InteractiveReadingSurface.tsx

"use client";

import { PersonaLearnerAgent } from "@/agents/PersonaLearner";

import {

ConversationState,

ConversationTurn,

SophiaAgent,

SophiaReading,

} from "@/agents/sophia";

import { useAuth } from "@/contexts/AuthContext";

import { TarotCard } from "@/types/tarot";

import { AnimatePresence, motion, useAnimation } from "framer-motion";

import {

BookOpen,

Eye,

EyeOff,

RefreshCw,

Save,

Shuffle,

Sparkles,

} from "lucide-react";

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

import { EnhancedShuffleAnimation } from "./EnhancedShuffleAnimation";

import {

EnhancedTarotSpreadLayouts,

SpreadType,

} from "./EnhancedTarotSpreadLayouts";

interface ReadingSession {

id: string;

spreadType: SpreadType;

cards: TarotCard[];

timestamp: Date;

interpretation?: string;

journalEntry?: string;

isGuest: boolean;

sophiaReading?: SophiaReading;

}

interface InteractiveReadingSurfaceProps {

selectedSpread: SpreadType;

onReadingComplete?: (session: ReadingSession) => void;

onBackToSelection?: () => void;

className?: string;

}

export const InteractiveReadingSurface: React.FC<InteractiveReadingSurfaceProps> = ({

selectedSpread,

onReadingComplete,

onBackToSelection,

className = "",

}) => {

// State management - Enhanced for conversational flow

const { user } = useAuth();

const isAuthenticated = !!user;

const [phase, setPhase] = useState<

| "preparation"

| "shuffling"

| "drawing"

| "revealing"

| "conversation"

| "interpreting"

| "complete"

>("preparation");

const [drawnCards, setDrawnCards] = useState<TarotCard[]>([]);

const [revealedCards, setRevealedCards] = useState<Set<number>>(new Set());

const [currentSession, setCurrentSession] = useState<ReadingSession | null>(

null

);

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

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

const [showSaveModal, setShowSaveModal] = useState(false);

// Additional state for legacy Sophia reading display

const [sophiaReading, setSophiaReading] = useState<SophiaReading | null>(

null

);

const [isGeneratingReading, setIsGeneratingReading] = useState(false);

const [showAllMeanings, setShowAllMeanings] = useState(false);

const [cardInterpretations, setCardInterpretations] = useState<

Record<number, string>

>({});

// Conversational state

const [conversationState, setConversationState] = useState<ConversationState>(

ConversationState.AWAITING\_DRAW

);

const [conversationHistory, setConversationHistory] = useState<

ConversationTurn[]

>([]);

const [currentTurn, setCurrentTurn] = useState<ConversationTurn | null>(null);

const [isProcessingTurn, setIsProcessingTurn] = useState(false);

const [finalReading, setFinalReading] = useState<SophiaReading | null>(null);

// Progressive Reveal System state

const [engagementLevelUp, setEngagementLevelUp] = useState<{

show: boolean;

newLevel: number;

thresholdMet: string;

} | null>(null);

// Agent instances

const sophiaAgent = new SophiaAgent();

const personaLearner = new PersonaLearnerAgent();

// Refs and animations

const surfaceRef = useRef<HTMLDivElement>(null);

const shuffleControls = useAnimation();

// Spread configuration mapping

const spreadRequirements = {

single: 1,

"three-card": 3,

"celtic-cross": 10,

horseshoe: 5,

relationship: 5,

custom: 7,

};

// Initialize reading session

useEffect(() => {

const sessionId = `reading\_${Date.now()}\_${Math.random()

.toString(36)

.substr(2, 9)}`;

const newSession: ReadingSession = {

id: sessionId,

spreadType: selectedSpread,

cards: [],

timestamp: new Date(),

isGuest: !isAuthenticated,

};

setCurrentSession(newSession);

}, [selectedSpread, isAuthenticated]);

// API call to draw cards

const drawCardsFromAPI = useCallback(

async (spreadType: SpreadType) => {

setIsLoading(true);

setError(null);

try {

const response = await fetch("/api/tarot/draw", {

method: "POST",

headers: { "Content-Type": "application/json" },

body: JSON.stringify({

spread: spreadType,

count: spreadRequirements[spreadType],

allowReversed: true,

userId: user?.id || null,

}),

});

const data = await response.json();

if (!data.success) {

throw new Error(data.error || "Failed to draw cards");

}

return data.data.cards;

} catch (err) {

const errorMessage =

err instanceof Error

? err.message

: "Failed to connect to cosmic energies";

setError(errorMessage);

throw err;

} finally {

setIsLoading(false);

}

},

[user?.id]

);

// Handle shuffle completion and card drawing

const handleShuffleComplete = useCallback(async () => {

try {

setPhase("drawing");

const cards = await drawCardsFromAPI(selectedSpread);

setDrawnCards(cards);

setPhase("revealing");

// Update session

if (currentSession) {

const updatedSession = { ...currentSession, cards };

setCurrentSession(updatedSession);

}

// Auto-start revealing after a brief pause

setTimeout(() => {

setRevealedCards(new Set([0])); // Reveal first card

}, 800);

} catch (err) {

setPhase("preparation");

console.error("Failed to draw cards:", err);

}

}, [selectedSpread, drawCardsFromAPI, currentSession]);

// Handle individual card reveal

const handleCardReveal = useCallback(

(card: TarotCard, index: number) => {

setRevealedCards((prev) => new Set([...prev, index]));

// Generate basic interpretation for the card

const interpretation = generateCardInterpretation(

card,

index,

selectedSpread

);

setCardInterpretations((prev) => ({ ...prev, [index]: interpretation }));

// Haptic feedback on mobile

if ("vibrate" in navigator) {

navigator.vibrate(75);

}

// Check if all cards are revealed

const totalCards = spreadRequirements[selectedSpread];

if (revealedCards.size + 1 >= totalCards) {

setTimeout(() => setPhase("interpreting"), 1000);

}

},

[revealedCards.size, selectedSpread]

);

// Generate contextual card interpretation

const generateCardInterpretation = (

card: TarotCard,

position: number,

spread: SpreadType

): string => {

const positionMeanings = {

single: [

"This card represents the universe's guidance for you right now.",

],

"three-card": [

"This card reveals the foundations and influences from your past.",

"This card illuminates your current situation and energy.",

"This card shows the potential path ahead and future possibilities.",

],

"celtic-cross": [

"Your present situation and current focus.",

"The challenge or opportunity crossing your path.",

"Distant past influences and foundations.",

"Recent past events affecting you now.",

"Possible outcomes if current path continues.",

"Immediate future and next steps.",

"Your approach and inner resources.",

"External influences and environment.",

"Your hopes, fears, and subconscious.",

"The ultimate outcome and resolution.",

],

};

const positionContext =

(positionMeanings as any)[spread]?.[position] ||

`Position ${position + 1} guidance:`;

const meaning = card.isReversed

? card.meaning.reversed

: card.meaning.upright;

return `${positionContext} ${meaning}`;

};

// Conversational Flow Methods

/\*\*

\* Start the conversational reading flow

\*/

const startConversation = useCallback(async () => {

if (!currentSession || drawnCards.length === 0) return;

try {

setIsProcessingTurn(true);

setPhase("conversation");

const context = {

userId: user?.id,

spreadType: selectedSpread,

sessionId: currentSession.id,

timestamp: new Date(),

cards: drawnCards,

};

const turn = await sophiaAgent.processReadingTurn(

currentSession.id,

ConversationState.AWAITING\_DRAW,

undefined,

drawnCards,

context

);

setCurrentTurn(turn);

setConversationState(turn.newState);

setConversationHistory([turn]);

} catch (error) {

console.error("Failed to start conversation:", error);

setError("Unable to begin the reading conversation. Please try again.");

} finally {

setIsProcessingTurn(false);

}

}, [currentSession, drawnCards, selectedSpread, user?.id, sophiaAgent]);

/\*\*

\* Process user input and advance conversation

\*/

const handleUserInput = useCallback(

async (userInput: string) => {

if (!currentSession || isProcessingTurn) return;

const responseStartTime = Date.now();

try {

setIsProcessingTurn(true);

// Log user response for learning (authenticated users only)

if (user?.id && currentTurn?.options) {

await personaLearner.logUserResponse(

user.id,

currentSession.id,

userInput,

{

options\_presented: currentTurn.options.map((opt) => opt.text),

response\_time\_ms: Date.now() - responseStartTime,

conversation\_state: conversationState,

}

);

}

const turn = await sophiaAgent.processReadingTurn(

currentSession.id,

conversationState,

userInput

);

setCurrentTurn(turn);

setConversationState(turn.newState);

setConversationHistory((prev) => [...prev, turn]);

// Log conversation turn for learning (authenticated users only)

if (user?.id) {

await personaLearner.logConversationTurn(

user.id,

currentSession.id,

turn.newState,

conversationHistory.length + 1,

turn.sophiaDialogue || "",

userInput,

turn.revealedCard

);

}

// If we have a final reading, update phase

if (turn.finalReading) {

setFinalReading(turn.finalReading);

setPhase("complete");

}

// Update revealed cards if a card was revealed in this turn

if (turn.revealedCard) {

const cardIndex = drawnCards.findIndex(

(card) => card.name === turn.revealedCard!.card.name

);

if (cardIndex !== -1) {

setRevealedCards((prev) => new Set([...prev, cardIndex]));

// Log card reveal engagement (authenticated users only)

if (user?.id) {

await personaLearner.logCardReveal(

user.id,

currentSession.id,

turn.revealedCard.card.name,

cardIndex,

{

clicked: true,

interpretation\_read: true,

}

);

}

}

}

} catch (error) {

console.error("Failed to process conversation turn:", error);

setError("Unable to process your response. Please try again.");

} finally {

setIsProcessingTurn(false);

}

},

[

currentSession,

conversationState,

isProcessingTurn,

sophiaAgent,

drawnCards,

user,

currentTurn,

personaLearner,

conversationHistory.length,

]

);

/\*\*

\* Handle saving the conversational reading with engagement level check

\*/

const handleSaveConversationalReading = useCallback(async () => {

if (!finalReading || !user?.id) return;

try {

const response = await fetch("/api/tarot/save-reading", {

method: "POST",

headers: {

"Content-Type": "application/json",

Authorization: `Bearer ${

await(user as any).getAccessToken?.() || ""

}`,

},

body: JSON.stringify({

userId: user.id,

spreadType: selectedSpread,

cards: drawnCards.map((card) => ({

id: card.id,

name: card.name,

position: drawnCards.indexOf(card).toString(),

isReversed: card.isReversed,

meaning: card.isReversed

? card.meaning.reversed

: card.meaning.upright,

})),

interpretation: finalReading.overall\_guidance,

question: "",

notes: "Conversational reading with Sophia",

cosmicInfluence: finalReading.spiritual\_insight,

drawId: currentSession?.id,

isPublic: false,

tags: ["sophia", "conversational", selectedSpread],

}),

});

if (response.ok) {

setShowSaveModal(false);

// Check and increment engagement level after successful save

await checkEngagementLevel();

// Optionally show success message

} else {

throw new Error("Failed to save reading");

}

} catch (error) {

console.error("Failed to save conversational reading:", error);

setError("Unable to save your reading. Please try again.");

}

}, [finalReading, user, selectedSpread, drawnCards, currentSession]);

/\*\*

\* Check and increment user engagement level (Progressive Reveal System)

\*/

const checkEngagementLevel = useCallback(async () => {

if (!user?.id) return; // Skip for guest users

try {

console.log(

"InteractiveReadingSurface: Checking engagement level after reading completion"

);

const result = await personaLearner.checkAndIncrementEngagementLevel(

user.id

);

if (result.levelIncreased) {

console.log(

`InteractiveReadingSurface: User leveled up! ${result.previousLevel} → ${result.newLevel} (${result.thresholdMet})`

);

// Show level up notification

setEngagementLevelUp({

show: true,

newLevel: result.newLevel,

thresholdMet: result.thresholdMet || "Level Up!",

});

// Auto-hide notification after 5 seconds

setTimeout(() => {

setEngagementLevelUp(null);

}, 5000);

} else {

console.log(

`InteractiveReadingSurface: No level increase. Current level: ${result.newLevel}`

);

}

} catch (error) {

console.error(

"InteractiveReadingSurface: Failed to check engagement level:",

error

);

}

}, [personaLearner, user?.id]);

// Auto-start conversation when cards are drawn

useEffect(() => {

if (

phase === "drawing" &&

drawnCards.length === spreadRequirements[selectedSpread]

) {

// Wait a moment for cards to be displayed, then start conversation

const timer = setTimeout(() => {

startConversation();

}, 1500);

return () => clearTimeout(timer);

}

}, [phase, drawnCards.length, selectedSpread, startConversation]);

// Handle save reading with PersonaLearner integration

const handleSaveReading = useCallback(

async (journalEntry?: string, userFeedback?: any) => {

if (!currentSession) return;

const sessionWithJournal = {

...currentSession,

journalEntry,

interpretation: sophiaReading

? `${sophiaReading.narrative}\n\n${sophiaReading.overall\_guidance}\n\n${sophiaReading.spiritual\_insight}`

: Object.values(cardInterpretations).join("\n\n"),

cards: drawnCards,

sophiaReading,

};

// Log interaction with PersonaLearner for learning (authenticated users only)

if (sophiaReading && user?.id) {

try {

await personaLearner.logInteraction(

user.id,

sophiaReading as any,

userFeedback

);

console.log("PersonaLearner: Interaction logged successfully");

} catch (error) {

console.warn("PersonaLearner: Failed to log interaction:", error);

// Continue with save even if logging fails

}

} else if (!user?.id) {

console.log(

"PersonaLearner: Skipping interaction logging for guest user"

);

}

// Save to localStorage for guest users or API for authenticated users

if (isAuthenticated && user) {

try {

// TODO: Implement API call to save reading

console.log("Saving to user account:", sessionWithJournal);

// Check engagement level after successful save

await checkEngagementLevel();

} catch (err) {

console.error("Failed to save reading:", err);

}

} else {

// Save to localStorage for guest users

const guestReadings = JSON.parse(

localStorage.getItem("guestReadings") || "[]"

);

guestReadings.push(sessionWithJournal);

localStorage.setItem(

"guestReadings",

JSON.stringify(guestReadings.slice(-10))

); // Keep last 10

}

onReadingComplete?.({

...sessionWithJournal,

sophiaReading: sophiaReading || undefined,

});

setShowSaveModal(false);

},

[

currentSession,

cardInterpretations,

drawnCards,

isAuthenticated,

user,

onReadingComplete,

sophiaReading,

personaLearner,

]

);

// Auto-reveal cards in sequence and generate Sophia reading

useEffect(() => {

if (phase === "revealing" && drawnCards.length > 0) {

const totalCards = drawnCards.length;

let currentCard = 0;

const revealInterval = setInterval(() => {

if (currentCard < totalCards) {

setRevealedCards((prev) => new Set([...prev, currentCard]));

currentCard++;

} else {

clearInterval(revealInterval);

setTimeout(async () => {

setPhase("interpreting");

await generateSophiaReading();

}, 1000);

}

}, 1200);

return () => clearInterval(revealInterval);

}

}, [phase, drawnCards.length]);

// Generate Sophia reading from Knowledge Pool

const generateSophiaReading = async () => {

if (!drawnCards.length || !currentSession) return;

setIsGeneratingReading(true);

try {

const readingContext = {

userId: user?.id,

spreadType: selectedSpread,

sessionId: currentSession.id,

timestamp: new Date(),

cards: drawnCards,

astrological\_context: {

// TODO: Add astrological context if available

},

};

const reading = await sophiaAgent.getReading(

drawnCards,

selectedSpread,

readingContext

);

setSophiaReading(reading);

// Update session with Sophia reading

if (currentSession) {

const updatedSession = {

...currentSession,

sophiaReading: reading,

interpretation: reading.narrative + "\n\n" + reading.overall\_guidance,

};

setCurrentSession(updatedSession);

}

} catch (error) {

console.error("Failed to generate Sophia reading:", error);

// Fallback to basic interpretations if Sophia reading fails

const fallbackInterpretations: Record<number, string> = {};

drawnCards.forEach((card, index) => {

fallbackInterpretations[index] = generateCardInterpretation(

card,

index,

selectedSpread

);

});

setCardInterpretations(fallbackInterpretations);

} finally {

setIsGeneratingReading(false);

}

};

// Phase-specific renders

const renderPreparation = () => (

<motion.div

className="text-center space-y-8"

initial={{ opacity: 0, y: 20 }}

animate={{ opacity: 1, y: 0 }}

exit={{ opacity: 0, y: -20 }}

>

<div className="space-y-4">

<motion.h2

className="text-3xl md:text-4xl font-bold bg-gradient-to-r from-purple-400 via-pink-400 to-cyan-400 bg-clip-text text-transparent"

animate={{ backgroundPosition: ["0%", "100%", "0%"] }}

transition={{ duration: 3, repeat: Infinity }}

>

Prepare Your Mind

</motion.h2>

<p className="text-purple-300 text-lg max-w-2xl mx-auto">

Take a moment to center yourself and focus on your question. The cards

will reveal what you need to know.

</p>

</div>

{/\* Meditation Timer \*/}

<motion.div

className="bg-purple-900/30 backdrop-blur-sm rounded-2xl p-6 max-w-md mx-auto"

initial={{ scale: 0.9, opacity: 0 }}

animate={{ scale: 1, opacity: 1 }}

transition={{ delay: 0.5 }}

>

<h3 className="text-purple-300 font-semibold mb-4">

{selectedSpread.replace("-", " ").toUpperCase()} SPREAD

</h3>

<p className="text-purple-400 text-sm mb-6">

{spreadRequirements[selectedSpread]} cards will be drawn to guide your

path

</p>

<motion.button

className="bg-gradient-to-r from-purple-600 to-pink-600 text-white px-8 py-4 rounded-xl font-semibold shadow-2xl shadow-purple-500/30 w-full flex items-center justify-center space-x-3"

onClick={() => setPhase("shuffling")}

whileHover={{ scale: 1.02, y: -2 }}

whileTap={{ scale: 0.98 }}

>

<Shuffle className="w-5 h-5" />

<span>Begin Shuffling</span>

<Sparkles className="w-5 h-5" />

</motion.button>

</motion.div>

</motion.div>

);

const renderShuffling = () => (

<motion.div

className="text-center space-y-8"

initial={{ opacity: 0, scale: 0.9 }}

animate={{ opacity: 1, scale: 1 }}

exit={{ opacity: 0, scale: 0.9 }}

>

<div className="space-y-4">

<motion.h2

className="text-2xl md:text-3xl font-semibold text-purple-300"

animate={{ scale: [1, 1.02, 1] }}

transition={{ duration: 2, repeat: Infinity }}

>

Shuffling Cosmic Energies...

</motion.h2>

<p className="text-purple-400">

The universe is aligning the perfect cards for your reading

</p>

</div>

<EnhancedShuffleAnimation

isShuffling={true}

onShuffleStart={() => {}}

onShuffleComplete={handleShuffleComplete}

size="large"

cardCount={78}

showCardPreview={true}

/>

</motion.div>

);

const renderReading = () => (

<div className="w-full">

{/\* Header \*/}

<motion.div

className="text-center mb-8"

initial={{ opacity: 0, y: -20 }}

animate={{ opacity: 1, y: 0 }}

>

<h2 className="text-2xl md:text-3xl font-bold text-white mb-2">

Your {selectedSpread.replace("-", " ")} Reading

</h2>

<p className="text-purple-300">

{phase === "conversation"

? "In conversation with Sophia"

: `${revealedCards.size} of ${drawnCards.length} cards revealed`}

</p>

</motion.div>

{/\* Conversational Reading Display \*/}

{phase === "conversation" && currentTurn && (

<motion.div

className="mb-8 bg-purple-900/30 backdrop-blur-sm rounded-2xl p-6"

data-testid="conversation-phase"

initial={{ opacity: 0, y: 20 }}

animate={{ opacity: 1, y: 0 }}

transition={{ delay: 0.2 }}

>

<div className="flex items-center space-x-3 mb-4">

<div className="w-10 h-10 bg-gradient-to-br from-purple-500 to-pink-500 rounded-full flex items-center justify-center">

<Sparkles className="w-5 h-5 text-white" />

</div>

<div>

<h3 className="text-purple-300 font-semibold">Sophia</h3>

<p className="text-purple-400 text-sm">

{conversationState === ConversationState.AWAITING\_DRAW

? "Drawing cards..."

: conversationState === ConversationState.REVEALING\_CARD\_1

? "Revealing first card..."

: conversationState === ConversationState.REVEALING\_CARD\_2

? "Revealing second card..."

: conversationState === ConversationState.REVEALING\_CARD\_3

? "Revealing third card..."

: conversationState === ConversationState.CARD\_INTERPRETATION

? "Interpreting your cards..."

: conversationState === ConversationState.ASKING\_QUESTION

? "Learning about you..."

: conversationState ===

ConversationState.AWAITING\_USER\_RESPONSE

? "Waiting for your response..."

: conversationState === ConversationState.PROVIDING\_GUIDANCE

? "Sharing guidance..."

: conversationState === ConversationState.FINAL\_SYNTHESIS

? "Weaving your reading together..."

: "In conversation with you"}

</p>

</div>

</div>

{/\* Sophia's Dialogue \*/}

<div className="space-y-4 text-purple-200">

<div className="bg-purple-800/20 rounded-lg p-4">

<p className="leading-relaxed">{currentTurn.sophiaDialogue}</p>

</div>

{/\* Show revealed card info if present \*/}

{currentTurn.revealedCard && (

<motion.div

className="bg-gradient-to-r from-purple-800/30 to-pink-800/30 rounded-lg p-4 border border-purple-500/30"

initial={{ opacity: 0, scale: 0.95 }}

animate={{ opacity: 1, scale: 1 }}

transition={{ delay: 0.5 }}

>

<div className="flex items-center space-x-3 mb-2">

<div className="w-6 h-6 bg-gradient-to-br from-gold-400 to-yellow-500 rounded-full flex items-center justify-center">

<Sparkles className="w-3 h-3 text-black" />

</div>

<h4 className="text-purple-300 font-medium">

{currentTurn.revealedCard.card.name}

</h4>

</div>

<p className="text-purple-200 text-sm leading-relaxed">

{typeof currentTurn.revealedCard.interpretation === "string"

? currentTurn.revealedCard.interpretation

: currentTurn.revealedCard.interpretation

?.personalized\_guidance || ""}

</p>

</motion.div>

)}

{/\* Interactive Question Display \*/}

{currentTurn.interactiveQuestion && (

<motion.div

className="bg-cyan-900/20 rounded-lg p-4 border border-cyan-500/30"

initial={{ opacity: 0, y: 10 }}

animate={{ opacity: 1, y: 0 }}

transition={{ delay: 0.7 }}

>

<h4 className="text-cyan-300 font-medium mb-2">

{currentTurn.interactiveQuestion.question}

</h4>

<p className="text-cyan-200 text-sm">

{currentTurn.interactiveQuestion.context}

</p>

</motion.div>

)}

</div>

{/\* Interactive Options \*/}

{currentTurn.options &&

currentTurn.options.length > 0 &&

!isProcessingTurn && (

<motion.div

className="mt-6 space-y-3"

initial={{ opacity: 0, y: 20 }}

animate={{ opacity: 1, y: 0 }}

transition={{ delay: 0.9 }}

>

<h4 className="text-purple-300 font-medium">

Choose your response:

</h4>

<div className="grid gap-3">

{currentTurn.options.map((option, index) => (

<motion.button

key={index}

className="bg-purple-700/30 hover:bg-purple-600/40 border border-purple-500/50 text-purple-200 p-4 rounded-lg text-left transition-all duration-200 group"

onClick={() => handleUserInput(option.text)}

whileHover={{ scale: 1.02, y: -2 }}

whileTap={{ scale: 0.98 }}

initial={{ opacity: 0, x: -20 }}

animate={{ opacity: 1, x: 0 }}

transition={{ delay: 1 + index \* 0.1 }}

>

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

<span className="leading-relaxed">{option.text}</span>

<div className="text-purple-400 group-hover:text-purple-300 transition-colors">

→

</div>

</div>

{option.hint && (

<p className="text-purple-400 text-sm mt-2 opacity-75">

{option.hint}

</p>

)}

</motion.button>

))}

</div>

</motion.div>

)}

{/\* Processing State \*/}

{isProcessingTurn && (

<motion.div

className="mt-6 flex items-center justify-center space-x-3 p-4"

initial={{ opacity: 0 }}

animate={{ opacity: 1 }}

>

<motion.div

className="w-4 h-4 border-2 border-purple-500 border-t-transparent rounded-full"

animate={{ rotate: 360 }}

transition={{ duration: 1, repeat: Infinity, ease: "linear" }}

/>

<p className="text-purple-300 text-sm">

Sophia is reflecting on your response...

</p>

</motion.div>

)}

</motion.div>

)}

{/\* Sophia Reading Display \*/}

{sophiaReading && phase === "interpreting" && (

<motion.div

className="mb-8 bg-purple-900/30 backdrop-blur-sm rounded-2xl p-6"

initial={{ opacity: 0, y: 20 }}

animate={{ opacity: 1, y: 0 }}

transition={{ delay: 0.5 }}

>

<div className="flex items-center space-x-3 mb-4">

<div className="w-10 h-10 bg-gradient-to-br from-purple-500 to-pink-500 rounded-full flex items-center justify-center">

<Sparkles className="w-5 h-5 text-white" />

</div>

<div>

<h3 className="text-purple-300 font-semibold">Sophia's Wisdom</h3>

<p className="text-purple-400 text-sm">

Reading from the Knowledge Pool

</p>

</div>

</div>

<div className="space-y-4 text-purple-200">

<div className="bg-purple-800/20 rounded-lg p-4">

<h4 className="text-purple-300 font-medium mb-2">Your Reading</h4>

<p className="leading-relaxed">{sophiaReading.narrative}</p>

</div>

{showAllMeanings && (

<motion.div

initial={{ opacity: 0, height: 0 }}

animate={{ opacity: 1, height: "auto" }}

exit={{ opacity: 0, height: 0 }}

className="space-y-3"

>

<div className="bg-purple-800/20 rounded-lg p-4">

<h4 className="text-purple-300 font-medium mb-2">Guidance</h4>

<p className="leading-relaxed">

{sophiaReading.overall\_guidance}

</p>

</div>

<div className="bg-purple-800/20 rounded-lg p-4">

<h4 className="text-purple-300 font-medium mb-2">

Spiritual Insight

</h4>

<p className="leading-relaxed">

{sophiaReading.spiritual\_insight}

</p>

</div>

{sophiaReading.card\_interpretations.length > 0 && (

<div className="bg-purple-800/20 rounded-lg p-4">

<h4 className="text-purple-300 font-medium mb-3">

Individual Card Meanings

</h4>

<div className="space-y-3">

{sophiaReading.card\_interpretations.map(

(interp, index) => (

<div

key={index}

className="border-l-2 border-purple-500 pl-3"

>

<p className="text-sm text-purple-400 mb-1">

{drawnCards[index]?.name} -{" "}

{interp.confidence\_score > 0.8

? "High"

: "Standard"}{" "}

Confidence

</p>

<p className="text-purple-200 text-sm leading-relaxed">

{interp.personalized\_guidance}

</p>

</div>

)

)}

</div>

</div>

)}

</motion.div>

)}

<div className="text-right">

<p className="text-purple-400 text-sm italic">

{sophiaReading.reader\_signature}

</p>

</div>

</div>

</motion.div>

)}

{/\* Loading state for Sophia reading \*/}

{isGeneratingReading && (

<motion.div

className="mb-8 bg-purple-900/30 backdrop-blur-sm rounded-2xl p-6 text-center"

initial={{ opacity: 0, y: 20 }}

animate={{ opacity: 1, y: 0 }}

>

<div className="flex items-center justify-center space-x-3 mb-4">

<motion.div

className="w-6 h-6 border-2 border-purple-500 border-t-transparent rounded-full"

animate={{ rotate: 360 }}

transition={{ duration: 1, repeat: Infinity, ease: "linear" }}

/>

<p className="text-purple-300">

Sophia is channeling wisdom from the Knowledge Pool...

</p>

</div>

</motion.div>

)}

{/\* Spread Layout - Show during revealing/interpreting phases, hide during conversation \*/}

{phase !== "conversation" && (

<EnhancedTarotSpreadLayouts

spreadType={selectedSpread}

cards={drawnCards}

onCardClick={handleCardReveal}

isRevealing={phase === "revealing" || phase === "interpreting"}

showBioluminescence={true}

isMobile={false}

/>

)}

{/\* Conversation Phase Card Display \*/}

{phase === "conversation" && drawnCards.length > 0 && (

<motion.div

className="mb-8"

initial={{ opacity: 0, y: 20 }}

animate={{ opacity: 1, y: 0 }}

transition={{ delay: 0.3 }}

>

<div className="grid grid-cols-1 md:grid-cols-3 lg:grid-cols-5 gap-4 max-w-4xl mx-auto">

{drawnCards.map((card, index) => (

<motion.div

key={`conversation-card-${index}`}

className={`relative aspect-[2/3] rounded-lg overflow-hidden ${

revealedCards.has(index)

? "bg-gradient-to-br from-purple-800 to-pink-800"

: "bg-purple-900/50"

}`}

initial={{ opacity: 0, scale: 0.8 }}

animate={{

opacity: revealedCards.has(index) ? 1 : 0.3,

scale: revealedCards.has(index) ? 1 : 0.9,

}}

transition={{ delay: index \* 0.1 }}

>

{revealedCards.has(index) ? (

<div className="p-3 h-full flex flex-col">

<h4 className="text-purple-200 font-medium text-sm mb-2 text-center">

{card.name}

</h4>

<div className="flex-1 flex items-center justify-center">

<div className="w-16 h-16 bg-gradient-to-br from-gold-400 to-yellow-500 rounded-full flex items-center justify-center">

<Sparkles className="w-8 h-8 text-black" />

</div>

</div>

<p className="text-purple-300 text-xs text-center mt-2">

{card.arcana === "major"

? "Major Arcana"

: `${card.suit} - Minor`}

</p>

</div>

) : (

<div className="h-full flex items-center justify-center">

<div className="w-12 h-12 border-2 border-purple-400/30 rounded-lg"></div>

</div>

)}

</motion.div>

))}

</div>

</motion.div>

)}

{/\* Controls \*/}

{(phase === "interpreting" || phase === "complete") && (

<motion.div

className="fixed bottom-6 left-1/2 transform -translate-x-1/2 z-50"

initial={{ opacity: 0, y: 20 }}

animate={{ opacity: 1, y: 0 }}

transition={{ delay: 1 }}

>

<div className="bg-black/80 backdrop-blur-lg rounded-2xl p-4 flex items-center space-x-4 shadow-2xl">

{phase === "interpreting" && (

<motion.button

className="flex items-center space-x-2 px-4 py-2 bg-purple-600 hover:bg-purple-700 text-white rounded-lg transition-colors"

onClick={() => setShowAllMeanings(!showAllMeanings)}

whileHover={{ scale: 1.05 }}

whileTap={{ scale: 0.95 }}

>

{showAllMeanings ? (

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

) : (

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

)}

<span>{showAllMeanings ? "Hide" : "Show"} Meanings</span>

</motion.button>

)}

<motion.button

className="flex items-center space-x-2 px-4 py-2 bg-pink-600 hover:bg-pink-700 text-white rounded-lg transition-colors"

onClick={() => {

if (phase === "complete" && finalReading) {

handleSaveConversationalReading();

} else {

setShowSaveModal(true);

}

}}

whileHover={{ scale: 1.05 }}

whileTap={{ scale: 0.95 }}

>

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

<span>Save Reading</span>

</motion.button>

<motion.button

className="flex items-center space-x-2 px-4 py-2 bg-cyan-600 hover:bg-cyan-700 text-white rounded-lg transition-colors"

onClick={onBackToSelection}

whileHover={{ scale: 1.05 }}

whileTap={{ scale: 0.95 }}

>

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

<span>New Reading</span>

</motion.button>

</div>

</motion.div>

)}

</div>

);

const renderCompleteReading = () => (

<motion.div

className="w-full max-w-4xl mx-auto"

initial={{ opacity: 0, y: 20 }}

animate={{ opacity: 1, y: 0 }}

exit={{ opacity: 0, y: -20 }}

>

{/\* Header \*/}

<motion.div

className="text-center mb-8"

initial={{ opacity: 0, y: -20 }}

animate={{ opacity: 1, y: 0 }}

>

<h2 className="text-2xl md:text-3xl font-bold text-white mb-2">

Your Complete Reading

</h2>

<p className="text-purple-300">

Sophia's personalized guidance for your{" "}

{selectedSpread.replace("-", " ")} spread

</p>

</motion.div>

{/\* Final Reading Display \*/}

{finalReading && (

<motion.div

className="mb-8 bg-purple-900/30 backdrop-blur-sm rounded-2xl p-6"

initial={{ opacity: 0, y: 20 }}

animate={{ opacity: 1, y: 0 }}

transition={{ delay: 0.2 }}

>

<div className="flex items-center space-x-3 mb-6">

<div className="w-12 h-12 bg-gradient-to-br from-purple-500 to-pink-500 rounded-full flex items-center justify-center">

<Sparkles className="w-6 h-6 text-white" />

</div>

<div>

<h3 className="text-purple-300 font-semibold text-lg">

Your Personalized Reading

</h3>

<p className="text-purple-400 text-sm">

Guided by {finalReading.reader\_signature}

</p>

</div>

</div>

<div className="space-y-6 text-purple-200">

{/\* Reading Narrative \*/}

<div className="bg-purple-800/20 rounded-lg p-5">

<h4 className="text-purple-300 font-medium mb-3 flex items-center">

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

Your Reading

</h4>

<p className="leading-relaxed text-lg">

{finalReading.narrative}

</p>

</div>

{/\* Overall Guidance \*/}

<div className="bg-pink-800/20 rounded-lg p-5">

<h4 className="text-pink-300 font-medium mb-3">

Guidance & Wisdom

</h4>

<p className="leading-relaxed">{finalReading.overall\_guidance}</p>

</div>

{/\* Spiritual Insight \*/}

<div className="bg-cyan-800/20 rounded-lg p-5">

<h4 className="text-cyan-300 font-medium mb-3">

Spiritual Insight

</h4>

<p className="leading-relaxed">

{finalReading.spiritual\_insight}

</p>

</div>

{/\* Individual Card Interpretations \*/}

{finalReading.card\_interpretations.length > 0 && (

<div className="bg-purple-800/20 rounded-lg p-5">

<h4 className="text-purple-300 font-medium mb-4">

Individual Card Wisdom

</h4>

<div className="space-y-4">

{finalReading.card\_interpretations.map((interp, index) => (

<motion.div

key={index}

className="border-l-3 border-purple-500 pl-4 py-2"

initial={{ opacity: 0, x: -20 }}

animate={{ opacity: 1, x: 0 }}

transition={{ delay: 0.1 \* index }}

>

<div className="flex items-center justify-between mb-2">

<h5 className="text-purple-300 font-medium">

{drawnCards[index]?.name}

</h5>

<span className="text-xs text-purple-400 bg-purple-800/30 px-2 py-1 rounded">

{interp.confidence\_score > 0.8

? "High Confidence"

: "Standard"}

</span>

</div>

<p className="text-purple-200 text-sm leading-relaxed mb-2">

{interp.personalized\_guidance}

</p>

{interp.practical\_advice && (

<p className="text-purple-300 text-xs italic">

💫 {interp.practical\_advice}

</p>

)}

</motion.div>

))}

</div>

</div>

)}

{/\* Conversation History Summary \*/}

{conversationHistory.length > 0 && (

<motion.div

className="bg-indigo-800/20 rounded-lg p-5"

initial={{ opacity: 0, y: 20 }}

animate={{ opacity: 1, y: 0 }}

transition={{ delay: 0.5 }}

>

<h4 className="text-indigo-300 font-medium mb-3">

Your Journey with Sophia

</h4>

<p className="text-indigo-200 text-sm mb-3">

This reading evolved through {conversationHistory.length}{" "}

meaningful exchanges

</p>

<div className="max-h-40 overflow-y-auto space-y-2">

{conversationHistory.slice(-3).map((turn, index) => (

<div

key={index}

className="text-indigo-200 text-xs p-2 bg-indigo-900/20 rounded"

>

<p className="opacity-75">

"{turn.sophiaDialogue?.substring(0, 100) || ""}..."

</p>

</div>

))}

</div>

</motion.div>

)}

</div>

</motion.div>

)}

{/\* Cards Display \*/}

<motion.div

className="mb-8"

initial={{ opacity: 0, y: 20 }}

animate={{ opacity: 1, y: 0 }}

transition={{ delay: 0.4 }}

>

<h4 className="text-purple-300 font-medium mb-4 text-center">

Your Cards

</h4>

<div className="grid grid-cols-1 md:grid-cols-3 lg:grid-cols-5 gap-4 max-w-4xl mx-auto">

{drawnCards.map((card, index) => (

<motion.div

key={`final-card-${index}`}

className="relative aspect-[2/3] rounded-lg overflow-hidden bg-gradient-to-br from-purple-800 to-pink-800 p-3"

initial={{ opacity: 0, scale: 0.8 }}

animate={{ opacity: 1, scale: 1 }}

transition={{ delay: 0.5 + index \* 0.1 }}

whileHover={{ scale: 1.05, y: -5 }}

>

<div className="h-full flex flex-col">

<h5 className="text-purple-200 font-medium text-sm mb-2 text-center">

{card.name}

</h5>

<div className="flex-1 flex items-center justify-center">

<div className="w-16 h-16 bg-gradient-to-br from-gold-400 to-yellow-500 rounded-full flex items-center justify-center">

<Sparkles className="w-8 h-8 text-black" />

</div>

</div>

<p className="text-purple-300 text-xs text-center mt-2">

{card.arcana === "major"

? "Major Arcana"

: `${card.suit} - Minor`}

</p>

{card.isReversed && (

<p className="text-pink-300 text-xs text-center mt-1">

Reversed

</p>

)}

</div>

</motion.div>

))}

</div>

</motion.div>

{/\* Controls \*/}

<motion.div

className="fixed bottom-6 left-1/2 transform -translate-x-1/2 z-50"

initial={{ opacity: 0, y: 20 }}

animate={{ opacity: 1, y: 0 }}

transition={{ delay: 0.6 }}

>

<div className="bg-black/80 backdrop-blur-lg rounded-2xl p-4 flex items-center space-x-4 shadow-2xl">

<motion.button

className="flex items-center space-x-2 px-4 py-2 bg-pink-600 hover:bg-pink-700 text-white rounded-lg transition-colors"

onClick={() => handleSaveConversationalReading()}

whileHover={{ scale: 1.05 }}

whileTap={{ scale: 0.95 }}

>

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

<span>Save Reading</span>

</motion.button>

<motion.button

className="flex items-center space-x-2 px-4 py-2 bg-cyan-600 hover:bg-cyan-700 text-white rounded-lg transition-colors"

onClick={onBackToSelection}

whileHover={{ scale: 1.05 }}

whileTap={{ scale: 0.95 }}

>

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

<span>New Reading</span>

</motion.button>

</div>

</motion.div>

</motion.div>

);

return (

<div

ref={surfaceRef}

className={`min-h-screen bg-gradient-to-br from-purple-900 via-blue-900 to-indigo-900 p-4 ${className}`}

>

<div className="max-w-6xl mx-auto pt-8">

{phase === "preparation" && renderPreparation()}

{phase === "shuffling" && renderShuffling()}

{(phase === "revealing" ||

phase === "interpreting" ||

phase === "conversation") &&

renderReading()}

{phase === "complete" && renderCompleteReading()}

{/\* Error Display \*/}

{error && (

<motion.div

className="fixed top-4 right-4 bg-red-900/90 backdrop-blur-sm text-white p-4 rounded-lg shadow-lg max-w-md z-50"

initial={{ opacity: 0, x: 50 }}

animate={{ opacity: 1, x: 0 }}

exit={{ opacity: 0, x: 50 }}

>

<div className="flex items-start space-x-3">

<div className="flex-shrink-0">

<svg

className="h-6 w-6 text-red-400"

fill="none"

viewBox="0 0 24 24"

stroke="currentColor"

>

<path

strokeLinecap="round"

strokeLinejoin="round"

strokeWidth={2}

d="M12 8v4m0 4h.01M21 12a9 9 0 11-18 0 9 9 0 0118 0z"

/>

</svg>

</div>

<div className="flex-1">

<h3 className="text-sm font-medium text-red-300">Error</h3>

<p className="mt-1 text-sm text-red-200">{error}</p>

</div>

<button

onClick={() => setError(null)}

className="flex-shrink-0 text-red-400 hover:text-red-300"

>

<svg

className="h-5 w-5"

fill="none"

viewBox="0 0 24 24"

stroke="currentColor"

>

<path

strokeLinecap="round"

strokeLinejoin="round"

strokeWidth={2}

d="M6 18L18 6M6 6l12 12"

/>

</svg>

</button>

</div>

</motion.div>

)}

{/\* Engagement Level Up Notification \*/}

<AnimatePresence>

{engagementLevelUp && (

<motion.div

className="fixed top-1/2 left-1/2 transform -translate-x-1/2 -translate-y-1/2 z-50"

initial={{ opacity: 0, scale: 0.5 }}

animate={{ opacity: 1, scale: 1 }}

exit={{ opacity: 0, scale: 0.5 }}

transition={{ type: "spring", damping: 15, stiffness: 300 }}

>

<div className="bg-gradient-to-r from-purple-900/95 to-pink-900/95 backdrop-blur-md rounded-2xl p-8 text-center shadow-2xl border border-purple-500/30">

<motion.div

className="text-6xl mb-4"

animate={{ rotate: [0, 360] }}

transition={{ duration: 2, ease: "linear" }}

>

🌟

</motion.div>

<h2 className="text-3xl font-bold text-white mb-2">

Level Up!

</h2>

<p className="text-xl text-purple-300 mb-4">

{engagementLevelUp.thresholdMet}

</p>

<p className="text-lg text-purple-200">

Welcome to Level {engagementLevelUp.newLevel}

</p>

</div>

</motion.div>

)}

</AnimatePresence>

</div>

</div>

);

};

Transitengine.ts

/\*\*

\* Transit Engine - Real-time planetary movement analysis

\* Uses Swiss Ephemeris for accurate astronomical calculations

\*/

import { AstronomicalCalculator } from '@/lib/astrology/AstronomicalCalculator';

import { SwissEphemerisShim } from '@/lib/astrology/SwissEphemerisShim';

import { AspectType, BirthData, Planet } from "../../types/astrology";

export interface PlanetaryPosition {

planet: Planet;

longitude: number;

latitude?: number;

distance?: number;

speed: number;

sign: string;

house?: number;

retrograde: boolean;

}

export interface TransitAspect {

transitPlanet: Planet;

natalPlanet: Planet;

aspect: AspectType;

orb: number;

exactDate: Date;

applying: boolean;

separating: boolean;

strength: 'strong' | 'moderate' | 'weak';

}

export interface DailyTransit {

date: Date;

planetaryPositions: PlanetaryPosition[];

majorAspects: TransitAspect[];

lunarPhase: {

phase: 'new' | 'waxing\_crescent' | 'first\_quarter' | 'waxing\_gibbous' | 'full' | 'waning\_gibbous' | 'last\_quarter' | 'waning\_crescent';

illumination: number;

nextPhaseDate: Date;

};

cosmicWeather: {

energy: 'high' | 'medium' | 'low';

focus: string[];

challenges: string[];

opportunities: string[];

};

}

export interface PersonalizedHoroscope {

date: Date;

overallTheme: string;

keyTransits: TransitAspect[];

dailyGuidance: {

love: string;

career: string;

health: string;

spiritual: string;

};

luckyNumbers: number[];

affirmation: string;

bestTimeOfDay: string;

}

export class TransitEngine {

private astroCalc: AstronomicalCalculator;

private ephemeris: SwissEphemerisShim;

constructor() {

this.astroCalc = new AstronomicalCalculator();

this.ephemeris = new SwissEphemerisShim();

}

/\*\*

\* Calculate current planetary positions

\*/

async getCurrentPlanetaryPositions(): Promise<PlanetaryPosition[]> {

const today = new Date();

const positions: PlanetaryPosition[] = [];

const planets: Planet[] = [

'sun', 'moon', 'mercury', 'venus', 'mars',

'jupiter', 'saturn', 'uranus', 'neptune', 'pluto'

];

for (const planet of planets) {

try {

const position = await AstronomicalCalculator.calculatePlanetaryPosition(planet, today);

const speed = await this.calculatePlanetarySpeed(planet, today, position);

const sign = this.getZodiacSign(position.longitude);

positions.push({

planet,

longitude: position.longitude,

latitude: position.latitude,

distance: position.distance,

speed,

sign,

retrograde: speed < 0

});

} catch (error) {

console.warn(`Failed to get position for ${planet}:`, error);

// Fallback to approximate position

positions.push(this.getFallbackPosition(planet, today));

}

}

return positions;

}

/\*\*

\* Calculate planetary speed (degrees per day)

\*/

private async calculatePlanetarySpeed(planet: Planet, date: Date, position?: { longitude: number; latitude: number; distance: number; speed?: number }): Promise<number> {

// If position already includes speed, use it

if (position && typeof position.speed === 'number') {

return position.speed;

}

try {

const today = await AstronomicalCalculator.calculatePlanetaryPosition(planet, date);

// If the position includes speed, use it

if (typeof today.speed === 'number') {

return today.speed;

}

// Otherwise calculate speed from positions

const tomorrow = new Date(date);

tomorrow.setDate(tomorrow.getDate() + 1);

const nextDay = await AstronomicalCalculator.calculatePlanetaryPosition(planet, tomorrow);

let speed = nextDay.longitude - today.longitude;

// Handle crossing 0° Aries

if (speed < -300) speed += 360;

if (speed > 300) speed -= 360;

return speed;

} catch (error) {

// Fallback average speeds (degrees per day)

const averageSpeeds: Record<Planet, number> = {

sun: 0.9856,

moon: 13.1764,

mercury: 1.59,

venus: 1.21,

mars: 0.64,

jupiter: 0.083,

saturn: 0.033,

uranus: 0.011,

neptune: 0.006,

pluto: 0.004

};

return averageSpeeds[planet] || 0.5;

}

}

/\*\*

\* Get zodiac sign from longitude

\*/

private getZodiacSign(longitude: number): string {

const signs = [

'Aries', 'Taurus', 'Gemini', 'Cancer',

'Leo', 'Virgo', 'Libra', 'Scorpio',

'Sagittarius', 'Capricorn', 'Aquarius', 'Pisces'

];

const signIndex = Math.floor(longitude / 30);

return signs[signIndex] || 'Aries';

}

/\*\*

\* Calculate transits to natal chart

\*/

async calculateTransits(birthData: BirthData, targetDate: Date = new Date()): Promise<TransitAspect[]> {

try {

const transitPositions = await this.getCurrentPlanetaryPositions();

// Use SwissEphemerisShim instead since AstronomicalCalculator doesn't have calculateBirthChart

const SwissEphemerisShim = (

await import("@/lib/astrology/SwissEphemerisShim")

).SwissEphemerisShim;

const natalChart = await SwissEphemerisShim.calculateFullChart(birthData);

const aspects: TransitAspect[] = [];

for (const transitPos of transitPositions) {

for (const natalPlanet of Object.keys(natalChart.planets) as Planet[]) {

const natalLongitude =

natalChart.planets[Number(natalPlanet)]?.longitude;

if (natalLongitude === undefined) continue;

const aspect = this.calculateAspect(

transitPos.longitude,

natalLongitude

);

if (aspect) {

const orb = this.calculateOrb(

transitPos.longitude,

natalLongitude,

aspect.type

);

if (orb <= this.getMaxOrb(aspect.type)) {

aspects.push({

transitPlanet: transitPos.planet,

natalPlanet,

aspect: aspect.type,

orb,

exactDate: this.calculateExactAspectDate(

transitPos,

natalLongitude,

aspect.type

),

applying: transitPos.speed > 0 ? orb > 0 : orb < 0,

separating: transitPos.speed > 0 ? orb < 0 : orb > 0,

strength: orb <= 1 ? "strong" : orb <= 3 ? "moderate" : "weak",

});

}

}

}

}

return aspects.sort((a, b) => a.orb - b.orb);

} catch (error) {

console.warn('Failed to calculate transits:', error);

return [];

}

}

/\*\*

\* Calculate aspect between two planetary positions

\*/

private calculateAspect(longitude1: number, longitude2: number): { type: AspectType; angle: number } | null {

let diff = Math.abs(longitude1 - longitude2);

if (diff > 180) diff = 360 - diff;

const aspects = [

{ type: 'conjunction' as AspectType, angle: 0, orb: 8 },

{ type: 'sextile' as AspectType, angle: 60, orb: 4 },

{ type: 'square' as AspectType, angle: 90, orb: 6 },

{ type: 'trine' as AspectType, angle: 120, orb: 6 },

{ type: 'opposition' as AspectType, angle: 180, orb: 8 }

];

for (const aspect of aspects) {

if (Math.abs(diff - aspect.angle) <= aspect.orb) {

return { type: aspect.type, angle: aspect.angle };

}

}

return null;

}

/\*\*

\* Calculate orb (exactness) of aspect

\*/

private calculateOrb(longitude1: number, longitude2: number, aspectType: AspectType): number {

let diff = longitude1 - longitude2;

if (diff > 180) diff -= 360;

if (diff < -180) diff += 360;

const aspectAngles = {

conjunction: 0,

sextile: 60,

square: 90,

trine: 120,

opposition: 180

};

const targetAngle = aspectAngles[aspectType];

return Math.abs(Math.abs(diff) - targetAngle);

}

/\*\*

\* Get maximum orb for aspect type

\*/

private getMaxOrb(aspectType: AspectType): number {

const orbs = {

conjunction: 8,

sextile: 4,

square: 6,

trine: 6,

opposition: 8

};

return orbs[aspectType] || 3;

}

/\*\*

\* Calculate exact date when aspect becomes exact

\*/

private calculateExactAspectDate(transitPos: PlanetaryPosition, natalLongitude: number, aspectType: AspectType): Date {

const aspectAngles = {

conjunction: 0,

sextile: 60,

square: 90,

trine: 120,

opposition: 180

};

const targetAngle = aspectAngles[aspectType];

let diff = transitPos.longitude - natalLongitude;

if (diff > 180) diff -= 360;

if (diff < -180) diff += 360;

const currentAngle = Math.abs(diff);

const degreesToTarget = Math.abs(currentAngle - targetAngle);

const daysToExact = degreesToTarget / Math.abs(transitPos.speed);

const exactDate = new Date();

exactDate.setDate(exactDate.getDate() + daysToExact);

return exactDate;

}

/\*\*

\* Generate daily transit report

\*/

async getDailyTransit(date: Date = new Date()): Promise<DailyTransit> {

const planetaryPositions = await this.getCurrentPlanetaryPositions();

const lunarPhase = await this.calculateLunarPhase(date);

const cosmicWeather = this.analyzeCosmicWeather(planetaryPositions);

return {

date,

planetaryPositions,

majorAspects: [], // Will be populated when natal chart is available

lunarPhase,

cosmicWeather

};

}

/\*\*

\* Calculate lunar phase

\*/

private async calculateLunarPhase(date: Date) {

try {

const sunPos = await AstronomicalCalculator.calculatePlanetaryPosition('sun', date);

const moonPos = await AstronomicalCalculator.calculatePlanetaryPosition('moon', date);

let diff = moonPos.longitude - sunPos.longitude;

if (diff < 0) diff += 360;

const phases = [

{ name: 'new', min: 0, max: 22.5 },

{ name: 'waxing\_crescent', min: 22.5, max: 67.5 },

{ name: 'first\_quarter', min: 67.5, max: 112.5 },

{ name: 'waxing\_gibbous', min: 112.5, max: 157.5 },

{ name: 'full', min: 157.5, max: 202.5 },

{ name: 'waning\_gibbous', min: 202.5, max: 247.5 },

{ name: 'last\_quarter', min: 247.5, max: 292.5 },

{ name: 'waning\_crescent', min: 292.5, max: 337.5 },

{ name: 'new', min: 337.5, max: 360 }

];

const currentPhase = phases.find(phase => diff >= phase.min && diff < phase.max) || phases[0];

const illumination = (1 - Math.cos(diff \* Math.PI / 180)) / 2;

// Calculate next phase date (simplified)

const nextPhaseDate = new Date(date);

nextPhaseDate.setDate(nextPhaseDate.getDate() + 7);

return {

phase: currentPhase.name as any,

illumination: Math.round(illumination \* 100) / 100,

nextPhaseDate

};

} catch (error) {

return {

phase: 'new' as any,

illumination: 0.5,

nextPhaseDate: new Date()

};

}

}

/\*\*

\* Analyze cosmic weather patterns

\*/

private analyzeCosmicWeather(positions: PlanetaryPosition[]) {

const retrogradeCount = positions.filter(p => p.retrograde).length;

const fastMovingPlanets = positions.filter(p => Math.abs(p.speed) > 1).length;

let energy: 'high' | 'medium' | 'low' = 'medium';

const focus: string[] = [];

const challenges: string[] = [];

const opportunities: string[] = [];

// Analyze energy level

if (fastMovingPlanets >= 3) energy = 'high';

if (retrogradeCount >= 3) energy = 'low';

// Analyze retrograde effects

if (retrogradeCount > 0) {

challenges.push('Retrograde energy calls for patience and reflection');

focus.push('Review and revision');

}

// Analyze planetary positions

const fireSignPlanets = positions.filter(p => ['Aries', 'Leo', 'Sagittarius'].includes(p.sign)).length;

const earthSignPlanets = positions.filter(p => ['Taurus', 'Virgo', 'Capricorn'].includes(p.sign)).length;

const airSignPlanets = positions.filter(p => ['Gemini', 'Libra', 'Aquarius'].includes(p.sign)).length;

const waterSignPlanets = positions.filter(p => ['Cancer', 'Scorpio', 'Pisces'].includes(p.sign)).length;

if (fireSignPlanets >= 3) {

focus.push('Action and initiative');

opportunities.push('Leadership opportunities');

}

if (earthSignPlanets >= 3) {

focus.push('Practical matters');

opportunities.push('Material progress');

}

if (airSignPlanets >= 3) {

focus.push('Communication and ideas');

opportunities.push('Networking and learning');

}

if (waterSignPlanets >= 3) {

focus.push('Emotions and intuition');

opportunities.push('Spiritual insights');

}

return {

energy,

focus,

challenges,

opportunities

};

}

/\*\*

\* Generate personalized horoscope

\*/

async generatePersonalizedHoroscope(birthData: BirthData, date: Date = new Date()): Promise<PersonalizedHoroscope> {

const transits = await this.calculateTransits(birthData, date);

const dailyTransit = await this.getDailyTransit(date);

// Select key transits (most exact and significant)

const keyTransits = transits

.filter(t => t.strength === 'strong' || t.strength === 'moderate')

.slice(0, 3);

// Generate guidance based on transits

const guidance = this.generateGuidanceFromTransits(keyTransits, dailyTransit);

return {

date,

overallTheme: this.generateOverallTheme(keyTransits, dailyTransit),

keyTransits,

dailyGuidance: guidance,

luckyNumbers: this.generateLuckyNumbers(date, birthData),

affirmation: this.generateAffirmation(keyTransits),

bestTimeOfDay: this.getBestTimeOfDay(dailyTransit)

};

}

/\*\*

\* Generate overall theme for the day

\*/

private generateOverallTheme(transits: TransitAspect[], dailyTransit: DailyTransit): string {

if (transits.length === 0) {

return `A day of ${dailyTransit.cosmicWeather.energy} cosmic energy. Focus on ${dailyTransit.cosmicWeather.focus[0] || 'personal growth'}.`;

}

const majorTransit = transits[0];

const themes = {

conjunction: 'new beginnings and focused energy',

sextile: 'harmonious opportunities and easy flow',

square: 'challenges that catalyze growth',

trine: 'natural talents and favorable conditions',

opposition: 'balance and integration of opposites'

};

return `${majorTransit.transitPlanet} ${majorTransit.aspect} ${majorTransit.natalPlanet} brings ${themes[majorTransit.aspect]}. ${dailyTransit.cosmicWeather.energy === 'high' ? 'High energy day' : dailyTransit.cosmicWeather.energy === 'low' ? 'Reflective day' : 'Balanced energy'}.`;

}

/\*\*

\* Generate daily guidance from transits

\*/

private generateGuidanceFromTransits(transits: TransitAspect[], dailyTransit: DailyTransit) {

const defaultGuidance = {

love: 'Open your heart to unexpected connections and deeper understanding.',

career: 'Focus on your long-term goals and take practical steps forward.',

health: 'Listen to your body and prioritize rest and nourishment.',

spiritual: 'Trust your intuition and seek moments of quiet reflection.'

};

if (transits.length === 0) return defaultGuidance;

const guidance = { ...defaultGuidance };

for (const transit of transits) {

if (transit.transitPlanet === 'venus' || transit.natalPlanet === 'venus') {

guidance.love = transit.aspect === 'conjunction' || transit.aspect === 'trine' ?

'Love energy is heightened. Express affection and enjoy beautiful moments.' :

'Relationship dynamics may need attention. Practice patience and understanding.';

}

if (transit.transitPlanet === 'mars' || transit.natalPlanet === 'mars') {

guidance.career = transit.aspect === 'conjunction' || transit.aspect === 'trine' ?

'Dynamic energy supports bold career moves and leadership.' :

'Channel competitive energy constructively. Avoid conflicts at work.';

}

if (transit.transitPlanet === 'moon' || transit.natalPlanet === 'moon') {

guidance.health = 'Pay attention to emotional well-being. Your feelings are especially important today.';

}

if (transit.transitPlanet === 'jupiter' || transit.natalPlanet === 'jupiter') {

guidance.spiritual = 'Opportunities for growth and expansion. Stay open to new perspectives.';

}

}

return guidance;

}

/\*\*

\* Generate lucky numbers based on birth data and date

\*/

private generateLuckyNumbers(date: Date, birthData: BirthData): number[] {

const day = date.getDate();

const month = date.getMonth() + 1;

const birthDay = new Date(birthData.birthDate).getDate();

const birthMonth = new Date(birthData.birthDate).getMonth() + 1;

return [

day,

month,

birthDay,

birthMonth,

(day + birthDay) % 9 + 1,

(month + birthMonth) % 9 + 1

].slice(0, 6);

}

/\*\*

\* Generate affirmation based on transits

\*/

private generateAffirmation(transits: TransitAspect[]): string {

const affirmations = [

'I trust the perfect timing of the universe.',

'I am aligned with my highest purpose.',

'I embrace change as an opportunity for growth.',

'I am open to the abundance flowing into my life.',

'I trust my intuition to guide me forward.',

'I radiate love and attract positive experiences.',

'I am exactly where I need to be in this moment.'

];

if (transits.length === 0) {

return affirmations[Math.floor(Math.random() \* affirmations.length)];

}

const majorTransit = transits[0];

if (majorTransit.aspect === 'conjunction') {

return 'I embrace new beginnings with confidence and clarity.';

} else if (majorTransit.aspect === 'trine') {

return 'I am in harmony with the natural flow of life.';

} else if (majorTransit.aspect === 'square') {

return 'I transform challenges into opportunities for growth.';

} else {

return affirmations[Math.floor(Math.random() \* affirmations.length)];

}

}

/\*\*

\* Determine best time of day based on cosmic weather

\*/

private getBestTimeOfDay(dailyTransit: DailyTransit): string {

const times = ['Early morning', 'Late morning', 'Afternoon', 'Evening', 'Night'];

if (dailyTransit.cosmicWeather.energy === 'high') {

return 'Late morning';

} else if (dailyTransit.cosmicWeather.energy === 'low') {

return 'Evening';

} else {

return 'Afternoon';

}

}

/\*\*

\* Fallback position when ephemeris fails

\*/

private getFallbackPosition(planet: Planet, date: Date): PlanetaryPosition {

// Simple fallback based on approximate positions

const approximatePositions = {

sun: (date.getMonth() \* 30 + date.getDate()) % 360,

moon: (date.getDate() \* 13) % 360,

mercury: (date.getMonth() \* 25 + date.getDate() \* 2) % 360,

venus: (date.getMonth() \* 20 + date.getDate() \* 1.5) % 360,

mars: (date.getMonth() \* 15 + date.getDate() \* 0.5) % 360,

jupiter: (date.getFullYear() % 12) \* 30,

saturn: (date.getFullYear() % 29) \* 12.4,

uranus: (date.getFullYear() % 84) \* 4.3,

neptune: (date.getFullYear() % 165) \* 2.2,

pluto: (date.getFullYear() % 248) \* 1.5

};

const longitude = approximatePositions[planet] || 0;

return {

planet,

longitude,

speed: 0.5,

sign: this.getZodiacSign(longitude),

retrograde: false

};

}

}

Astrology-horoscopes-test.tsx

import { BirthData } from "../../src/types/astrology";

// Mock fetch globally

global.fetch = jest.fn();

describe("Daily Oracle Astrology Horoscopes", () => {

const zodiacSigns = [

"aries",

"taurus",

"gemini",

"cancer",

"leo",

"virgo",

"libra",

"scorpio",

"sagittarius",

"capricorn",

"aquarius",

"pisces",

];

const zodiacElements = {

aries: "fire",

leo: "fire",

sagittarius: "fire",

taurus: "earth",

virgo: "earth",

capricorn: "earth",

gemini: "air",

libra: "air",

aquarius: "air",

cancer: "water",

scorpio: "water",

pisces: "water",

};

const zodiacModalities = {

aries: "cardinal",

cancer: "cardinal",

libra: "cardinal",

capricorn: "cardinal",

taurus: "fixed",

leo: "fixed",

scorpio: "fixed",

aquarius: "fixed",

gemini: "mutable",

virgo: "mutable",

sagittarius: "mutable",

pisces: "mutable",

};

beforeEach(() => {

jest.clearAllMocks();

});

describe("Complete Zodiac Horoscope Coverage", () => {

zodiacSigns.forEach((sign) => {

it(`should provide complete horoscope data for ${sign}`, async () => {

const mockHoroscopeResponse = {

success: true,

data: {

horoscopes: {

[sign]: {

db\_identifier: `DB\_ENTRY\_072625\_horoscope\_${sign}`,

sign: sign,

element: zodiacElements[sign],

modality: zodiacModalities[sign],

ruling\_planet: sign === "aries" ? "mars" : "test\_planet",

daily: `Today ${sign} experiences profound cosmic alignment...`,

love: `Romantic energies for ${sign} are heightened today...`,

career: `Professional opportunities align for ${sign}...`,

mood: `${sign} feels energetically balanced and focused`,

self\_growth: `Personal development for ${sign} centers on...`,

planetary\_influences: {

primary: `Jupiter in favorable aspect enhances ${sign} energy`,

secondary: `Venus supports ${sign} relationships today`,

challenging: `Saturn requires ${sign} to face responsibilities`,

},

degrees: Math.random() \* 30, // 0-30 degrees within sign

house\_position: Math.floor(Math.random() \* 12) + 1,

aspects: [

{ planet: "venus", aspect: "trine", influence: "positive" },

{

planet: "mars",

aspect: "square",

influence: "challenging",

},

],

lucky\_numbers: [3, 7, 12],

colors: ["blue", "silver"],

keywords: ["growth", "harmony", "transformation"],

},

},

},

};

(global.fetch as jest.Mock).mockResolvedValueOnce({

ok: true,

json: async () => mockHoroscopeResponse,

});

const response = await fetch(

`/api/daily-oracle/astrology/horoscope/${sign}`

);

const data = await response.json();

const horoscope = data.data.horoscopes[sign];

// Validate required fields

expect(horoscope.db\_identifier).toMatch(

new RegExp(`^DB\_ENTRY\_\\d{6}\_horoscope\_${sign}$`)

);

expect(horoscope.sign).toBe(sign);

expect(horoscope.element).toBe(zodiacElements[sign]);

expect(horoscope.modality).toBe(zodiacModalities[sign]);

// Validate content areas

expect(horoscope.daily).toBeDefined();

expect(horoscope.daily.length).toBeGreaterThan(50);

expect(horoscope.love).toBeDefined();

expect(horoscope.career).toBeDefined();

expect(horoscope.mood).toBeDefined();

expect(horoscope.self\_growth).toBeDefined();

// Validate astrological data

expect(horoscope.degrees).toBeGreaterThanOrEqual(0);

expect(horoscope.degrees).toBeLessThan(30);

expect(horoscope.house\_position).toBeGreaterThanOrEqual(1);

expect(horoscope.house\_position).toBeLessThanOrEqual(12);

expect(horoscope.aspects).toBeInstanceOf(Array);

expect(horoscope.planetary\_influences).toBeDefined();

});

});

it("should provide all 12 horoscopes in single response", async () => {

const mockAllHoroscopesResponse = {

success: true,

data: {

date: "2025-07-26",

horoscopes: Object.fromEntries(

zodiacSigns.map((sign) => [

sign,

{

db\_identifier: `DB\_ENTRY\_072625\_horoscope\_${sign}`,

sign: sign,

element: zodiacElements[sign],

modality: zodiacModalities[sign],

daily: `Daily guidance for ${sign}...`,

love: `Love forecast for ${sign}...`,

career: `Career insights for ${sign}...`,

mood: `Emotional tone for ${sign}`,

self\_growth: `Growth opportunities for ${sign}...`,

},

])

),

metadata: {

total\_signs: 12,

elements\_covered: ["fire", "earth", "air", "water"],

modalities\_covered: ["cardinal", "fixed", "mutable"],

},

},

};

(global.fetch as jest.Mock).mockResolvedValueOnce({

ok: true,

json: async () => mockAllHoroscopesResponse,

});

const response = await fetch(

"/api/daily-oracle/astrology/horoscopes/all"

);

const data = await response.json();

// Validate all signs present

zodiacSigns.forEach((sign) => {

expect(data.data.horoscopes[sign]).toBeDefined();

expect(data.data.horoscopes[sign].sign).toBe(sign);

});

// Validate metadata

expect(data.data.metadata.total\_signs).toBe(12);

expect(data.data.metadata.elements\_covered).toEqual([

"fire",

"earth",

"air",

"water",

]);

expect(data.data.metadata.modalities\_covered).toEqual([

"cardinal",

"fixed",

"mutable",

]);

});

});

describe("Planetary Transits Integration", () => {

it("should reflect current planetary transits in horoscopes", async () => {

const mockTransitResponse = {

success: true,

data: {

current\_transits: {

date: "2025-07-26",

major\_transits: [

{

planet: "saturn",

sign: "pisces",

aspect: "trine",

target\_planet: "uranus",

target\_sign: "gemini",

influence: "Structured innovation and practical breakthroughs",

},

{

planet: "venus",

position: "galactic\_center",

influence: "Deep cosmic love and universal connection",

},

],

},

horoscopes: {

pisces: {

daily:

"With Saturn in your sign trining Uranus, structured innovation flows naturally...",

transit\_specific:

"Saturn in Pisces brings grounded spirituality to your core identity",

},

gemini: {

daily:

"Uranus in your sign receives supportive energy from Saturn...",

transit\_specific: "Revolutionary ideas gain practical foundation",

},

},

},

};

(global.fetch as jest.Mock).mockResolvedValueOnce({

ok: true,

json: async () => mockTransitResponse,

});

const response = await fetch("/api/daily-oracle/astrology/transits");

const data = await response.json();

// Validate transit data

expect(data.data.current\_transits.major\_transits).toBeInstanceOf(Array);

expect(data.data.current\_transits.major\_transits.length).toBeGreaterThan(

0

);

// Validate transit-specific horoscope content

expect(data.data.horoscopes.pisces.transit\_specific).toContain("Saturn");

expect(data.data.horoscopes.gemini.transit\_specific).toContain("Uranus");

});

});

describe("Elemental and Modal Analysis", () => {

it("should provide elemental energy analysis", async () => {

const mockElementalResponse = {

success: true,

data: {

elemental\_analysis: {

fire\_signs: {

general\_energy: "Passionate and action-oriented today",

signs: ["aries", "leo", "sagittarius"],

dominant\_theme: "Creative manifestation",

},

earth\_signs: {

general\_energy: "Grounded and practical focus",

signs: ["taurus", "virgo", "capricorn"],

dominant\_theme: "Material world mastery",

},

air\_signs: {

general\_energy: "Mental clarity and communication",

signs: ["gemini", "libra", "aquarius"],

dominant\_theme: "Intellectual breakthroughs",

},

water\_signs: {

general\_energy: "Emotional depth and intuition",

signs: ["cancer", "scorpio", "pisces"],

dominant\_theme: "Psychic sensitivity heightened",

},

},

modal\_analysis: {

cardinal\_signs: {

energy: "Initiating new cycles",

signs: ["aries", "cancer", "libra", "capricorn"],

},

fixed\_signs: {

energy: "Maintaining and stabilizing",

signs: ["taurus", "leo", "scorpio", "aquarius"],

},

mutable\_signs: {

energy: "Adapting and transforming",

signs: ["gemini", "virgo", "sagittarius", "pisces"],

},

},

},

};

(global.fetch as jest.Mock).mockResolvedValueOnce({

ok: true,

json: async () => mockElementalResponse,

});

const response = await fetch(

"/api/daily-oracle/astrology/elemental-analysis"

);

const data = await response.json();

// Validate elemental analysis

const elements = [

"fire\_signs",

"earth\_signs",

"air\_signs",

"water\_signs",

];

elements.forEach((element) => {

expect(data.data.elemental\_analysis[element]).toBeDefined();

expect(

data.data.elemental\_analysis[element].general\_energy

).toBeDefined();

expect(data.data.elemental\_analysis[element].signs).toBeInstanceOf(

Array

);

expect(data.data.elemental\_analysis[element].signs.length).toBe(3);

});

// Validate modal analysis

const modalities = ["cardinal\_signs", "fixed\_signs", "mutable\_signs"];

modalities.forEach((modality) => {

expect(data.data.modal\_analysis[modality]).toBeDefined();

expect(data.data.modal\_analysis[modality].energy).toBeDefined();

expect(data.data.modal\_analysis[modality].signs).toBeInstanceOf(Array);

expect(data.data.modal\_analysis[modality].signs.length).toBe(4);

});

});

});

describe("Personalized Horoscope Features", () => {

it("should incorporate birth data for personalized readings", async () => {

const mockBirthData: BirthData = {

name: "Test User",

birthDate: "1990-07-26",

birthTime: "14:30",

birthLocation: "San Francisco, USA",

city: "San Francisco",

country: "USA",

latitude: 37.7749,

longitude: -122.4194,

timezone: "America/Los\_Angeles",

};

const mockPersonalizedResponse = {

success: true,

data: {

personalized\_horoscope: {

sun\_sign: "cancer",

rising\_sign: "scorpio",

moon\_sign: "pisces",

birth\_chart\_highlights: {

sun\_house: 8,

moon\_house: 4,

rising\_degree: 15.7,

},

daily\_focus:

"Your Cancer sun in the 8th house emphasizes transformation...",

transit\_impacts: [

{

transit: "Moon conjunct natal Venus",

effect: "Heightened emotional sensitivity in relationships",

},

],

personalization\_factors: {

location\_influences:

"West Coast water energy amplifies your Cancer nature",

birth\_time\_precision:

"Accurate birth time enables precise house cusps",

},

},

},

};

(global.fetch as jest.Mock).mockResolvedValueOnce({

ok: true,

json: async () => mockPersonalizedResponse,

});

const response = await fetch("/api/daily-oracle/astrology/personalized", {

method: "POST",

headers: { "Content-Type": "application/json" },

body: JSON.stringify({ birthData: mockBirthData }),

});

const data = await response.json();

const personal = data.data.personalized\_horoscope;

expect(personal.sun\_sign).toBe("cancer");

expect(personal.birth\_chart\_highlights).toBeDefined();

expect(personal.daily\_focus).toBeDefined();

expect(personal.transit\_impacts).toBeInstanceOf(Array);

expect(personal.personalization\_factors).toBeDefined();

});

});

describe("Horoscope Quality Validation", () => {

zodiacSigns.forEach((sign) => {

it(`should ensure ${sign} horoscope meets quality standards`, async () => {

const mockQualityResponse = {

success: true,

data: {

horoscopes: {

[sign]: {

daily: "A".repeat(100), // Minimum length

love: "B".repeat(80),

career: "C".repeat(90),

mood: "D".repeat(50),

self\_growth: "E".repeat(75),

quality\_metrics: {

word\_count: {

daily: 100,

love: 80,

career: 90,

mood: 50,

self\_growth: 75,

},

readability\_score: 85,

empowerment\_tone: true,

actionable\_guidance: true,

avoids\_fatalism: true,

},

},

},

},

};

(global.fetch as jest.Mock).mockResolvedValueOnce({

ok: true,

json: async () => mockQualityResponse,

});

const response = await fetch(

`/api/daily-oracle/astrology/quality/${sign}`

);

const data = await response.json();

const horoscope = data.data.horoscopes[sign];

const quality = horoscope.quality\_metrics;

// Validate minimum content length

expect(quality.word\_count.daily).toBeGreaterThanOrEqual(50);

expect(quality.word\_count.love).toBeGreaterThanOrEqual(40);

expect(quality.word\_count.career).toBeGreaterThanOrEqual(40);

// Validate quality standards

expect(quality.readability\_score).toBeGreaterThanOrEqual(70);

expect(quality.empowerment\_tone).toBe(true);

expect(quality.actionable\_guidance).toBe(true);

expect(quality.avoids\_fatalism).toBe(true);

});

});

});

});

VirtualReaderDisplay.tsx

"use client";

import { PersonaLearnerAgentClient } from "@/agents/PersonaLearner-client";

import { useAuth } from "@/contexts/AuthContext";

import { User } from "@supabase/supabase-js";

import { AnimatePresence, motion } from "framer-motion";

import { Eye, Lock, Sparkles, Unlock } from "lucide-react";

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

interface VirtualReaderDisplayProps {

readerId?: string;

size?: "small" | "medium" | "large";

showLevel?: boolean;

showProgress?: boolean;

className?: string;

}

interface EngagementData {

currentLevel: number;

levelName: string;

metrics: {

completedReadings: number;

conversationTurns: number;

questionsAnswered: number;

sessionCount: number;

};

nextThreshold?: {

level: number;

readings: number;

turns: number;

questions: number;

};

progressToNext?: number;

}

export const VirtualReaderDisplay: React.FC<VirtualReaderDisplayProps> = ({

readerId = "sophia",

size = "medium",

showLevel = true,

showProgress = false,

className = "",

}) => {

const { user } = useAuth() as { user: User | null };

const [engagementData, setEngagementData] = useState<EngagementData | null>(

null

);

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

const [imageError, setImageError] = useState(false);

const [previousLevel, setPreviousLevel] = useState<number | null>(null);

const personaLearner = new PersonaLearnerAgentClient();

// Size configurations

const sizeConfig = {

small: {

container: "w-16 h-24",

image: "w-full h-full",

text: "text-xs",

badge: "text-xs px-2 py-1",

},

medium: {

container: "w-32 h-48",

image: "w-full h-full",

text: "text-sm",

badge: "text-sm px-3 py-1",

},

large: {

container: "w-48 h-72",

image: "w-full h-full",

text: "text-base",

badge: "text-base px-4 py-2",

},

};

const config = sizeConfig[size];

// Load engagement data

useEffect(() => {

const loadEngagementData = async () => {

if (!user) {

// Guest users see level 1

setEngagementData({

currentLevel: 1,

levelName: "Guest User",

metrics: {

completedReadings: 0,

conversationTurns: 0,

questionsAnswered: 0,

sessionCount: 0,

},

});

setIsLoading(false);

return;

}

try {

const analysis = await personaLearner.getEngagementAnalysis(user.id);

// Check if level increased since last view

const lastKnownLevel = localStorage.getItem(

`last\_known\_level\_${user.id}`

);

if (lastKnownLevel) {

const lastLevel = parseInt(lastKnownLevel);

if (analysis.currentLevel > lastLevel) {

setPreviousLevel(lastLevel);

}

}

// Store current level

localStorage.setItem(

`last\_known\_level\_${user.id}`,

analysis.currentLevel.toString()

);

setEngagementData(analysis);

} catch (error) {

console.error(

"VirtualReaderDisplay: Failed to load engagement data:",

error

);

// Fallback to level 1

setEngagementData({

currentLevel: 1,

levelName: "Novice Seeker",

metrics: {

completedReadings: 0,

conversationTurns: 0,

questionsAnswered: 0,

sessionCount: 0,

},

});

} finally {

setIsLoading(false);

}

};

loadEngagementData();

}, [user, personaLearner]);

// Clear level up indicator after animation

useEffect(() => {

if (previousLevel !== null) {

const timer = setTimeout(() => {

setPreviousLevel(null);

}, 3000);

return () => clearTimeout(timer);

}

}, [previousLevel]);

const getImagePath = (level: number): string => {

// Ensure level is within valid range

const validLevel = Math.max(1, Math.min(5, level));

return `/images/readers/${readerId}/level\_${validLevel}.png`;

};

const getLevelColor = (level: number): string => {

switch (level) {

case 1:

return "from-purple-600 to-purple-800";

case 2:

return "from-purple-500 to-pink-600";

case 3:

return "from-pink-500 to-purple-600";

case 4:

return "from-purple-400 to-pink-500";

case 5:

return "from-gold-400 to-yellow-500";

default:

return "from-purple-600 to-purple-800";

}

};

const getLevelIcon = (level: number) => {

if (level >= 5) return <Unlock className="w-4 h-4" />;

if (level >= 3) return <Eye className="w-4 h-4" />;

return <Lock className="w-4 h-4" />;

};

if (isLoading) {

return (

<div

className={`${config.container} ${className} flex items-center justify-center`}

>

<motion.div

className="w-8 h-8 border-2 border-purple-500 border-t-transparent rounded-full"

animate={{ rotate: 360 }}

transition={{ duration: 1, repeat: Infinity, ease: "linear" }}

/>

</div>

);

}

return (

<div className={`relative ${config.container} ${className}`}>

{/\* Level Up Animation Overlay \*/}

<AnimatePresence>

{previousLevel !== null && engagementData && (

<motion.div

className="absolute inset-0 z-20 flex items-center justify-center bg-black/80 rounded-lg"

initial={{ opacity: 0 }}

animate={{ opacity: 1 }}

exit={{ opacity: 0 }}

transition={{ duration: 0.5 }}

>

<motion.div

className="text-center"

initial={{ scale: 0.5, y: 20 }}

animate={{ scale: 1, y: 0 }}

exit={{ scale: 0.5, y: -20 }}

transition={{ type: "spring", damping: 15, stiffness: 300 }}

>

<motion.div

className="text-gold-400 mb-2"

animate={{ rotate: [0, 360] }}

transition={{ duration: 2, ease: "linear" }}

>

<Sparkles className="w-8 h-8 mx-auto" />

</motion.div>

<div className={`text-white font-bold ${config.text}`}>

Level Up!

</div>

<div className={`text-gold-300 ${config.text}`}>

{previousLevel} → {engagementData.currentLevel}

</div>

</motion.div>

</motion.div>

)}

</AnimatePresence>

{/\* Main Reader Image \*/}

<motion.div

className="relative w-full h-full rounded-lg overflow-hidden bg-gradient-to-br from-purple-900/50 to-black/50"

whileHover={{ scale: 1.02 }}

transition={{ type: "spring", damping: 20, stiffness: 300 }}

>

{!imageError ? (

<motion.img

src={getImagePath(engagementData?.currentLevel || 1)}

alt={`${readerId} - Level ${engagementData?.currentLevel || 1}`}

className={`${config.image} object-cover transition-opacity duration-500`}

onError={() => setImageError(true)}

initial={{ opacity: 0 }}

animate={{ opacity: 1 }}

key={engagementData?.currentLevel} // Re-animate on level change

transition={{ duration: 0.8 }}

/>

) : (

// Fallback display if image fails to load

<div

className={`${config.image} bg-gradient-to-br from-purple-800 to-purple-900 flex items-center justify-center`}

>

<div className="text-center text-purple-300">

<Sparkles className="w-8 h-8 mx-auto mb-2" />

<div className={config.text}>

Level {engagementData?.currentLevel || 1}

</div>

</div>

</div>

)}

{/\* Level Badge \*/}

{showLevel && engagementData && (

<motion.div

className={`absolute top-2 right-2 bg-gradient-to-r ${getLevelColor(

engagementData.currentLevel

)}

text-white ${

config.badge

} rounded-full font-bold shadow-lg backdrop-blur-sm border border-white/20

flex items-center space-x-1`}

initial={{ opacity: 0, scale: 0.8 }}

animate={{ opacity: 1, scale: 1 }}

transition={{ delay: 0.3 }}

>

{getLevelIcon(engagementData.currentLevel)}

<span>{engagementData.currentLevel}</span>

</motion.div>

)}

{/\* Cosmic Glow Effect \*/}

<motion.div

className="absolute inset-0 bg-gradient-to-t from-transparent via-transparent to-purple-500/10 pointer-events-none"

animate={{

opacity: [0.5, 0.8, 0.5],

}}

transition={{

duration: 3,

repeat: Infinity,

ease: "easeInOut",

}}

/>

</motion.div>

{/\* Level Name and Progress \*/}

{(showLevel || showProgress) && engagementData && (

<motion.div

className="mt-2 text-center"

initial={{ opacity: 0, y: 10 }}

animate={{ opacity: 1, y: 0 }}

transition={{ delay: 0.5 }}

>

{showLevel && (

<div className={`${config.text} text-purple-300 font-medium`}>

{engagementData.levelName}

</div>

)}

{showProgress && engagementData.nextThreshold && (

<div className="mt-1">

<div className={`${config.text} text-purple-400 mb-1`}>

Progress to Level {engagementData.nextThreshold.level}

</div>

<div className="w-full bg-purple-900/50 rounded-full h-2">

<motion.div

className={`h-2 bg-gradient-to-r ${getLevelColor(

engagementData.nextThreshold.level

)} rounded-full`}

initial={{ width: 0 }}

animate={{ width: `${engagementData.progressToNext || 0}%` }}

transition={{ duration: 1, delay: 0.7 }}

/>

</div>

<div className={`${config.text} text-purple-500 mt-1`}>

{Math.round(engagementData.progressToNext || 0)}%

</div>

</div>

)}

</motion.div>

)}

</div>

);

};