How it Works

**🎯 Kurzora "How It Works" Page - Complete UI Analysis**

**13-Point Framework for Immediate Cursor Implementation**

**1. UI Components & Layout**

**Interactive Elements**

**Primary Interactive Components:**

* **ProcessStepCard** (6 signal generation steps with hover animations)
* **SignalScoreExplainer** (interactive score range cards with examples)
* **PerformanceMetrics** (animated counters with real-time updates)
* **ProcessFlowDiagram** (interactive step-by-step flow visualization)
* **LiveExampleModal** (popup showing real signal examples)
* **AnimatedStatsCounter** (number animations for performance metrics)

**Enhanced Navigation & Controls:**

* Smooth scroll navigation between sections
* "Try Live Example" interactive buttons
* Expandable detail sections for each process step
* Mobile-optimized accordion for smaller screens

**React + TypeScript Component Structure**

// Complete How It Works Page Architecture

<HowItWorksPage>

<div className="min-h-screen bg-gradient-to-br from-slate-950 via-blue-950 to-slate-950">

{/\* Enhanced Navigation \*/}

<HowItWorksNavigation

onScrollToSection={handleScrollToSection}

currentSection={activeSection}

/>

{/\* Hero Section with Animation \*/}

<HeroSection

title="How Our Signals Work"

subtitle="Kurzora combines institutional-grade algorithms..."

ctaButton={<LiveExampleButton />}

/>

{/\* Interactive Process Flow \*/}

<ProcessFlowSection>

<SectionHeader

title="Our Signal Generation Process"

description="Six-step institutional-grade analysis"

/>

<InteractiveProcessFlow

steps={signalProcessSteps}

onStepClick={handleStepDetail}

activeStep={selectedStep}

/>

</ProcessFlowSection>

{/\* Enhanced Signal Score Explanation \*/}

<SignalScoreSection>

<InteractiveScoreExplainer

scoreRanges={scoreRanges}

onScoreClick={handleScoreExample}

showExamples={showExamples}

/>

</SignalScoreSection>

{/\* Live Performance Metrics \*/}

<PerformanceSection>

<AnimatedPerformanceMetrics

metrics={performanceData}

isVisible={isMetricsVisible}

onMetricClick={handleMetricDetail}

/>

<BacktestingChart

data={backtestingData}

timeRange={selectedTimeRange}

/>

</PerformanceSection>

{/\* Risk Disclosure with Emphasis \*/}

<RiskDisclosureSection enhanced={true} />

{/\* Enhanced Footer \*/}

<Footer variant="how-it-works" />

</div>

{/\* Modals and Overlays \*/}

{showLiveExample && (

<LiveExampleModal

onClose={() => setShowLiveExample(false)}

exampleSignal={selectedExample}

/>

)}

{showStepDetail && (

<ProcessStepDetailModal

step={selectedStep}

onClose={() => setShowStepDetail(false)}

/>

)}

</HowItWorksPage>

**Responsive Design Considerations**

// Mobile-first responsive components

const ProcessStepCard: React.FC<ProcessStepProps> = ({ step, index, onClick }) => (

<Card className={`

group cursor-pointer transition-all duration-300

bg-slate-800/50 backdrop-blur-sm border-slate-700

hover:bg-slate-700/50 hover:border-blue-500/50 hover:scale-105

md:hover:scale-110 lg:hover:scale-105

${step.isActive ? 'ring-2 ring-blue-500' : ''}

`}>

<CardHeader className="pb-4">

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

<div className={`

p-3 rounded-full transition-colors duration-300

${step.bgColor} group-hover:${step.hoverColor}

`}>

<step.icon className={`h-6 w-6 ${step.iconColor} group-hover:scale-110 transition-transform`} />

</div>

<div className="flex-1">

<CardTitle className="text-lg text-white group-hover:text-blue-300 transition-colors">

{step.title}

</CardTitle>

<div className="text-sm text-slate-500">Step {index + 1}</div>

</div>

</div>

</CardHeader>

<CardContent className="pt-0">

<p className="text-slate-400 group-hover:text-slate-300 transition-colors mb-4">

{step.description}

</p>

<div className="flex items-center text-sm text-blue-400 group-hover:text-blue-300">

<span>Learn more</span>

<ChevronRight className="h-4 w-4 ml-1 group-hover:translate-x-1 transition-transform" />

</div>

</CardContent>

</Card>

);

**Loading States and Error Handling**

// Loading skeleton for process steps

const ProcessStepSkeleton: React.FC = () => (

<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">

{Array.from({ length: 6 }).map((\_, i) => (

<Card key={i} className="bg-slate-800/50 border-slate-700">

<CardHeader>

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

<div className="w-12 h-12 bg-slate-700 rounded-full animate-pulse" />

<div className="space-y-2 flex-1">

<div className="h-4 bg-slate-700 rounded animate-pulse" />

<div className="h-3 bg-slate-700/70 rounded animate-pulse w-20" />

</div>

</div>

</CardHeader>

<CardContent>

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

<div className="h-3 bg-slate-700/70 rounded animate-pulse" />

<div className="h-3 bg-slate-700/70 rounded animate-pulse w-3/4" />

</div>

</CardContent>

</Card>

))}

</div>

);

// Error boundary for the page

class HowItWorksErrorBoundary extends React.Component<

{ children: React.ReactNode },

{ hasError: boolean; error?: Error }

> {

constructor(props: any) {

super(props);

this.state = { hasError: false };

}

static getDerivedStateFromError(error: Error) {

return { hasError: true, error };

}

render() {

if (this.state.hasError) {

return (

<div className="min-h-screen bg-slate-950 flex items-center justify-center p-4">

<Card className="max-w-md mx-auto bg-slate-800 border-red-500/50">

<CardContent className="p-6 text-center">

<AlertTriangle className="h-12 w-12 text-red-400 mx-auto mb-4" />

<h3 className="text-lg font-semibold text-white mb-2">

Something went wrong

</h3>

<p className="text-slate-400 mb-4">

We're having trouble loading this page. Please try again.

</p>

<Button onClick={() => window.location.reload()}>

Refresh Page

</Button>

</CardContent>

</Card>

</div>

);

}

return this.props.children;

}

}

**2. State Management (Zustand)**

**Store Structure**

// How It Works specific store

interface HowItWorksStore {

// UI State

activeSection: string;

selectedStep: ProcessStep | null;

showLiveExample: boolean;

showStepDetail: boolean;

selectedExample: SignalExample | null;

// Performance Data

performanceMetrics: PerformanceMetrics;

backtestingData: BacktestingData[];

isMetricsVisible: boolean;

// Interactive State

animationsEnabled: boolean;

currentScoreExample: ScoreRange | null;

selectedTimeRange: TimeRange;

// Actions

setActiveSection: (section: string) => void;

selectStep: (step: ProcessStep) => void;

openLiveExample: (example: SignalExample) => void;

closeLiveExample: () => void;

toggleAnimations: () => void;

updatePerformanceMetrics: (metrics: PerformanceMetrics) => void;

setTimeRange: (range: TimeRange) => void;

}

const useHowItWorksStore = create<HowItWorksStore>((set, get) => ({

// Initial state

activeSection: 'hero',

selectedStep: null,

showLiveExample: false,

showStepDetail: false,

selectedExample: null,

performanceMetrics: {

winRate: 68,

tradesAnalyzed: 180000,

avgROI: 6,

sharpeRatio: 1.8,

maxDrawdown: 12

},

backtestingData: [],

isMetricsVisible: false,

animationsEnabled: true,

currentScoreExample: null,

selectedTimeRange: '1Y',

// Actions

setActiveSection: (section) => set({ activeSection: section }),

selectStep: (step) => set({

selectedStep: step,

showStepDetail: true

}),

openLiveExample: (example) => set({

selectedExample: example,

showLiveExample: true

}),

closeLiveExample: () => set({

showLiveExample: false,

selectedExample: null

}),

toggleAnimations: () => set((state) => ({

animationsEnabled: !state.animationsEnabled

})),

updatePerformanceMetrics: (metrics) => set({ performanceMetrics: metrics }),

setTimeRange: (range) => set({ selectedTimeRange: range }),

}));

**Local vs Global State Decisions**

// Local state for component-specific interactions

const useProcessStepInteractions = () => {

const [hoveredStep, setHoveredStep] = useState<string | null>(null);

const [expandedCards, setExpandedCards] = useState<Set<string>>(new Set());

const toggleCardExpansion = useCallback((stepId: string) => {

setExpandedCards(prev => {

const newSet = new Set(prev);

if (newSet.has(stepId)) {

newSet.delete(stepId);

} else {

newSet.add(stepId);

}

return newSet;

});

}, []);

return {

hoveredStep,

setHoveredStep,

expandedCards,

toggleCardExpansion

};

};

// Global state for cross-component data

const usePerformanceData = () => {

const { performanceMetrics, updatePerformanceMetrics } = useHowItWorksStore();

// Simulate real-time updates

useEffect(() => {

const interval = setInterval(() => {

// Slight variations to simulate live data

updatePerformanceMetrics({

...performanceMetrics,

tradesAnalyzed: performanceMetrics.tradesAnalyzed + Math.floor(Math.random() \* 5),

});

}, 30000); // Update every 30 seconds

return () => clearInterval(interval);

}, [performanceMetrics, updatePerformanceMetrics]);

return performanceMetrics;

};

**Optimistic Updates**

// Optimistic state updates for better UX

const useOptimisticInteractions = () => {

const [pendingActions, setPendingActions] = useState<Set<string>>(new Set());

const handleOptimisticStepSelection = useCallback(async (step: ProcessStep) => {

const actionId = `step-${step.id}`;

// Optimistically update UI

setPendingActions(prev => new Set(prev).add(actionId));

useHowItWorksStore.getState().selectStep(step);

try {

// Simulate API call for step analytics

await trackStepInteraction(step.id);

} catch (error) {

// Revert on error

console.error('Failed to track step interaction:', error);

} finally {

setPendingActions(prev => {

const newSet = new Set(prev);

newSet.delete(actionId);

return newSet;

});

}

}, []);

return {

pendingActions,

handleOptimisticStepSelection

};

};

**3. API Contracts & Integration**

**API Endpoints**

// API endpoints for How It Works page

interface HowItWorksAPI {

// Performance metrics endpoint

GET: '/api/v1/how-it-works/performance';

// Live signal examples

GET: '/api/v1/how-it-works/examples';

// Process step details

GET: '/api/v1/how-it-works/steps/:stepId';

// User interaction tracking

POST: '/api/v1/analytics/page-interaction';

// Backtesting data

GET: '/api/v1/how-it-works/backtesting';

}

// Request/Response schemas

interface PerformanceMetricsResponse {

data: {

winRate: number;

tradesAnalyzed: number;

avgROI: number;

sharpeRatio: number;

maxDrawdown: number;

lastUpdated: string;

marketConditions: 'BULL' | 'BEAR' | 'SIDEWAYS';

};

metadata: {

updateFrequency: string;

dataSource: string;

period: string;

};

}

interface SignalExampleResponse {

data: SignalExample[];

pagination: {

total: number;

page: number;

limit: number;

};

}

interface SignalExample {

id: string;

ticker: string;

signalType: 'BUY' | 'SELL';

score: number;

entryPrice: number;

stopLoss: number;

takeProfit: number;

reasoning: string;

indicators: {

rsi: number;

macd: number;

volume: number;

support: number;

};

outcome?: {

actualExit: number;

profit: number;

duration: string;

};

timestamp: string;

}

interface ProcessStepDetail {

id: string;

title: string;

description: string;

technicalDetails: string;

algorithmType: string;

parameters: Record<string, any>;

examples: SignalExample[];

performance: {

accuracy: number;

avgConfidence: number;

falsePositiveRate: number;

};

}

**API Client Implementation**

// API client for How It Works data

class HowItWorksAPIClient {

private baseURL = process.env.VITE\_API\_URL;

async getPerformanceMetrics(): Promise<PerformanceMetricsResponse> {

const response = await fetch(`${this.baseURL}/api/v1/how-it-works/performance`, {

headers: {

'Authorization': `Bearer ${getAuthToken()}`,

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

},

});

if (!response.ok) {

throw new APIError('Failed to fetch performance metrics', response.status);

}

return response.json();

}

async getSignalExamples(limit: number = 10): Promise<SignalExampleResponse> {

const response = await fetch(

`${this.baseURL}/api/v1/how-it-works/examples?limit=${limit}&type=demo`,

{

headers: {

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

},

}

);

if (!response.ok) {

throw new APIError('Failed to fetch signal examples', response.status);

}

return response.json();

}

async getStepDetails(stepId: string): Promise<ProcessStepDetail> {

const response = await fetch(`${this.baseURL}/api/v1/how-it-works/steps/${stepId}`);

if (!response.ok) {

throw new APIError('Failed to fetch step details', response.status);

}

return response.json();

}

async trackPageInteraction(interaction: PageInteraction): Promise<void> {

// Fire and forget analytics

fetch(`${this.baseURL}/api/v1/analytics/page-interaction`, {

method: 'POST',

headers: {

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

},

body: JSON.stringify(interaction),

}).catch(console.error);

}

}

// Custom hooks for API integration

const usePerformanceMetrics = () => {

return useQuery({

queryKey: ['performance-metrics'],

queryFn: () => howItWorksAPI.getPerformanceMetrics(),

staleTime: 5 \* 60 \* 1000, // 5 minutes

refetchInterval: 60 \* 1000, // 1 minute

retry: 3,

});

};

const useSignalExamples = () => {

return useQuery({

queryKey: ['signal-examples'],

queryFn: () => howItWorksAPI.getSignalExamples(),

staleTime: 10 \* 60 \* 1000, // 10 minutes

retry: 2,

});

};

**Error Response Formats**

interface APIErrorResponse {

error: {

code: string;

message: string;

details?: Record<string, any>;

timestamp: string;

requestId: string;

};

}

// Common error codes for How It Works page

enum HowItWorksErrorCodes {

PERFORMANCE\_DATA\_UNAVAILABLE = 'PERFORMANCE\_DATA\_UNAVAILABLE',

SIGNAL\_EXAMPLES\_NOT\_FOUND = 'SIGNAL\_EXAMPLES\_NOT\_FOUND',

STEP\_DETAILS\_NOT\_FOUND = 'STEP\_DETAILS\_NOT\_FOUND',

RATE\_LIMITED = 'RATE\_LIMITED',

MAINTENANCE\_MODE = 'MAINTENANCE\_MODE',

}

// Error handling utility

const handleAPIError = (error: APIError) => {

switch (error.code) {

case HowItWorksErrorCodes.PERFORMANCE\_DATA\_UNAVAILABLE:

return "Performance data is temporarily unavailable. Showing cached results.";

case HowItWorksErrorCodes.SIGNAL\_EXAMPLES\_NOT\_FOUND:

return "Signal examples are currently being updated. Please try again shortly.";

case HowItWorksErrorCodes.RATE\_LIMITED:

return "Too many requests. Please wait a moment before trying again.";

default:

return "Something went wrong. Please refresh the page or try again later.";

}

};

**4. Performance & Optimization**

**Lazy Loading Strategies**

// Lazy load heavy components

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

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

const BacktestingChart = lazy(() => import('../components/charts/BacktestingChart'));

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

// Lazy load with loading fallback

const LazyBacktestingChart = () => (

<Suspense fallback={<ChartSkeleton />}>

<BacktestingChart />

</Suspense>

);

// Intersection Observer for performance sections

const useIntersectionObserver = (threshold = 0.1) => {

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

const ref = useRef<HTMLDivElement>(null);

useEffect(() => {

const observer = new IntersectionObserver(

([entry]) => {

if (entry.isIntersecting) {

setIsVisible(true);

observer.disconnect(); // Only trigger once

}

},

{ threshold }

);

if (ref.current) {

observer.observe(ref.current);

}

return () => observer.disconnect();

}, [threshold]);

return [ref, isVisible] as const;

};

**Memoization Opportunities**

// Memoized process step cards

const ProcessStepCard = React.memo<ProcessStepProps>(({

step,

index,

isActive,

onClick

}) => {

const handleClick = useCallback(() => {

onClick(step);

}, [onClick, step]);

return (

<Card

className={`process-step-card ${isActive ? 'active' : ''}`}

onClick={handleClick}

>

{/\* Card content \*/}

</Card>

);

}, (prevProps, nextProps) => {

return (

prevProps.step.id === nextProps.step.id &&

prevProps.isActive === nextProps.isActive &&

prevProps.index === nextProps.index

);

});

// Memoized performance calculations

const useCalculatedMetrics = (rawMetrics: PerformanceMetrics) => {

return useMemo(() => {

return {

...rawMetrics,

winRateFormatted: `${rawMetrics.winRate}%`,

tradesFormatted: rawMetrics.tradesAnalyzed.toLocaleString(),

roiFormatted: `${rawMetrics.avgROI}%`,

sharpeFormatted: rawMetrics.sharpeRatio.toFixed(2),

risk Score: calculateRiskScore(rawMetrics),

};

}, [rawMetrics]);

};

// Memoized signal examples

const useFilteredExamples = (examples: SignalExample[], scoreFilter: number) => {

return useMemo(() => {

return examples.filter(example => example.score >= scoreFilter);

}, [examples, scoreFilter]);

};

**Bundle Splitting Considerations**

// Route-level splitting

const HowItWorksPage = lazy(() => import('./pages/HowItWorks'));

// Feature-level splitting

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

const InteractiveCharts = lazy(() => import('./components/charts/InteractiveCharts'));

// Vendor splitting in vite.config.ts

export default defineConfig({

build: {

rollupOptions: {

output: {

manualChunks: {

'chart-libs': ['chart.js', 'react-chartjs-2'],

'animation-libs': ['framer-motion', 'lottie-react'],

'ui-components': ['lucide-react', '@radix-ui/react-dialog'],

},

},

},

},

});

**Caching Strategies**

// React Query caching configuration

const queryClient = new QueryClient({

defaultOptions: {

queries: {

staleTime: 5 \* 60 \* 1000, // 5 minutes

cacheTime: 30 \* 60 \* 1000, // 30 minutes

retry: (failureCount, error) => {

if (error.status === 404) return false;

return failureCount < 3;

},

},

},

});

// Local storage for user preferences

const usePersistedPreferences = () => {

const [preferences, setPreferences] = useState(() => {

const saved = localStorage.getItem('how-it-works-preferences');

return saved ? JSON.parse(saved) : {

animationsEnabled: true,

autoPlayExamples: false,

preferredTimeRange: '1Y',

};

});

useEffect(() => {

localStorage.setItem('how-it-works-preferences', JSON.stringify(preferences));

}, [preferences]);

return [preferences, setPreferences];

};

// Service worker for static asset caching

// sw.js

self.addEventListener('fetch', (event) => {

if (event.request.destination === 'image' ||

event.request.url.includes('/static/')) {

event.respondWith(

caches.match(event.request).then((response) => {

return response || fetch(event.request);

})

);

}

});

**5. Database Schema**

**PostgreSQL Table Structures**

-- How It Works page analytics table

CREATE TABLE how\_it\_works\_analytics (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

user\_id UUID REFERENCES users(id),

session\_id VARCHAR(255) NOT NULL,

page\_section VARCHAR(100) NOT NULL,

interaction\_type VARCHAR(50) NOT NULL, -- 'view', 'click', 'hover', 'scroll'

interaction\_target VARCHAR(100), -- 'step\_card', 'live\_example', 'performance\_metric'

interaction\_data JSONB,

timestamp TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

user\_agent TEXT,

referrer TEXT,

-- Indexes for performance

INDEX idx\_how\_it\_works\_analytics\_user\_id (user\_id),

INDEX idx\_how\_it\_works\_analytics\_timestamp (timestamp),

INDEX idx\_how\_it\_works\_analytics\_section (page\_section),

INDEX idx\_how\_it\_works\_analytics\_interaction (interaction\_type)

);

-- Performance metrics cache table

CREATE TABLE performance\_metrics\_cache (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

metric\_type VARCHAR(50) NOT NULL, -- 'overall', 'by\_timeframe', 'by\_market\_condition'

timeframe VARCHAR(20), -- '1D', '1W', '1M', '3M', '1Y', 'ALL'

market\_condition VARCHAR(20), -- 'BULL', 'BEAR', 'SIDEWAYS', 'ALL'

-- Metric values

win\_rate DECIMAL(5,2) NOT NULL,

trades\_analyzed INTEGER NOT NULL,

avg\_roi DECIMAL(5,2) NOT NULL,

sharpe\_ratio DECIMAL(5,3),

max\_drawdown DECIMAL(5,2),

profit\_factor DECIMAL(5,3),

-- Metadata

calculation\_date TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

data\_source VARCHAR(50) DEFAULT 'polygon',

is\_current BOOLEAN DEFAULT true,

-- Constraints

CONSTRAINT valid\_win\_rate CHECK (win\_rate >= 0 AND win\_rate <= 100),

CONSTRAINT valid\_roi CHECK (avg\_roi >= -100),

CONSTRAINT valid\_drawdown CHECK (max\_drawdown >= 0 AND max\_drawdown <= 100),

-- Indexes

INDEX idx\_performance\_metrics\_type\_timeframe (metric\_type, timeframe),

INDEX idx\_performance\_metrics\_current (is\_current),

INDEX idx\_performance\_metrics\_date (calculation\_date)

);

-- Signal examples for How It Works page

CREATE TABLE signal\_examples (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

ticker VARCHAR(10) NOT NULL,

signal\_type VARCHAR(10) NOT NULL CHECK (signal\_type IN ('BUY', 'SELL')),

signal\_score INTEGER NOT NULL CHECK (signal\_score >= 0 AND signal\_score <= 100),

-- Price data

entry\_price DECIMAL(10,4) NOT NULL,

stop\_loss DECIMAL(10,4),

take\_profit DECIMAL(10,4),

-- Technical indicators

indicators JSONB NOT NULL,

-- Signal reasoning

reasoning TEXT NOT NULL,

technical\_summary TEXT,

-- Outcome (if available)

outcome JSONB,

-- Metadata

signal\_timestamp TIMESTAMP WITH TIME ZONE NOT NULL,

is\_demo BOOLEAN DEFAULT true,

is\_featured BOOLEAN DEFAULT false,

display\_order INTEGER,

-- Timestamps

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

-- Indexes

INDEX idx\_signal\_examples\_ticker (ticker),

INDEX idx\_signal\_examples\_score (signal\_score),

INDEX idx\_signal\_examples\_demo (is\_demo),

INDEX idx\_signal\_examples\_featured (is\_featured),

INDEX idx\_signal\_examples\_timestamp (signal\_timestamp)

);

-- Process steps configuration

CREATE TABLE process\_steps (

id VARCHAR(50) PRIMARY KEY,

title VARCHAR(200) NOT NULL,

description TEXT NOT NULL,

technical\_details TEXT,

icon VARCHAR(50) NOT NULL,

step\_order INTEGER NOT NULL,

-- Display configuration

bg\_color VARCHAR(50),

icon\_color VARCHAR(50),

hover\_color VARCHAR(50),

-- Content

algorithm\_type VARCHAR(100),

parameters JSONB,

-- Performance metrics for this step

accuracy DECIMAL(5,2),

avg\_confidence DECIMAL(5,2),

false\_positive\_rate DECIMAL(5,2),

-- Status

is\_active BOOLEAN DEFAULT true,

-- Timestamps

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

-- Constraints

CONSTRAINT unique\_step\_order UNIQUE(step\_order),

-- Indexes

INDEX idx\_process\_steps\_order (step\_order),

INDEX idx\_process\_steps\_active (is\_active)

);

**Indexes for Performance Optimization**

-- Composite indexes for common queries

CREATE INDEX idx\_analytics\_user\_session\_time

ON how\_it\_works\_analytics (user\_id, session\_id, timestamp DESC);

CREATE INDEX idx\_performance\_metrics\_lookup

ON performance\_metrics\_cache (metric\_type, timeframe, market\_condition, is\_current);

CREATE INDEX idx\_signal\_examples\_display

ON signal\_examples (is\_demo, is\_featured, display\_order, signal\_score DESC);

-- Partial indexes for specific use cases

CREATE INDEX idx\_analytics\_recent\_interactions

ON how\_it\_works\_analytics (interaction\_type, timestamp)

WHERE timestamp >= NOW() - INTERVAL '7 days';

CREATE INDEX idx\_current\_metrics\_only

ON performance\_metrics\_cache (metric\_type, timeframe)

WHERE is\_current = true;

-- GIN index for JSONB columns

CREATE INDEX idx\_signal\_examples\_indicators

ON signal\_examples USING GIN (indicators);

CREATE INDEX idx\_analytics\_interaction\_data

ON how\_it\_works\_analytics USING GIN (interaction\_data);

**Migration Scripts**

-- Migration: Add process steps data

INSERT INTO process\_steps (id, title, description, technical\_details, icon, step\_order, bg\_color, icon\_color, algorithm\_type, accuracy) VALUES

('multi\_timeframe', 'Multi Time Frame', 'Confirm trends, reduce false signals, and make more informed decisions.', 'Analyzes price action across 1H, 4H, 1D, and 1W timeframes using proprietary trend confirmation algorithms.', 'Layers', 1, 'bg-blue-500/20', 'text-blue-400', 'Multi-Timeframe Trend Analysis', 87.3),

('support\_resistance', 'Multi-Level Support & Resistance', 'Provide clearer entry/exit points and enhance trend confirmation.', 'Identifies dynamic and static support/resistance levels using volume-weighted price analysis and institutional order flow.', 'Signal', 2, 'bg-purple-500/20', 'text-purple-400', 'Support/Resistance Detection', 91.2),

('options\_indicators', 'Options Indicators', 'Identifying trends, volatility, and potential entry/exit points.', 'Incorporates options flow analysis, put/call ratios, and implied volatility for comprehensive market sentiment.', 'Gauge', 3, 'bg-green-500/20', 'text-green-400', 'Options Flow Analysis', 78.6),

('risk\_management', 'Risk Management', 'Every signal includes calculated stop-loss and take-profit levels to maintain favorable risk-reward ratios.', 'Automated position sizing and risk calculation using advanced portfolio theory and volatility-adjusted metrics.', 'AlertTriangle', 4, 'bg-amber-500/20', 'text-amber-400', 'Risk Management Engine', 95.1),

('validation\_scoring', 'Validation & Scoring', 'Each signal receives a score (0-100) based on strength of confirmation.', 'Machine learning scoring model trained on 180,000+ historical trades with continuous model optimization.', 'CheckCircle', 5, 'bg-red-500/20', 'text-red-400', 'ML Validation Engine', 89.7),

('ai\_enhancement', 'AI Enhancement', 'Machine learning algorithms continuously improve signal quality.', 'Deep learning models analyze market patterns, news sentiment, and macroeconomic factors for enhanced accuracy.', 'BrainCircuit', 6, 'bg-indigo-500/20', 'text-indigo-400', 'AI Pattern Recognition', 92.4);

-- Migration: Add sample signal examples

INSERT INTO signal\_examples (ticker, signal\_type, signal\_score, entry\_price, stop\_loss, take\_profit, indicators, reasoning, is\_demo, is\_featured) VALUES

('AAPL', 'BUY', 87, 175.25, 170.00, 185.00,

'{"rsi": 45.2, "macd": 0.85, "volume": 125.3, "support": 174.50}',

'Strong support level confluence at $174.50 with oversold RSI and bullish MACD crossover. Options flow shows institutional accumulation.',

true, true),

('TSLA', 'SELL', 91, 245.80, 255.00, 225.00,

'{"rsi": 78.1, "macd": -1.23, "volume": 89.7, "resistance": 246.00}',

'Overbought conditions with resistance at $246. Heavy put volume and bearish divergence in momentum indicators.',

true, true),

('NVDA', 'BUY', 82, 415.30, 405.00, 435.00,

'{"rsi": 52.3, "macd": 2.1, "volume": 156.8, "support": 414.00}',

'Breakout above key resistance with strong volume confirmation. AI sector momentum and bullish options positioning.',

true, false);

**6. User Experience**

**Loading States and Skeleton Screens**

// Skeleton components for different sections

const HeroSectionSkeleton: React.FC = () => (

<div className="text-center py-16">

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

<div className="h-12 bg-slate-700 rounded-lg mx-auto w-2/3 animate-pulse" />

<div className="h-6 bg-slate-700/70 rounded mx-auto w-3/4 animate-pulse" />

<div className="h-6 bg-slate-700/70 rounded mx-auto w-1/2 animate-pulse" />

</div>

</div>

);

const PerformanceMetricsSkeleton: React.FC = () => (

<div className="grid grid-cols-1 md:grid-cols-3 gap-6">

{Array.from({ length: 3 }).map((\_, i) => (

<div key={i} className="bg-slate-700/50 rounded-lg p-4">

<div className="h-8 bg-slate-600 rounded mb-2 animate-pulse" />

<div className="h-4 bg-slate-600/70 rounded w-24 mx-auto animate-pulse" />

</div>

))}

</div>

);

// Progressive loading for performance metrics

const useProgressiveLoading = () => {

const [loadingStage, setLoadingStage] = useState<'hero' | 'process' | 'metrics' | 'complete'>('hero');

useEffect(() => {

const stages = [

{ stage: 'hero', delay: 0 },

{ stage: 'process', delay: 500 },

{ stage: 'metrics', delay: 1000 },

{ stage: 'complete', delay: 1500 },

];

stages.forEach(({ stage, delay }) => {

setTimeout(() => setLoadingStage(stage as any), delay);

});

}, []);

return loadingStage;

};

**Error Boundaries and Fallback UI**

// Section-specific error boundaries

const ProcessStepsErrorBoundary: React.FC<{ children: React.ReactNode }> = ({ children }) => {

return (

<ErrorBoundary

fallback={

<Card className="bg-slate-800/50 border-amber-500/50 p-6">

<div className="text-center">

<AlertTriangle className="h-12 w-12 text-amber-400 mx-auto mb-4" />

<h3 className="text-lg font-semibold text-white mb-2">

Process Information Unavailable

</h3>

<p className="text-slate-400 mb-4">

We're having trouble loading the signal process details.

Our core functionality remains unaffected.

</p>

<Button variant="outline" onClick={() => window.location.reload()}>

Try Again

</Button>

</div>

</Card>

}

>

{children}

</ErrorBoundary>

);

};

// Network error handling

const useNetworkStatus = () => {

const [isOnline, setIsOnline] = useState(navigator.onLine);

const [showOfflineMessage, setShowOfflineMessage] = useState(false);

useEffect(() => {

const handleOnline = () => {

setIsOnline(true);

setShowOfflineMessage(false);

};

const handleOffline = () => {

setIsOnline(false);

setShowOfflineMessage(true);

};

window.addEventListener('online', handleOnline);

window.addEventListener('offline', handleOffline);

return () => {

window.removeEventListener('online', handleOnline);

window.removeEventListener('offline', handleOffline);

};

}, []);

return { isOnline, showOfflineMessage };

};

// Offline indicator component

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

const { showOfflineMessage } = useNetworkStatus();

if (!showOfflineMessage) return null;

return (

<div className="fixed top-4 right-4 z-50 bg-amber-500 text-white px-4 py-2 rounded-lg shadow-lg">

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

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

<span className="text-sm">You're offline. Some features may be limited.</span>

</div>

</div>

);

};

**Accessibility Considerations**

// ARIA labels and semantic markup

const ProcessStepCard: React.FC<ProcessStepProps> = ({ step, index, onClick }) => (

<Card

role="button"

tabIndex={0}

aria-label={`Step ${index + 1}: ${step.title}. ${step.description}`}

aria-describedby={`step-detail-${step.id}`}

className="focus:ring-2 focus:ring-blue-400 focus:outline-none cursor-pointer"

onClick={onClick}

onKeyDown={(e) => {

if (e.key === 'Enter' || e.key === ' ') {

e.preventDefault();

onClick();

}

}}

>

<CardHeader>

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

<div className={`p-3 rounded-full ${step.bgColor}`}>

<step.icon className={`h-6 w-6 ${step.iconColor}`} />

</div>

<CardTitle className="text-lg text-white">{step.title}</CardTitle>

</div>

</CardHeader>

<CardContent>

<p id={`step-detail-${step.id}`} className="text-slate-400">

{step.description}

</p>

</CardContent>

</Card>

);

// Screen reader announcements

const useAccessibilityAnnouncements = () => {

const announce = useCallback((message: string, priority: 'polite' | 'assertive' = 'polite') => {

const announcement = document.createElement('div');

announcement.setAttribute('aria-live', priority);

announcement.setAttribute('aria-atomic', 'true');

announcement.className = 'sr-only';

announcement.textContent = message;

document.body.appendChild(announcement);

setTimeout(() => {

document.body.removeChild(announcement);

}, 1000);

}, []);

return announce;

};

// Keyboard navigation for modals

const useModalKeyboardNavigation = (isOpen: boolean, onClose: () => void) => {

useEffect(() => {

if (!isOpen) return;

const handleKeyDown = (e: KeyboardEvent) => {

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

onClose();

}

// Trap focus within modal

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

const modal = document.querySelector('[role="dialog"]');

if (!modal) return;

const focusableElements = modal.querySelectorAll(

'a[href], button, textarea, input, select, [tabindex]:not([tabindex="-1"])'

);

const firstElement = focusableElements[0] as HTMLElement;

const lastElement = focusableElements[focusableElements.length - 1] as HTMLElement;

if (e.shiftKey && document.activeElement === firstElement) {

e.preventDefault();

lastElement.focus();

} else if (!e.shiftKey && document.activeElement === lastElement) {

e.preventDefault();

firstElement.focus();

}

}

};

document.addEventListener('keydown', handleKeyDown);

return () => document.removeEventListener('keydown', handleKeyDown);

}, [isOpen, onClose]);

};

**Animation and Transition Requirements**

// Framer Motion variants for page sections

const pageVariants = {

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

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

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

};

const staggerChildren = {

animate: {

transition: {

staggerChildren: 0.1

}

}

};

const cardVariants = {

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

animate: {

opacity: 1,

scale: 1,

transition: {

type: "spring",

stiffness: 300,

damping: 30

}

},

hover: {

scale: 1.05,

y: -5,

transition: {

type: "spring",

stiffness: 400,

damping: 25

}

}

};

// Animated counter component

const AnimatedCounter: React.FC<{

value: number;

duration?: number;

suffix?: string;

}> = ({ value, duration = 2000, suffix = '' }) => {

const [displayValue, setDisplayValue] = useState(0);

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

const ref = useRef<HTMLDivElement>(null);

useEffect(() => {

const observer = new IntersectionObserver(

([entry]) => {

if (entry.isIntersecting) {

setIsVisible(true);

observer.disconnect();

}

},

{ threshold: 0.5 }

);

if (ref.current) {

observer.observe(ref.current);

}

return () => observer.disconnect();

}, []);

useEffect(() => {

if (!isVisible) return;

const startTime = Date.now();

const startValue = displayValue;

const animate = () => {

const now = Date.now();

const progress = Math.min((now - startTime) / duration, 1);

const easeOutCubic = 1 - Math.pow(1 - progress, 3);

const currentValue = startValue + (value - startValue) \* easeOutCubic;

setDisplayValue(Math.round(currentValue));

if (progress < 1) {

requestAnimationFrame(animate);

}

};

requestAnimationFrame(animate);

}, [isVisible, value, duration, displayValue]);

return (

<div ref={ref} className="text-3xl font-bold text-emerald-400">

{displayValue.toLocaleString()}{suffix}

</div>

);

};

// Scroll-triggered animations

const useScrollAnimation = (threshold = 0.1) => {

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

const ref = useRef<HTMLDivElement>(null);

useEffect(() => {

const observer = new IntersectionObserver(

([entry]) => {

setIsVisible(entry.isIntersecting);

},

{ threshold }

);

if (ref.current) {

observer.observe(ref.current);

}

return () => observer.disconnect();

}, [threshold]);

return [ref, isVisible] as const;

};

**7. Integration Points**

**Navigation Patterns and Routing**

// Route configuration for How It Works

const howItWorksRoutes = {

main: '/how-it-works',

stepDetail: '/how-it-works/step/:stepId',

liveExample: '/how-it-works/example/:exampleId',

performance: '/how-it-works/performance',

} as const;

// Navigation breadcrumbs

const HowItWorksBreadcrumbs: React.FC<{ currentPath: string }> = ({ currentPath }) => {

const navigate = useNavigate();

const breadcrumbs = useMemo(() => {

const paths = [

{ label: 'Home', path: '/', icon: Home },

{ label: 'How It Works', path: '/how-it-works', icon: Info },

];

if (currentPath.includes('/step/')) {

const stepId = currentPath.split('/step/')[1];

paths.push({

label: `Step: ${stepId}`,

path: currentPath,

icon: Layers

});

}

return paths;

}, [currentPath]);

return (

<nav aria-label="Breadcrumb" className="mb-6">

<ol className="flex items-center space-x-2 text-sm text-slate-400">

{breadcrumbs.map((crumb, index) => (

<li key={crumb.path} className="flex items-center">

{index > 0 && <ChevronRight className="h-4 w-4 mx-2" />}

<button

onClick={() => navigate(crumb.path)}

className="flex items-center space-x-1 hover:text-white transition-colors"

>

<crumb.icon className="h-4 w-4" />

<span>{crumb.label}</span>

</button>

</li>

))}

</ol>

</nav>

);

};

**Cross-Component State Synchronization**

// Shared state management for How It Works interactions

const useHowItWorksIntegration = () => {

const {

selectedStep,

selectedExample,

updatePerformanceMetrics

} = useHowItWorksStore();

const { trackEvent } = useAnalytics();

const navigate = useNavigate();

// Navigate to signals page with pre-selected filters

const navigateToSignalsWithFilters = useCallback((filters: SignalFilters) => {

const signalsStore = useSignalsStore.getState();

signalsStore.setFilters(filters);

navigate('/signals', {

state: {

from: 'how-it-works',

preselectedFilters: filters

}

});

}, [navigate]);

// Navigate to specific signal detail

const navigateToSignalExample = useCallback((example: SignalExample) => {

navigate(`/signals/${example.ticker}`, {

state: {

from: 'how-it-works-example',

exampleSignal: example

}

});

}, [navigate]);

// Track user engagement

const trackStepInteraction = useCallback((stepId: string, interactionType: string) => {

trackEvent('how\_it\_works\_step\_interaction', {

step\_id: stepId,

interaction\_type: interactionType,

timestamp: new Date().toISOString(),

});

}, [trackEvent]);

return {

selectedStep,

selectedExample,

navigateToSignalsWithFilters,

navigateToSignalExample,

trackStepInteraction,

};

};

// Event bus for cross-component communication

class HowItWorksEventBus extends EventTarget {

emit(eventType: string, data: any) {

this.dispatchEvent(new CustomEvent(eventType, { detail: data }));

}

on(eventType: string, handler: (event: CustomEvent) => void) {

this.addEventListener(eventType, handler as EventListener);

}

off(eventType: string, handler: (event: CustomEvent) => void) {

this.removeEventListener(eventType, handler as EventListener);

}

}

const howItWorksEventBus = new HowItWorksEventBus();

// Hook to use event bus

const useHowItWorksEvents = () => {

const emitStepSelected = useCallback((step: ProcessStep) => {

howItWorksEventBus.emit('step:selected', step);

}, []);

const emitExampleViewed = useCallback((example: SignalExample) => {

howItWorksEventBus.emit('example:viewed', example);

}, []);

const emitMetricsViewed = useCallback((section: string) => {

howItWorksEventBus.emit('metrics:viewed', { section });

}, []);

return {

emitStepSelected,

emitExampleViewed,

emitMetricsViewed,

};

};

**Shared Components and State**

// Shared modal system

const useModalSystem = () => {

const [modals, setModals] = useState<Map<string, ModalConfig>>(new Map());

const openModal = useCallback((id: string, config: ModalConfig) => {

setModals(prev => new Map(prev).set(id, config));

}, []);

const closeModal = useCallback((id: string) => {

setModals(prev => {

const newMap = new Map(prev);

newMap.delete(id);

return newMap;

});

}, []);

const closeAllModals = useCallback(() => {

setModals(new Map());

}, []);

return {

modals,

openModal,

closeModal,

closeAllModals,

};

};

// Global toast notification system

const useToastNotifications = () => {

const [toasts, setToasts] = useState<Toast[]>([]);

const addToast = useCallback((toast: Omit<Toast, 'id'>) => {

const id = Math.random().toString(36).substr(2, 9);

setToasts(prev => [...prev, { ...toast, id }]);

// Auto-remove after duration

setTimeout(() => {

removeToast(id);

}, toast.duration || 5000);

}, []);

const removeToast = useCallback((id: string) => {

setToasts(prev => prev.filter(toast => toast.id !== id));

}, []);

return {

toasts,

addToast,

removeToast,

};

};

// Performance monitoring integration

const usePagePerformance = () => {

const [metrics, setMetrics] = useState<PerformanceMetrics | null>(null);

useEffect(() => {

// Performance observer for page metrics

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

const entries = list.getEntries();

entries.forEach((entry) => {

if (entry.entryType === 'navigation') {

const navEntry = entry as PerformanceNavigationTiming;

setMetrics({

loadTime: navEntry.loadEventEnd - navEntry.loadEventStart,

domContentLoaded: navEntry.domContentLoadedEventEnd - navEntry.domContentLoadedEventStart,

firstPaint: performance.getEntriesByType('paint')[0]?.startTime || 0,

firstContentfulPaint: performance.getEntriesByType('paint')[1]?.startTime || 0,

});

}

});

});

observer.observe({ entryTypes: ['navigation', 'paint'] });

return () => observer.disconnect();

}, []);

return metrics;

};

**8. Testing Strategy**

**Unit Test Requirements**

// Test file: HowItWorks.test.tsx

import { render, screen, fireEvent, waitFor } from '@testing-library/react';

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

import { BrowserRouter } from 'react-router-dom';

import { HowItWorks } from './HowItWorks';

// Test wrapper with all providers

const TestWrapper: React.FC<{ children: React.ReactNode }> = ({ children }) => {

const queryClient = new QueryClient({

defaultOptions: {

queries: { retry: false },

mutations: { retry: false },

},

});

return (

<QueryClientProvider client={queryClient}>

<BrowserRouter>

<LanguageProvider>

{children}

</LanguageProvider>

</BrowserRouter>

</QueryClientProvider>

);

};

describe('HowItWorks Page', () => {

beforeEach(() => {

// Reset mocks

jest.clearAllMocks();

});

test('renders all process steps', () => {

render(<HowItWorks />, { wrapper: TestWrapper });

expect(screen.getByText('Multi Time Frame')).toBeInTheDocument();

expect(screen.getByText('Multi-Level Support & Resistance')).toBeInTheDocument();

expect(screen.getByText('Options Indicators')).toBeInTheDocument();

expect(screen.getByText('Risk Management')).toBeInTheDocument();

expect(screen.getByText('Validation & Scoring')).toBeInTheDocument();

expect(screen.getByText('AI Enhancement')).toBeInTheDocument();

});

test('renders signal score ranges with correct colors', () => {

render(<HowItWorks />, { wrapper: TestWrapper });

const strongScore = screen.getByText('Strong (80-100)');

const validScore = screen.getByText('Valid (60-79)');

const weakScore = screen.getByText('Weak (40-59)');

const ignoreScore = screen.getByText('Ignore (0-39)');

expect(strongScore).toBeInTheDocument();

expect(validScore).toBeInTheDocument();

expect(weakScore).toBeInTheDocument();

expect(ignoreScore).toBeInTheDocument();

});

test('displays performance metrics', () => {

render(<HowItWorks />, { wrapper: TestWrapper });

expect(screen.getByText('68%')).toBeInTheDocument();

expect(screen.getByText('180,000+')).toBeInTheDocument();

expect(screen.getByText('6%')).toBeInTheDocument();

});

test('handles step card interactions', async () => {

const mockTrackEvent = jest.fn();

(useAnalytics as jest.Mock).mockReturnValue({ trackEvent: mockTrackEvent });

render(<HowItWorks />, { wrapper: TestWrapper });

const stepCard = screen.getByRole('button', { name: /Multi Time Frame/ });

fireEvent.click(stepCard);

await waitFor(() => {

expect(mockTrackEvent).toHaveBeenCalledWith('step\_interaction', {

step\_id: 'multi\_timeframe',

interaction\_type: 'click'

});

});

});

});

// Component-specific tests

describe('ProcessStepCard', () => {

const mockStep = {

id: 'test-step',

title: 'Test Step',

description: 'Test description',

icon: Layers,

bgColor: 'bg-blue-500/20',

iconColor: 'text-blue-400',

};

test('renders step information correctly', () => {

render(

<ProcessStepCard

step={mockStep}

index={0}

onClick={jest.fn()}

/>,

{ wrapper: TestWrapper }

);

expect(screen.getByText('Test Step')).toBeInTheDocument();

expect(screen.getByText('Test description')).toBeInTheDocument();

expect(screen.getByText('Step 1')).toBeInTheDocument();

});

test('handles click events', () => {

const mockOnClick = jest.fn();

render(

<ProcessStepCard

step={mockStep}

index={0}

onClick={mockOnClick}

/>,

{ wrapper: TestWrapper }

);

fireEvent.click(screen.getByRole('button'));

expect(mockOnClick).toHaveBeenCalledWith(mockStep);

});

test('handles keyboard navigation', () => {

const mockOnClick = jest.fn();

render(

<ProcessStepCard

step={mockStep}

index={0}

onClick={mockOnClick}

/>,

{ wrapper: TestWrapper }

);

const card = screen.getByRole('button');

fireEvent.keyDown(card, { key: 'Enter' });

expect(mockOnClick).toHaveBeenCalledWith(mockStep);

fireEvent.keyDown(card, { key: ' ' });

expect(mockOnClick).toHaveBeenCalledTimes(2);

});

});

**Integration Test Scenarios**

// Integration tests for API interactions

describe('HowItWorks API Integration', () => {

beforeEach(() => {

// Setup MSW (Mock Service Worker) handlers

server.use(

rest.get('/api/v1/how-it-works/performance', (req, res, ctx) => {

return res(ctx.json({

data: {

winRate: 68,

tradesAnalyzed: 180000,

avgROI: 6,

sharpeRatio: 1.8,

maxDrawdown: 12,

lastUpdated: '2024-01-15T10:00:00Z',

marketConditions: 'BULL'

}

}));

}),

rest.get('/api/v1/how-it-works/examples', (req, res, ctx) => {

return res(ctx.json({

data: mockSignalExamples,

pagination: { total: 10, page: 1, limit: 10 }

}));

})

);

});

test('loads and displays performance metrics from API', async () => {

render(<HowItWorks />, { wrapper: TestWrapper });

await waitFor(() => {

expect(screen.getByText('68%')).toBeInTheDocument();

expect(screen.getByText('180,000+')).toBeInTheDocument();

});

});

test('handles API errors gracefully', async () => {

server.use(

rest.get('/api/v1/how-it-works/performance', (req, res, ctx) => {

return res(ctx.status(500), ctx.json({ error: 'Internal server error' }));

})

);

render(<HowItWorks />, { wrapper: TestWrapper });

await waitFor(() => {

expect(screen.getByText(/something went wrong/i)).toBeInTheDocument();

});

});

test('retries failed requests', async () => {

let callCount = 0;

server.use(

rest.get('/api/v1/how-it-works/performance', (req, res, ctx) => {

callCount++;

if (callCount < 3) {

return res(ctx.status(500));

}

return res(ctx.json({ data: mockPerformanceData }));

})

);

render(<HowItWorks />, { wrapper: TestWrapper });

await waitFor(() => {

expect(screen.getByText('68%')).toBeInTheDocument();

});

expect(callCount).toBe(3);

});

});

// Navigation integration tests

describe('HowItWorks Navigation', () => {

test('navigates back to home', () => {

const mockNavigate = jest.fn();

(useNavigate as jest.Mock).mockReturnValue(mockNavigate);

render(<HowItWorks />, { wrapper: TestWrapper });

fireEvent.click(screen.getByText('← Back to Home'));

expect(mockNavigate).toHaveBeenCalledWith('/');

});

test('opens live example modal', async () => {

render(<HowItWorks />, { wrapper: TestWrapper });

const liveExampleButton = screen.getByText('Try Live Example');

fireEvent.click(liveExampleButton);

await waitFor(() => {

expect(screen.getByRole('dialog')).toBeInTheDocument();

});

});

});

**Mock Data Structures**

// Mock data for testing

export const mockProcessSteps: ProcessStep[] = [

{

id: 'multi\_timeframe',

title: 'Multi Time Frame',

description: 'Confirm trends, reduce false signals, and make more informed decisions.',

icon: Layers,

bgColor: 'bg-blue-500/20',

iconColor: 'text-blue-400',

stepOrder: 1,

algorithmType: 'Multi-Timeframe Trend Analysis',

accuracy: 87.3,

},

// ... other steps

];

export const mockPerformanceData: PerformanceMetrics = {

winRate: 68,

tradesAnalyzed: 180000,

avgROI: 6,

sharpeRatio: 1.8,

maxDrawdown: 12,

lastUpdated: '2024-01-15T10:00:00Z',

marketConditions: 'BULL',

};

export const mockSignalExamples: SignalExample[] = [

{

id: '1',

ticker: 'AAPL',

signalType: 'BUY',

score: 87,

entryPrice: 175.25,

stopLoss: 170.00,

takeProfit: 185.00,

indicators: {

rsi: 45.2,

macd: 0.85,

volume: 125.3,

support: 174.50,

},

reasoning: 'Strong support level confluence at $174.50 with oversold RSI and bullish MACD crossover.',

timestamp: '2024-01-15T09:30:00Z',

},

// ... other examples

];

// Mock API responses

export const mockAPIResponses = {

performanceMetrics: {

data: mockPerformanceData,

metadata: {

updateFrequency: '5 minutes',

dataSource: 'polygon',

period: 'last\_12\_months',

},

},

signalExamples: {

data: mockSignalExamples,

pagination: {

total: 10,

page: 1,

limit: 10,

},

},

};

// Test utilities

export const renderWithProviders = (ui: React.ReactElement) => {

return render(ui, { wrapper: TestWrapper });

};

export const createMockIntersectionObserver = () => {

const mockIntersectionObserver = jest.fn();

mockIntersectionObserver.mockReturnValue({

observe: () => null,

unobserve: () => null,

disconnect: () => null

});

window.IntersectionObserver = mockIntersectionObserver;

};

**Edge Cases to Handle**

// Edge case handling

describe('HowItWorks Edge Cases', () => {

test('handles empty performance metrics', () => {

server.use(

rest.get('/api/v1/how-it-works/performance', (req, res, ctx) => {

return res(ctx.json({ data: null }));

})

);

render(<HowItWorks />, { wrapper: TestWrapper });

expect(screen.getByText(/performance data unavailable/i)).toBeInTheDocument();

});

test('handles malformed API response', () => {

server.use(

rest.get('/api/v1/how-it-works/performance', (req, res, ctx) => {

return res(ctx.text('Not JSON'));

})

);

render(<HowItWorks />, { wrapper: TestWrapper });

expect(screen.getByText(/something went wrong/i)).toBeInTheDocument();

});

test('handles missing translation keys', () => {

const mockT = jest.fn().mockReturnValue('Translation missing');

(useLanguage as jest.Mock).mockReturnValue({ t: mockT });

render(<HowItWorks />, { wrapper: TestWrapper });

expect(mockT).toHaveBeenCalledWith('features.multiTimeframe');

});

test('handles slow network conditions', async () => {

server.use(

rest.get('/api/v1/how-it-works/performance', (req, res, ctx) => {

return res(ctx.delay(5000), ctx.json(mockAPIResponses.performanceMetrics));

})

);

render(<HowItWorks />, { wrapper: TestWrapper });

// Should show loading state

expect(screen.getByTestId('performance-skeleton')).toBeInTheDocument();

// Should eventually load

await waitFor(

() => {

expect(screen.getByText('68%')).toBeInTheDocument();

},

{ timeout: 6000 }

);

});

});

**9. Charts & Data Visualizations**

**Chart Libraries and Configurations**

// Backtesting performance chart with Chart.js

import { Line } from 'react-chartjs-2';

import {

Chart as ChartJS,

CategoryScale,

LinearScale,

PointElement,

LineElement,

Title,

Tooltip,

Legend,

Filler

} from 'chart.js';

ChartJS.register(

CategoryScale,

LinearScale,

PointElement,

LineElement,

Title,

Tooltip,

Legend,

Filler

);

const BacktestingChart: React.FC<{

data: BacktestingDataPoint[];

timeRange: TimeRange;

}> = ({ data, timeRange }) => {

const chartData = useMemo(() => ({

labels: data.map(point => format(new Date(point.date), 'MMM yyyy')),

datasets: [

{

label: 'Cumulative Returns',

data: data.map(point => point.cumulativeReturn),

borderColor: 'rgb(34, 197, 94)',

backgroundColor: 'rgba(34, 197, 94, 0.1)',

fill: true,

tension: 0.4,

pointRadius: 0,

pointHoverRadius: 6,

borderWidth: 2,

},

{

label: 'Benchmark (S&P 500)',

data: data.map(point => point.benchmarkReturn),

borderColor: 'rgb(148, 163, 184)',

backgroundColor: 'transparent',

fill: false,

tension: 0.4,

pointRadius: 0,

pointHoverRadius: 6,

borderWidth: 1,

borderDash: [5, 5],

}

]

}), [data]);

const options = useMemo(() => ({

responsive: true,

maintainAspectRatio: false,

plugins: {

legend: {

position: 'top' as const,

labels: {

color: 'rgb(148, 163, 184)',

usePointStyle: true,

}

},

tooltip: {

mode: 'index' as const,

intersect: false,

backgroundColor: 'rgba(15, 23, 42, 0.9)',

titleColor: 'rgb(248, 250, 252)',

bodyColor: 'rgb(148, 163, 184)',

borderColor: 'rgb(59, 130, 246)',

borderWidth: 1,

callbacks: {

label: (context: any) => {

return `${context.dataset.label}: ${context.parsed.y.toFixed(2)}%`;

}

}

},

},

scales: {

x: {

grid: {

color: 'rgba(148, 163, 184, 0.1)',

},

ticks: {

color: 'rgb(148, 163, 184)',

}

},

y: {

grid: {

color: 'rgba(148, 163, 184, 0.1)',

},

ticks: {

color: 'rgb(148, 163, 184)',

callback: (value: any) => `${value}%`

}

}

},

interaction: {

mode: 'nearest' as const,

axis: 'x' as const,

intersect: false,

},

}), []);

return (

<div className="h-64 md:h-80">

<Line data={chartData} options={options} />

</div>

);

};

**Real-time Data Updates and Animations**

// Real-time performance metrics with animations

const AnimatedPerformanceChart: React.FC<{

metrics: PerformanceMetrics;

isVisible: boolean;

}> = ({ metrics, isVisible }) => {

const [animatedMetrics, setAnimatedMetrics] = useState(metrics);

const [isAnimating, setIsAnimating] = useState(false);

// Animate metrics changes

useEffect(() => {

if (!isVisible) return;

setIsAnimating(true);

const animate = () => {

const duration = 1500;

const startTime = Date.now();

const startMetrics = animatedMetrics;

const animateFrame = () => {

const elapsed = Date.now() - startTime;

const progress = Math.min(elapsed / duration, 1);

// Easing function

const easeOutCubic = 1 - Math.pow(1 - progress, 3);

setAnimatedMetrics({

winRate: lerp(startMetrics.winRate, metrics.winRate, easeOutCubic),

tradesAnalyzed: Math.floor(lerp(startMetrics.tradesAnalyzed, metrics.tradesAnalyzed, easeOutCubic)),

avgROI: lerp(startMetrics.avgROI, metrics.avgROI, easeOutCubic),

});

if (progress < 1) {

requestAnimationFrame(animateFrame);

} else {

setIsAnimating(false);

}

};

requestAnimationFrame(animateFrame);

};

animate();

}, [metrics, isVisible, animatedMetrics]);

const lerp = (start: number, end: number, progress: number) => {

return start + (end - start) \* progress;

};

return (

<div className="grid grid-cols-1 md:grid-cols-3 gap-6">

<MetricCard

title="Average Win Rate"

value={animatedMetrics.winRate}

suffix="%"

icon={TrendingUp}

isAnimating={isAnimating}

color="emerald"

/>

<MetricCard

title="Trades Analyzed"

value={animatedMetrics.tradesAnalyzed}

suffix="+"

icon={BarChart}

isAnimating={isAnimating}

color="blue"

formatter="number"

/>

<MetricCard

title="Average ROI per Trade"

value={animatedMetrics.avgROI}

suffix="%"

icon={DollarSign}

isAnimating={isAnimating}

color="green"

/>

</div>

);

};

// Individual metric card with animation

const MetricCard: React.FC<{

title: string;

value: number;

suffix?: string;

icon: React.ComponentType<any>;

isAnimating: boolean;

color: string;

formatter?: 'number' | 'percentage' | 'currency';

}> = ({ title, value, suffix = '', icon: Icon, isAnimating, color, formatter = 'number' }) => {

const formatValue = (val: number) => {

switch (formatter) {

case 'number':

return Math.floor(val).toLocaleString();

case 'percentage':

return val.toFixed(1);

case 'currency':

return val.toFixed(2);

default:

return val.toString();

}

};

const colorClasses = {

emerald: 'text-emerald-400 bg-emerald-500/10',

blue: 'text-blue-400 bg-blue-500/10',

green: 'text-green-400 bg-green-500/10',

};

return (

<Card className="bg-slate-700/50 rounded-lg p-6 transition-all duration-300 hover:bg-slate-700/70">

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

<Icon className={`h-8 w-8 ${colorClasses[color as keyof typeof colorClasses].split(' ')[0]}`} />

{isAnimating && (

<div className="animate-pulse">

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

</div>

)}

</div>

<div className={`text-3xl font-bold mb-1 transition-all duration-300 ${isAnimating ? 'scale-105' : ''}`}>

<span className={colorClasses[color as keyof typeof colorClasses].split(' ')[0]}>

{formatValue(value)}{suffix}

</span>

</div>

<div className="text-slate-400 text-sm">{title}</div>

</Card>

);

};

**Interactive Chart Components**

// Interactive signal score distribution chart

const SignalScoreDistribution: React.FC<{

data: ScoreDistributionData[];

onScoreRangeClick: (range: ScoreRange) => void;

}> = ({ data, onScoreRangeClick }) => {

const [hoveredRange, setHoveredRange] = useState<string | null>(null);

const chartData = {

labels: data.map(d => d.label),

datasets: [{

label: 'Signal Distribution',

data: data.map(d => d.count),

backgroundColor: data.map((d, index) =>

hoveredRange === d.range ? d.hoverColor : d.color

),

borderColor: data.map(d => d.borderColor),

borderWidth: 2,

}]

};

const options = {

responsive: true,

maintainAspectRatio: false,

plugins: {

legend: {

display: false,

},

tooltip: {

callbacks: {

label: (context: any) => {

const total = data.reduce((sum, item) => sum + item.count, 0);

const percentage = ((context.parsed / total) \* 100).toFixed(1);

return `${context.parsed} signals (${percentage}%)`;

}

}

}

},

onHover: (event: any, elements: any[]) => {

if (elements.length > 0) {

const index = elements[0].index;

setHoveredRange(data[index].range);

} else {

setHoveredRange(null);

}

},

onClick: (event: any, elements: any[]) => {

if (elements.length > 0) {

const index = elements[0].index;

onScoreRangeClick(data[index]);

}

},

};

return (

<div className="h-64">

<Bar data={chartData} options={options} />

</div>

);

};

// Process flow visualization with interactive steps

const InteractiveProcessFlow: React.FC<{

steps: ProcessStep[];

activeStep: string | null;

onStepClick: (step: ProcessStep) => void;

}> = ({ steps, activeStep, onStepClick }) => {

return (

<div className="relative">

{/\* Connection lines \*/}

<svg className="absolute inset-0 w-full h-full pointer-events-none">

{steps.slice(0, -1).map((\_, index) => (

<g key={index}>

<defs>

<linearGradient

id={`gradient-${index}`}

x1="0%"

y1="0%"

x2="100%"

y2="0%"

>

<stop offset="0%" stopColor="rgb(59, 130, 246)" stopOpacity="0.5" />

<stop offset="100%" stopColor="rgb(147, 51, 234)" stopOpacity="0.5" />

</linearGradient>

</defs>

<path

d={`M ${(index + 1) \* 200 - 50} 100 Q ${(index + 1) \* 200 + 25} 100 ${(index + 1) \* 200 + 100} 100`}

stroke={`url(#gradient-${index})`}

strokeWidth="2"

fill="none"

className="animate-pulse"

/>

</g>

))}

</svg>

{/\* Step circles \*/}

<div className="flex justify-between items-center relative z-10">

{steps.map((step, index) => (

<div

key={step.id}

className="flex flex-col items-center cursor-pointer group"

onClick={() => onStepClick(step)}

>

<div

className={`

w-20 h-20 rounded-full flex items-center justify-center

transition-all duration-300 border-2

${activeStep === step.id

? 'bg-blue-500 border-blue-400 scale-110'

: 'bg-slate-700 border-slate-600 group-hover:border-blue-400 group-hover:scale-105'

}

`}

>

<step.icon className={`h-8 w-8 ${activeStep === step.id ? 'text-white' : 'text-slate-300'}`} />

</div>

<div className="mt-2 text-center">

<div className={`text-sm font-medium ${activeStep === step.id ? 'text-blue-400' : 'text-white'}`}>

{step.title}

</div>

<div className="text-xs text-slate-400">Step {index + 1}</div>

</div>

</div>

))}

</div>

</div>

);

};

**Responsive Chart Behavior**

// Responsive chart container with mobile optimizations

const ResponsiveChartContainer: React.FC<{

children: React.ReactNode;

title: string;

height?: {

mobile: string;

desktop: string;

};

}> = ({ children, title, height = { mobile: '250px', desktop: '400px' } }) => {

const [isMobile, setIsMobile] = useState(false);

useEffect(() => {

const checkMobile = () => {

setIsMobile(window.innerWidth < 768);

};

checkMobile();

window.addEventListener('resize', checkMobile);

return () => window.removeEventListener('resize', checkMobile);

}, []);

return (

<div className="w-full">

<h3 className="text-lg font-semibold text-white mb-4">{title}</h3>

<div

className="w-full"

style={{

height: isMobile ? height.mobile : height.desktop

}}

>

{children}

</div>

</div>

);

};

// Mobile-optimized chart options

const getMobileChartOptions = (baseOptions: any) => ({

...baseOptions,

plugins: {

...baseOptions.plugins,

legend: {

...baseOptions.plugins?.legend,

position: 'bottom',

labels: {

...baseOptions.plugins?.legend?.labels,

boxWidth: 12,

padding: 10,

}

},

tooltip: {

...baseOptions.plugins?.tooltip,

titleFont: { size: 12 },

bodyFont: { size: 11 },

}

},

scales: {

...baseOptions.scales,

x: {

...baseOptions.scales?.x,

ticks: {

...baseOptions.scales?.x?.ticks,

maxTicksLimit: 6,

fontSize: 10,

}

},

y: {

...baseOptions.scales?.y,

ticks: {

...baseOptions.scales?.y?.ticks,

maxTicksLimit: 5,

fontSize: 10,

}

}

}

});

**Chart Data Processing and Calculations**

// Data processing utilities for charts

class ChartDataProcessor {

static processBacktestingData(rawData: RawBacktestingData[]): BacktestingDataPoint[] {

return rawData.map((point, index) => {

const cumulativeReturn = this.calculateCumulativeReturn(rawData.slice(0, index + 1));

const benchmarkReturn = this.calculateBenchmarkReturn(point.date);

return {

date: point.date,

cumulativeReturn,

benchmarkReturn,

drawdown: this.calculateDrawdown(rawData.slice(0, index + 1)),

winRate: this.calculateRunningWinRate(rawData.slice(0, index + 1)),

};

});

}

static calculateCumulativeReturn(data: RawBacktestingData[]): number {

return data.reduce((cumulative, point) => {

return cumulative \* (1 + point.return / 100);

}, 1) - 1;

}

static calculateDrawdown(data: RawBacktestingData[]): number {

const cumulativeReturns = data.map((\_, index) =>

this.calculateCumulativeReturn(data.slice(0, index + 1))

);

let maxDrawdown = 0;

let peak = cumulativeReturns[0];

for (const value of cumulativeReturns) {

if (value > peak) {

peak = value;

}

const drawdown = (peak - value) / peak;

maxDrawdown = Math.max(maxDrawdown, drawdown);

}

return maxDrawdown \* 100;

}

static calculateRunningWinRate(data: RawBacktestingData[]): number {

const wins = data.filter(point => point.return > 0).length;

return (wins / data.length) \* 100;

}

static processSignalDistribution(signals: Signal[]): ScoreDistributionData[] {

const ranges = [

{ min: 80, max: 100, label: 'Strong (80-100)', color: 'rgba(34, 197, 94, 0.8)', range: 'strong' },

{ min: 60, max: 79, label: 'Valid (60-79)', color: 'rgba(59, 130, 246, 0.8)', range: 'valid' },

{ min: 40, max: 59, label: 'Weak (40-59)', color: 'rgba(251, 191, 36, 0.8)', range: 'weak' },

{ min: 0, max: 39, label: 'Ignore (0-39)', color: 'rgba(239, 68, 68, 0.8)', range: 'ignore' },

];

return ranges.map(range => {

const count = signals.filter(signal =>

signal.score >= range.min && signal.score <= range.max

).length;

return {

...range,

count,

hoverColor: range.color.replace('0.8', '1'),

borderColor: range.color.replace('0.8', '1'),

};

});

}

static smoothData(data: number[], windowSize: number = 5): number[] {

const smoothed: number[] = [];

for (let i = 0; i < data.length; i++) {

const start = Math.max(0, i - Math.floor(windowSize / 2));

const end = Math.min(data.length, i + Math.ceil(windowSize / 2));

const window = data.slice(start, end);

const average = window.reduce((sum, val) => sum + val, 0) / window.length;

smoothed.push(average);

}

return smoothed;

}

}

// Real-time data streaming for charts

const useRealTimeChartData = (endpoint: string, interval: number = 30000) => {

const [data, setData] = useState<any[]>([]);

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

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

useEffect(() => {

const fetchData = async () => {

try {

const response = await fetch(endpoint);

if (!response.ok) throw new Error('Failed to fetch data');

const newData = await response.json();

setData(prevData => {

// Keep only last 100 data points for performance

const combined = [...prevData, ...newData].slice(-100);

return combined;

});

setError(null);

} catch (err) {

setError(err instanceof Error ? err.message : 'Unknown error');

} finally {

setIsLoading(false);

}

};

// Initial fetch

fetchData();

// Set up interval for real-time updates

const intervalId = setInterval(fetchData, interval);

return () => clearInterval(intervalId);

}, [endpoint, interval]);

return { data, isLoading, error };

};

**10. Visual Data Elements**

**Progress Indicators and Dynamic Counters**

// Circular progress indicator for signal confidence

const CircularProgress: React.FC<{

value: number;

max: number;

size: number;

strokeWidth: number;

color: string;

label?: string;

animated?: boolean;

}> = ({ value, max, size, strokeWidth, color, label, animated = true }) => {

const [animatedValue, setAnimatedValue] = useState(0);

const radius = (size - strokeWidth) / 2;

const circumference = radius \* 2 \* Math.PI;

const progress = (animatedValue / max) \* 100;

const strokeDasharray = `${circumference} ${circumference}`;

const strokeDashoffset = circumference - (progress / 100) \* circumference;

useEffect(() => {

if (!animated) {

setAnimatedValue(value);

return;

}

const duration = 1500;

const startTime = Date.now();

const startValue = animatedValue;

const animate = () => {

const elapsed = Date.now() - startTime;

const progress = Math.min(elapsed / duration, 1);

const easeOutCubic = 1 - Math.pow(1 - progress, 3);

setAnimatedValue(startValue + (value - startValue) \* easeOutCubic);

if (progress < 1) {

requestAnimationFrame(animate);

}

};

requestAnimationFrame(animate);

}, [value, animated, animatedValue]);

return (

<div className="relative inline-flex items-center justify-center">

<svg

className="transform -rotate-90"

width={size}

height={size}

>

{/\* Background circle \*/}

<circle

cx={size / 2}

cy={size / 2}

r={radius}

stroke="rgb(51, 65, 85)"

strokeWidth={strokeWidth}

fill="transparent"

/>

{/\* Progress circle \*/}

<circle

cx={size / 2}

cy={size / 2}

r={radius}

stroke={color}

strokeWidth={strokeWidth}

fill="transparent"

strokeDasharray={strokeDasharray}

strokeDashoffset={strokeDashoffset}

strokeLinecap="round"

className="transition-all duration-500 ease-out"

/>

</svg>

<div className="absolute inset-0 flex flex-col items-center justify-center">

<span className="text-lg font-bold text-white">

{Math.round(animatedValue)}

</span>

{label && (

<span className="text-xs text-slate-400">{label}</span>

)}

</div>

</div>

);

};

// Linear progress bar with animated fill

const LinearProgress: React.FC<{

value: number;

max: number;

height?: string;

color?: string;

backgroundColor?: string;

label?: string;

showPercentage?: boolean;

}> = ({

value,

max,

height = 'h-3',

color = 'bg-blue-500',

backgroundColor = 'bg-slate-700',

label,

showPercentage = true

}) => {

const [animatedValue, setAnimatedValue] = useState(0);

const percentage = (animatedValue / max) \* 100;

useEffect(() => {

const timer = setTimeout(() => {

setAnimatedValue(value);

}, 100);

return () => clearTimeout(timer);

}, [value]);

return (

<div className="w-full">

{label && (

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

<span className="text-sm text-slate-300">{label}</span>

{showPercentage && (

<span className="text-sm text-slate-400">{Math.round(percentage)}%</span>

)}

</div>

)}

<div className={`w-full ${height} ${backgroundColor} rounded-full overflow-hidden`}>

<div

className={`${height} ${color} rounded-full transition-all duration-1000 ease-out`}

style={{ width: `${percentage}%` }}

/>

</div>

</div>

);

};

// Animated number counter with formatting options

const AnimatedCounter: React.FC<{

value: number;

duration?: number;

formatter?: (value: number) => string;

className?: string;

prefix?: string;

suffix?: string;

}> = ({

value,

duration = 2000,

formatter = (v) => v.toString(),

className = '',

prefix = '',

suffix = ''

}) => {

const [displayValue, setDisplayValue] = useState(0);

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

const ref = useRef<HTMLDivElement>(null);

useEffect(() => {

const observer = new IntersectionObserver(

([entry]) => {

if (entry.isIntersecting) {

setIsVisible(true);

observer.disconnect();

}

},

{ threshold: 0.5 }

);

if (ref.current) {

observer.observe(ref.current);

}

return () => observer.disconnect();

}, []);

useEffect(() => {

if (!isVisible) return;

const startTime = Date.now();

const startValue = displayValue;

const animate = () => {

const elapsed = Date.now() - startTime;

const progress = Math.min(elapsed / duration, 1);

// Easing function for smooth animation

const easeOutExpo = progress === 1 ? 1 : 1 - Math.pow(2, -10 \* progress);

const currentValue = startValue + (value - startValue) \* easeOutExpo;

setDisplayValue(currentValue);

if (progress < 1) {

requestAnimationFrame(animate);

}

};

requestAnimationFrame(animate);

}, [isVisible, value, duration, displayValue]);

return (

<div ref={ref} className={className}>

{prefix}{formatter(displayValue)}{suffix}

</div>

);

};

**Icon Systems and Visual Hierarchy**

// Icon mapping for different signal types and statuses

const SignalIcon: React.FC<{

type: 'BUY' | 'SELL' | 'HOLD';

score: number;

size?: 'sm' | 'md' | 'lg';

animated?: boolean;

}> = ({ type, score, size = 'md', animated = false }) => {

const sizeClasses = {

sm: 'h-4 w-4',

md: 'h-6 w-6',

lg: 'h-8 w-8'

};

const getIconAndColor = () => {

if (score >= 80) {

return {

icon: type === 'BUY' ? TrendingUp : TrendingDown,

color: 'text-emerald-400',

bgColor: 'bg-emerald-500/20'

};

} else if (score >= 60) {

return {

icon: type === 'BUY' ? ArrowUp : ArrowDown,

color: 'text-blue-400',

bgColor: 'bg-blue-500/20'

};

} else if (score >= 40) {

return {

icon: Minus,

color: 'text-yellow-400',

bgColor: 'bg-yellow-500/20'

};

} else {

return {

icon: X,

color: 'text-red-400',

bgColor: 'bg-red-500/20'

};

}

};

const { icon: Icon, color, bgColor } = getIconAndColor();

return (

<div className={`

${bgColor} p-2 rounded-full inline-flex items-center justify-center

${animated ? 'animate-pulse' : ''}

`}>

<Icon className={`${sizeClasses[size]} ${color}`} />

</div>

);

};

// Status indicator with pulse animation

const StatusIndicator: React.FC<{

status: 'active' | 'inactive' | 'warning' | 'error';

label: string;

size?: 'sm' | 'md' | 'lg';

showPulse?: boolean;

}> = ({ status, label, size = 'md', showPulse = true }) => {

const statusConfig = {

active: {

color: 'bg-emerald-500',

textColor: 'text-emerald-400',

borderColor: 'border-emerald-500/30'

},

inactive: {

color: 'bg-slate-500',

textColor: 'text-slate-400',

borderColor: 'border-slate-500/30'

},

warning: {

color: 'bg-yellow-500',

textColor: 'text-yellow-400',

borderColor: 'border-yellow-500/30'

},

error: {

color: 'bg-red-500',

textColor: 'text-red-400',

borderColor: 'border-red-500/30'

}

};

const sizeClasses = {

sm: 'h-2 w-2',

md: 'h-3 w-3',

lg: 'h-4 w-4'

};

const config = statusConfig[status];

return (

<div className={`

flex items-center space-x-2 px-3 py-1 rounded-full border

${config.borderColor} bg-slate-800/50

`}>

<div className="relative">

<div className={`

${sizeClasses[size]} ${config.color} rounded-full

`} />

{showPulse && status === 'active' && (

<div className={`

absolute inset-0 ${config.color} rounded-full animate-ping opacity-30

`} />

)}

</div>

<span className={`text-sm ${config.textColor}`}>

{label}

</span>

</div>

);

};

// Achievement badges and awards

const AchievementBadge: React.FC<{

title: string;

icon: React.ComponentType<any>;

color: string;

description?: string;

earned?: boolean;

}> = ({ title, icon: Icon, color, description, earned = false }) => {

return (

<div className={`

relative p-4 rounded-lg border transition-all duration-300

${earned

? `${color} border-current bg-current/10 scale-100`

: 'border-slate-600 bg-slate-800/50 opacity-50 scale-95'

}

hover:scale-105

`}>

{earned && (

<div className="absolute -top-2 -right-2">

<div className="bg-emerald-500 rounded-full p-1">

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

</div>

</div>

)}

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

<div className={`

p-2 rounded-full

${earned ? 'bg-current/20' : 'bg-slate-700'}

`}>

<Icon className={`h-6 w-6 ${earned ? 'text-current' : 'text-slate-400'}`} />

</div>

<div>

<h4 className={`font-medium ${earned ? 'text-white' : 'text-slate-400'}`}>

{title}

</h4>

{description && (

<p className={`text-sm ${earned ? 'text-slate-300' : 'text-slate-500'}`}>

{description}

</p>

)}

</div>

</div>

</div>

);

};

**Color-coded Status Systems**

// Signal quality color system

const getSignalQualityColor = (score: number) => {

if (score >= 80) {

return {

background: 'bg-gradient-to-r from-emerald-900/50 to-emerald-800/30',

border: 'border-emerald-700',

text: 'text-emerald-400',

icon: 'text-emerald-400',

badge: 'bg-emerald-500/20 text-emerald-400'

};

} else if (score >= 60) {

return {

background: 'bg-gradient-to-r from-blue-900/50 to-blue-800/30',

border: 'border-blue-700',

text: 'text-blue-400',

icon: 'text-blue-400',

badge: 'bg-blue-500/20 text-blue-400'

};

} else if (score >= 40) {

return {

background: 'bg-gradient-to-r from-yellow-900/50 to-yellow-800/30',

border: 'border-yellow-700',

text: 'text-yellow-400',

icon: 'text-yellow-400',

badge: 'bg-yellow-500/20 text-yellow-400'

};

} else {

return {

background: 'bg-gradient-to-r from-red-900/50 to-red-800/30',

border: 'border-red-700',

text: 'text-red-400',

icon: 'text-red-400',

badge: 'bg-red-500/20 text-red-400'

};

}

};

// Market condition indicator

const MarketConditionIndicator: React.FC<{

condition: 'BULL' | 'BEAR' | 'SIDEWAYS';

confidence: number;

}> = ({ condition, confidence }) => {

const conditionConfig = {

BULL: {

color: 'text-emerald-400',

bgColor: 'bg-emerald-500/20',

icon: TrendingUp,

label: 'Bull Market'

},

BEAR: {

color: 'text-red-400',

bgColor: 'bg-red-500/20',

icon: TrendingDown,

label: 'Bear Market'

},

SIDEWAYS: {

color: 'text-blue-400',

bgColor: 'bg-blue-500/20',

icon: Minus,

label: 'Sideways'

}

};

const config = conditionConfig[condition];

return (

<div className={`

flex items-center space-x-3 p-3 rounded-lg

${config.bgColor} border border-current/30

`}>

<config.icon className={`h-5 w-5 ${config.color}`} />

<div>

<div className={`font-medium ${config.color}`}>

{config.label}

</div>

<div className="text-sm text-slate-400">

{confidence}% confidence

</div>

</div>

<CircularProgress

value={confidence}

max={100}

size={40}

strokeWidth={3}

color={config.color.replace('text-', 'rgb(').replace('-400', ', 0.8)')}

/>

</div>

);

};

// Risk level indicator with gradient

const RiskLevelIndicator: React.FC<{

level: 'LOW' | 'MEDIUM' | 'HIGH';

value: number;

}> = ({ level, value }) => {

const riskConfig = {

LOW: {

color: 'from-emerald-500 to-green-400',

textColor: 'text-emerald-400',

icon: Shield

},

MEDIUM: {

color: 'from-yellow-500 to-orange-400',

textColor: 'text-yellow-400',

icon: AlertTriangle

},

HIGH: {

color: 'from-red-500 to-pink-400',

textColor: 'text-red-400',

icon: AlertOctagon

}

};

const config = riskConfig[level];

return (

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

<div className={`

w-12 h-12 rounded-full bg-gradient-to-r ${config.color}

flex items-center justify-center

`}>

<config.icon className="h-6 w-6 text-white" />

</div>

<div>

<div className={`font-medium ${config.textColor}`}>

{level} Risk

</div>

<div className="text-slate-400">

{value}% max exposure

</div>

</div>

</div>

);

};

**Typography Scale and Visual Feedback**

// Typography system for How It Works page

const Typography = {

// Headings

h1: 'text-4xl md:text-5xl font-bold text-white',

h2: 'text-2xl md:text-3xl font-bold text-white',

h3: 'text-xl md:text-2xl font-semibold text-white',

h4: 'text-lg md:text-xl font-semibold text-white',

// Body text

body: 'text-base text-slate-400',

bodyLarge: 'text-lg text-slate-400',

bodySmall: 'text-sm text-slate-400',

// Labels and captions

label: 'text-sm font-medium text-white',

caption: 'text-xs text-slate-500',

// Special text

accent: 'text-blue-400 font-medium',

success: 'text-emerald-400 font-medium',

warning: 'text-yellow-400 font-medium',

error: 'text-red-400 font-medium',

// Interactive text

link: 'text-blue-400 hover:text-blue-300 transition-colors cursor-pointer',

buttonText: 'text-white font-medium',

};

// Visual feedback for interactive elements

const InteractiveCard: React.FC<{

children: React.ReactNode;

onClick?: () => void;

isActive?: boolean;

isLoading?: boolean;

}> = ({ children, onClick, isActive = false, isLoading = false }) => {

const [isPressed, setIsPressed] = useState(false);

return (

<div

className={`

relative transition-all duration-200 cursor-pointer

${isActive

? 'ring-2 ring-blue-500 bg-blue-500/10'

: 'hover:bg-slate-700/50'

}

${isPressed ? 'scale-95' : 'scale-100'}

${isLoading ? 'opacity-50 pointer-events-none' : ''}

`}

onClick={onClick}

onMouseDown={() => setIsPressed(true)}

onMouseUp={() => setIsPressed(false)}

onMouseLeave={() => setIsPressed(false)}

>

{children}

{/\* Loading overlay \*/}

{isLoading && (

<div className="absolute inset-0 flex items-center justify-center bg-slate-900/50 rounded-lg">

<div className="w-6 h-6 border-2 border-blue-400 border-t-transparent rounded-full animate-spin" />

</div>

)}

{/\* Active indicator \*/}

{isActive && (

<div className="absolute top-2 right-2">

<div className="w-3 h-3 bg-blue-500 rounded-full animate-pulse" />

</div>

)}

</div>

);

};

// Tooltip component for additional information

const Tooltip: React.FC<{

content: string;

children: React.ReactNode;

position?: 'top' | 'bottom' | 'left' | 'right';

}> = ({ content, children, position = 'top' }) => {

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

const positionClasses = {

top: 'bottom-full left-1/2 transform -translate-x-1/2 mb-2',

bottom: 'top-full left-1/2 transform -translate-x-1/2 mt-2',

left: 'right-full top-1/2 transform -translate-y-1/2 mr-2',

right: 'left-full top-1/2 transform -translate-y-1/2 ml-2'

};

return (

<div

className="relative inline-block"

onMouseEnter={() => setIsVisible(true)}

onMouseLeave={() => setIsVisible(false)}

>

{children}

{isVisible && (

<div className={`

absolute z-50 px-3 py-2 text-sm text-white bg-slate-900

rounded-lg border border-slate-700 shadow-lg whitespace-nowrap

${positionClasses[position]}

animate-in fade-in-0 zoom-in-95 duration-200

`}>

{content}

<div className={`

absolute w-2 h-2 bg-slate-900 border border-slate-700 rotate-45

${position === 'top' ? 'top-full left-1/2 transform -translate-x-1/2 -translate-y-1/2' : ''}

${position === 'bottom' ? 'bottom-full left-1/2 transform -translate-x-1/2 translate-y-1/2' : ''}

${position === 'left' ? 'left-full top-1/2 transform -translate-x-1/2 -translate-y-1/2' : ''}

${position === 'right' ? 'right-full top-1/2 transform translate-x-1/2 -translate-y-1/2' : ''}

`} />

</div>

)}

</div>

);

};

**11. Security & Validation**

**Input Validation Schemas**

import { z } from 'zod';

// Validation schemas for How It Works interactions

const StepInteractionSchema = z.object({

stepId: z.string().min(1, 'Step ID is required'),

interactionType: z.enum(['view', 'click', 'hover', 'expand']),

timestamp: z.date(),

sessionId: z.string().uuid('Invalid session ID'),

metadata: z.record(z.unknown()).optional(),

});

const ExampleSignalRequestSchema = z.object({

filters: z.object({

scoreRange: z.object({

min: z.number().min(0).max(100),

max: z.number().min(0).max(100),

}).optional(),

signalType: z.enum(['BUY', 'SELL', 'ALL']).optional(),

timeframe: z.enum(['1D', '1W', '1M', '3M', '1Y']).optional(),

}).optional(),

limit: z.number().min(1).max(50).default(10),

offset: z.number().min(0).default(0),

});

const PerformanceMetricsRequestSchema = z.object({

timeRange: z.enum(['1M', '3M', '6M', '1Y', 'ALL']).default('1Y'),

marketCondition: z.enum(['BULL', 'BEAR', 'SIDEWAYS', 'ALL']).optional(),

includeDrawdown: z.boolean().default(true),

includeBenchmark: z.boolean().default(true),

});

// Validation hooks

const useValidatedStepInteraction = () => {

const validateAndTrack = useCallback(async (data: unknown) => {

try {

const validatedData = StepInteractionSchema.parse(data);

// Additional business logic validation

const allowedSteps = ['multi\_timeframe', 'support\_resistance', 'options\_indicators', 'risk\_management', 'validation\_scoring', 'ai\_enhancement'];

if (!allowedSteps.includes(validatedData.stepId)) {

throw new Error('Invalid step ID');

}

// Track the interaction

await trackStepInteraction(validatedData);

return { success: true, data: validatedData };

} catch (error) {

if (error instanceof z.ZodError) {

return {

success: false,

error: 'Invalid input data',

details: error.errors

};

}

return {

success: false,

error: error instanceof Error ? error.message : 'Unknown error'

};

}

}, []);

return validateAndTrack;

};

// Form validation for user preferences

const UserPreferencesSchema = z.object({

animationsEnabled: z.boolean(),

autoPlayExamples: z.boolean(),

preferredTimeRange: z.enum(['1M', '3M', '6M', '1Y']),

language: z.enum(['en', 'de', 'ar']),

theme: z.enum(['dark', 'light']).default('dark'),

notificationPreferences: z.object({

emailAlerts: z.boolean(),

pushNotifications: z.boolean(),

telegramAlerts: z.boolean(),

}),

});

**Authentication Requirements**

// Authentication context for How It Works page

interface AuthContext {

user: User | null;

isAuthenticated: boolean;

hasPermission: (permission: string) => boolean;

requireAuth: () => void;

}

// Permission-based access control

const useHowItWorksPermissions = () => {

const { user, hasPermission } = useAuth();

const permissions = useMemo(() => ({

viewBasicInfo: true, // Public access

viewDetailedExamples: hasPermission('view\_signal\_examples'),

viewPerformanceMetrics: hasPermission('view\_performance\_data'),

interactWithExamples: user !== null,

exportData: hasPermission('export\_data'),

accessAdvancedFeatures: hasPermission('access\_premium\_features'),

}), [user, hasPermission]);

return permissions;

};

// Protected component wrapper

const ProtectedSection: React.FC<{

permission: keyof ReturnType<typeof useHowItWorksPermissions>;

fallback?: React.ReactNode;

children: React.ReactNode;

}> = ({ permission, fallback, children }) => {

const permissions = useHowItWorksPermissions();

const { requireAuth } = useAuth();

if (!permissions[permission]) {

if (fallback) {

return <>{fallback}</>;

}

return (

<Card className="bg-slate-800/50 border-blue-500/30 p-6">

<div className="text-center">

<Lock className="h-12 w-12 text-blue-400 mx-auto mb-4" />

<h3 className="text-lg font-semibold text-white mb-2">

Premium Feature

</h3>

<p className="text-slate-400 mb-4">

Sign in to access detailed signal examples and performance data.

</p>

<Button onClick={requireAuth} className="bg-blue-600 hover:bg-blue-700">

Sign In to Continue

</Button>

</div>

</Card>

);

}

return <>{children}</>;

};

// Session management for analytics

const useSecureSession = () => {

const [sessionId] = useState(() => {

// Generate secure session ID

return crypto.randomUUID();

});

const [sessionStartTime] = useState(() => new Date());

const getSessionInfo = useCallback(() => ({

sessionId,

startTime: sessionStartTime,

duration: Date.now() - sessionStartTime.getTime(),

userAgent: navigator.userAgent,

referrer: document.referrer,

}), [sessionId, sessionStartTime]);

return {

sessionId,

getSessionInfo,

};

};

**Data Sanitization and XSS Prevention**

import DOMPurify from 'dompurify';

// HTML sanitization utilities

const sanitizeHTML = (html: string): string => {

return DOMPurify.sanitize(html, {

ALLOWED\_TAGS: ['b', 'i', 'em', 'strong', 'code', 'pre'],

ALLOWED\_ATTR: [],

});

};

// Text sanitization for user inputs

const sanitizeText = (text: string): string => {

return text

.replace(/[<>]/g, '') // Remove angle brackets

.replace(/javascript:/gi, '') // Remove javascript protocol

.replace(/on\w+=/gi, '') // Remove event handlers

.trim();

};

// Safe component for rendering dynamic content

const SafeHTML: React.FC<{

content: string;

className?: string;

}> = ({ content, className }) => {

const sanitizedContent = useMemo(() => {

return sanitizeHTML(content);

}, [content]);

return (

<div

className={className}

dangerouslySetInnerHTML={{ \_\_html: sanitizedContent }}

/>

);

};

// Input sanitization hook

const useSanitizedInput = (initialValue: string = '') => {

const [value, setValue] = useState(initialValue);

const [sanitizedValue, setSanitizedValue] = useState(initialValue);

const handleChange = useCallback((newValue: string) => {

setValue(newValue);

setSanitizedValue(sanitizeText(newValue));

}, []);

return {

value,

sanitizedValue,

onChange: handleChange,

};

};

// XSS prevention for dynamic URLs

const sanitizeURL = (url: string): string => {

try {

const parsed = new URL(url);

// Only allow specific protocols

if (!['http:', 'https:', 'mailto:'].includes(parsed.protocol)) {

return '#';

}

// Block javascript and data URLs

if (parsed.protocol === 'javascript:' || parsed.protocol === 'data:') {

return '#';

}

return parsed.toString();

} catch {

return '#';

}

};

**Rate Limiting and API Security**

// Client-side rate limiting

class ClientRateLimit {

private requests: Map<string, number[]> = new Map();

private limits: Map<string, { count: number; window: number }> = new Map();

constructor() {

// Set up rate limits for different endpoints

this.limits.set('step-interaction', { count: 10, window: 60000 }); // 10 per minute

this.limits.set('example-request', { count: 5, window: 60000 }); // 5 per minute

this.limits.set('performance-data', { count: 20, window: 300000 }); // 20 per 5 minutes

}

canMakeRequest(endpoint: string): boolean {

const now = Date.now();

const limit = this.limits.get(endpoint);

if (!limit) return true;

const requests = this.requests.get(endpoint) || [];

// Remove old requests outside the window

const validRequests = requests.filter(time => now - time < limit.window);

if (validRequests.length >= limit.count) {

return false;

}

// Add current request

validRequests.push(now);

this.requests.set(endpoint, validRequests);

return true;

}

getTimeUntilReset(endpoint: string): number {

const limit = this.limits.get(endpoint);

if (!limit) return 0;

const requests = this.requests.get(endpoint) || [];

if (requests.length === 0) return 0;

const oldestRequest = Math.min(...requests);

const resetTime = oldestRequest + limit.window;

return Math.max(0, resetTime - Date.now());

}

}

const rateLimit = new ClientRateLimit();

// Rate-limited API hook

const useRateLimitedAPI = () => {

const [isBlocked, setIsBlocked] = useState(false);

const [timeUntilReset, setTimeUntilReset] = useState(0);

const makeRequest = useCallback(async (

endpoint: string,

requestFn: () => Promise<any>

) => {

if (!rateLimit.canMakeRequest(endpoint)) {

const resetTime = rateLimit.getTimeUntilReset(endpoint);

setIsBlocked(true);

setTimeUntilReset(resetTime);

// Auto-unblock when rate limit resets

setTimeout(() => {

setIsBlocked(false);

setTimeUntilReset(0);

}, resetTime);

throw new Error(`Rate limit exceeded. Try again in ${Math.ceil(resetTime / 1000)} seconds.`);

}

return await requestFn();

}, []);

return {

makeRequest,

isBlocked,

timeUntilReset,

};

};

// CSRF protection for API requests

const useCSRFProtection = () => {

const [csrfToken, setCSRFToken] = useState<string | null>(null);

useEffect(() => {

// Get CSRF token from meta tag or cookie

const token = document.querySelector('meta[name="csrf-token"]')?.getAttribute('content') ||

document.cookie.split('; ').find(row => row.startsWith('csrf-token='))?.split('=')[1];

setCSRFToken(token || null);

}, []);

const addCSRFHeaders = useCallback((headers: HeadersInit = {}): HeadersInit => {

if (csrfToken)