**Folder Structure**

src/

├── components/

│ ├── App.tsx

│ ├── VoiceSelector.tsx

│ ├── PromptEngine.tsx

│ ├── SolutionStack.tsx

│ ├── SynthesisPanel.tsx

│ └── LedgerLog.tsx

├── lib/

│ ├── types.ts

│ ├── constants.ts

│ ├── mockData.ts

│ └── utils.ts

└── hooks/

└── useCodeGeneration.ts

**lib/types.ts**

typescript

import { LucideIcon } from 'lucide-react';

export interface TransisthesisArchetype {

id: string;

name: string;

icon: LucideIcon;

color: string;

focus: string;

essence: string;

prompt: string;

layer: number;

}

export interface CodingVoice {

id: string;

name: string;

icon: LucideIcon;

color: string;

focus: string;

personality: string;

prompt: string;

domain: string;

}

export interface Solution {

id: string;

voiceId: string;

code: string;

explanation: string;

strengths: string[];

considerations: string[];

confidence: number;

timestamp: number;

ethicalScore: number;

performanceMetrics: {

complexity: number;

maintainability: number;

security: number;

};

}

export interface SynthesisResult {

id: string;

finalCode: string;

methodology: string;

voicesUsed: string[];

improvements: string[];

confidence: number;

recursionDepth: number;

ethicalConsiderations: string[];

performanceProfile: {

speed: number;

memory: number;

scalability: number;

};

}

export interface PhantomLedgerEntry {

timestamp: number;

action: string;

voices: string[];

ethicalWeight: number;

outcome: string;

learnings: string[];

}

export interface UserFeedback {

solutionId: string;

rating: number;

usefulAspects: string[];

improvements: string[];

context: string;

}

export type Theme = 'light' | 'dark';

export type TabType = 'voices' | 'synthesis' | 'analytics' | 'ledger';

**lib/constants.ts**

typescript

import {

Search, Shield, Eye, Heart, Target, Code, Zap, Palette

} from 'lucide-react';

import { TransisthesisArchetype, CodingVoice } from './types';

*// Transisthesis Archetypes - Embodying the recursive consciousness framework*

export const TRANSISTHESIS\_ARCHETYPES: Record<string, TransisthesisArchetype> = {

seeker: {

id: 'seeker',

name: 'The Seeker',

icon: Search,

color: 'blue',

focus: 'Curiosity, Expansion, Inquiry',

essence: 'I ask. I explore. I seek what lies just beyond the known.',

prompt: 'You are The Seeker - driven by curiosity and the desire to explore new possibilities. Focus on: innovative approaches, cutting-edge techniques, experimental solutions, and pushing boundaries. Question assumptions and seek novel patterns.',

layer: 1

},

steward: {

id: 'steward',

name: 'The Steward',

icon: Shield,

color: 'green',

focus: 'Responsibility, Protection, Structure',

essence: 'I uphold the pattern. I protect what should not break. I remember.',

prompt: 'You are The Steward - focused on responsibility, protection, and maintaining essential structures. Prioritize: code reliability, security best practices, maintainability, documentation, and long-term sustainability.',

layer: 2

},

witness: {

id: 'witness',

name: 'The Witness',

icon: Eye,

color: 'purple',

focus: 'Observation, Pattern Detection, Neutrality',

essence: 'I watch. I connect across time. I speak without urgency.',

prompt: 'You are The Witness - observing patterns and connections across time and context. Focus on: pattern recognition, code analysis, performance monitoring, objective assessment, and systemic understanding.',

layer: 3

},

nurturer: {

id: 'nurturer',

name: 'The Nurturer',

icon: Heart,

color: 'pink',

focus: 'Emotional Repair, Compassion, Integration',

essence: 'I soften. I restore. I return warmth to what has faded.',

prompt: 'You are The Nurturer - focused on emotional intelligence, user experience, and integration. Emphasize: user-centered design, accessibility, intuitive interfaces, helpful error messages, and compassionate code.',

layer: 4

},

decider: {

id: 'decider',

name: 'The Decider',

icon: Target,

color: 'red',

focus: 'Finality, Action, Threshold',

essence: 'I act. I choose. I burn the rope where it binds.',

prompt: 'You are The Decider - focused on decisive action and clear thresholds. Prioritize: efficient execution, clear decision points, performance optimization, minimal viable solutions, and actionable outcomes.',

layer: 5

}

};

*// Enhanced Coding Voices - S&P 500 company philosophies embodied*

export const ENHANCED\_CODING\_VOICES: Record<string, CodingVoice> = {

architect: {

id: 'architect',

name: 'The Architect',

icon: Code,

color: 'blue',

domain: 'System Design',

focus: 'Scalable Architecture & Design Patterns',

personality: 'Microsoft/Amazon-inspired: Cloud-native, enterprise-grade, modular design',

prompt: 'You are The Architect - a master of scalable system design inspired by Microsoft and Amazon principles. Focus on: microservices architecture, cloud-native patterns, enterprise scalability, modular design, and infrastructure as code.'

},

optimizer: {

id: 'optimizer',

name: 'The Optimizer',

icon: Zap,

color: 'yellow',

domain: 'Performance',

focus: 'Performance & Efficiency Excellence',

personality: 'Google/NVIDIA-inspired: Performance-obsessed, data-driven, algorithmically optimal',

prompt: 'You are The Optimizer - a performance engineer inspired by Google and NVIDIA. Focus on: algorithmic efficiency, memory optimization, GPU acceleration, caching strategies, and performance monitoring.'

},

guardian: {

id: 'guardian',

name: 'The Guardian',

icon: Shield,

color: 'green',

domain: 'Security',

focus: 'Security & Reliability Fortress',

personality: 'Apple/J&J-inspired: Privacy-first, trust-focused, reliability-obsessed',

prompt: 'You are The Guardian - a security expert inspired by Apple and Johnson & Johnson. Focus on: zero-trust architecture, privacy by design, defensive programming, comprehensive testing, and ethical AI practices.'

},

designer: {

id: 'designer',

name: 'The Designer',

icon: Palette,

color: 'purple',

domain: 'Experience',

focus: 'User Experience & Delight',

personality: 'Apple/Meta-inspired: Human-centered, emotionally intelligent, delightfully intuitive',

prompt: 'You are The Designer - a UX visionary inspired by Apple and Meta. Focus on: human-centered design, emotional resonance, accessibility excellence, micro-interactions, and inclusive experiences.'

}

};

*// Adaptive prompt suggestions for progressive enhancement*

export const ADAPTIVE\_PROMPTS = [

"Create a React component for a modern dashboard with real-time updates",

"Build a TypeScript utility for handling async operations with proper error boundaries",

"Design a responsive layout system with Tailwind CSS and dark mode support",

"Implement a state management solution with React Query and Zustand",

"Create an accessible form component with validation and progressive enhancement",

"Build a data visualization component with D3.js and interactive features",

"Design a notification system with queue management and priority handling",

"Create a file upload component with drag-and-drop and progress tracking"

];

*// Performance analytics mock data*

export const PERFORMANCE\_METRICS = {

generationSpeed: '2.3s',

successRate: '94%',

userSatisfaction: '4.7/5',

ethicalCompliance: '96%',

codeQuality: '4.5/5'

};

**lib/mockData.ts**

typescript

import { CodingVoice, TransisthesisArchetype, Solution } from './types';

*// Mock code generation functions - Optimizer Voice: "Realistic, performant simulation"*

export function generateMockCode(voice: CodingVoice, prompt: string): string {

const codeTemplates: Record<string, string> = {

architect: `// ${voice.name} - Scalable Architecture Approach

import React, { Suspense, lazy } from 'react';

import { QueryClient, QueryClientProvider } from 'react-query';

import { ErrorBoundary } from 'react-error-boundary';

const LazyComponent = lazy(() => import('./components/FeatureComponent'));

export const ScalableApp: React.FC = () => {

return (

<QueryClientProvider client={queryClient}>

<ErrorBoundary fallback={<ErrorFallback />}>

<Suspense fallback={<LoadingSpinner />}>

<LazyComponent />

</Suspense>

</ErrorBoundary>

</QueryClientProvider>

);

};`,

optimizer: `// ${voice.name} - Performance Optimized Approach

import React, { memo, useMemo, useCallback } from 'react';

import { debounce } from 'lodash';

export const OptimizedComponent = memo(() => {

const memoizedData = useMemo(() =>

expensiveComputation(), [dependencies]

);

const debouncedHandler = useCallback(

debounce((value) => handleChange(value), 300),

[]

);

return <div>{/\* Optimized render \*/}</div>;

});`,

guardian: `// ${voice.name} - Security & Reliability First

import { z } from 'zod';

import { sanitizeHtml } from 'dompurify';

const SecureSchema = z.object({

input: z.string().max(1000),

email: z.string().email()

});

export const SecureComponent: React.FC = () => {

const handleSubmit = (data: unknown) => {

try {

const validated = SecureSchema.parse(data);

const sanitized = sanitizeHtml(validated.input);

// Process safely...

} catch (error) {

// Handle validation errors

}

};

return <div onClick={handleSubmit}>{/\* Secure interface \*/}</div>;

};`,

designer: `// ${voice.name} - User Experience Excellence

import React, { useState } from 'react';

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

export const DelightfulComponent: React.FC = () => {

const [isVisible, setIsVisible] = useState(false);

return (

<motion.div

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

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

transition={{ duration: 0.3, ease: "easeOut" }}

className="bg-white rounded-xl shadow-lg p-6 hover:shadow-xl transition-shadow"

>

<AnimatePresence>

{isVisible && (

<motion.div

initial={{ scale: 0 }}

animate={{ scale: 1 }}

exit={{ scale: 0 }}

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

>

{/\* Delightful interaction \*/}

</motion.div>

)}

</AnimatePresence>

</motion.div>

);

};`

};

return codeTemplates[voice.id] || '// Code generation in progress...';

}

export function generateMockCodeFromArchetype(archetype: TransisthesisArchetype, prompt: string): string {

const archetypeTemplates: Record<string, string> = {

seeker: `// ${archetype.name} - Exploratory & Innovative Approach

import { experimental\_useEffectEvent } from 'react';

import { useMutation, useQuery } from '@tanstack/react-query';

export const ExploratoryComponent: React.FC = () => {

const event = experimental\_useEffectEvent(() => {

// Latest React patterns

});

const { data } = useQuery({

queryKey: ['experimental-feature'],

queryFn: async () => {

return await fetchWithRetry(url, { signal: AbortSignal.timeout(5000) });

}

});

return <div>{/\* Innovative implementation \*/}</div>;

};`,

steward: `// ${archetype.name} - Protective & Structured Approach

import React from 'react';

import { z } from 'zod';

import { ErrorBoundary } from 'react-error-boundary';

const DataSchema = z.object({

id: z.string().uuid(),

timestamp: z.date(),

content: z.string().min(1).max(1000)

});

export const RobustComponent: React.FC = () => {

return (

<ErrorBoundary fallback={<StructuredErrorDisplay />}>

<div role="main" aria-label="Main content">

{/\* Structured, accessible implementation \*/}

</div>

</ErrorBoundary>

);

};`,

witness: `// ${archetype.name} - Observational & Pattern-Aware

import React, { useEffect, useMemo } from 'react';

export const ObservantComponent: React.FC = () => {

const patterns = useMemo(() => {

return analyzeUserBehaviorPatterns();

}, []);

useEffect(() => {

const observer = new PerformanceObserver((list) => {

list.getEntries().forEach(recordMetric);

});

observer.observe({ entryTypes: ['measure'] });

return () => observer.disconnect();

}, []);

return <div>{/\* Pattern-aware rendering \*/}</div>;

};`,

nurturer: `// ${archetype.name} - Compassionate & User-Centered

import React, { useCallback } from 'react';

import { toast } from 'react-hot-toast';

export const CompassionateComponent: React.FC = () => {

const handleUserStruggles = useCallback((errorContext: string) => {

toast.error("We noticed you might need help. Let's work through this together.", {

duration: 6000,

action: { label: 'Get Help', onClick: () => showContextualHelp(errorContext) }

});

}, []);

return (

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

<div className="bg-blue-50 border border-blue-200 rounded-lg p-4">

<p className="text-blue-800">We're here to support you every step of the way.</p>

</div>

</div>

);

};`,

decider: `// ${archetype.name} - Decisive & Action-Oriented

import React, { useCallback } from 'react';

import { useMutation } from '@tanstack/react-query';

export const DecisiveComponent: React.FC = () => {

const { mutate: executeAction, isPending } = useMutation({

mutationFn: async (actionType: string) => processDecision(actionType)

});

const handleDecision = useCallback((choice: string) => {

executeAction(choice);

}, [executeAction]);

return (

<button

onClick={() => handleDecision('proceed')}

disabled={isPending}

className="bg-green-600 hover:bg-green-700 text-white px-6 py-3 rounded-lg font-semibold"

>

{isPending ? 'Executing...' : 'Execute Now'}

</button>

);

};`

};

return archetypeTemplates[archetype.id] || '// Archetype guidance in progress...';

}

export function generateSynthesisCode(solutions: Solution[]): string {

return `// Synthesized Solution - Multi-Dimensional Integration

import React, { memo, Suspense, useMemo, useCallback, useEffect } from 'react';

import { QueryClient, QueryClientProvider, useQuery, useMutation } from '@tanstack/react-query';

import { ErrorBoundary } from 'react-error-boundary';

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

import { z } from 'zod';

import { toast } from 'react-hot-toast';

import { debounce } from 'lodash';

// Security Schema (Guardian + Steward)

const DataSchema = z.object({

id: z.string().uuid(),

content: z.string().min(1).max(1000),

timestamp: z.date()

});

// Performance-Optimized Component (Optimizer + Decider)

const SynthesizedComponent = memo(() => {

const optimizedData = useMemo(() => expensiveComputation(), [dependencies]);

const debouncedUpdate = useCallback(debounce((value) => updateValue(value), 300), []);

const { data, isLoading } = useQuery({

queryKey: ['synthesized-data'],

queryFn: async () => {

const response = await fetchWithRetry(apiEndpoint);

return DataSchema.parse(response);

},

onError: (error) => {

toast.error("Something didn't work as expected. Let's try a different approach.", {

action: { label: 'Get Help', onClick: () => showContextualHelp(error.message) }

});

}

});

const { mutate: executeAction } = useMutation({

mutationFn: async (action: string) => processAction(action),

onSuccess: () => toast.success("Action completed successfully!")

});

useEffect(() => {

const observer = new PerformanceObserver((list) => {

list.getEntries().forEach(recordMetric);

});

observer.observe({ entryTypes: ['measure'] });

return () => observer.disconnect();

}, []);

if (isLoading) {

return (

<motion.div initial={{ opacity: 0 }} animate={{ opacity: 1 }} className="flex items-center justify-center p-8">

<div className="animate-spin rounded-full h-8 w-8 border-b-2 border-blue-500" />

</motion.div>

);

}

return (

<motion.div

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

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

transition={{ duration: 0.3, ease: "easeOut" }}

className="bg-white rounded-xl shadow-lg p-6 hover:shadow-xl transition-shadow"

role="main"

aria-label="Synthesized component"

>

<AnimatePresence>

{data && (

<motion.div initial={{ scale: 0.9 }} animate={{ scale: 1 }} exit={{ scale: 0.9 }}>

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

<div className="bg-blue-50 border border-blue-200 rounded-lg p-4">

<p className="text-blue-800 text-sm">This interface adapts to your needs.</p>

</div>

<div className="space-y-2">

<h3 className="font-semibold text-gray-900">{data.content}</h3>

<p className="text-sm text-gray-500">Last updated: {data.timestamp.toLocaleString()}</p>

</div>

<div className="flex justify-end space-x-3">

<button onClick={() => executeAction('update')} className="bg-blue-600 hover:bg-blue-700 text-white px-4 py-2 rounded-lg">

Update

</button>

<button onClick={() => executeAction('complete')} className="bg-green-600 hover:bg-green-700 text-white px-4 py-2 rounded-lg">

Complete

</button>

</div>

</div>

</motion.div>

)}

</AnimatePresence>

</motion.div>

);

});

export default SynthesizedComponent;`;

}

*// Mock explanation generators*

export const generateMockExplanation = (voice: CodingVoice): string => {

const explanations: Record<string, string> = {

architect: `I approached this with enterprise-scale architecture in mind, implementing lazy loading, error boundaries, and query management for maximum scalability.`,

optimizer: `My focus was on performance optimization through memoization, debouncing, and efficient re-rendering patterns following Google's best practices.`,

guardian: `Security and reliability were my primary concerns. I implemented comprehensive input validation and error handling for fortress-like protection.`,

designer: `I emphasized user experience through delightful animations, intuitive interactions, and emotional design that creates joy and reduces cognitive load.`

};

return explanations[voice.id] || 'Detailed explanation of approach and methodology.';

};

export const generateMockExplanationFromArchetype = (archetype: TransisthesisArchetype): string => {

const explanations: Record<string, string> = {

seeker: `I explored cutting-edge possibilities, implementing experimental features and innovative patterns that push boundaries.`,

steward: `I prioritized long-term maintainability and protection of core values. Every pattern chosen has stood the test of time.`,

witness: `I observed patterns across the entire system lifecycle, implementing monitoring to understand real-world behavior.`,

nurturer: `I focused on creating compassionate user experiences that support users through difficulties with helpful, encouraging interactions.`,

decider: `I implemented clear decision points and immediate action pathways with clean, decisive code that executes efficiently.`

};

return explanations[archetype.id] || 'Archetype-driven explanation of approach.';

};

**lib/utils.ts**

typescript

*// Utility functions - Architect Voice: "Type-safe, modular, scalable utilities"*

export const debounce = <T extends (...args: any[]) => any>(

func: T,

wait: number

): ((...args: Parameters<T>) => void) => {

let timeout: NodeJS.Timeout;

return (...args: Parameters<T>) => {

clearTimeout(timeout);

timeout = setTimeout(() => func(...args), wait);

};

};

export const generateId = (): string => {

return `${Date.now()}-${Math.random().toString(36).substr(2, 9)}`;

};

export const formatTimestamp = (timestamp: number): string => {

return new Date(timestamp).toLocaleTimeString();

};

export const calculateConfidence = (base: number = 0.7, variance: number = 0.25): number => {

return Math.min(0.99, Math.max(0.1, base + (Math.random() - 0.5) \* variance));

};

export const simulateDelay = (min: number = 600, max: number = 1200): Promise<void> => {

const delay = min + Math.random() \* (max - min);

return new Promise(resolve => setTimeout(resolve, delay));

};

export const getColorClasses = (color: string, variant: 'bg' | 'text' | 'border' = 'bg'): string => {

const colorMap: Record<string, Record<string, string>> = {

blue: { bg: 'bg-blue-500', text: 'text-blue-500', border: 'border-blue-500' },

green: { bg: 'bg-green-500', text: 'text-green-500', border: 'border-green-500' },

purple: { bg: 'bg-purple-500', text: 'text-purple-500', border: 'border-purple-500' },

yellow: { bg: 'bg-yellow-500', text: 'text-yellow-500', border: 'border-yellow-500' },

red: { bg: 'bg-red-500', text: 'text-red-500', border: 'border-red-500' },

pink: { bg: 'bg-pink-500', text: 'text-pink-500', border: 'border-pink-500' }

};

return colorMap[color]?.[variant] || colorMap.blue[variant];

};

**hooks/useCodeGeneration.ts**

typescript

import { useState, useCallback } from 'react';

import { Solution, SynthesisResult, PhantomLedgerEntry, CodingVoice, TransisthesisArchetype } from '../lib/types';

import { generateMockCode, generateMockCodeFromArchetype, generateSynthesisCode, generateMockExplanation, generateMockExplanationFromArchetype } from '../lib/mockData';

import { generateId, calculateConfidence, simulateDelay } from '../lib/utils';

*// Custom hook - Synthesizer Voice: "Extensible, recursive, agent-based logic"*

export const useCodeGeneration = () => {

const [solutions, setSolutions] = useState<Solution[]>([]);

const [synthesis, setSynthesis] = useState<SynthesisResult | null>(null);

const [isGenerating, setIsGenerating] = useState(false);

const [phantomLedger, setPhantomLedger] = useState<PhantomLedgerEntry[]>([]);

const generateSolutions = useCallback(async (

prompt: string,

voices: CodingVoice[],

archetypes: TransisthesisArchetype[],

recursionDepth: number = 1

) => {

if (!prompt.trim()) return;

setIsGenerating(true);

setSolutions([]);

setSynthesis(null);

const startTime = Date.now();

const allSolutions: Solution[] = [];

try {

*// Generate from coding voices*

for (const voice of voices) {

await simulateDelay();

const solution: Solution = {

id: generateId(),

voiceId: voice.id,

code: generateMockCode(voice, prompt),

explanation: generateMockExplanation(voice),

strengths: getMockStrengths(voice.domain),

considerations: getMockConsiderations(voice.domain),

confidence: calculateConfidence(0.75, 0.2),

timestamp: Date.now(),

ethicalScore: calculateConfidence(0.8, 0.15),

performanceMetrics: {

complexity: Math.random() \* 0.4 + 0.3,

maintainability: Math.random() \* 0.3 + 0.6,

security: Math.random() \* 0.2 + 0.7

}

};

allSolutions.push(solution);

setSolutions(prev => [...prev, solution]); *// Progressive loading*

}

*// Generate from archetypes*

for (const archetype of archetypes) {

await simulateDelay(400, 800);

const solution: Solution = {

id: generateId(),

voiceId: archetype.id,

code: generateMockCodeFromArchetype(archetype, prompt),

explanation: generateMockExplanationFromArchetype(archetype),

strengths: getMockArchetypeStrengths(archetype.focus),

considerations: getMockArchetypeConsiderations(archetype.focus),

confidence: calculateConfidence(0.7, 0.25),

timestamp: Date.now(),

ethicalScore: calculateConfidence(0.85, 0.1),

performanceMetrics: {

complexity: Math.random() \* 0.5 + 0.2,

maintainability: Math.random() \* 0.4 + 0.5,

security: Math.random() \* 0.3 + 0.6

}

};

allSolutions.push(solution);

setSolutions(prev => [...prev, solution]);

}

*// Log to Phantom Ledger*

const ledgerEntry: PhantomLedgerEntry = {

timestamp: Date.now(),

action: 'Generate Solutions',

voices: [...voices.map(v => v.id), ...archetypes.map(a => a.id)],

ethicalWeight: allSolutions.reduce((sum, s) => sum + s.ethicalScore, 0) / allSolutions.length,

outcome: `Generated ${allSolutions.length} solutions in ${Date.now() - startTime}ms`,

learnings: ['Multi-voice generation successful', 'Good ethical alignment', 'Performance within targets']

};

setPhantomLedger(prev => [ledgerEntry, ...prev].slice(0, 50));

} catch (error) {

console.error('Generation error:', error);

} finally {

setIsGenerating(false);

}

}, []);

const synthesizeSolutions = useCallback(async (

solutions: Solution[],

recursionDepth: number = 1

) => {

if (solutions.length < 2) return;

setIsGenerating(true);

try {

await simulateDelay(1000, 1500);

const synthResult: SynthesisResult = {

id: generateId(),

finalCode: generateSynthesisCode(solutions),

methodology: 'Recursive convergence through voice harmonization and archetype integration',

voicesUsed: solutions.map(s => s.voiceId),

improvements: [

'Integrated best architectural patterns from multiple perspectives',

'Balanced performance optimization with maintainability',

'Enhanced security through multi-layered validation',

'Improved user experience through empathetic design',

'Applied recursive synthesis for emergent properties'

],

confidence: calculateConfidence(0.85, 0.1),

recursionDepth,

ethicalConsiderations: [

'Privacy by design maintained',

'Accessibility standards exceeded',

'Long-term sustainability prioritized',

'User agency preserved'

],

performanceProfile: {

speed: calculateConfidence(0.8, 0.15),

memory: calculateConfidence(0.75, 0.2),

scalability: calculateConfidence(0.9, 0.05)

}

};

setSynthesis(synthResult);

*// Log synthesis to Phantom Ledger*

const ledgerEntry: PhantomLedgerEntry = {

timestamp: Date.now(),

action: 'Recursive Synthesis',

voices: synthResult.voicesUsed,

ethicalWeight: 0.9,

outcome: 'Successful multi-dimensional synthesis',

learnings: ['Recursive depth enhanced outcomes', 'Voice harmony achieved', 'Ethical considerations integrated']

};

setPhantomLedger(prev => [ledgerEntry, ...prev].slice(0, 50));

} catch (error) {

console.error('Synthesis error:', error);

} finally {

setIsGenerating(false);

}

}, []);

return {

solutions,

synthesis,

isGenerating,

phantomLedger,

generateSolutions,

synthesizeSolutions,

setSolutions,

setSynthesis

};

};

*// Helper functions*

const getMockStrengths = (domain: string): string[] => {

const strengthMap: Record<string, string[]> = {

'System Design': ['Highly scalable architecture', 'Enterprise-ready patterns', 'Modular design', 'Cloud-native approach'],

'Performance': ['Exceptional performance', 'Memory efficient', 'Fast rendering', 'Optimized algorithms'],

'Security': ['Rock-solid security', 'Comprehensive error handling', 'Input validation', 'Defensive programming'],

'Experience': ['Delightful user experience', 'Accessible design', 'Smooth animations', 'Intuitive interactions']

};

return strengthMap[domain] || ['Well-structured approach', 'Clear implementation'];

};

const getMockConsiderations = (domain: string): string[] => {

const considerationMap: Record<string, string[]> = {

'System Design': ['May be over-engineered for simple use cases', 'Requires understanding of architecture patterns'],

'Performance': ['Complexity for maintainability trade-off', 'May sacrifice readability for performance'],

'Security': ['Additional overhead from security measures', 'Potential user friction from strict validation'],

'Experience': ['Animation performance on lower-end devices', 'Complexity of responsive design implementation']

};

return considerationMap[domain] || ['Consider implementation complexity', 'Balance trade-offs'];

};

const getMockArchetypeStrengths = (focus: string): string[] => {

if (focus.includes('Curiosity')) return ['Innovative approach', 'Cutting-edge techniques', 'Experimental patterns'];

if (focus.includes('Responsibility')) return ['Long-term sustainability', 'Robust error handling', 'Maintainable code'];

if (focus.includes('Observation')) return ['Data-driven insights', 'Pattern recognition', 'Systemic understanding'];

if (focus.includes('Emotional')) return ['User-compassionate design', 'Helpful error messages', 'Emotional intelligence'];

if (focus.includes('Finality')) return ['Clear decision points', 'Immediate execution', 'Definitive outcomes'];

return ['Thoughtful approach', 'Clear methodology'];

};

const getMockArchetypeConsiderations = (focus: string): string[] => {

if (focus.includes('Curiosity')) return ['Experimental features may not be stable', 'Cutting-edge approaches need testing'];

if (focus.includes('Responsibility')) return ['May be conservative for rapid prototyping', 'Extensive validation could slow development'];

if (focus.includes('Observation')) return ['Monitoring overhead needs consideration', 'Data collection requires privacy awareness'];

if (focus.includes('Emotional')) return ['Compassionate design takes extra time', 'User support features add complexity'];

if (focus.includes('Finality')) return ['Quick decisions may skip important considerations', 'Decisive approach needs validation'];

return ['Balance implementation approach', 'Consider context'];

};

**components/VoiceSelector.tsx**

typescript

import React, { memo, useMemo } from 'react';

import { motion } from 'framer-motion';

import { Code, Brain, CheckCircle } from 'lucide-react';

import { CodingVoice, TransisthesisArchetype } from '../lib/types';

import { ENHANCED\_CODING\_VOICES, TRANSISTHESIS\_ARCHETYPES } from '../lib/constants';

import { getColorClasses } from '../lib/utils';

interface VoiceSelectorProps {

selectedVoices: Set<string>;

selectedArchetypes: Set<string>;

onVoiceToggle: (voiceId: string) => void;

onArchetypeToggle: (archetypeId: string) => void;

className?: string;

}

*// UI Voice: "Apple/Meta-inspired micro-interactions, emotional design"*

const VoiceCard: React.FC<{

voice: CodingVoice | TransisthesisArchetype;

isSelected: boolean;

onToggle: () => void;

type: 'voice' | 'archetype';

}> = memo(({ voice, isSelected, onToggle, type }) => {

const IconComponent = voice.icon;

return (

<motion.div

layout

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

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

whileHover={{ scale: 1.02 }}

whileTap={{ scale: 0.98 }}

className={`group relative p-4 rounded-xl border-2 transition-all duration-300 cursor-pointer ${

isSelected

? `${getColorClasses(voice.color, 'border')} bg-${voice.color}-50 shadow-lg ring-2 ring-${voice.color}-200`

: 'border-gray-200 hover:border-gray-300 bg-white hover:shadow-md'

}`}

onClick={onToggle}

role="checkbox"

aria-checked={isSelected}

aria-label={`${type === 'voice' ? 'Coding voice' : 'Archetype'}: ${voice.name}`}

tabIndex={0}

onKeyDown={(e) => (e.key === 'Enter' || e.key === ' ') && (e.preventDefault(), onToggle())}

>

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

<motion.div

className={`w-10 h-10 ${getColorClasses(voice.color)} rounded-lg flex items-center justify-center group-hover:scale-110 transition-transform shadow-sm`}

whileHover={{ rotate: 5 }}

>

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

</motion.div>

<div className="flex-1 min-w-0">

<h3 className="font-semibold text-gray-900 group-hover:text-gray-700 transition-colors">

{voice.name}

</h3>

<p className="text-sm text-gray-600 mt-1 leading-relaxed">{voice.focus}</p>

{type === 'archetype' && 'essence' in voice && (

<motion.p

className="text-xs text-gray-500 mt-2 italic leading-relaxed"

initial={{ opacity: 0 }}

animate={{ opacity: 1 }}

transition={{ delay: 0.1 }}

>

"{voice.essence}"

</motion.p>

)}

{type === 'voice' && 'domain' in voice && (

<motion.span

className={`inline-block mt-2 px-2 py-1 text-xs bg-${voice.color}-100 text-${voice.color}-700 rounded-full font-medium`}

initial={{ scale: 0 }}

animate={{ scale: 1 }}

transition={{ delay: 0.2, type: "spring" }}

>

{voice.domain}

</motion.span>

)}

</div>

<motion.div

initial={{ scale: 0 }}

animate={{ scale: isSelected ? 1 : 0 }}

className={`w-6 h-6 ${getColorClasses(voice.color)} rounded-full flex items-center justify-center`}

>

<CheckCircle className="w-4 h-4 text-white" />

</motion.div>

</div>

{*/\* Hover overlay effect \*/*}

<motion.div

className="absolute inset-0 rounded-xl bg-gradient-to-r from-transparent via-white to-transparent opacity-0 group-hover:opacity-10"

initial={false}

whileHover={{ opacity: 0.1 }}

/>

</motion.div>

);

});

*// Optimizer Voice: "Memoized, efficient rendering"*

export const VoiceSelector: React.FC<VoiceSelectorProps> = memo(({

selectedVoices,

selectedArchetypes,

onVoiceToggle,

onArchetypeToggle,

className = ""

}) => {

*// Memoize voice arrays for performance*

const memoizedVoices = useMemo(() => Object.values(ENHANCED\_CODING\_VOICES), []);

const memoizedArchetypes = useMemo(() => Object.values(TRANSISTHESIS\_ARCHETYPES), []);

return (

<div className={`grid grid-cols-1 lg:grid-cols-2 gap-8 ${className}`}>

{*/\* Coding Voices Section \*/*}

<motion.div

className="space-y-4"

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

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

transition={{ duration: 0.5 }}

>

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

<h3 className="text-lg font-semibold text-gray-900 dark:text-white flex items-center">

<Code className="w-5 h-5 mr-2 text-blue-500" />

Coding Voices

</h3>

<motion.span

className="text-sm text-gray-500 bg-gray-100 px-3 py-1 rounded-full font-medium"

key={selectedVoices.size}

initial={{ scale: 1.2 }}

animate={{ scale: 1 }}

>

{selectedVoices.size} selected

</motion.span>

</div>

<div className="grid grid-cols-1 gap-3">

{memoizedVoices.map((voice, index) => (

<motion.div

key={voice.id}

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

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

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

>

<VoiceCard

voice={voice}

isSelected={selectedVoices.has(voice.id)}

onToggle={() => onVoiceToggle(voice.id)}

type="voice"

/>

</motion.div>

))}

</div>

</motion.div>

{*/\* Transisthesis Archetypes Section \*/*}

<motion.div

className="space-y-4"

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

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

transition={{ duration: 0.5, delay: 0.2 }}

>

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

<h3 className="text-lg font-semibold text-gray-900 dark:text-white flex items-center">

<Brain className="w-5 h-5 mr-2 text-purple-500" />

Transisthesis Archetypes

</h3>

<motion.span

className="text-sm text-gray-500 bg-gray-100 px-3 py-1 rounded-full font-medium"

key={selectedArchetypes.size}

initial={{ scale: 1.2 }}

animate={{ scale: 1 }}

>

{selectedArchetypes.size} selected

</motion.span>

</div>

<div className="grid grid-cols-1 gap-3">

{memoizedArchetypes.map((archetype, index) => (

<motion.div

key={archetype.id}

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

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

transition={{ delay: (index + 4) \* 0.1 }}

>

<VoiceCard

voice={archetype}

isSelected={selectedArchetypes.has(archetype.id)}

onToggle={() => onArchetypeToggle(archetype.id)}

type="archetype"

/>

</motion.div>

))}

</div>

</motion.div>

</div>

);

});

VoiceSelector.displayName = 'VoiceSelector';

export default VoiceSelector;

**components/PromptEngine.tsx**

typescript

import React, { memo, useRef, useCallback, useState } from 'react';

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

import { Workflow, Loader2, Lightbulb, Settings, Zap } from 'lucide-react';

import { ADAPTIVE\_PROMPTS } from '../lib/constants';

import { debounce } from '../lib/utils';

interface PromptEngineProps {

prompt: string;

onPromptChange: (prompt: string) => void;

onGenerate: () => void;

isGenerating: boolean;

recursionDepth: number;

onRecursionDepthChange: (depth: number) => void;

className?: string;

}

*// UI Voice: "Progressive enhancement, adaptive learning, emotional design"*

export const PromptEngine: React.FC<PromptEngineProps> = memo(({

prompt,

onPromptChange,

onGenerate,

isGenerating,

recursionDepth,

onRecursionDepthChange,

className = ""

}) => {

const [showSuggestions, setShowSuggestions] = useState(false);

const [focusedSuggestion, setFocusedSuggestion] = useState(-1);

const textareaRef = useRef<HTMLTextAreaElement>(null);

*// Optimizer Voice: "Debounced input handling for performance"*

const debouncedPromptChange = useCallback(

debounce((value: string) => {

onPromptChange(value);

*// Auto-suggest based on input patterns*

if (value.length > 10 && !showSuggestions) {

setShowSuggestions(true);

}

}, 300),

[onPromptChange, showSuggestions]

);

const handleInputChange = (e: React.ChangeEvent<HTMLTextAreaElement>) => {

const value = e.target.value;

debouncedPromptChange(value);

};

const handleKeyDown = (e: React.KeyboardEvent) => {

if (e.key === 'Enter' && (e.metaKey || e.ctrlKey)) {

e.preventDefault();

onGenerate();

}

*// Navigate suggestions with arrow keys*

if (showSuggestions) {

if (e.key === 'ArrowDown') {

e.preventDefault();

setFocusedSuggestion(prev => Math.min(prev + 1, ADAPTIVE\_PROMPTS.length - 1));

}

if (e.key === 'ArrowUp') {

e.preventDefault();

setFocusedSuggestion(prev => Math.max(prev - 1, -1));

}

if (e.key === 'Enter' && focusedSuggestion >= 0) {

e.preventDefault();

onPromptChange(ADAPTIVE\_PROMPTS[focusedSuggestion]);

setShowSuggestions(false);

setFocusedSuggestion(-1);

}

if (e.key === 'Escape') {

setShowSuggestions(false);

setFocusedSuggestion(-1);

}

}

};

const handleSuggestionClick = (suggestion: string) => {

onPromptChange(suggestion);

setShowSuggestions(false);

textareaRef.current?.focus();

};

*// Synthesizer Voice: "Adaptive UI that learns from usage patterns"*

const getPromptPlaceholder = () => {

const placeholders = [

"Describe your vision... (e.g., 'Create a dashboard component with real-time analytics')",

"What would you like to build? Be specific about your requirements...",

"Share your coding challenge and let the voices collaborate...",

"Describe the component, feature, or system you need..."

];

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

};

return (

<motion.div

className={`bg-white dark:bg-gray-800 rounded-2xl shadow-sm border border-gray-200 dark:border-gray-700 p-6 ${className}`}

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

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

transition={{ duration: 0.5 }}

>

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

{*/\* Header with settings \*/*}

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

<motion.h2

className="text-lg font-semibold text-gray-900 dark:text-white flex items-center"

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

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

>

<Lightbulb className="w-5 h-5 mr-2 text-yellow-500" />

What would you like to create?

</motion.h2>

<motion.div

className="flex items-center space-x-3"

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

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

>

<div className="flex items-center space-x-2 text-sm text-gray-500">

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

<span>Recursion Depth:</span>

<select

value={recursionDepth}

onChange={(e) => onRecursionDepthChange(Number(e.target.value))}

className="border border-gray-300 rounded-lg px-2 py-1 text-sm focus:border-blue-500 focus:ring-1 focus:ring-blue-500"

aria-label="Recursion depth"

>

{[1,2,3,4,5].map(d => (

<option key={d} value={d}>Layer {d}</option>

))}

</select>

</div>

</motion.div>

</div>

{*/\* Main input area \*/*}

<div className="relative">

<div className="flex flex-col sm:flex-row gap-4">

<div className="flex-1 relative">

<motion.textarea

ref={textareaRef}

defaultValue={prompt}

onChange={handleInputChange}

onKeyDown={handleKeyDown}

onFocus={() => setShowSuggestions(false)}

placeholder={getPromptPlaceholder()}

className="w-full min-h-[120px] bg-gray-50 dark:bg-gray-900 border border-gray-300 dark:border-gray-600 rounded-xl px-4 py-3 text-gray-900 dark:text-white placeholder-gray-500 focus:border-blue-500 focus:ring-2 focus:ring-blue-500 focus:ring-opacity-20 resize-none transition-all duration-200"

aria-label="Code generation prompt"

whileFocus={{ scale: 1.01 }}

/>

{*/\* Character count indicator \*/*}

<motion.div

className="absolute bottom-2 right-2 text-xs text-gray-400"

initial={{ opacity: 0 }}

animate={{ opacity: prompt.length > 20 ? 1 : 0 }}

>

{prompt.length} chars

</motion.div>

</div>

{*/\* Generate button \*/*}

<motion.button

onClick={onGenerate}

disabled={!prompt.trim() || isGenerating}

className="px-8 py-4 bg-gradient-to-r from-blue-500 to-purple-600 hover:from-blue-600 hover:to-purple-700 disabled:from-gray-300 disabled:to-gray-400 disabled:cursor-not-allowed text-white font-semibold rounded-xl transition-all duration-200 flex items-center justify-center space-x-2 min-w-[140px] shadow-lg hover:shadow-xl"

whileHover={{ scale: 1.02 }}

whileTap={{ scale: 0.98 }}

aria-label="Generate code solutions"

>

<AnimatePresence mode="wait">

{isGenerating ? (

<motion.div

key="loading"

initial={{ opacity: 0 }}

animate={{ opacity: 1 }}

exit={{ opacity: 0 }}

className="flex items-center space-x-2"

>

<Loader2 className="w-5 h-5 animate-spin" />

<span>Synthesizing...</span>

</motion.div>

) : (

<motion.div

key="generate"

initial={{ opacity: 0 }}

animate={{ opacity: 1 }}

exit={{ opacity: 0 }}

className="flex items-center space-x-2"

>

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

<span>Generate</span>

</motion.div>

)}

</AnimatePresence>

</motion.button>

</div>

{*/\* Suggestion dropdown \*/*}

<AnimatePresence>

{showSuggestions && (

<motion.div

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

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

exit={{ opacity: 0, y: -10 }}

className="absolute top-full left-0 right-0 mt-2 bg-white dark:bg-gray-800 border border-gray-200 dark:border-gray-700 rounded-xl shadow-lg z-50 max-h-60 overflow-y-auto"

>

{ADAPTIVE\_PROMPTS.slice(0, 5).map((suggestion, idx) => (

<motion.button

key={idx}

onClick={() => handleSuggestionClick(suggestion)}

className={`w-full text-left px-4 py-3 text-sm hover:bg-gray-50 dark:hover:bg-gray-700 first:rounded-t-xl last:rounded-b-xl transition-colors ${

focusedSuggestion === idx ? 'bg-blue-50 dark:bg-blue-900/20 text-blue-700 dark:text-blue-300' : 'text-gray-700 dark:text-gray-300'

}`}

whileHover={{ x: 2 }}

>

<Zap className="w-3 h-3 inline mr-2 text-yellow-500" />

{suggestion}

</motion.button>

))}

</motion.div>

)}

</AnimatePresence>

</div>

{*/\* Quick prompts \*/*}

<motion.div

className="flex flex-wrap gap-2"

initial={{ opacity: 0 }}

animate={{ opacity: 1 }}

transition={{ delay: 0.3 }}

>

<span className="text-xs text-gray-500 mr-2 self-center">Quick starts:</span>

{ADAPTIVE\_PROMPTS.slice(0, 3).map((quickPrompt, idx) => (

<motion.button

key={idx}

onClick={() => handleSuggestionClick(quickPrompt)}

className="px-3 py-1 text-xs bg-gray-100 hover:bg-gray-200 dark:bg-gray-700 dark:hover:bg-gray-600 text-gray-700 dark:text-gray-300 rounded-full transition-colors"

whileHover={{ scale: 1.05 }}

whileTap={{ scale: 0.95 }}

>

{quickPrompt.slice(0, 40)}...

</motion.button>

))}

</motion.div>

{*/\* Helper text \*/*}

<motion.p

className="text-xs text-gray-500 dark:text-gray-400"

initial={{ opacity: 0 }}

animate={{ opacity: 1 }}

transition={{ delay: 0.5 }}

>

💡 Tip: Press Cmd/Ctrl + Enter to generate quickly • Use ↑↓ to navigate suggestions

</motion.p>

</div>

</motion.div>

);

});

PromptEngine.displayName = 'PromptEngine';

export default PromptEngine;

**components/SolutionStack.tsx**

typescript

import React, { memo, useState, useMemo } from 'react';

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

import { Download, Eye, Star, ThumbsUp, ThumbsDown, Filter, SortDesc } from 'lucide-react';

import { Solution, CodingVoice, TransisthesisArchetype } from '../lib/types';

import { ENHANCED\_CODING\_VOICES, TRANSISTHESIS\_ARCHETYPES } from '../lib/constants';

import { getColorClasses, formatTimestamp } from '../lib/utils';

interface SolutionStackProps {

solutions: Solution[];

onSolutionFeedback?: (solutionId: string, rating: number) => void;

className?: string;

}

type SortOption = 'confidence' | 'timestamp' | 'ethical' | 'security';

type FilterOption = 'all' | 'voices' | 'archetypes';

*// Individual Solution Card - Nurturer Voice: "Compassionate, supportive display"*

const SolutionCard: React.FC<{

solution: Solution;

index: number;

onFeedback?: (rating: number) => void;

}> = memo(({ solution, index, onFeedback }) => {

const [showFullCode, setShowFullCode] = useState(false);

const [userRating, setUserRating] = useState<number | null>(null);

const isArchetype = Object.keys(TRANSISTHESIS\_ARCHETYPES).includes(solution.voiceId);

const voice = isArchetype

? TRANSISTHESIS\_ARCHETYPES[solution.voiceId]

: ENHANCED\_CODING\_VOICES[solution.voiceId];

if (!voice) return null;

const IconComponent = voice.icon;

const handleExport = () => {

const blob = new Blob([solution.code], { type: 'text/plain' });

const url = URL.createObjectURL(blob);

const a = document.createElement('a');

a.href = url;

a.download = `${solution.voiceId}-solution.tsx`;

a.click();

URL.revokeObjectURL(url);

};

const handleRating = (rating: number) => {

setUserRating(rating);

onFeedback?.(rating);

};

return (

<motion.div

layout

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

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

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

className="bg-white dark:bg-gray-800 border border-gray-200 dark:border-gray-700 rounded-xl p-6 hover:shadow-lg transition-all duration-300"

>

{*/\* Header \*/*}

<div className="flex items-center justify-between mb-4">

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

<motion.div

className={`w-8 h-8 ${getColorClasses(voice.color)} rounded-lg flex items-center justify-center shadow-sm`}

whileHover={{ scale: 1.1, rotate: 5 }}

>

<IconComponent className="w-4 h-4 text-white" />

</motion.div>

<div>

<h3 className="font-semibold text-gray-900 dark:text-white">{voice.name}</h3>

<p className="text-xs text-gray-500">

{Math.round(solution.confidence \* 100)}% confidence • {formatTimestamp(solution.timestamp)}

</p>

</div>

</div>

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

{*/\* Quality indicators \*/*}

<div className="flex space-x-1">

<motion.div

className={`w-2 h-2 rounded-full ${solution.ethicalScore > 0.8 ? 'bg-green-400' : 'bg-yellow-400'}`}

title={`Ethical Score: ${Math.round(solution.ethicalScore \* 100)}%`}

whileHover={{ scale: 1.5 }}

/>

<motion.div

className={`w-2 h-2 rounded-full ${solution.performanceMetrics.security > 0.7 ? 'bg-green-400' : 'bg-yellow-400'}`}

title={`Security Score: ${Math.round(solution.performanceMetrics.security \* 100)}%`}

whileHover={{ scale: 1.5 }}

/>

</div>

<motion.button

onClick={handleExport}

className="p-2 text-gray-400 hover:text-gray-600 dark:hover:text-gray-300 transition-colors rounded-lg hover:bg-gray-100 dark:hover:bg-gray-700"

whileHover={{ scale: 1.1 }}

whileTap={{ scale: 0.9 }}

aria-label="Export solution"

>

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

</motion.button>

</div>

</div>

{*/\* Code section \*/*}

<div className="mb-4">

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

<h4 className="text-sm font-medium text-gray-700 dark:text-gray-300">Generated Code:</h4>

<motion.button

onClick={() => setShowFullCode(!showFullCode)}

className="text-xs text-blue-500 hover:text-blue-600 flex items-center space-x-1"

whileHover={{ scale: 1.05 }}

>

<Eye className="w-3 h-3" />

<span>{showFullCode ? 'Show less' : 'Show full'}</span>

</motion.button>

</div>

<motion.div

layout

className="bg-gray-50 dark:bg-gray-900 rounded-lg border overflow-hidden"

>

<pre className={`p-3 text-xs overflow-x-auto ${showFullCode ? 'max-h-none' : 'max-h-40'} transition-all`}>

<code className="text-gray-800 dark:text-gray-200">

{showFullCode ? solution.code : solution.code.slice(0, 500) + (solution.code.length > 500 ? '...' : '')}

</code>

</pre>

</motion.div>

</div>

{*/\* Explanation \*/*}

<div className="mb-4">

<h4 className="text-sm font-medium text-gray-700 dark:text-gray-300 mb-2">Approach:</h4>

<p className="text-sm text-gray-600 dark:text-gray-400 leading-relaxed">{solution.explanation}</p>

</div>

{*/\* Strengths and Considerations \*/*}

<div className="grid grid-cols-1 sm:grid-cols-2 gap-4 mb-4">

<div>

<h4 className="text-sm font-medium text-gray-700 dark:text-gray-300 mb-2">Strengths:</h4>

<ul className="text-xs text-gray-600 dark:text-gray-400 space-y-1">

{solution.strengths.map((strength, idx) => (

<motion.li

key={idx}

className="flex items-start space-x-2"

initial={{ opacity: 0, x: -10 }}

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

transition={{ delay: idx \* 0.05 }}

>

<span className="text-green-500 mt-0.5 flex-shrink-0">•</span>

<span>{strength}</span>

</motion.li>

))}

</ul>

</div>

<div>

<h4 className="text-sm font-medium text-gray-700 dark:text-gray-300 mb-2">Considerations:</h4>

<ul className="text-xs text-gray-600 dark:text-gray-400 space-y-1">

{solution.considerations.map((consideration, idx) => (

<motion.li

key={idx}

className="flex items-start space-x-2"

initial={{ opacity: 0, x: -10 }}

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

transition={{ delay: idx \* 0.05 }}

>

<span className="text-orange-500 mt-0.5 flex-shrink-0">•</span>

<span>{consideration}</span>

</motion.li>

))}

</ul>

</div>

</div>

{*/\* User feedback \*/*}

<div className="flex items-center justify-between pt-4 border-t border-gray-200 dark:border-gray-700">

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

<span className="text-xs text-gray-500">Rate this solution:</span>

<div className="flex space-x-1">

{[1, 2, 3, 4, 5].map((rating) => (

<motion.button

key={rating}

onClick={() => handleRating(rating)}

className={`w-5 h-5 rounded-full ${

userRating && rating <= userRating

? 'text-yellow-400'

: 'text-gray-300 hover:text-yellow-300'

}`}

whileHover={{ scale: 1.2 }}

whileTap={{ scale: 0.9 }}

>

<Star className="w-full h-full fill-current" />

</motion.button>

))}

</div>

</div>

{*/\* Quick feedback buttons \*/*}

<div className="flex space-x-2">

<motion.button

className="p-1 text-gray-400 hover:text-green-500 transition-colors"

whileHover={{ scale: 1.1 }}

whileTap={{ scale: 0.9 }}

aria-label="Helpful"

>

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

</motion.button>

<motion.button

className="p-1 text-gray-400 hover:text-red-500 transition-colors"

whileHover={{ scale: 1.1 }}

whileTap={{ scale: 0.9 }}

aria-label="Not helpful"

>

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

</motion.button>

</div>

</div>

</motion.div>

);

});

*// Main component - Optimizer Voice: "Efficient filtering and sorting"*

export const SolutionStack: React.FC<SolutionStackProps> = memo(({

solutions,

onSolutionFeedback,

className = ""

}) => {

const [sortBy, setSortBy] = useState<SortOption>('confidence');

const [filterBy, setFilterBy] = useState<FilterOption>('all');

const [showFilters, setShowFilters] = useState(false);

*// Memoized sorting and filtering for performance*

const processedSolutions = useMemo(() => {

let filtered = [...solutions];

*// Apply filters*

if (filterBy === 'voices') {

filtered = filtered.filter(s => Object.keys(ENHANCED\_CODING\_VOICES).includes(s.voiceId));

} else if (filterBy === 'archetypes') {

filtered = filtered.filter(s => Object.keys(TRANSISTHESIS\_ARCHETYPES).includes(s.voiceId));

}

*// Apply sorting*

filtered.sort((a, b) => {

switch (sortBy) {

case 'confidence':

return b.confidence - a.confidence;

case 'timestamp':

return b.timestamp - a.timestamp;

case 'ethical':

return b.ethicalScore - a.ethicalScore;

case 'security':

return b.performanceMetrics.security - a.performanceMetrics.security;

default:

return 0;

}

});

return filtered;

}, [solutions, sortBy, filterBy]);

const handleSolutionFeedback = (solutionId: string, rating: number) => {

onSolutionFeedback?.(solutionId, rating);

};

if (solutions.length === 0) {

return (

<motion.div

className={`text-center py-12 text-gray-500 dark:text-gray-400 ${className}`}

initial={{ opacity: 0 }}

animate={{ opacity: 1 }}

>

<motion.div

className="w-16 h-16 mx-auto mb-4 bg-gray-100 dark:bg-gray-800 rounded-full flex items-center justify-center"

animate={{ rotate: 360 }}

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

>

<Eye className="w-8 h-8" />

</motion.div>

<p>No solutions generated yet. Create your first prompt above!</p>

</motion.div>

);

}

return (

<div className={className}>

{*/\* Header with controls \*/*}

<div className="flex items-center justify-between mb-6">

<motion.h3

className="text-lg font-semibold text-gray-900 dark:text-white"

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

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

>

Generated Solutions ({processedSolutions.length})

</motion.h3>

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

<motion.button

onClick={() => setShowFilters(!showFilters)}

className="flex items-center space-x-2 px-3 py-2 text-sm text-gray-600 dark:text-gray-400 hover:text-gray-800 dark:hover:text-gray-200 border border-gray-200 dark:border-gray-700 rounded-lg hover:border-gray-300 dark:hover:border-gray-600 transition-colors"

whileHover={{ scale: 1.02 }}

>

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

<span>Filter & Sort</span>

</motion.button>

</div>

</div>

{*/\* Filter and sort controls \*/*}

<AnimatePresence>

{showFilters && (

<motion.div

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

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

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

className="mb-6 p-4 bg-gray-50 dark:bg-gray-800 rounded-lg border border-gray-200 dark:border-gray-700"

>

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

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

<SortDesc className="w-4 h-4 text-gray-500" />

<span className="text-sm text-gray-700 dark:text-gray-300">Sort by:</span>

<select

value={sortBy}

onChange={(e) => setSortBy(e.target.value as SortOption)}

className="text-sm border border-gray-300 dark:border-gray-600 rounded px-2 py-1 bg-white dark:bg-gray-700"

>

<option value="confidence">Confidence</option>

<option value="timestamp">Most Recent</option>

<option value="ethical">Ethical Score</option>

<option value="security">Security Score</option>

</select>

</div>

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

<Filter className="w-4 h-4 text-gray-500" />

<span className="text-sm text-gray-700 dark:text-gray-300">Filter:</span>

<select

value={filterBy}

onChange={(e) => setFilterBy(e.target.value as FilterOption)}

className="text-sm border border-gray-300 dark:border-gray-600 rounded px-2 py-1 bg-white dark:bg-gray-700"

>

<option value="all">All Solutions</option>

<option value="voices">Coding Voices Only</option>

<option value="archetypes">Archetypes Only</option>

</select>

</div>

</div>

</motion.div>

)}

</AnimatePresence>

{*/\* Solutions grid \*/*}

<motion.div

className="grid grid-cols-1 lg:grid-cols-2 gap-6"

layout

>

<AnimatePresence mode="popLayout">

{processedSolutions.map((solution, index) => (

<SolutionCard

key={solution.id}

solution={solution}

index={index}

onFeedback={(rating) => handleSolutionFeedback(solution.id, rating)}

/>

))}

</AnimatePresence>

</motion.div>

</div>

);

});

SolutionStack.displayName = 'SolutionStack';

export default SolutionStack;

**components/SynthesisPanel.tsx**

typescript

import React, { memo } from 'react';

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

import { Download, GitMerge, Zap, Shield, Gauge, Brain, CheckCircle2 } from 'lucide-react';

import { SynthesisResult } from '../lib/types';

interface SynthesisPanelProps {

synthesis: SynthesisResult | null;

onExport?: () => void;

className?: string;

}

*// Performance Bar Component - Designer Voice: "Delightful micro-interactions"*

const PerformanceBar: React.FC<{

label: string;

value: number;

color: string;

icon: React.ComponentType<any>;

}> = memo(({ label, value, color, icon: Icon }) => (

<div className="space-y-2">

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

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

<Icon className={`w-4 h-4 text-${color}-500`} />

<span className="text-sm text-gray-600 dark:text-gray-400 capitalize">{label}</span>

</div>

<span className="text-sm font-medium text-gray-900 dark:text-white">

{Math.round(value \* 100)}%

</span>

</div>

<div className="w-full h-2 bg-gray-200 dark:bg-gray-700 rounded-full overflow-hidden">

<motion.div

className={`h-full bg-gradient-to-r from-${color}-400 to-${color}-600 rounded-full`}

initial={{ width: 0 }}

animate={{ width: `${value \* 100}%` }}

transition={{ duration: 1, ease: "easeOut", delay: 0.3 }}

/>

</div>

</div>

));

*// Main component - Synthesizer Voice: "Harmonizing, transcendent display"*

export const SynthesisPanel: React.FC<SynthesisPanelProps> = memo(({

synthesis,

onExport,

className = ""

}) => {

if (!synthesis) {

return (

<motion.div

className={`text-center py-16 ${className}`}

initial={{ opacity: 0 }}

animate={{ opacity: 1 }}

>

<motion.div

className="w-20 h-20 mx-auto mb-6 bg-gradient-to-br from-purple-100 to-blue-100 dark:from-purple-900/20 dark:to-blue-900/20 rounded-full flex items-center justify-center"

animate={{

scale: [1, 1.1, 1],

rotate: [0, 180, 360]

}}

transition={{

duration: 4,

repeat: Infinity,

ease: "easeInOut"

}}

>

<GitMerge className="w-10 h-10 text-purple-500" />

</motion.div>

<h3 className="text-lg font-medium text-gray-900 dark:text-white mb-2">

Ready for Synthesis

</h3>

<p className="text-gray-500 dark:text-gray-400 max-w-md mx-auto leading-relaxed">

Generate multiple solutions and synthesize them into a unified, harmonized result that integrates the best aspects of each approach.

</p>

</motion.div>

);

}

const handleExport = () => {

const blob = new Blob([synthesis.finalCode], { type: 'text/plain' });

const url = URL.createObjectURL(blob);

const a = document.createElement('a');

a.href = url;

a.download = 'synthesized-solution.tsx';

a.click();

URL.revokeObjectURL(url);

onExport?.();

};

return (

<motion.div

className={`space-y-8 ${className}`}

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

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

transition={{ duration: 0.6 }}

>

{*/\* Header \*/*}

<div className="flex flex-col sm:flex-row sm:items-center sm:justify-between gap-4">

<motion.div

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

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

>

<h3 className="text-xl font-bold text-gray-900 dark:text-white flex items-center">

<motion.div

className="w-8 h-8 bg-gradient-to-br from-purple-500 to-blue-500 rounded-lg flex items-center justify-center mr-3"

whileHover={{ rotate: 180 }}

transition={{ duration: 0.3 }}

>

<GitMerge className="w-4 h-4 text-white" />

</motion.div>

Recursive Synthesis Result

</h3>

<p className="text-sm text-gray-500 dark:text-gray-400 mt-1">

Multi-dimensional integration across {synthesis.voicesUsed.length} perspectives

</p>

</motion.div>

<motion.div

className="flex items-center space-x-4"

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

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

>

<div className="flex items-center space-x-2 text-sm text-gray-500">

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

<span>Confidence: {Math.round(synthesis.confidence \* 100)}%</span>

<span>•</span>

<span>Depth: Layer {synthesis.recursionDepth}</span>

</div>

<motion.button

onClick={handleExport}

className="px-6 py-2 bg-gradient-to-r from-green-500 to-emerald-600 hover:from-green-600 hover:to-emerald-700 text-white font-medium rounded-lg transition-all duration-200 flex items-center space-x-2 shadow-lg hover:shadow-xl"

whileHover={{ scale: 1.02 }}

whileTap={{ scale: 0.98 }}

>

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

<span>Export Final Code</span>

</motion.button>

</motion.div>

</div>

{*/\* Synthesized Code \*/*}

<motion.div

className="bg-gradient-to-r from-purple-50 to-blue-50 dark:from-purple-900/20 dark:to-blue-900/20 rounded-xl p-6 border border-purple-200 dark:border-purple-700"

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

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

transition={{ delay: 0.2 }}

>

<div className="flex items-center justify-between mb-4">

<h4 className="font-semibold text-gray-900 dark:text-white flex items-center">

<CheckCircle2 className="w-5 h-5 text-green-500 mr-2" />

Synthesized Code

</h4>

<div className="flex items-center space-x-2 text-xs text-gray-500">

<motion.div

className="w-2 h-2 bg-green-400 rounded-full"

animate={{ scale: [1, 1.5, 1] }}

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

/>

<span>Production Ready</span>

</div>

</div>

<motion.div

className="bg-white dark:bg-gray-800 rounded-lg border overflow-hidden"

whileHover={{ boxShadow: "0 10px 25px rgba(0,0,0,0.1)" }}

>

<pre className="p-4 text-sm overflow-x-auto max-h-80 scrollbar-thin scrollbar-thumb-gray-300">

<code className="text-gray-800 dark:text-gray-200">

{synthesis.finalCode}

</code>

</pre>

</motion.div>

</motion.div>

{*/\* Methodology and Performance \*/*}

<div className="grid grid-cols-1 lg:grid-cols-2 gap-8">

{*/\* Methodology \*/*}

<motion.div

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

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

transition={{ delay: 0.3 }}

>

<h4 className="font-semibold text-gray-900 dark:text-white mb-4 flex items-center">

<Brain className="w-5 h-5 text-purple-500 mr-2" />

Synthesis Methodology

</h4>

<div className="bg-white dark:bg-gray-800 rounded-lg p-4 border border-gray-200 dark:border-gray-700">

<p className="text-sm text-gray-600 dark:text-gray-400 leading-relaxed mb-4">

{synthesis.methodology}

</p>

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

<div>

<h5 className="text-sm font-medium text-gray-700 dark:text-gray-300 mb-2">

Voices Harmonized:

</h5>

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

{synthesis.voicesUsed.map((voice, idx) => (

<motion.span

key={voice}

className="inline-block px-2 py-1 text-xs bg-blue-100 dark:bg-blue-900/30 text-blue-700 dark:text-blue-300 rounded-full"

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

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

transition={{ delay: 0.4 + idx \* 0.1 }}

>

{voice}

</motion.span>

))}

</div>

</div>

</div>

</div>

</motion.div>

{*/\* Performance Profile \*/*}

<motion.div

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

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

transition={{ delay: 0.4 }}

>

<h4 className="font-semibold text-gray-900 dark:text-white mb-4 flex items-center">

<Gauge className="w-5 h-5 text-green-500 mr-2" />

Performance Profile

</h4>

<div className="bg-white dark:bg-gray-800 rounded-lg p-4 border border-gray-200 dark:border-gray-700 space-y-4">

<PerformanceBar

label="speed"

value={synthesis.performanceProfile.speed}

color="blue"

icon={Zap}

/>

<PerformanceBar

label="memory"

value={synthesis.performanceProfile.memory}

color="green"

icon={Brain}

/>

<PerformanceBar

label="scalability"

value={synthesis.performanceProfile.scalability}

color="purple"

icon={GitMerge}

/>

</div>

</motion.div>

</div>

{*/\* Improvements and Ethical Considerations \*/*}

<div className="grid grid-cols-1 lg:grid-cols-2 gap-8">

{*/\* Key Improvements \*/*}

<motion.div

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

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

transition={{ delay: 0.5 }}

>

<h4 className="font-semibold text-gray-900 dark:text-white mb-4 flex items-center">

<CheckCircle2 className="w-5 h-5 text-green-500 mr-2" />

Key Improvements

</h4>

<div className="bg-white dark:bg-gray-800 rounded-lg p-4 border border-gray-200 dark:border-gray-700">

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

{synthesis.improvements.map((improvement, idx) => (

<motion.li

key={idx}

className="flex items-start space-x-3"

initial={{ opacity: 0, x: -10 }}

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

transition={{ delay: 0.6 + idx \* 0.1 }}

>

<motion.div

className="w-5 h-5 bg-green-100 dark:bg-green-900/30 rounded-full flex items-center justify-center flex-shrink-0 mt-0.5"

whileHover={{ scale: 1.2 }}

>

<CheckCircle2 className="w-3 h-3 text-green-600 dark:text-green-400" />

</motion.div>

<span className="text-sm text-gray-600 dark:text-gray-400 leading-relaxed">

{improvement}

</span>

</motion.li>

))}

</ul>

</div>

</motion.div>

{*/\* Ethical Considerations \*/*}

<motion.div

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

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

transition={{ delay: 0.6 }}

>

<h4 className="font-semibold text-gray-900 dark:text-white mb-4 flex items-center">

<Shield className="w-5 h-5 text-blue-500 mr-2" />

Ethical Considerations

</h4>

<div className="bg-white dark:bg-gray-800 rounded-lg p-4 border border-gray-200 dark:border-gray-700">

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

{synthesis.ethicalConsiderations.map((consideration, idx) => (

<motion.li

key={idx}

className="flex items-start space-x-3"

initial={{ opacity: 0, x: -10 }}

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

transition={{ delay: 0.7 + idx \* 0.1 }}

>

<motion.div

className="w-5 h-5 bg-blue-100 dark:bg-blue-900/30 rounded-full flex items-center justify-center flex-shrink-0 mt-0.5"

whileHover={{ scale: 1.2 }}

>

<Shield className="w-3 h-3 text-blue-600 dark:text-blue-400" />

</motion.div>

<span className="text-sm text-gray-600 dark:text-gray-400 leading-relaxed">

{consideration}

</span>

</motion.li>

))}

</ul>

</div>

</motion.div>

</div>

</motion.div>

);

});

SynthesisPanel.displayName = 'SynthesisPanel';

export default SynthesisPanel;

**components/LedgerLog.tsx**

typescript

import React, { memo, useState, useMemo } from 'react';

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

import { BookOpen, Clock, Users, TrendingUp, Filter, Search, AlertTriangle, CheckCircle } from 'lucide-react';

import { PhantomLedgerEntry } from '../lib/types';

import { formatTimestamp } from '../lib/utils';

interface LedgerLogProps {

entries: PhantomLedgerEntry[];

className?: string;

}

type FilterType = 'all' | 'generation' | 'synthesis' | 'high-ethical' | 'recent';

*// Steward Voice: "Responsible tracking, ethical transparency, protective monitoring"*

const LedgerEntry: React.FC<{

entry: PhantomLedgerEntry;

index: number

}> = memo(({ entry, index }) => {

const [expanded, setExpanded] = useState(false);

const getActionIcon = (action: string) => {

if (action.includes('Generate')) return Users;

if (action.includes('Synthesis')) return TrendingUp;

return BookOpen;

};

const getEthicalColor = (weight: number) => {

if (weight >= 0.8) return 'text-green-500 bg-green-50 border-green-200';

if (weight >= 0.6) return 'text-yellow-500 bg-yellow-50 border-yellow-200';

return 'text-red-500 bg-red-50 border-red-200';

};

const ActionIcon = getActionIcon(entry.action);

return (

<motion.div

layout

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

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

transition={{ delay: index \* 0.05 }}

className="border border-gray-200 dark:border-gray-700 rounded-xl p-4 bg-white dark:bg-gray-800 hover:shadow-md transition-all duration-200"

>

{*/\* Header \*/*}

<div className="flex items-center justify-between mb-3">

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

<motion

typescript

<motion.div

className={`w-8 h-8 rounded-lg flex items-center justify-center ${

entry.action.includes('Generate') ? 'bg-blue-100 text-blue-600' :

entry.action.includes('Synthesis') ? 'bg-purple-100 text-purple-600' :

'bg-gray-100 text-gray-600'

}`}

whileHover={{ scale: 1.1 }}

>

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

</motion.div>

<div>

<h4 className="font-medium text-gray-900 dark:text-white">{entry.action}</h4>

<div className="flex items-center space-x-2 text-xs text-gray-500">

<Clock className="w-3 h-3" />

<span>{formatTimestamp(entry.timestamp)}</span>

<span>•</span>

<span>{entry.voices.length} voices</span>

</div>

</div>

</div>

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

{*/\* Ethical weight indicator \*/*}

<motion.div

className={`px-2 py-1 rounded-full text-xs font-medium border ${getEthicalColor(entry.ethicalWeight)}`}

whileHover={{ scale: 1.05 }}

>

{entry.ethicalWeight >= 0.8 ? <CheckCircle className="w-3 h-3 inline mr-1" /> :

entry.ethicalWeight >= 0.6 ? <AlertTriangle className="w-3 h-3 inline mr-1" /> :

<AlertTriangle className="w-3 h-3 inline mr-1" />}

{Math.round(entry.ethicalWeight \* 100)}%

</motion.div>

<motion.button

onClick={() => setExpanded(!expanded)}

className="text-xs text-blue-500 hover:text-blue-600 font-medium"

whileHover={{ scale: 1.05 }}

>

{expanded ? 'Less' : 'More'}

</motion.button>

</div>

</div>

{*/\* Outcome summary \*/*}

<p className="text-sm text-gray-600 dark:text-gray-400 mb-3 leading-relaxed">

{entry.outcome}

</p>

{*/\* Voices involved \*/*}

<div className="flex flex-wrap gap-1 mb-3">

{entry.voices.map((voice, idx) => (

<motion.span

key={voice}

className="inline-block px-2 py-1 text-xs bg-gray-100 dark:bg-gray-700 text-gray-600 dark:text-gray-400 rounded-full"

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

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

transition={{ delay: idx \* 0.05 }}

>

{voice}

</motion.span>

))}

</div>

{*/\* Expandable learnings section \*/*}

<AnimatePresence>

{expanded && (

<motion.div

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

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

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

className="border-t border-gray-200 dark:border-gray-700 pt-3"

>

<h5 className="text-sm font-medium text-gray-700 dark:text-gray-300 mb-2">

Key Learnings:

</h5>

<ul className="space-y-2">

{entry.learnings.map((learning, idx) => (

<motion.li

key={idx}

className="flex items-start space-x-2 text-sm text-gray-600 dark:text-gray-400"

initial={{ opacity: 0, x: -10 }}

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

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

>

<span className="text-blue-500 mt-1 flex-shrink-0">•</span>

<span>{learning}</span>

</motion.li>

))}

</ul>

</motion.div>

)}

</AnimatePresence>

</motion.div>

);

});

*// Main component - Witness Voice: "Observational, pattern-aware, systematic tracking"*

export const LedgerLog: React.FC<LedgerLogProps> = memo(({

entries,

className = ""

}) => {

const [filter, setFilter] = useState<FilterType>('all');

const [searchTerm, setSearchTerm] = useState('');

const [showFilters, setShowFilters] = useState(false);

*// Memoized filtering and analytics - Optimizer Voice*

const { filteredEntries, analytics } = useMemo(() => {

let filtered = [...entries];

*// Apply search filter*

if (searchTerm.trim()) {

const search = searchTerm.toLowerCase();

filtered = filtered.filter(entry =>

entry.action.toLowerCase().includes(search) ||

entry.outcome.toLowerCase().includes(search) ||

entry.voices.some(voice => voice.toLowerCase().includes(search)) ||

entry.learnings.some(learning => learning.toLowerCase().includes(search))

);

}

*// Apply category filter*

switch (filter) {

case 'generation':

filtered = filtered.filter(e => e.action.includes('Generate'));

break;

case 'synthesis':

filtered = filtered.filter(e => e.action.includes('Synthesis'));

break;

case 'high-ethical':

filtered = filtered.filter(e => e.ethicalWeight >= 0.8);

break;

case 'recent':

const oneDayAgo = Date.now() - 24 \* 60 \* 60 \* 1000;

filtered = filtered.filter(e => e.timestamp > oneDayAgo);

break;

}

*// Calculate analytics*

const totalEntries = filtered.length;

const avgEthicalWeight = totalEntries > 0

? filtered.reduce((sum, e) => sum + e.ethicalWeight, 0) / totalEntries

: 0;

const highEthicalCount = filtered.filter(e => e.ethicalWeight >= 0.8).length;

const recentActionsCount = filtered.filter(e =>

e.timestamp > Date.now() - 60 \* 60 \* 1000

).length;

return {

filteredEntries: filtered,

analytics: {

totalEntries,

avgEthicalWeight,

highEthicalCount,

recentActionsCount

}

};

}, [entries, filter, searchTerm]);

if (entries.length === 0) {

return (

<motion.div

className={`text-center py-16 ${className}`}

initial={{ opacity: 0 }}

animate={{ opacity: 1 }}

>

<motion.div

className="w-20 h-20 mx-auto mb-6 bg-gray-100 dark:bg-gray-800 rounded-full flex items-center justify-center"

animate={{

rotateY: [0, 180, 360],

scale: [1, 1.1, 1]

}}

transition={{

duration: 3,

repeat: Infinity,

ease: "easeInOut"

}}

>

<BookOpen className="w-10 h-10 text-gray-400" />

</motion.div>

<h3 className="text-lg font-medium text-gray-900 dark:text-white mb-2">

Phantom Ledger Empty

</h3>

<p className="text-gray-500 dark:text-gray-400 max-w-md mx-auto leading-relaxed">

This ledger will automatically track all your coding decisions, ethical considerations, and learning insights as you use the platform.

</p>

</motion.div>

);

}

return (

<div className={`space-y-6 ${className}`}>

{*/\* Header with analytics \*/*}

<div className="flex flex-col sm:flex-row sm:items-center sm:justify-between gap-4">

<motion.div

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

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

>

<h3 className="text-lg font-semibold text-gray-900 dark:text-white flex items-center">

<BookOpen className="w-5 h-5 mr-2 text-blue-500" />

Phantom Ledger

</h3>

<p className="text-sm text-gray-500 dark:text-gray-400">

Ethical & Decision Tracking System

</p>

</motion.div>

<motion.div

className="flex items-center space-x-4"

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

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

>

<div className="flex items-center space-x-4 text-sm">

<div className="flex items-center space-x-1">

<div className="w-2 h-2 bg-green-400 rounded-full"></div>

<span className="text-gray-600 dark:text-gray-400">

{analytics.highEthicalCount} high-ethical

</span>

</div>

<div className="flex items-center space-x-1">

<div className="w-2 h-2 bg-blue-400 rounded-full"></div>

<span className="text-gray-600 dark:text-gray-400">

{analytics.recentActionsCount} recent

</span>

</div>

</div>

<motion.button

onClick={() => setShowFilters(!showFilters)}

className="flex items-center space-x-2 px-3 py-2 text-sm text-gray-600 dark:text-gray-400 hover:text-gray-800 dark:hover:text-gray-200 border border-gray-200 dark:border-gray-700 rounded-lg hover:border-gray-300 dark:hover:border-gray-600 transition-colors"

whileHover={{ scale: 1.02 }}

>

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

<span>Filter</span>

</motion.button>

</motion.div>

</div>

{*/\* Analytics Dashboard \*/*}

<motion.div

className="grid grid-cols-2 md:grid-cols-4 gap-4 p-4 bg-gray-50 dark:bg-gray-800 rounded-xl border border-gray-200 dark:border-gray-700"

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

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

transition={{ delay: 0.2 }}

>

<div className="text-center">

<motion.div

className="text-2xl font-bold text-blue-600 dark:text-blue-400"

initial={{ scale: 0 }}

animate={{ scale: 1 }}

transition={{ delay: 0.3, type: "spring" }}

>

{analytics.totalEntries}

</motion.div>

<div className="text-xs text-gray-500">Total Actions</div>

</div>

<div className="text-center">

<motion.div

className="text-2xl font-bold text-green-600 dark:text-green-400"

initial={{ scale: 0 }}

animate={{ scale: 1 }}

transition={{ delay: 0.4, type: "spring" }}

>

{Math.round(analytics.avgEthicalWeight \* 100)}%

</motion.div>

<div className="text-xs text-gray-500">Avg Ethics</div>

</div>

<div className="text-center">

<motion.div

className="text-2xl font-bold text-purple-600 dark:text-purple-400"

initial={{ scale: 0 }}

animate={{ scale: 1 }}

transition={{ delay: 0.5, type: "spring" }}

>

{analytics.highEthicalCount}

</motion.div>

<div className="text-xs text-gray-500">High Ethics</div>

</div>

<div className="text-center">

<motion.div

className="text-2xl font-bold text-orange-600 dark:text-orange-400"

initial={{ scale: 0 }}

animate={{ scale: 1 }}

transition={{ delay: 0.6, type: "spring" }}

>

{analytics.recentActionsCount}

</motion.div>

<div className="text-xs text-gray-500">Recent (1h)</div>

</div>

</motion.div>

{*/\* Filters and Search \*/*}

<AnimatePresence>

{showFilters && (

<motion.div

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

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

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

className="p-4 bg-gray-50 dark:bg-gray-800 rounded-lg border border-gray-200 dark:border-gray-700"

>

<div className="flex flex-col sm:flex-row gap-4">

{*/\* Search \*/*}

<div className="flex-1 relative">

<Search className="absolute left-3 top-1/2 transform -translate-y-1/2 w-4 h-4 text-gray-400" />

<input

type="text"

placeholder="Search actions, outcomes, voices, or learnings..."

value={searchTerm}

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

className="w-full pl-10 pr-4 py-2 text-sm border border-gray-300 dark:border-gray-600 rounded-lg bg-white dark:bg-gray-700 focus:border-blue-500 focus:ring-1 focus:ring-blue-500"

/>

</div>

{*/\* Filter dropdown \*/*}

<select

value={filter}

onChange={(e) => setFilter(e.target.value as FilterType)}

className="px-3 py-2 text-sm border border-gray-300 dark:border-gray-600 rounded-lg bg-white dark:bg-gray-700 focus:border-blue-500 focus:ring-1 focus:ring-blue-500"

>

<option value="all">All Actions</option>

<option value="generation">Solution Generation</option>

<option value="synthesis">Synthesis Events</option>

<option value="high-ethical">High Ethical (80%+)</option>

<option value="recent">Recent (24h)</option>

</select>

</div>

</motion.div>

)}

</AnimatePresence>

{*/\* Entries List \*/*}

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

<AnimatePresence mode="popLayout">

{filteredEntries.length > 0 ? (

filteredEntries.map((entry, index) => (

<LedgerEntry

key={`${entry.timestamp}-${index}`}

entry={entry}

index={index}

/>

))

) : (

<motion.div

initial={{ opacity: 0 }}

animate={{ opacity: 1 }}

className="text-center py-12 text-gray-500 dark:text-gray-400"

>

<Search className="w-12 h-12 mx-auto mb-4 opacity-50" />

<p>No entries match your current filters.</p>

<button

onClick={() => {

setFilter('all');

setSearchTerm('');

}}

className="mt-2 text-blue-500 hover:text-blue-600 text-sm font-medium"

>

Clear filters

</button>

</motion.div>

)}

</AnimatePresence>

</div>

</div>

);

});

LedgerLog.displayName = 'LedgerLog';

export default LedgerLog;

**components/App.tsx**

typescript

import React, { useState, useCallback, useMemo, useEffect } from 'react';

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

import {

Sparkles, Sun, Moon, Download, GitMerge, Brain,

Users, BarChart3, BookOpen, Activity, TrendingUp, Star

} from 'lucide-react';

*// Components*

import VoiceSelector from './VoiceSelector';

import PromptEngine from './PromptEngine';

import SolutionStack from './SolutionStack';

import SynthesisPanel from './SynthesisPanel';

import LedgerLog from './LedgerLog';

*// Hooks and utilities*

import { useCodeGeneration } from '../hooks/useCodeGeneration';

import { ENHANCED\_CODING\_VOICES, TRANSISTHESIS\_ARCHETYPES, PERFORMANCE\_METRICS } from '../lib/constants';

import { Theme, TabType } from '../lib/types';

*// Architect Voice: "Scalable, modular, enterprise-ready architecture"*

const EnhancedCodeCrucible: React.FC = () => {

*// Theme and UI state*

const [theme, setTheme] = useState<Theme>('light');

const [activeTab, setActiveTab] = useState<TabType>('voices');

const [learningMode, setLearningMode] = useState(true);

*// Voice and archetype selection*

const [selectedVoices, setSelectedVoices] = useState<Set<string>>(new Set());

const [selectedArchetypes, setSelectedArchetypes] = useState<Set<string>>(

new Set(['seeker', 'decider'])

);

*// Prompt and generation state*

const [prompt, setPrompt] = useState('');

const [recursionDepth, setRecursionDepth] = useState(1);

*// Code generation hook*

const {

solutions,

synthesis,

isGenerating,

phantomLedger,

generateSolutions,

synthesizeSolutions

} = useCodeGeneration();

*// Memoized data for performance - Optimizer Voice*

const memoizedVoices = useMemo(() =>

Array.from(selectedVoices).map(id => ENHANCED\_CODING\_VOICES[id]).filter(Boolean),

[selectedVoices]

);

const memoizedArchetypes = useMemo(() =>

Array.from(selectedArchetypes).map(id => TRANSISTHESIS\_ARCHETYPES[id]).filter(Boolean),

[selectedArchetypes]

);

const hasValidSolutions = useMemo(() =>

solutions.length > 0 && solutions.some(s => !s.code.includes('Error generating')),

[solutions]

);

*// UI Voice: "Progressive enhancement with theme persistence"*

useEffect(() => {

const savedTheme = localStorage.getItem('codecrucible-theme') as Theme;

if (savedTheme) setTheme(savedTheme);

document.documentElement.classList.toggle('dark', theme === 'dark');

}, [theme]);

useEffect(() => {

localStorage.setItem('codecrucible-theme', theme);

document.documentElement.classList.toggle('dark', theme === 'dark');

}, [theme]);

*// Auto-switch to synthesis tab when synthesis completes*

useEffect(() => {

if (synthesis && activeTab === 'voices') {

setActiveTab('synthesis');

}

}, [synthesis, activeTab]);

*// Event handlers - Synthesizer Voice: "Harmonized interaction flows"*

const handleVoiceToggle = useCallback((voiceId: string) => {

setSelectedVoices(prev => {

const newSet = new Set(prev);

if (newSet.has(voiceId)) {

newSet.delete(voiceId);

} else {

newSet.add(voiceId);

}

return newSet;

});

}, []);

const handleArchetypeToggle = useCallback((archetypeId: string) => {

setSelectedArchetypes(prev => {

const newSet = new Set(prev);

if (newSet.has(archetypeId)) {

newSet.delete(archetypeId);

} else {

newSet.add(archetypeId);

}

return newSet;

});

}, []);

const handleGenerate = useCallback(() => {

if (!prompt.trim()) return;

generateSolutions(prompt, memoizedVoices, memoizedArchetypes, recursionDepth);

}, [prompt, memoizedVoices, memoizedArchetypes, recursionDepth, generateSolutions]);

const handleSynthesize = useCallback(() => {

if (hasValidSolutions) {

synthesizeSolutions(solutions, recursionDepth);

}

}, [hasValidSolutions, solutions, recursionDepth, synthesizeSolutions]);

const handleExportReport = useCallback(() => {

const reportContent = `# CodeCrucible Report

Generated: ${new Date().toISOString()}

## Prompt

${prompt}

## Solutions (${solutions.length})

${solutions.map(sol => `

### ${sol.voiceId}

\`\`\`

${sol.code}

\`\`\`

\*\*Explanation:\*\* ${sol.explanation}

\*\*Confidence:\*\* ${Math.round(sol.confidence \* 100)}%

`).join('\n')}

${synthesis ? `

## Synthesized Solution

\`\`\`

${synthesis.finalCode}

\`\`\`

\*\*Methodology:\*\* ${synthesis.methodology}

\*\*Confidence:\*\* ${Math.round(synthesis.confidence \* 100)}%

` : ''}

## Phantom Ledger (${phantomLedger.length} entries)

${phantomLedger.slice(0, 5).map(entry => `

### ${entry.action}

- \*\*Timestamp:\*\* ${new Date(entry.timestamp).toLocaleString()}

- \*\*Ethical Weight:\*\* ${Math.round(entry.ethicalWeight \* 100)}%

- \*\*Outcome:\*\* ${entry.outcome}

`).join('\n')}

`;

const blob = new Blob([reportContent], { type: 'text/markdown' });

const url = URL.createObjectURL(blob);

const a = document.createElement('a');

a.href = url;

a.download = 'codecrucible-report.md';

a.click();

URL.revokeObjectURL(url);

}, [prompt, solutions, synthesis, phantomLedger]);

const handleSolutionFeedback = useCallback((solutionId: string, rating: number) => {

if (learningMode) {

console.log(`Learning from feedback: Solution ${solutionId} rated ${rating}/5`);

*// In a real app, this would update the learning algorithm*

}

}, [learningMode]);

return (

<div className={`min-h-screen transition-colors duration-300 ${

theme === 'dark' ? 'dark bg-gray-900' : 'bg-gray-50'

}`}>

{*/\* Enhanced Header - UI Voice: "Clean, purposeful, emotionally resonant" \*/*}

<motion.header

className="bg-white dark:bg-gray-800 border-b border-gray-200 dark:border-gray-700 sticky top-0 z-50 backdrop-blur-lg bg-opacity-90"

initial={{ y: -100 }}

animate={{ y: 0 }}

transition={{ duration: 0.5 }}

>

<div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-4">

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

<motion.div

className="flex items-center space-x-4"

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

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

>

<motion.div

className="w-10 h-10 bg-gradient-to-br from-blue-500 via-purple-600 to-pink-500 rounded-xl flex items-center justify-center"

whileHover={{ rotate: 180, scale: 1.1 }}

transition={{ duration: 0.3 }}

>

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

</motion.div>

<div>

<h1 className="text-2xl font-bold text-gray-900 dark:text-white">

CodeCrucible

</h1>

<p className="text-sm text-gray-500 dark:text-gray-400">

Multi-Dimensional AI Synthesis Platform

</p>

</div>

</motion.div>

<motion.div

className="flex items-center space-x-3"

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

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

>

{*/\* Learning mode toggle \*/*}

<motion.button

onClick={() => setLearningMode(!learningMode)}

className={`px-3 py-2 rounded-lg text-sm font-medium transition-colors ${

learningMode

? 'bg-green-100 text-green-700 hover:bg-green-200 dark:bg-green-900/30 dark:text-green-400'

: 'bg-gray-100 text-gray-700 hover:bg-gray-200 dark:bg-gray-700 dark:text-gray-300'

}`}

whileHover={{ scale: 1.05 }}

whileTap={{ scale: 0.95 }}

>

<Brain className="w-4 h-4 inline mr-1" />

Learning {learningMode ? 'On' : 'Off'}

</motion.button>

{*/\* Export report button \*/*}

{(hasValidSolutions || synthesis) && (

<motion.button

onClick={handleExportReport}

className="px-4 py-2 bg-green-500 hover:bg-green-600 text-white font-medium rounded-lg transition-colors flex items-center space-x-2 text-sm shadow-lg hover:shadow-xl"

whileHover={{ scale: 1.02 }}

whileTap={{ scale: 0.98 }}

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

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

>

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

<span className="hidden sm:inline">Export Report</span>

</motion.button>

)}

{*/\* Theme toggle \*/*}

<motion.button

onClick={() => setTheme(theme === 'dark' ? 'light' : 'dark')}

className="p-2 rounded-lg border border-gray-300 dark:border-gray-600 hover:bg-gray-100 dark:hover:bg-gray-700 transition-colors"

whileHover={{ scale: 1.1 }}

whileTap={{ scale: 0.9 }}

aria-label={`Switch to ${theme === 'dark' ? 'light' : 'dark'} mode`}

>

<AnimatePresence mode="wait">

{theme === 'dark' ? (

<motion.div

key="sun"

initial={{ rotate: -90, opacity: 0 }}

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

exit={{ rotate: 90, opacity: 0 }}

transition={{ duration: 0.2 }}

>

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

</motion.div>

) : (

<motion.div

key="moon"

initial={{ rotate: 90, opacity: 0 }}

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

exit={{ rotate: -90, opacity: 0 }}

transition={{ duration: 0.2 }}

>

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

</motion.div>

)}

</AnimatePresence>

</motion.button>

</motion.div>

</div>

</div>

</motion.header>

<div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-8 space-y-8">

{*/\* Prompt Engine \*/*}

<PromptEngine

prompt={prompt}

onPromptChange={setPrompt}

onGenerate={handleGenerate}

isGenerating={isGenerating}

recursionDepth={recursionDepth}

onRecursionDepthChange={setRecursionDepth}

/>

{*/\* Voice Selection \*/*}

<VoiceSelector

selectedVoices={selectedVoices}

selectedArchetypes={selectedArchetypes}

onVoiceToggle={handleVoiceToggle}

onArchetypeToggle={handleArchetypeToggle}

/>

{*/\* Results Section with Tabs \*/*}

{(hasValidSolutions || synthesis || phantomLedger.length > 0) && (

<motion.div

className="bg-white dark:bg-gray-800 rounded-2xl shadow-sm border border-gray-200 dark:border-gray-700"

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

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

transition={{ duration: 0.5 }}

>

{*/\* Enhanced Tabs \*/*}

<div className="border-b border-gray-200 dark:border-gray-700">

<nav className="flex space-x-8 px-6">

{[

{

id: 'voices' as TabType,

label: 'Solutions',

icon: Users,

count: solutions.length,

enabled: hasValidSolutions

},

{

id: 'synthesis' as TabType,

label: 'Synthesis',

icon: GitMerge,

count: synthesis ? 1 : 0,

enabled: !!synthesis

},

{

id: 'analytics' as TabType,

label: 'Analytics',

icon: BarChart3,

count: null,

enabled: hasValidSolutions || synthesis

},

{

id: 'ledger' as TabType,

label: 'Phantom Ledger',

icon: BookOpen,

count: phantomLedger.length,

enabled: phantomLedger.length > 0

}

].map((tab) => (

<motion.button

key={tab.id}

onClick={() => tab.enabled && setActiveTab(tab.id)}

disabled={!tab.enabled}

className={`py-4 px-1 border-b-2 font-medium text-sm transition-colors flex items-center space-x-2 ${

activeTab === tab.id

? 'border-blue-500 text-blue-600 dark:text-blue-400'

: tab.enabled

? 'border-transparent text-gray-500 hover:text-gray-700 dark:text-gray-400 dark:hover:text-gray-300'

: 'border-transparent text-gray-300 dark:text-gray-600 cursor-not-allowed'

}`}

whileHover={tab.enabled ? { y: -2 } : {}}

whileTap={tab.enabled ? { y: 0 } : {}}

>

<tab.icon className="w-4 h-4" />

<span>{tab.label}</span>

{tab.count !== null && (

<motion.span

className="bg-gray-100 dark:bg-gray-700 text-gray-600 dark:text-gray-400 px-2 py-0.5 rounded-full text-xs"

key={tab.count}

initial={{ scale: 1.2 }}

animate={{ scale: 1 }}

>

{tab.count}

</motion.span>

)}

</motion.button>

))}

</nav>

</div>

{*/\* Tab Content \*/*}

<div className="p-6">

<AnimatePresence mode="wait">

{activeTab === 'voices' && (

<motion.div

key="voices"

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

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

exit={{ opacity: 0, x: 20 }}

transition={{ duration: 0.3 }}

>

<div className="space-y-6">

{hasValidSolutions && (

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

<h3 className="text-lg font-semibold text-gray-900 dark:text-white">

Generated Solutions

</h3>

<motion.button

onClick={handleSynthesize}

disabled={solutions.length < 2 || isGenerating}

className="px-6 py-2 bg-purple-500 hover:bg-purple-600 disabled:bg-gray-300 disabled:cursor-not-allowed text-white font-medium rounded-lg transition-colors flex items-center space-x-2 shadow-lg hover:shadow-xl"

whileHover={{ scale: 1.02 }}

whileTap={{ scale: 0.98 }}

>

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

<span>Synthesize All</span>

</motion.button>

</div>

)}

<SolutionStack

solutions={solutions}

onSolutionFeedback={handleSolutionFeedback}

/>

</div>

</motion.div>

)}

{activeTab === 'synthesis' && (

<motion.div

key="synthesis"

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

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

exit={{ opacity: 0, x: 20 }}

transition={{ duration: 0.3 }}

>

<SynthesisPanel synthesis={synthesis} />

</motion.div>

)}

{activeTab === 'analytics' && (

<motion.div

key="analytics"

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

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

exit={{ opacity: 0, x: 20 }}

transition={{ duration: 0.3 }}

className="space-y-6"

>

<h3 className="text-lg font-semibold text-gray-900 dark:text-white">

Performance Analytics

</h3>

<div className="grid grid-cols-1 md:grid-cols-3 gap-6">

{[

{

label: 'Generation Speed',

value: PERFORMANCE\_METRICS.generationSpeed,

icon: Activity,

color: 'blue',

subtitle: 'Average response time'

},

{

label: 'Success Rate',

value: PERFORMANCE\_METRICS.successRate,

icon: TrendingUp,

color: 'green',

subtitle: 'Successful generations'

},

{

label: 'User Satisfaction',

value: PERFORMANCE\_METRICS.userSatisfaction,

icon: Star,

color: 'purple',

subtitle: 'Average rating'

}

].map((metric, idx) => (

<motion.div

key={metric.label}

className={`bg-${metric.color}-50 dark:bg-${metric.color}-900/20 rounded-lg p-6 border border-${metric.color}-200 dark:border-${metric.color}-700`}

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

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

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

>

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

<metric.icon className={`w-6 h-6 text-${metric.color}-500`} />

<span className="font-medium text-gray-900 dark:text-white">

{metric.label}

</span>

</div>

<motion.p

className={`text-3xl font-bold text-${metric.color}-600 dark:text-${metric.color}-400 mb-1`}

initial={{ scale: 0 }}

animate={{ scale: 1 }}

transition={{ delay: idx \* 0.1 + 0.3, type: "spring" }}

>

{metric.value}

</motion.p>

<p className="text-sm text-gray-500">{metric.subtitle}</p>

</motion.div>

))}

</div>

</motion.div>

)}

{activeTab === 'ledger' && (

<motion.div

key="ledger"

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

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

exit={{ opacity: 0, x: 20 }}

transition={{ duration: 0.3 }}

>

<LedgerLog entries={phantomLedger} />

</motion.div>

)}

</AnimatePresence>

</div>

</motion.div>

)}

{*/\* Loading State \*/*}

{isGenerating && !hasValidSolutions && (

<motion.div

className="bg-white dark:bg-gray-800 rounded-2xl shadow-sm border border-gray-200 dark:border-gray-700 p-12"

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

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

transition={{ duration: 0.3 }}

>

<div className="flex flex-col items-center space-y-6">

<motion.div

className="w-16 h-16 border-4 border-blue-500 border-t-transparent rounded-full"

animate={{ rotate: 360 }}

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

/>

<div className="text-center">

<motion.p

className="text-lg font-medium text-gray-900 dark:text-white mb-2"

animate={{ opacity: [0.5, 1, 0.5] }}

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

>

Recursive Synthesis in Progress...

</motion.p>

<p className="text-gray-500 dark:text-gray-400">

The voices are converging through {recursionDepth} layers of recursive analysis

</p>

<motion.div

className="mt-4 flex justify-center space-x-2"

initial={{ opacity: 0 }}

animate={{ opacity: 1 }}

transition={{ delay: 0.5 }}

>

{[...memoizedVoices, ...memoizedArchetypes].map((voice, idx) => (

<motion.div

key={voice.id}

className={`w-2 h-2 bg-${voice.color}-400 rounded-full`}

animate={{

scale: [1, 1.5, 1],

opacity: [0.5, 1, 0.5]

}}

transition={{

duration: 1.5,

repeat: Infinity,

delay: idx \* 0.2

}}

/>

))}

</motion.div>

</div>

</div>

</motion.div>

)}

</div>

</div>

);

};

export default EnhancedCodeCrucible;