import React, { useState, useEffect, useRef } from "react";

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

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

import { InvokeLLM } from "@/integrations/Core";

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

import { Card } from "@/components/ui/card";

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

import { Badge } from "@/components/ui/badge";

import { Send, Upload, ArrowLeft, Loader2 } from "lucide-react";

import { createPageUrl } from "@/utils";

import { format } from "date-fns";

import ChatMessages from "../components/chat/ChatMessages";

import DocumentUpload from "../components/chat/DocumentUpload";

import DocumentSidebar from "../components/chat/DocumentSidebar";

export default function ChatPage() {

const location = useLocation();

const urlParams = new URLSearchParams(location.search);

const caseId = urlParams.get('caseId');

const [legalCase, setLegalCase] = useState(null);

const [documents, setDocuments] = useState([]);

const [messages, setMessages] = useState([]);

const [newMessage, setNewMessage] = useState("");

const [isLoading, setIsLoading] = useState(true);

const [isSending, setIsSending] = useState(false);

const [showUpload, setShowUpload] = useState(false);

const [error, setError] = useState(null);

const messagesEndRef = useRef(null);

useEffect(() => {

if (caseId) {

loadCaseData();

} else {

setError("No case ID provided.");

setIsLoading(false);

}

}, [caseId]);

useEffect(() => {

scrollToBottom();

}, [messages]);

const scrollToBottom = () => {

messagesEndRef.current?.scrollIntoView({ behavior: "smooth" });

};

const loadCaseData = async () => {

setIsLoading(true);

setError(null);

try {

const caseData = await LegalCase.get(caseId);

if (!caseData) {

throw new Error("Case not found.");

}

const [documentsData, messagesData] = await Promise.all([

Document.filter({ case\_id: caseId }, "-created\_date"),

ChatMessage.filter({ case\_id: caseId }, "created\_date")

]);

setLegalCase(caseData);

setDocuments(documentsData);

setMessages(messagesData);

} catch (err) {

console.error("Error loading case data:", err);

setError(`Failed to load case data: ${err.message}`);

setLegalCase(null);

}

setIsLoading(false);

};

const handleSendMessage = async () => {

if (!newMessage.trim() || isSending) return;

const userMessage = newMessage;

setNewMessage("");

setIsSending(true);

// Add user message

const userMessageData = {

case\_id: caseId,

message: userMessage,

sender: "user",

message\_type: "text"

};

const savedUserMessage = await ChatMessage.create(userMessageData);

setMessages(prev => [...prev, savedUserMessage]);

try {

// Prepare context from documents

const documentContext = documents.map(doc =>

`Document: ${doc.title}\nContent: ${doc.extracted\_content || 'Content not extracted'}`

).join('\n\n');

const prompt = `You are a versatile and professional legal AI assistant. Your primary goal is to accurately and helpfully respond to the user's specific query.

CASE INFORMATION:

Case Title: ${legalCase?.title}

Case Type: ${legalCase?.case\_type}

Case Description: ${legalCase?.description}

Current Jurisdiction: ${legalCase?.jurisdiction}

Documents in this case:

${documentContext}

User's question: ${userMessage}

YOUR TASK:

1. \*\*Understand the User's Intent:\*\* First, determine what the user is asking for. Are they asking for a document summary, a legal definition, research, or advice on taking action?

2. \*\*Fulfill the Core Request:\*\* Provide a direct answer to the user's question. If they ask for a summary of documents, provide a detailed summary based on the provided context.

3. \*\*Apply Jurisdictional Analysis (ONLY IF RELEVANT):\*\* If the user's query involves filing a lawsuit, claim or any other legal document determining next legal steps, or asks where to take action, you MUST use the following jurisdictional analysis framework to ensure accuracy. DO NOT perform this analysis if the user is simply asking for information or a document summary.

\*\*Jurisdictional Framework:\*\*

- \*\*Federal, State, Municipal and Local Indicators:\*\* Look for federal, state, municipal and local indicators. federal, state, municipal and local property, federal, state, municipal and local laws, rules, regulations and government agencies, etc.

- \*\*Conclusion:\*\* If Federal, State, Municipal and Local indicators exist, jurisdiction should be stated accordingly.

- \*\*Reasoning:\*\* Briefly explain why.

4. \*\*Offer Further Assistance (ONLY IF RELEVANT):\*\* If you discuss filing any legal documents, you MUST also offer to help draft the necessary pro se documents for the correct court or agency.

CRITICAL RULES:

- Always recommend the appropriate jurisdiction and venue according to the uploaded documents and the express intentions and questions of the user.

- If your analysis suggests an update to the case data (like a deadline), include it in the 'updated\_case\_data' field.

`;

const responseSchema = {

type: "object",

properties: {

jurisdiction\_analysis: {

type: "object",

description: "Systematic analysis of proper jurisdiction and venue (ONLY include if relevant to the query)",

properties: {

federal\_indicators\_found: {

type: "array",

items: { type: "string" },

description: "List any federal,state, municipal, local indicators identified"

},

proper\_jurisdiction: {

type: "string",

enum: ["federal", "state", "uncertain"],

description: "Determined proper jurisdiction"

},

reasoning: {

type: "string",

description: "Explanation of jurisdiction determination"

}

}

},

response\_text: {

type: "string",

description: "The conversational response including legal guidance"

},

updated\_case\_data: {

type: "object",

description: "Case updates based on analysis",

properties: {

deadline: {

type: "string",

format: "date",

description: "Calculated deadline in YYYY-MM-DD format"

},

priority: {

type: "string",

enum: ["low", "medium", "high", "urgent"]

},

status: {

type: "string",

enum: ["active", "research", "drafting", "filing", "completed", "closed"]

},

jurisdiction: {

type: "string",

description: "Updated jurisdiction (e.g., 'Federal Court - Eastern District of Pennsylvania')"

}

}

}

},

required: ["response\_text"]

};

const aiResponseObject = await InvokeLLM({

prompt,

add\_context\_from\_internet: true,

response\_json\_schema: responseSchema

});

let aiMessageContent = aiResponseObject.response\_text;

// Add jurisdiction analysis to the response, if it was performed

if (aiResponseObject.jurisdiction\_analysis && aiResponseObject.jurisdiction\_analysis.reasoning) {

const analysis = aiResponseObject.jurisdiction\_analysis;

aiMessageContent += `\n\n\*\*JURISDICTION ANALYSIS:\*\*\n`;

if (analysis.federal\_state\_municipal\_local\_indicators\_found && analysis.federal\_state\_municipal\_local\_found.length > 0) {

aiMessageContent += `Federal, State, Municipal and local indicators identified: ${analysis.federal\_state\_municipal\_local\_found.join(', ')}\n`;

}

aiMessageContent += `Proper jurisdiction: ${analysis.proper\_jurisdiction.toUpperCase()} COURT/AGENCY\n`;

aiMessageContent += `Reasoning: ${analysis.reasoning}`;

}

// Check if there's data to update

if (aiResponseObject.updated\_case\_data && Object.keys(aiResponseObject.updated\_case\_data).length > 0) {

const updates = aiResponseObject.updated\_case\_data;

await LegalCase.update(caseId, updates);

setLegalCase(prev => ({ ...prev, ...updates }));

const updatedFields = Object.keys(updates).map(key => {

// Add T00:00:00 to avoid timezone issues when parsing date-only strings

if (key === 'deadline' && updates[key]) return `deadline to ${format(new Date(updates[key] + 'T00:00:00'), 'MMMM d, yyyy')}`;

return `${key} to "${updates[key]}"`;

}).join(', ');

aiMessageContent += `\n\n\*System Update: I've updated the case ${updatedFields}.\*`;

}

// Add AI response to chat

const aiMessageData = {

case\_id: caseId,

message: aiMessageContent,

sender: "assistant",

message\_type: "text"

};

const savedAiMessage = await ChatMessage.create(aiMessageData);

setMessages(prev => [...prev, savedAiMessage]);

} catch (error) {

console.error("Error getting AI response:", error);

const errorMessageData = {

case\_id: caseId,

message: "I apologize, but I'm having trouble processing your request right now. Please try again.",

sender: "assistant",

message\_type: "text"

};

const savedErrorMessage = await ChatMessage.create(errorMessageData);

setMessages(prev => [...prev, savedErrorMessage]);

}

setIsSending(false);

};

const handleKeyPress = (e) => {

if (e.key === 'Enter' && !e.shiftKey) {

e.preventDefault();

handleSendMessage();

}

};

const handleDocumentUploaded = () => {

loadCaseData();

setShowUpload(false);

};

if (isLoading) {

return (

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

<div className="text-center">

<Loader2 className="w-8 h-8 animate-spin mx-auto mb-4 text-[var(--navy)]" />

<p className="text-slate-600">Loading case data...</p>

</div>

</div>

);

}

if (error || !legalCase) {

return (

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

<Card className="p-8 text-center max-w-md">

<h2 className="text-xl font-semibold text-[var(--navy)] mb-2">Error Loading Case</h2>

<p className="text-slate-600 mb-4">{error || "The requested case could not be found."}</p>

<p className="text-sm text-slate-500 mb-4">Case ID: ${caseId}</p>

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

<Button variant="outline">

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

Back to Cases

</Button>

</Link>

</Card>

</div>

);

}

return (

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

<div className="flex-1 flex flex-col">

{/\* Header \*/}

<div className="bg-white/90 backdrop-blur-sm border-b border-slate-200 p-6">

<div className="flex items-center justify-between">

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

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

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

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

</Button>

</Link>

<div>

<h1 className="text-xl font-bold text-[var(--navy)]">{legalCase.title}</h1>

<div className="flex items-center gap-2 mt-1">

<Badge variant="outline" className="text-xs">

{legalCase.case\_type?.replace(/\_/g, ' ')}

</Badge>

{legalCase.jurisdiction && (

<span className="text-sm text-slate-500">{legalCase.jurisdiction}</span>

)}

</div>

</div>

</div>

<Button

onClick={() => setShowUpload(true)}

variant="outline"

className="flex items-center gap-2"

>

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

Upload Document

</Button>

</div>

</div>

{/\* Chat Area \*/}

<div className="flex-1 flex">

<div className="flex-1 flex flex-col">

<div className="flex-1 overflow-auto p-6">

<ChatMessages messages={messages} />

<div ref={messagesEndRef} />

</div>

{/\* Message Input \*/}

<div className="bg-white/90 backdrop-blur-sm border-t border-slate-200 p-4">

<div className="flex gap-3">

<div className="flex-1">

<Textarea

value={newMessage}

onChange={(e) => setNewMessage(e.target.value)}

onKeyPress={handleKeyPress}

placeholder="Ask your legal AI assistant anything..."

className="min-h-[60px] resize-none border-slate-200 focus:border-[var(--gold)] focus:ring-[var(--gold)]"

disabled={isSending}

/>

</div>

<Button

onClick={handleSendMessage}

disabled={!newMessage.trim() || isSending}

className="bg-gradient-to-r from-[var(--navy)] to-[var(--navy-light)] hover:shadow-lg transition-all duration-300 px-6"

>

{isSending ? (

<Loader2 className="w-4 h-4 animate-spin" />

) : (

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

)}

</Button>

</div>

</div>

</div>

{/\* Document Sidebar \*/}

<DocumentSidebar documents={documents} />

</div>

</div>

{/\* Upload Modal \*/}

<DocumentUpload

open={showUpload}

onOpenChange={setShowUpload}

caseId={caseId}

onDocumentUploaded={handleDocumentUploaded}

/>

</div>

);

}