files, persona,
job, timestamp).
                </li>
                <li>
                  <strong>Extracted Sections:</strong> Key sections from the documents deemed
relevant, along with a
                  summary, page numbers, relevance score, and an importance rank.
                </li>
                <li>
```

```
                <strong>Sub-section Analysis:</strong> More granular details extracted from
relevant sub-sections,
                explaining their specific content and relevance to the job, also with an importance
rank.
              </li>
            </ul>
          </li>
          <li>
            <strong>Results Display:</strong> The structured JSON response from the AI is
then parsed and
            displayed in a user-friendly format, categorized into Metadata, Extracted Sections,
and Sub-section
            Analysis.
          </li>
        </ol>
        <p className="text-slate-700 dark:text-slate-300 mt-4">
          This approach ensures that the AI's output is focused, relevant, and directly
actionable for the
          specified user role and task.
        </p>
      </div>
    </div>
  )}
 </div>
 </div>
 )
}
```

Json File :

```
{
  "name": "pdf-intelligence-app",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint"
```

```
},
"dependencies": {
  "@hookform/resolvers": "^3.9.1",
  "@radix-ui/react-accordion": "1.2.2",
  "@radix-ui/react-alert-dialog": "1.1.4",
  "@radix-ui/react-aspect-ratio": "1.1.1",
  "@radix-ui/react-avatar": "1.1.2",
  "@radix-ui/react-checkbox": "1.1.3",
  "@radix-ui/react-collapsible": "1.1.2",
  "@radix-ui/react-context-menu": "2.2.4",
  "@radix-ui/react-dialog": "1.1.4",
  "@radix-ui/react-dropdown-menu": "2.1.4",
  "@radix-ui/react-hover-card": "1.1.4",
  "@radix-ui/react-label": "2.1.1",
  "@radix-ui/react-menubar": "1.1.4",
  "@radix-ui/react-navigation-menu": "1.2.3",
  "@radix-ui/react-popover": "1.1.4",
  "@radix-ui/react-progress": "1.1.1",
  "@radix-ui/react-radio-group": "1.2.2",
  "@radix-ui/react-scroll-area": "1.2.2",
  "@radix-ui/react-select": "2.1.4",
  "@radix-ui/react-separator": "1.1.1",
  "@radix-ui/react-slider": "1.2.2",
  "@radix-ui/react-slot": "1.1.1",
  "@radix-ui/react-switch": "1.1.2",
  "@radix-ui/react-tabs": "1.1.2",
  "@radix-ui/react-toast": "1.2.4",
  "@radix-ui/react-toggle": "1.1.1",
  "@radix-ui/react-toggle-group": "1.1.1",
  "@radix-ui/react-tooltip": "1.1.6",
  "autoprefixer": "^10.0.1",
  "class-variance-authority": "^0.7.1",
  "clsx": "^2.1.1",
  "cmdk": "1.0.4",
  "date-fns": "4.1.0",
  "embla-carousel-react": "8.5.1",
  "geist": "^1.3.1",
  "input-otp": "1.4.1",
  "lucide-react": "^0.454.0",
  "next": "14.0.0",
  "next-themes": "^0.4.4",
  "react": "^18",
  "react-dom": "^18",
  "react-dropzone": "^14.2.3",
```

```
    "react-hook-form": "^7.54.1",
    "react-resizable-panels": "^2.1.7",
    "recharts": "2.15.0",
    "sonner": "^1.7.1",
    "tailwind-merge": "^2.5.5",
    "tailwindcss-animate": "^1.0.7",
    "vaul": "^0.9.6",
    "zod": "^3.24.1",
    "pdfjs-dist": "^3.11.174"
  },
  "devDependencies": {
    "@types/node": "^20",
    "@types/react": "^18",
    "@types/react-dom": "^18",
    "postcss": "^8",
    "tailwindcss": "^3.4.17",
    "typescript": "^5",
    "eslint": "^8",
    "eslint-config-next": "14.0.0"
  }
}
```