To make this Resume Screening Application more robust, user-friendly, and ready for public use, there are a few key areas that can be extended and improved. Here's an outline of additional features and enhancements that could be added:

1. File Type and Size Validation

Prevent users from uploading unsupported file types or overly large files. This helps to maintain performance and ensure the system only processes what it can handle.

2. User Feedback During Upload

Adding a progress bar or a visual indicator during the file upload process could enhance user experience, especially when uploading multiple large files.

3. Downloadable Results or Export Option

Users may want to download the analyzed results in CSV format to keep track of successful candidates.

4. Improved Status Messaging

Instead of random success or failure, the app could give feedback based on keyword matching or specific attributes (e.g., education level, years of experience) from the resume. You could add a simple parser to check for certain qualifications.

5. Error Handling and Notifications

Notify users of errors, such as when a file fails to upload or analyze. This includes handling edge cases like empty or corrupted files.

6. Accessibility Improvements

Ensure the app is accessible to users with disabilities by improving keyboard navigation and adding ARIA labels for screen readers.

7. Security Enhancements

Though it's simulated here, in a real-world deployment, you'd want to sanitize file uploads and ensure files are not malicious.

I'll go ahead and extend the code to include some of these improvements for a more feature-complete application:

---

Extended Code with File Validation, Notifications, and Export Option

"use client"

import { useState } from "react"

import { Check, X, Upload, RefreshCw, FileText, Download } from "lucide-react"

import { Button } from "@/components/ui/button"

import { Input } from "@/components/ui/input"

import { Table, TableBody, TableCell, TableHead, TableHeader, TableRow } from "@/components/ui/table"

const analyzeResumes = async (files: File[]) => {

await new Promise(resolve => setTimeout(resolve, 2000)) // Simulating API call

return files.map((file, index) => ({

id: index + 1,

name: file.name,

successful: Math.random() > 0.5,

email: `candidate${index + 1}@example.com`,

phone: `+1 (555) ${String(Math.floor(Math.random() \* 900) + 100).padStart(3, '0')}-${String(Math.floor(Math.random() \* 9000) + 1000)}`,

}))

}

export default function ResumeScreener() {

const [resumes, setResumes] = useState<File[]>([])

const [results, setResults] = useState<any[]>([])

const [isAnalyzing, setIsAnalyzing] = useState(false)

const [errorMessage, setErrorMessage] = useState<string | null>(null)

const maxFileSizeMB = 5

// Function to validate file type and size

const validateFile = (file: File) => {

const validTypes = ['application/pdf', 'application/msword', 'application/vnd.openxmlformats-officedocument.wordprocessingml.document']

if (!validTypes.includes(file.type)) {

return `Invalid file type for ${file.name}. Only PDF, DOC, and DOCX files are allowed.`

}

if (file.size > maxFileSizeMB \* 1024 \* 1024) {

return `File ${file.name} is too large. Max size is ${maxFileSizeMB}MB.`

}

return null

}

const handleFileUpload = (event: React.ChangeEvent<HTMLInputElement>) => {

if (event.target.files) {

const uploadedFiles = Array.from(event.target.files)

const errors = uploadedFiles.map(file => validateFile(file)).filter(error => error !== null)

if (errors.length > 0) {

setErrorMessage(errors.join('\n'))

setResumes([]) // Clear the previous valid files

} else {

setErrorMessage(null)

setResumes(uploadedFiles)

}

}

}

const handleAnalysis = async () => {

setIsAnalyzing(true)

const analysisResults = await analyzeResumes(resumes)

setResults(analysisResults)

setIsAnalyzing(false)

}

// Function to export results as CSV

const exportResultsAsCSV = () => {

const csvContent = "data:text/csv;charset=utf-8,"

+ ["ID,Name,Status,Email,Phone"]

.concat(results.map(result =>

`${result.id},${result.name},${result.successful ? "Successful" : "Unsuccessful"},${result.successful ? result.email : "-"},${result.successful ? result.phone : "-"}`))

.join("\n")

const encodedUri = encodeURI(csvContent)

const link = document.createElement("a")

link.setAttribute("href", encodedUri)

link.setAttribute("download", "resume\_analysis\_results.csv")

document.body.appendChild(link)

link.click()

}

return (

<div className="container mx-auto p-4 max-w-4xl">

<h1 className="text-2xl font-bold mb-4">Resume Screening Application</h1>

<div className="mb-4">

<Input

type="file"

accept=".pdf,.doc,.docx"

multiple

onChange={handleFileUpload}

className="mb-2"

/>

{errorMessage && <p className="text-red-500 text-sm mb-2">{errorMessage}</p>}

<p className="text-sm text-gray-500 mb-2">

{resumes.length} {resumes.length === 1 ? 'resume' : 'resumes'} selected

</p>

<Button

onClick={handleAnalysis}

disabled={resumes.length === 0 || isAnalyzing}

className="w-full"

>

{isAnalyzing ? (

<>

<RefreshCw className="mr-2 h-4 w-4 animate-spin" />

Analyzing Resumes...

</>

) : (

<>

<Upload className="mr-2 h-4 w-4" />

Analyze Resumes

</>

)}

</Button>

</div>

{results.length > 0 && (

<>

<Table>

<TableHeader>

<TableRow>

<TableHead>ID</TableHead>

<TableHead>Name</TableHead>

<TableHead>Status</TableHead>

<TableHead>Email</TableHead>

<TableHead>Phone</TableHead>

</TableRow>

</TableHeader>

<TableBody>

{results.map((result) => (

<TableRow key={result.id}>

<TableCell>{result.id}</TableCell>

<TableCell>{result.name}</TableCell>

<TableCell>

{result.successful ? (

<Check className="text-green-500" />

) : (

<X className="text-red-500" />

)}

</TableCell>

<TableCell>{result.successful ? result.email : "-"}</TableCell>

<TableCell>{result.successful ? result.phone : "-"}</TableCell>

</TableRow>

))}

</TableBody>

</Table>

<Button onClick={exportResultsAsCSV} className="mt-4">

<Download className="mr-2 h-4 w-4" />

Export Results as CSV

</Button>

</>

)}

</div>

)

}

Changes and New Features:

1. File Validation:

Checks the type and size of uploaded files. Files that don't meet the criteria trigger an error message.

Error messages are displayed below the file upload input.

2. CSV Export:

Added a button to export the results in CSV format, making it easy for users to download and keep the analysis.

3. User Feedback:

Shows error messages if files are invalid or too large.