**ReAlign MVP - Scaffold**

version **1.5**

date May, 21, 2025

// src/pages/TransactionView.tsx

import { useState, useEffect } from “react”; // Added useEffect

import { Input } from “@/components/ui/input”;

import { Button } from “@/components/ui/button”;

import { Textarea } from “@/components/ui/textarea”;

import { Card, CardContent, CardHeader, CardTitle } from “@/components/ui/card”; // Added more Card parts

import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from “@/components/ui/select”; // Added more Select parts

import { UploadCloud, Send, FilePlus2, Clock, Eye, MessageCircle, CheckCircle, AlertCircle } from “lucide-react”; // Added more icons

import { motion } from “framer-motion”;

import imageCompression from “browser-image-compression”;

// --- Mock API Call Functions ---

// These would be replaced by actual API calls to your backend

const apiUpdateTransactionPhase = async (transactionId: string, phase: string) => {

console.log(`API: Updating transaction ${transactionId} phase to: ${phase}`);

// Simulating API call

await new Promise(resolve => setTimeout(resolve, 500));

// On success: return { success: true };

// On failure: return { success: false, error: "Failed to update phase" };

return { success: true };

};

const apiFetchTransactionDetails = async (transactionId: string) => {

console.log(`API: Fetching transaction ${transactionId} details`);

await new Promise(resolve => setTimeout(resolve, 500));

// Return mock data including messages, requests, current phase etc.

return {

id: transactionId,

title: "123 Main St Short Sale",

propertyAddress: "123 Main St, Anytown, USA",

currentPhase: "File Intake", // Fetched from backend

parties: ROLES, // Assuming ROLES is also part of fetched data

messages: [

{ id: "seed", text: "Transaction initiated. Welcome!", sender: "System", timestamp: new Date().toISOString(), replies: [] }

],

documentRequests: [

{ id: "req1", docType: "Paystubs (30 Days)", assignedTo: "Seller", status: "pending", dueDate: "2025-06-01", revisionNote: "" },

{ id: "req2", docType: "Bank Statements (2 Months)", assignedTo: "Seller", status: "overdue", dueDate: "2025-05-28", revisionNote: "" }

]

};

};

const apiPostMessage = async (transactionId: string, message: object) => {

console.log(`API: Posting message to transaction ${transactionId}:`, message);

await new Promise(resolve => setTimeout(resolve, 500));

return { success: true, data: message }; // Return the posted message, possibly with server-generated ID/timestamp

};

const apiUploadFile = async (transactionId: string, file: File, docType: string, visibility: string) => {

console.log(`API: Uploading file for transaction ${transactionId}: ${file.name}, Type: ${docType}, Visibility: ${visibility}`);

const formData = new FormData();

formData.append("file", file);

formData.append("docType", docType);

formData.append("visibility", visibility);

// Actual fetch call to Supabase Storage or your backend

await new Promise(resolve => setTimeout(resolve, 1000));

return { success: true, fileUrl: `https://fakecloud.com/${file.name}` };

};

const apiTriggerDocumentRequest = async (transactionId: string, requestDetails: object) => {

console.log(`API: Triggering document request for transaction ${transactionId}:`, requestDetails);

await new Promise(resolve => setTimeout(resolve, 500));

return { success: true, data: requestDetails };

};

const apiUpdateDocumentRequestStatus = async (requestId: string, status: string, note?: string) => {

console.log(`API: Updating doc request ${requestId} to status ${status} with note: ${note || 'N/A'}`);

await new Promise(resolve => setTimeout(resolve, 500));

return { success: true };

}

// --- Mock Auth Context ---

// In a real app, this would come from your authentication library (e.g., Supabase Auth)

const useAuth = () => {

return {

user: {

id: "user-negotiator-123",

role: "negotiator", // Can be 'negotiator', 'seller', 'buyer', etc.

name: "Negotiator Nick"

},

isLoading: false,

};

};

// --- Defined Constants (as per original scaffold) ---

const PHASES = [

“Transaction Initiated”,

“Property Listed”,

“Initial Document Collection”,

“Offer Received”,

“Offer Submitted”,

“Lender Review”,

“BPO Ordered”,

“Approval Received”,

“In Closing”

]; // Updated to match Figma Spec

const ROLES = [ // This would typically be fetched with transaction data

{ id: "party1", name: “Seller Sam”, role: "Seller", status: “⚠️ Pending”, lastAction: “Viewed update 2h ago” },

{ id: "party2", name: “Buyer Bob”, role: "Buyer", status: “✅ Complete”, lastAction: “Uploaded doc yesterday” },

// ... other roles

];

const REQUEST\_TEMPLATES = [

“Paystubs (30 Days)”,

“Bank Statements (2 Months)”,

“4506-C Form”,

“Short Sale Affidavit”,

];

// --- Mock Components (for illustrative purposes) ---

// These would be your actual shadcn/ui based components defined in Component Spec

const PhaseTracker = ({ currentPhase, creationDate }: { currentPhase: string, creationDate?: Date }) => (

<Card className="mb-4"><CardContent><p className="p-4">Phase Tracker: <strong>{currentPhase}</strong> {creationDate ? `(Since: ${creationDate.toLocaleDateString()})` : ''}</p></CardContent></Card>

);

const PartyCard = ({ roleInfo }: { roleInfo: typeof ROLES[0] }) => (

<Card className="mb-2"><CardContent className="p-3"><strong>{roleInfo.name}</strong> ({roleInfo.role}) - Status: {roleInfo.status}</CardContent></Card>

);

const MessageThread = ({ messages, onSendMessage, onReply, currentUserRole }: { messages: any[], onSendMessage: (text: string) => void, onReply: (id: string, text: string) => void, currentUserRole: string }) => {

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

const canPostNew = currentUserRole === 'negotiator';

return (

<Card className="mb-4">

<CardHeader><CardTitle>Messages</CardTitle></CardHeader>

<CardContent>

{messages.map(msg => <div key={msg.id} className="mb-2 p-2 border rounded"><strong>{msg.sender || 'User'}</strong>: {msg.text} <span className="text-xs text-gray-500">{new Date(msg.timestamp).toLocaleTimeString()}</span></div>)}

{canPostNew && (

<div className="mt-2 flex gap-2">

<Input value={newMessage} onChange={(e) => setNewMessage(e.target.value)} placeholder="Type your message..." />

<Button onClick={() => { onSendMessage(newMessage); setNewMessage(""); }}><Send size={18} /></Button>

</div>

)}

{/\* Reply UI would be more complex, integrated with each message \*/}

</CardContent>

</Card>

);

};

const DocRequestList = ({ requests, onMakeRequest, onResetToPending, currentUserRole }: { requests: any[], onMakeRequest: (template: string) => void, onResetToPending: (id:string, note: string) => void, currentUserRole: string}) => {

const [revisionNote, setRevisionNote] = useState("");

const [selectedRequestForNote, setSelectedRequestForNote] = useState<string | null>(null);

return (

<Card className="mb-4">

<CardHeader><CardTitle>Document Requests</CardTitle></CardHeader>

<CardContent>

{requests.map(req => (

<div key={req.id} className="mb-2 p-2 border rounded">

<p><strong>{req.docType}</strong> for {req.assignedTo} - Status: {req.status}</p>

{req.dueDate && <p className="text-sm">Due: {req.dueDate}</p>}

{req.revisionNote && req.status === 'pending' && <p className="text-sm text-orange-600">Note: {req.revisionNote}</p>}

{currentUserRole === 'negotiator' && req.status !== 'complete' && (

<>

<Button variant="outline" size="sm" onClick={() => setSelectedRequestForNote(req.id)}>Reset to Pending w/ Note</Button>

{selectedRequestForNote === req.id && (

<div className="mt-1">

<Textarea placeholder="Reason for revision..." value={revisionNote} onChange={e => setRevisionNote(e.target.value)} />

<Button size="sm" onClick={() => { onResetToPending(req.id, revisionNote); setRevisionNote(""); setSelectedRequestForNote(null); }}>Save Note & Reset</Button>

</div>

)}

</>

)}

</div>

))}

{currentUserRole === 'negotiator' && (

<div className="mt-2">

<Select onValueChange={onMakeRequest}>

<SelectTrigger><SelectValue placeholder="Request a new document..." /></SelectTrigger>

<SelectContent>

{REQUEST\_TEMPLATES.map(template => <SelectItem key={template} value={template}>{template}</SelectItem>)}

</SelectContent>

</Select>

</div>

)}

</CardContent>

</Card>

);

};

const UploadWidget = ({ onFileUpload, transactionId }: { onFileUpload: (file: File, docType: string, visibility: 'private' | 'shared') => void, transactionId: string }) => {

const [selectedFile, setSelectedFile] = useState<File | null>(null);

const [docType, setDocType] = useState(REQUEST\_TEMPLATES[0]); // Default or let user pick

const [visibility, setVisibility] = useState<'private' | 'shared'>('private');

// Internal state for retry UI, error messages, etc. would be within the actual UploadWidget component

return (

<Card className="mb-4">

<CardHeader><CardTitle>Upload Document</CardTitle></CardHeader>

<CardContent>

<Input type="file" onChange={(e) => setSelectedFile(e.target.files ? e.target.files[0] : null)} className="mb-2"/>

{/\* Simplified: In reality, docType might be linked to a specific request \*/}

<Select value={docType} onValueChange={setDocType} >

<SelectTrigger><SelectValue placeholder="Select document type" /></SelectTrigger>

<SelectContent>{REQUEST\_TEMPLATES.map(dt => <SelectItem key={dt} value={dt}>{dt}</SelectItem>)}</SelectContent>

</Select>

<Select value={visibility} onValueChange={(v: 'private' | 'shared') => setVisibility(v)} >

<SelectTrigger><SelectValue placeholder="Select visibility" /></SelectTrigger>

<SelectContent>

<SelectItem value="private">Private (Negotiator & Uploader)</SelectItem>

<SelectItem value="shared">Shared (All Parties)</SelectItem>

</SelectContent>

</Select>

<Button onClick={() => selectedFile && onFileUpload(selectedFile, docType, visibility)} disabled={!selectedFile} className="mt-2 w-full">

<UploadCloud size={18} className="mr-2"/> Upload

</Button>

<p className="text-xs text-gray-500 mt-1">Uploads are private to you and the negotiator by default.</p>

{/\* Actual UploadWidget would have its own progress, error, retry UI \*/}

</CardContent>

</Card>

);

};

// --- Main TransactionView Component ---

export default function TransactionView() {

// Assuming transactionId is available (e.g., from URL params)

const transactionId = "txn-abc-123"; // Placeholder

const { user, isLoading: authLoading } = useAuth();

const isNegotiator = user?.role === 'negotiator';

const [phase, setPhase] = useState(“File Intake”);

const [transactionTitle, setTransactionTitle] = useState("");

const [partyData, setPartyData] = useState<typeof ROLES>([]);

const [messages, setMessages] = useState<any[]>([]);

const [documentRequests, setDocumentRequests] = useState<any[]>([]);

const [uploading, setUploading] = useState(false);

const [error, setError] = useState<string | null>(null); // For general errors

// Fetch initial transaction data

useEffect(() => {

if (transactionId) {

apiFetchTransactionDetails(transactionId).then(data => {

setTransactionTitle(data.title);

setPhase(data.currentPhase);

setPartyData(data.parties);

setMessages(data.messages);

setDocumentRequests(data.documentRequests);

}).catch(err => setError("Failed to load transaction details."));

}

}, [transactionId]);

const handlePhaseChange = async (newPhase: string) => {

setPhase(newPhase); // Optimistic update

const result = await apiUpdateTransactionPhase(transactionId, newPhase);

if (!result.success) {

setError("Failed to update phase. Please try again.");

// Optionally revert phase: await apiFetchTransactionDetails... to get actual current phase

}

};

const handleSendMessage = async (text: string) => {

if (!text.trim() || !isNegotiator) return; // Only negotiators can start new top-level messages

const newMessageObj = {

id: `msg-${Date.now()}`, // Client-side temp ID

text: text,

senderId: user.id,

senderName: user.name, // Assuming user object has name

timestamp: new Date().toISOString(),

replies: [],

};

setMessages(prev => [newMessageObj, ...prev]); // Optimistic update

const result = await apiPostMessage(transactionId, newMessageObj);

if (!result.success) {

setError("Failed to send message.");

setMessages(prev => prev.filter(m => m.id !== newMessageObj.id)); // Revert

} else {

// Optionally update message with server-generated ID/timestamp

// setMessages(prev => prev.map(m => m.id === newMessageObj.id ? result.data : m));

}

};

const handleReply = (messageId: string, replyText: string) => {

// Similar to handleSendMessage but with replyTo: messageId

// API call would associate reply with parent message

console.log(`Replying to ${messageId}: ${replyText}`);

// Update local state optimistically and then call API

};

const handleFileUpload = async (file: File, docType: string, visibility: 'private' | 'shared') => {

if (!file) return;

setUploading(true);

setError(null);

try {

const compressedFile = await imageCompression(file, {

maxSizeMB: 1, // As per spec

maxWidthOrHeight: 1920,

useWebWorker: true,

});

const result = await apiUploadFile(transactionId, compressedFile, docType, visibility);

if (result.success) {

// Call onUploadComplete logic from props if this were a self-contained widget

// e.g., refresh document list or show success message

console.log("Upload successful, URL:", result.fileUrl);

// Potentially update a list of uploaded documents here

} else {

setError("Upload failed. The UploadWidget would show specific retry options.");

}

} catch (uploadError: any) {

console.error(“Upload processing failed”, uploadError);

setError(uploadError.message || “File processing error occurred.”);

}

setUploading(false);

};

const triggerDocumentRequest = async (template: string) => {

if (!isNegotiator || !template) return;

// For MVP, assign to a placeholder or require selection

const assignedToUserId = ROLES.find(r => r.role === 'Seller')?.id || 'seller-placeholder-id'; // Example

const requestDetails = {

docType: template,

assignedTo: assignedToUserId, // In a real app, negotiator would select party

status: 'requested',

// dueDate: ... // Optionally set due date

};

// Optimistically update UI or wait for API response

const result = await apiTriggerDocumentRequest(transactionId, requestDetails);

if (result.success) {

// Add to local 'documentRequests' state or refetch

// setDocumentRequests(prev => [...prev, result.data]);

sendMultiChannelNotification(template, "A new document has been requested.");

} else {

setError("Failed to send document request.");

}

};

const handleResetToPending = async (requestId: string, note: string) => {

if (!isNegotiator) return;

const result = await apiUpdateDocumentRequestStatus(requestId, 'pending', note);

if (result.success) {

setDocumentRequests(prevReqs => prevReqs.map(r => r.id === requestId ? {...r, status: 'pending', revisionNote: note} : r));

sendMultiChannelNotification(requestId, `Document request updated: ${note}`);

} else {

setError("Failed to reset document request status.");

}

};

// Updated to accept a generic message or specific details like a note

const sendMultiChannelNotification = (itemIdentifier: string, message: string, note?: string) => {

// MVP placeholder - backend should handle actual dispatch via email, SMS, etc.

console.log(`Notification Triggered for item ${itemIdentifier}: ${message} ${note ? `(Note: ${note})` : ''}`);

// In real app: await apiSendNotification({ transactionId, itemIdentifier, message, note });

};

if (authLoading) {

return <div>Loading authentication details...</div>;

}

if (!user) {

// This case should be handled by a higher-order component or router redirecting to login

return <div>Not authenticated. Redirecting...</div>;

}

if (error) {

// Simple global error display for now

return <div className="p-4 text-red-600 bg-red-100 border border-red-600 rounded">Error: {error} <Button onClick={() => setError(null)}>Dismiss</Button></div>;

}

return (

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

<header className="mb-6">

<h1 className="text-2xl font-bold">{transactionTitle || "Transaction Details"}</h1>

<p className="text-sm text-gray-600">{transactionId}</p>

</header>

{isNegotiator && (

<div className="mb-4">

<label htmlFor="phase-select" className="block text-sm font-medium text-gray-700 mb-1">Current Phase:</label>

<Select value={phase} onValueChange={handlePhaseChange}>

<SelectTrigger id="phase-select">

<SelectValue placeholder="Select phase" />

</SelectTrigger>

<SelectContent>

{PHASES.map((p) => <SelectItem key={p} value={p}>{p}</SelectItem>)}

</SelectContent>

</Select>

</div>

)}

<PhaseTracker currentPhase={phase} />

<div className="grid md:grid-cols-2 gap-4 mb-4">

<div>

<Card>

<CardHeader><CardTitle>Parties Involved</CardTitle></CardHeader>

<CardContent>

{partyData.map(p => <PartyCard key={p.id} roleInfo={p} />)}

</CardContent>

</Card>

</div>

<div>

{/\* This is a simplified UploadWidget call.

The actual UploadWidget component would handle its internal state like selected file,

upload progress, error messages, and retry UI.

onFileUpload is a callback for when the widget has a file ready for actual upload processing.

\*/}

<UploadWidget onFileUpload={handleFileUpload} transactionId={transactionId} />

</div>

</div>

<DocRequestList

requests={documentRequests}

onMakeRequest={triggerDocumentRequest}

onResetToPending={handleResetToPending}

currentUserRole={user.role}

/>

<MessageThread

messages={messages}

onSendMessage={handleSendMessage}

onReply={handleReply}

currentUserRole={user.role}

/>

{uploading && (

<div className="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center">

<div className="bg-white p-4 rounded shadow-lg">Uploading, please wait...</div>

</div>

)}

</div>

);

}