import React, { useState, useCallback } from "react";

import { LegalCase, Document } from "@/entities/all";

import { UploadFile, ExtractDataFromUploadedFile, InvokeLLM } from "@/integrations/Core";

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

import { Card, CardContent, CardHeader, CardTitle } from "@/components/ui/card";

import { Textarea } from "@/components/ui/textarea";

import { ArrowLeft, Upload, FileText, Bot, BrainCircuit, CheckCircle, Loader2 } from "lucide-react";

import { Link, useNavigate } from "react-router-dom";

import { createPageUrl } from "@/utils";

import AIAnalysisProgress from "../components/cases/AIAnalysisProgress";

export default function CreateCasePage() {

const [description, setDescription] = useState("");

const [files, setFiles] = useState([]);

const [dragActive, setDragActive] = useState(false);

const [isProcessing, setIsProcessing] = useState(false);

const [analysisSteps, setAnalysisSteps] = useState([]);

const [fileErrors, setFileErrors] = useState([]);

const [processingError, setProcessingError] = useState("");

const navigate = useNavigate();

const MAX\_FILE\_SIZE = 10 \* 1024 \* 1024; // 10MB in bytes

const validateFile = (file) => {

const errors = [];

if (file.size > MAX\_FILE\_SIZE) {

errors.push(`${file.name}: File size (${(file.size / 1024 / 1024).toFixed(1)}MB) exceeds 10MB limit`);

}

const allowedTypes = ['pdf', 'png', 'jpg', 'jpeg', 'docx', 'txt'];

const fileExtension = file.name.split('.').pop().toLowerCase();

if (!allowedTypes.includes(fileExtension)) {

errors.push(`${file.name}: File type not supported. Please use PDF, DOCX, TXT, or image files`);

}

return errors;

};

const handleDrag = useCallback((e) => {

e.preventDefault();

e.stopPropagation();

if (e.type === "dragenter" || e.type === "dragover") {

setDragActive(true);

} else if (e.type === "dragleave") {

setDragActive(false);

}

}, []);

const handleDrop = useCallback((e) => {

e.preventDefault();

e.stopPropagation();

setDragActive(false);

if (e.dataTransfer.files && e.dataTransfer.files[0]) {

const newFiles = Array.from(e.dataTransfer.files);

const allErrors = [];

const validFiles = [];

newFiles.forEach(file => {

const fileErrors = validateFile(file);

if (fileErrors.length === 0) {

validFiles.push(file);

} else {

allErrors.push(...fileErrors);

}

});

setFiles(prev => [...prev, ...validFiles]);

setFileErrors(allErrors);

}

}, []);

const handleFileSelect = (e) => {

if (e.target.files && e.target.files[0]) {

const newFiles = Array.from(e.target.files);

const allErrors = [];

const validFiles = [];

newFiles.forEach(file => {

const fileErrors = validateFile(file);

if (fileErrors.length === 0) {

validFiles.push(file);

} else {

allErrors.push(...fileErrors);

}

});

setFiles(prev => [...prev, ...validFiles]);

setFileErrors(allErrors);

e.target.value = null; // Clear input field to allow re-uploading the same file after removal

}

};

const removeFile = (indexToRemove) => {

setFiles(prev => prev.filter((\_, index) => index !== indexToRemove));

setFileErrors([]); // Clear errors when files are removed

};

const handleCreateCase = async () => {

if (!description.trim()) {

setProcessingError("Please provide a description for the legal issue.");

return;

}

setIsProcessing(true);

setFileErrors([]);

setProcessingError("");

setAnalysisSteps([]);

let stepIndex = -1;

const addStep = (title, status = 'processing') => {

stepIndex++;

setAnalysisSteps(prev => [...prev, { title, status }]);

};

const updateCurrentStepStatus = (status, errorMessage = "") => {

setAnalysisSteps(prev => {

const newSteps = [...prev];

if(newSteps[stepIndex]) {

newSteps[stepIndex].status = status;

if (errorMessage) {

newSteps[stepIndex].error = errorMessage;

}

}

return newSteps;

});

};

try {

let uploadedFiles = [];

let documentContents = [];

if (files.length > 0) {

addStep("Uploading documents...");

try {

uploadedFiles = await Promise.all(

files.map(file => UploadFile({ file }))

);

updateCurrentStepStatus('complete');

} catch (uploadError) {

console.error("Upload error:", uploadError);

updateCurrentStepStatus('error', 'Failed to upload one or more files');

setProcessingError("Failed to upload documents. Please try again with smaller files or check your internet connection.");

setIsProcessing(false);

return;

}

addStep("Analyzing document contents...");

documentContents = await Promise.all(

uploadedFiles.map(async ({ file\_url }, index) => {

let extractedContent = "";

try {

const extractResult = await ExtractDataFromUploadedFile({

file\_url,

json\_schema: { type: "object", properties: { content: { type: "string" } } }

});

if (extractResult.status === "success" && extractResult.output?.content) {

extractedContent = extractResult.output.content;

}

} catch (e) {

console.error("Extraction failed for file:", files[index].name, e);

extractedContent = `[Content extraction failed for ${files[index].name}]`;

}

return `Document ${index + 1} (${files[index].name}):\n${extractedContent}`;

})

);

updateCurrentStepStatus('complete');

}

addStep("AI is structuring your case...");

let newCase; // Declare newCase here so it's accessible after the try-catch

try {

const caseSchema = LegalCase.schema();

const caseProperties = caseSchema.properties;

delete caseProperties.status;

const documentsContext = documentContents.length > 0

? `Content from Uploaded Documents:\n---\n${documentContents.join("\n\n---\n\n")}\n---\n\n`

: '';

const prompt = `You are an expert legal AI assistant specializing in case intake and analysis. Based on the user's description${documentContents.length > 0 ? ' and the content of the provided documents' : ''}, analyze the information and structure it into a new legal case.

User's Description:

---

${description}

---

${documentsContext}Your task is to analyze all this information and generate a structured JSON object for the new legal case.

- Provide a concise but descriptive 'title' (required).

- Determine the most appropriate 'case\_type' from: personal\_injury, contract\_dispute, family\_law, criminal\_defense, employment, real\_estate, corporate, intellectual\_property, immigration, other (required).

- Infer the 'jurisdiction' if possible from the text.

- Summarize the key facts and issues in the 'description'.

- Assess the 'priority' as: low, medium, high, or urgent.

- If any deadlines are mentioned, identify the most immediate one for the 'deadline' field in YYYY-MM-DD format.

Ensure all required fields are filled appropriately. The response must be a valid JSON object.`;

const aiResponse = await InvokeLLM({

prompt,

response\_json\_schema: { type: "object", properties: caseProperties }

});

if (!aiResponse || !aiResponse.title || !aiResponse.case\_type) {

throw new Error("AI response is incomplete or invalid. Missing title or case\_type.");

}

updateCurrentStepStatus('complete');

addStep("Creating the legal case...");

newCase = await LegalCase.create({

...aiResponse,

status: 'active',

title: aiResponse.title || 'Untitled Legal Case',

case\_type: aiResponse.case\_type || 'other',

description: aiResponse.description || description

});

if (!newCase || !newCase.id) {

throw new Error("Database error: Case was not created successfully, returned invalid ID.");

}

if (uploadedFiles.length > 0) {

await Promise.all(

uploadedFiles.map((uploadedFile, index) =>

Document.create({

case\_id: newCase.id,

title: files[index].name.replace(/\.[^/.]+$/, ""),

file\_url: uploadedFile.file\_url,

file\_type: files[index].name.split('.').pop().toLowerCase(),

extracted\_content: documentContents[index]?.split(':\n')[1] || '',

document\_category: 'other'

})

)

);

}

updateCurrentStepStatus('complete');

} catch (processingLogicError) {

console.error("AI structuring or Case Creation error:", processingLogicError);

updateCurrentStepStatus('error', processingLogicError.message);

setProcessingError(`Analysis failed: ${processingLogicError.message}. Please simplify your description or try again.`);

setIsProcessing(false);

return;

}

addStep("Redirecting to your new case...");

updateCurrentStepStatus('complete');

setTimeout(() => navigate(createPageUrl(`Chat?caseId=${newCase.id}`)), 1500);

} catch (error) {

console.error("General error creating case with AI:", error);

// Ensure the step is marked as error if the overall process fails

// This might not apply if the error happens before any step is added

if (analysisSteps.length === 0 || analysisSteps[analysisSteps.length - 1].status !== 'error') {

setAnalysisSteps(prev => [...prev, { title: 'Process failed', status: 'error' }]);

} else {

updateCurrentStepStatus('error', 'An unexpected error occurred.');

}

setProcessingError(`An unexpected error occurred: ${error.message}. Please try again.`);

setIsProcessing(false);

}

};

if (isProcessing) {

return (

<AIAnalysisProgress

steps={analysisSteps}

error={processingError}

onRetry={() => {

setIsProcessing(false);

setProcessingError("");

setAnalysisSteps([]);

}}

/>

);

}

return (

<div className="min-h-screen bg-gradient-to-br from-slate-50 to-blue-50 p-6">

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

<div className="flex items-center gap-4 mb-8">

<Link to={createPageUrl("Cases")}>

<Button variant="outline" size="icon">

<ArrowLeft className="w-4 h-4" />

</Button>

</Link>