**Dashboard**

**Kurzora Dashboard - Complete UI Analysis**

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

**1. UI Components & Layout**

**Interactive Elements**

**Primary Dashboard Components:**

* **Metric Cards** (4 main KPIs: Today's Signals, Active Signals, Avg Score, Success Rate)
* **Performance Summary Cards** (Best Performer, Active Signals Count, Latest Signal, Alerts)
* **Recent Trades Panel** with navigation to Historical Trades
* **WinRateGauge** (circular chart component)
* **PortfolioPerformanceChart** (line chart vs S&P 500)
* **SignalHeatmap** (main data table with filtering)
* **SignalModal** (detailed signal view with paper trading)
* **WelcomeBanner** (conditional onboarding)

**Interactive Controls:**

* Auto-refresh toggle for real-time updates
* Signal filtering (timeframe, score, sector, market)
* Paper trading modal with risk management
* Navigation buttons and routing
* Portfolio balance adjustment
* Risk percentage slider (0.5-10%)

**React + TypeScript Component Structure**

// Complete Dashboard Architecture

<Layout>

<div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-8">

{/\* Conditional Welcome Banner \*/}

{showWelcome && (

<WelcomeBanner

planName={selectedPlan?.name}

onDismiss={handleDismissWelcome}

onExploreFeatures={handleExploreFeatures}

onCompleteSetup={handleCompleteSetup}

/>

)}

{/\* Welcome Header \*/}

<div className="mb-8">

<h1 className="text-3xl font-bold text-white mb-2">

Welcome, {user.name}!

</h1>

<p className="text-slate-400 mb-2">

Welcome to your Kurzora trading intelligence dashboard

</p>

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

💡 Click any signal below or go to the Signals tab to get started.

</p>

</div>

{/\* Key Metrics Cards Grid \*/}

<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6 mb-8">

<MetricCard

title="Today's Signals"

value="12"

subtitle="3 Strong, 9 Valid"

icon={Bell}

iconColor="text-blue-400"

/>

{/\* ... other metric cards \*/}

</div>

{/\* Performance Summary + Recent Trades \*/}

<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-5 gap-6 mb-8">

<div className="lg:col-span-4 grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6">

<PerformanceCard type="bestPerformer" />

<PerformanceCard type="activeSignals" />

<PerformanceCard type="latestSignal" />

<PerformanceCard type="alerts" />

</div>

<div className="lg:col-span-1">

<RecentTradesPanel />

</div>

</div>

{/\* Charts Section \*/}

<div className="grid grid-cols-1 lg:grid-cols-5 gap-8 mb-8">

<div className="lg:col-span-2">

<WinRateGauge winRate={61} totalTrades={127} winningTrades={77} />

</div>

<div className="lg:col-span-3">

<PortfolioPerformanceChart />

</div>

</div>

{/\* Signal Heatmap \*/}

<div className="mb-8">

<SignalHeatmap onOpenSignalModal={handleOpenSignalModal} />

</div>

{/\* Disclaimer \*/}

<div className="text-xs text-gray-500 text-center mt-4">

Simulation disclaimer text

</div>

{/\* Signal Modal \*/}

<SignalModal

isOpen={signalModalOpen}

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

signal={selectedSignal}

onExecuteTrade={handleExecuteTrade}

existingPositions={existingPositions}

/>

</div>

</Layout>

**Tailwind CSS Classes**

/\* Layout Structure \*/

.max-w-7xl.mx-auto.px-4.sm:px-6.lg:px-8.py-8 /\* Main container \*/

/\* Card Styling \*/

.bg-slate-900/50.backdrop-blur-sm.border-blue-800/30 /\* Metric cards \*/

.hover:bg-slate-900/70.transition-all.duration-300 /\* Hover effects \*/

/\* Grid Layouts \*/

.grid.grid-cols-1.md:grid-cols-2.lg:grid-cols-4.gap-6 /\* Responsive grids \*/

.grid.grid-cols-1.lg:grid-cols-5.gap-8 /\* Charts layout \*/

/\* Typography \*/

.text-3xl.font-bold.text-white /\* Main headings \*/

.text-slate-400 /\* Secondary text \*/

.text-emerald-400 /\* Success states \*/

/\* Interactive Elements \*/

.data-[state=checked]:bg-emerald-600 /\* Switch component \*/

.bg-emerald-600.hover:bg-emerald-700 /\* Primary buttons \*/

**Responsive Design Implementation**

// Mobile-first breakpoints

const breakpoints = {

sm: '640px', // Small tablets

md: '768px', // Tablets

lg: '1024px', // Desktops

xl: '1280px' // Large desktops

}

// Responsive patterns used

grid-cols-1 md:grid-cols-2 lg:grid-cols-4 // Metric cards

grid-cols-1 lg:grid-cols-5 // Charts section

flex-col lg:flex-row lg:items-center // Header elements

space-y-4 lg:space-y-0 // Vertical spacing adjustments

**Loading States & Error Handling**

// Dashboard Loading Skeleton

const DashboardSkeleton = () => (

<div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-8">

<div className="animate-pulse">

{/\* Header skeleton \*/}

<div className="h-8 bg-slate-700 rounded w-1/3 mb-2" />

<div className="h-4 bg-slate-700 rounded w-1/2 mb-8" />

{/\* Metric cards skeleton \*/}

<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6 mb-8">

{[...Array(4)].map((\_, i) => (

<div key={i} className="h-24 bg-slate-700 rounded-lg" />

))}

</div>

{/\* Charts skeleton \*/}

<div className="grid grid-cols-1 lg:grid-cols-5 gap-8 mb-8">

<div className="lg:col-span-2 h-64 bg-slate-700 rounded-lg" />

<div className="lg:col-span-3 h-64 bg-slate-700 rounded-lg" />

</div>

</div>

</div>

)

// Error boundary for dashboard

const DashboardErrorBoundary = ({ children }) => (

<ErrorBoundary

fallback={

<div className="text-center text-white p-8">

<h2 className="text-xl font-bold mb-4">Dashboard temporarily unavailable</h2>

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

Refresh Dashboard

</Button>

</div>

}

>

{children}

</ErrorBoundary>

)

**2. State Management (Zustand)**

**Store Structure**

interface DashboardStore {

// Dashboard Metrics

metrics: {

todaysSignals: number

activeSignals: number

avgSignalScore: number

successRate: number

newSignalsLastHour: number

}

// Performance Data

performance: {

bestPerformer: {

symbol: string

profit: number

percentage: number

}

latestSignal: {

symbol: string

score: number

timeAgo: string

}

alerts: {

count: number

type: string

}

}

// Portfolio Data

portfolio: {

balance: number

performance: number

vs\_sp500: number

chartData: PortfolioDataPoint[]

}

// Trading Data

trading: {

winRate: number

totalTrades: number

winningTrades: number

losingTrades: number

recentTrades: Trade[]

}

// Signals Data

signals: {

heatmapData: Signal[]

filters: {

timeframe: string

minScore: number

sector: string

market: string

}

autoRefresh: boolean

lastUpdated: string

}

// UI State

ui: {

showWelcome: boolean

selectedPlan: any

signalModalOpen: boolean

selectedSignal: Signal | null

loading: boolean

error: string | null

}

// Actions

updateMetrics: () => Promise<void>

updatePortfolioData: () => Promise<void>

updateSignalsData: () => Promise<void>

setSignalFilters: (filters: Partial<SignalFilters>) => void

openSignalModal: (signal: Signal) => void

closeSignalModal: () => void

executePaperTrade: (tradeData: any) => Promise<void>

dismissWelcome: () => void

toggleAutoRefresh: () => void

}

**Zustand Implementation**

const useDashboardStore = create<DashboardStore>((set, get) => ({

// Initial state

metrics: {

todaysSignals: 0,

activeSignals: 0,

avgSignalScore: 0,

successRate: 0,

newSignalsLastHour: 0

},

performance: {

bestPerformer: { symbol: '', profit: 0, percentage: 0 },

latestSignal: { symbol: '', score: 0, timeAgo: '' },

alerts: { count: 0, type: '' }

},

portfolio: {

balance: 8000,

performance: 0,

vs\_sp500: 0,

chartData: []

},

trading: {

winRate: 0,

totalTrades: 0,

winningTrades: 0,

losingTrades: 0,

recentTrades: []

},

signals: {

heatmapData: [],

filters: {

timeframe: '1D',

minScore: 70,

sector: 'all',

market: 'global'

},

autoRefresh: true,

lastUpdated: ''

},

ui: {

showWelcome: false,

selectedPlan: null,

signalModalOpen: false,

selectedSignal: null,

loading: false,

error: null

},

// Actions

updateMetrics: async () => {

set({ ui: { ...get().ui, loading: true } })

try {

const response = await fetch('/api/dashboard/metrics')

const metrics = await response.json()

set({ metrics, ui: { ...get().ui, loading: false } })

} catch (error) {

set({ ui: { ...get().ui, error: error.message, loading: false } })

}

},

updatePortfolioData: async () => {

try {

const response = await fetch('/api/dashboard/portfolio')

const portfolio = await response.json()

set({ portfolio })

} catch (error) {

console.error('Failed to update portfolio data:', error)

}

},

updateSignalsData: async () => {

try {

const { filters } = get().signals

const response = await fetch(`/api/signals?${new URLSearchParams(filters)}`)

const heatmapData = await response.json()

set({

signals: {

...get().signals,

heatmapData,

lastUpdated: new Date().toISOString()

}

})

} catch (error) {

console.error('Failed to update signals data:', error)

}

},

setSignalFilters: (newFilters) => {

const currentFilters = get().signals.filters

set({

signals: {

...get().signals,

filters: { ...currentFilters, ...newFilters }

}

})

// Automatically refresh data when filters change

get().updateSignalsData()

},

openSignalModal: (signal) => {

set({

ui: {

...get().ui,

signalModalOpen: true,

selectedSignal: signal

}

})

},

closeSignalModal: () => {

set({

ui: {

...get().ui,

signalModalOpen: false,

selectedSignal: null

}

})

},

executePaperTrade: async (tradeData) => {

try {

const response = await fetch('/api/paper-trades', {

method: 'POST',

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

body: JSON.stringify(tradeData)

})

if (response.ok) {

// Update recent trades

const updatedTrades = [tradeData, ...get().trading.recentTrades.slice(0, 4)]

set({

trading: {

...get().trading,

recentTrades: updatedTrades

}

})

get().closeSignalModal()

}

} catch (error) {

console.error('Failed to execute paper trade:', error)

}

},

toggleAutoRefresh: () => {

const autoRefresh = !get().signals.autoRefresh

set({

signals: {

...get().signals,

autoRefresh

}

})

}

}))

**Local vs Global State Decisions**

// Local State (Component Level)

const [portfolioBalance, setPortfolioBalance] = useState(8000) // Modal-specific

const [customRiskPercent, setCustomRiskPercent] = useState([2]) // Modal-specific

const [showWelcome, setShowWelcome] = useState(false) // Page-specific

// Global State (Zustand Store)

const { metrics, portfolio, signals } = useDashboardStore() // Dashboard data

const { user } = useAuthStore() // User authentication

const { theme, language } = useSettingsStore() // App settings

// Persistent State (localStorage)

localStorage.setItem('dashboardFilters', JSON.stringify(filters))

localStorage.setItem('portfolioBalance', portfolioBalance.toString())

**Real-time Updates with Auto-refresh**

// Auto-refresh implementation

const useAutoRefresh = () => {

const { autoRefresh, updateMetrics, updateSignalsData } = useDashboardStore()

useEffect(() => {

if (!autoRefresh) return

const interval = setInterval(() => {

updateMetrics()

updateSignalsData()

}, 2 \* 60 \* 1000) // Every 2 minutes

return () => clearInterval(interval)

}, [autoRefresh, updateMetrics, updateSignalsData])

}

// WebSocket for real-time signal updates

const useRealTimeSignals = () => {

const { updateSignalsData } = useDashboardStore()

useEffect(() => {

const ws = new WebSocket(process.env.REACT\_APP\_WS\_URL)

ws.onmessage = (event) => {

const data = JSON.parse(event.data)

if (data.type === 'signal\_update') {

updateSignalsData()

}

}

return () => ws.close()

}, [updateSignalsData])

}

**3. API Contracts & Integration**

**API Endpoints**

**Dashboard Metrics API**

// GET /api/dashboard/metrics

interface DashboardMetricsResponse {

todaysSignals: number

activeSignals: number

avgSignalScore: number

successRate: number

newSignalsLastHour: number

bestPerformer: {

symbol: string

profit: number

percentage: number

}

latestSignal: {

symbol: string

score: number

timeAgo: string

}

alerts: {

count: number

description: string

}

}

**Portfolio Performance API**

// GET /api/dashboard/portfolio

interface PortfolioResponse {

balance: number

performance: number

vs\_sp500: number

chartData: Array<{

date: string

portfolio: number

sp500: number

}>

winRate: number

totalTrades: number

winningTrades: number

losingTrades: number

}

**Signals Heatmap API**

// GET /api/signals?timeframe=1D&minScore=70&sector=all&market=global

interface SignalsResponse {

signals: Array<{

ticker: string

name: string

price: number

change: number

signals: {

'1H': number

'4H': number

'1D': number

'1W': number

}

finalScore: number

sector: string

market: string

}>

summary: {

strong: number // 90-100

valid: number // 80-89

weak: number // 70-79

total: number

}

lastUpdated: string

}

**Recent Trades API**

// GET /api/trades/recent?limit=5

interface RecentTradesResponse {

trades: Array<{

id: string

symbol: string

type: 'buy' | 'sell'

profit: number

percentage: number

date: string

status: 'closed' | 'open'

}>

}

**Paper Trading API**

// POST /api/paper-trades

interface PaperTradeRequest {

symbol: string

name: string

entryPrice: number

shares: number

stopLoss: number

takeProfit: number

investmentAmount: number

signalScore: number

riskPercentage: number

}

interface PaperTradeResponse {

id: string

status: 'created' | 'error'

message: string

trade: {

id: string

symbol: string

entryPrice: number

shares: number

stopLoss: number

takeProfit: number

createdAt: string

}

}

**Real-time Data Integration**

// WebSocket Events

interface WebSocketMessage {

type: 'signal\_update' | 'price\_update' | 'trade\_execution'

data: any

timestamp: string

}

// Signal Updates

type: 'signal\_update'

data: {

symbol: string

newScore: number

timeframe: string

updatedAt: string

}

// Price Updates

type: 'price\_update'

data: {

symbol: string

price: number

change: number

changePercent: number

}

// Trade Execution

type: 'trade\_execution'

data: {

tradeId: string

symbol: string

type: 'stop\_loss' | 'take\_profit' | 'manual\_close'

profit: number

percentage: number

}

**Error Response Format**

interface APIError {

error: {

code: 'VALIDATION\_ERROR' | 'UNAUTHORIZED' | 'RATE\_LIMITED' | 'INTERNAL\_ERROR'

message: string

details?: any

}

timestamp: string

path: string

}

// Error handling hook

const useAPIError = () => {

const handleError = (error: APIError) => {

switch (error.error.code) {

case 'UNAUTHORIZED':

// Redirect to login

break

case 'RATE\_LIMITED':

// Show rate limit message

break

default:

// Show generic error

break

}

}

return { handleError }

}

**4. Performance & Optimization**

**Lazy Loading Implementation**

// Component lazy loading

const WinRateGauge = lazy(() => import('../components/dashboard/WinRateGauge'))

const PortfolioPerformanceChart = lazy(() => import('../components/dashboard/PortfolioPerformanceChart'))

const SignalHeatmap = lazy(() => import('../components/dashboard/SignalHeatmap'))

const SignalModal = lazy(() => import('../components/signals/SignalModal'))

// Conditional loading based on viewport

const useLazyComponents = () => {

const [showCharts, setShowCharts] = useState(false)

useEffect(() => {

const observer = new IntersectionObserver(

(entries) => {

if (entries[0].isIntersecting) {

setShowCharts(true)

observer.disconnect()

}

},

{ threshold: 0.1 }

)

const chartsSection = document.getElementById('charts-section')

if (chartsSection) {

observer.observe(chartsSection)

}

return () => observer.disconnect()

}, [])

return showCharts

}

// Usage in Dashboard

const Dashboard = () => {

const showCharts = useLazyComponents()

return (

<div>

{/\* Metrics always visible \*/}

<MetricCards />

{/\* Charts loaded when visible \*/}

<div id="charts-section">

{showCharts ? (

<Suspense fallback={<ChartsSkeleton />}>

<WinRateGauge />

<PortfolioPerformanceChart />

</Suspense>

) : (

<ChartsSkeleton />

)}

</div>

</div>

)

}

**Memoization Strategies**

// Component memoization

const MemoizedMetricCard = React.memo(MetricCard)

const MemoizedPerformanceCard = React.memo(PerformanceCard)

const MemoizedSignalRow = React.memo(SignalRow)

// Expensive calculations

const portfolioMetrics = useMemo(() => {

return calculatePortfolioMetrics(trades, portfolioBalance)

}, [trades, portfolioBalance])

const filteredSignals = useMemo(() => {

return filterAndSortSignals(signals, filters)

}, [signals, filters])

const chartData = useMemo(() => {

return processChartData(rawPortfolioData, timeframe)

}, [rawPortfolioData, timeframe])

// Callback memoization

const handleSignalClick = useCallback((signal: Signal) => {

openSignalModal(signal)

}, [openSignalModal])

const handleFilterChange = useCallback((newFilters: Partial<SignalFilters>) => {

setSignalFilters(newFilters)

}, [setSignalFilters])

// Debounced operations

const debouncedFilterUpdate = useDebouncedCallback(

(filters: SignalFilters) => {

updateSignalsData(filters)

},

500

)

**Bundle Splitting & Code Optimization**

// Route-based splitting

const Dashboard = lazy(() => import('./pages/Dashboard'))

// Feature-based splitting

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

const AdvancedFilters = lazy(() => import('./components/filters/AdvancedFilters'))

// Third-party library splitting

const recharts = () => import('recharts')

const tradingView = () => import('./lib/tradingview-widget')

// Critical path optimization

const criticalComponents = [

'MetricCards',

'WelcomeBanner',

'NavigationBar'

]

// Preload critical data

const preloadDashboardData = async () => {

return Promise.all([

fetch('/api/dashboard/metrics'),

fetch('/api/signals?limit=10'),

fetch('/api/trades/recent?limit=5')

])

}

**5. Database Schema**

**PostgreSQL Tables**

-- Dashboard metrics (aggregated data)

CREATE TABLE dashboard\_metrics (

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

user\_id UUID NOT NULL REFERENCES users(id),

-- Daily metrics

todays\_signals INTEGER DEFAULT 0,

active\_signals INTEGER DEFAULT 0,

avg\_signal\_score DECIMAL(5,2) DEFAULT 0,

success\_rate DECIMAL(5,2) DEFAULT 0,

new\_signals\_last\_hour INTEGER DEFAULT 0,

-- Performance metrics

best\_performer\_symbol VARCHAR(10),

best\_performer\_profit DECIMAL(10,2),

best\_performer\_percentage DECIMAL(5,2),

latest\_signal\_symbol VARCHAR(10),

latest\_signal\_score INTEGER,

latest\_signal\_time TIMESTAMP WITH TIME ZONE,

alerts\_count INTEGER DEFAULT 0,

alerts\_description TEXT,

-- Timestamps

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

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

-- Portfolio performance tracking

CREATE TABLE portfolio\_performance (

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

user\_id UUID NOT NULL REFERENCES users(id),

-- Portfolio data

balance DECIMAL(12,2) NOT NULL,

total\_performance DECIMAL(5,2) DEFAULT 0,

vs\_sp500\_performance DECIMAL(5,2) DEFAULT 0,

-- Trading statistics

win\_rate DECIMAL(5,2) DEFAULT 0,

total\_trades INTEGER DEFAULT 0,

winning\_trades INTEGER DEFAULT 0,

losing\_trades INTEGER DEFAULT 0,

-- Daily snapshots

portfolio\_value DECIMAL(12,2),

sp500\_value DECIMAL(12,2),

date DATE NOT NULL,

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

UNIQUE(user\_id, date)

);

-- Trading signals (real-time data)

CREATE TABLE trading\_signals (

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

-- Stock information

ticker VARCHAR(10) NOT NULL,

company\_name VARCHAR(255) NOT NULL,

current\_price DECIMAL(10,2) NOT NULL,

price\_change DECIMAL(10,2) NOT NULL,

price\_change\_percent DECIMAL(5,2) NOT NULL,

-- Signal scores by timeframe

score\_1h INTEGER CHECK (score\_1h >= 0 AND score\_1h <= 100),

score\_4h INTEGER CHECK (score\_4h >= 0 AND score\_4h <= 100),

score\_1d INTEGER CHECK (score\_1d >= 0 AND score\_1d <= 100),

score\_1w INTEGER CHECK (score\_1w >= 0 AND score\_1w <= 100),

-- Calculated final score

final\_score INTEGER CHECK (final\_score >= 0 AND final\_score <= 100),

signal\_strength VARCHAR(20) CHECK (signal\_strength IN ('strong', 'valid', 'weak')),

-- Classification

sector VARCHAR(50),

market VARCHAR(50) DEFAULT 'US',

-- Technical indicators

rsi\_value DECIMAL(5,2),

macd\_value DECIMAL(8,4),

volume\_ratio DECIMAL(4,2),

support\_level DECIMAL(10,2),

resistance\_level DECIMAL(10,2),

-- Timestamps

signal\_generated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

expires\_at TIMESTAMP WITH TIME ZONE,

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

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

-- Paper trades

CREATE TABLE paper\_trades (

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

user\_id UUID NOT NULL REFERENCES users(id),

signal\_id UUID REFERENCES trading\_signals(id),

-- Trade details

symbol VARCHAR(10) NOT NULL,

company\_name VARCHAR(255) NOT NULL,

entry\_price DECIMAL(10,2) NOT NULL,

shares INTEGER NOT NULL,

investment\_amount DECIMAL(12,2) NOT NULL,

-- Risk management

stop\_loss DECIMAL(10,2) NOT NULL,

take\_profit DECIMAL(10,2) NOT NULL,

risk\_percentage DECIMAL(5,2) NOT NULL,

risk\_amount DECIMAL(10,2) NOT NULL,

-- Trade status

status VARCHAR(20) DEFAULT 'open' CHECK (status IN ('open', 'closed', 'stopped')),

exit\_price DECIMAL(10,2),

exit\_reason VARCHAR(50),

-- Performance

current\_price DECIMAL(10,2),

unrealized\_pnl DECIMAL(10,2),

realized\_pnl DECIMAL(10,2),

-- Signal context

signal\_score INTEGER,

signal\_timeframe VARCHAR(5),

-- Timestamps

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

closed\_at TIMESTAMP WITH TIME ZONE,

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

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

-- User preferences and settings

CREATE TABLE dashboard\_preferences (

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

user\_id UUID NOT NULL REFERENCES users(id),

-- Display preferences

default\_timeframe VARCHAR(5) DEFAULT '1D',

min\_signal\_score INTEGER DEFAULT 70,

preferred\_sectors TEXT[], -- Array of sector names

auto\_refresh\_enabled BOOLEAN DEFAULT true,

refresh\_interval\_seconds INTEGER DEFAULT 120,

-- Chart preferences

chart\_theme VARCHAR(20) DEFAULT 'dark',

show\_portfolio\_comparison BOOLEAN DEFAULT true,

default\_chart\_period VARCHAR(5) DEFAULT '3M',

-- Trading preferences

default\_portfolio\_balance DECIMAL(12,2) DEFAULT 10000,

default\_risk\_percentage DECIMAL(3,1) DEFAULT 2.0,

max\_risk\_percentage DECIMAL(3,1) DEFAULT 5.0,

-- Notification preferences

enable\_signal\_alerts BOOLEAN DEFAULT true,

enable\_trade\_alerts BOOLEAN DEFAULT true,

enable\_performance\_alerts BOOLEAN DEFAULT false,

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

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

UNIQUE(user\_id)

);

-- Indexes for performance

CREATE INDEX idx\_dashboard\_metrics\_user\_id ON dashboard\_metrics(user\_id);

CREATE INDEX idx\_dashboard\_metrics\_calculated\_at ON dashboard\_metrics(calculated\_at DESC);

CREATE INDEX idx\_portfolio\_performance\_user\_id ON portfolio\_performance(user\_id);

CREATE INDEX idx\_portfolio\_performance\_date ON portfolio\_performance(date DESC);

CREATE INDEX idx\_trading\_signals\_ticker ON trading\_signals(ticker);

CREATE INDEX idx\_trading\_signals\_final\_score ON trading\_signals(final\_score DESC);

CREATE INDEX idx\_trading\_signals\_generated\_at ON trading\_signals(signal\_generated\_at DESC);

CREATE INDEX idx\_trading\_signals\_sector ON trading\_signals(sector);

CREATE INDEX idx\_trading\_signals\_strength ON trading\_signals(signal\_strength);

CREATE INDEX idx\_paper\_trades\_user\_id ON paper\_trades(user\_id);

CREATE INDEX idx\_paper\_trades\_symbol ON paper\_trades(symbol);

CREATE INDEX idx\_paper\_trades\_status ON paper\_trades(status);

CREATE INDEX idx\_paper\_trades\_opened\_at ON paper\_trades(opened\_at DESC);

CREATE INDEX idx\_dashboard\_preferences\_user\_id ON dashboard\_preferences(user\_id);

-- Composite indexes for common queries

CREATE INDEX idx\_signals\_score\_time ON trading\_signals(final\_score DESC, signal\_generated\_at DESC);

CREATE INDEX idx\_trades\_user\_status ON paper\_trades(user\_id, status, opened\_at DESC);

**6. User Experience**

**Loading States Implementation**

// Dashboard loading skeleton

const DashboardSkeleton = () => (

<div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-8">

<div className="animate-pulse">

{/\* Header skeleton \*/}

<div className="mb-8">

<div className="h-8 bg-slate-700 rounded w-1/3 mb-2" />

<div className="h-4 bg-slate-700 rounded w-1/2 mb-2" />

<div className="h-4 bg-slate-700 rounded w-1/4" />

</div>

{/\* Metric cards skeleton \*/}

<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6 mb-8">

{[...Array(4)].map((\_, i) => (

<div key={i} className="bg-slate-800/50 rounded-lg p-6">

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

<div className="h-4 bg-slate-700 rounded w-20" />

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

</div>

<div className="h-8 bg-slate-700 rounded w-16 mb-2" />

<div className="h-3 bg-slate-700 rounded w-24" />

</div>

))}

</div>

{/\* Performance cards skeleton \*/}

<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-5 gap-6 mb-8">

<div className="lg:col-span-4 grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6">

{[...Array(4)].map((\_, i) => (

<div key={i} className="bg-slate-800/50 rounded-lg p-4">

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

<div className="w-10 h-10 bg-slate-700 rounded-lg" />

<div className="flex-1">

<div className="h-3 bg-slate-700 rounded w-20 mb-2" />

<div className="h-5 bg-slate-700 rounded w-16 mb-1" />

<div className="h-3 bg-slate-700 rounded w-24" />

</div>

</div>

</div>

))}

</div>

<div className="lg:col-span-1">

<div className="bg-slate-800/50 rounded-lg p-4 h-full">

<div className="h-4 bg-slate-700 rounded w-24 mb-4" />

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

{[...Array(3)].map((\_, i) => (

<div key={i} className="flex justify-between items-center">

<div>

<div className="h-3 bg-slate-700 rounded w-12 mb-1" />

<div className="h-3 bg-slate-700 rounded w-16" />

</div>

<div className="h-3 bg-slate-700 rounded w-12" />

</div>

))}

</div>

</div>

</div>

</div>

{/\* Charts skeleton \*/}

<div className="grid grid-cols-1 lg:grid-cols-5 gap-8 mb-8">

<div className="lg:col-span-2 h-64 bg-slate-800/50 rounded-lg" />

<div className="lg:col-span-3 h-64 bg-slate-800/50 rounded-lg" />

</div>

{/\* Heatmap skeleton \*/}

<div className="bg-slate-800/50 rounded-lg p-6">

<div className="h-6 bg-slate-700 rounded w-1/4 mb-6" />

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

{[...Array(5)].map((\_, i) => (

<div key={i} className="flex items-center space-x-4">

<div className="h-4 bg-slate-700 rounded w-16" />

<div className="h-4 bg-slate-700 rounded w-24" />

<div className="h-8 bg-slate-700 rounded w-12" />

<div className="h-8 bg-slate-700 rounded w-12" />

<div className="h-8 bg-slate-700 rounded w-12" />

<div className="h-8 bg-slate-700 rounded w-12" />

<div className="h-8 bg-slate-700 rounded w-16" />

</div>

))}

</div>

</div>

</div>

</div>

)

// Progressive loading with Intersection Observer

const useProgressiveLoading = () => {

const [visibleSections, setVisibleSections] = useState<Set<string>>(new Set(['header']))

useEffect(() => {

const observer = new IntersectionObserver(

(entries) => {

entries.forEach((entry) => {

if (entry.isIntersecting) {

setVisibleSections(prev => new Set([...prev, entry.target.id]))

}

})

},

{ threshold: 0.1 }

)

const sections = ['metrics', 'performance', 'charts', 'heatmap']

sections.forEach(section => {

const element = document.getElementById(section)

if (element) observer.observe(element)

})

return () => observer.disconnect()

}, [])

return visibleSections

}

**Error Boundaries & Fallback UI**

// Dashboard-specific error boundary

class DashboardErrorBoundary extends Component<ErrorBoundaryProps, ErrorBoundaryState> {

constructor(props: ErrorBoundaryProps) {

super(props)

this.state = { hasError: false, error: null, errorInfo: null }

}

static getDerivedStateFromError(error: Error): Partial<ErrorBoundaryState> {

return { hasError: true, error }

}

componentDidCatch(error: Error, errorInfo: ErrorInfo) {

console.error('Dashboard error:', error, errorInfo)

// Report to error tracking service

if (process.env.NODE\_ENV === 'production') {

reportError(error, {

component: 'Dashboard',

errorInfo,

userId: this.props.userId

})

}

}

render() {

if (this.state.hasError) {

return (

<div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-8">

<div className="text-center">

<div className="mb-8">

<AlertTriangle className="h-16 w-16 text-red-400 mx-auto mb-4" />

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

Dashboard temporarily unavailable

</h2>

<p className="text-slate-400 mb-6">

We're experiencing technical difficulties. Please try refreshing the page.

</p>

</div>

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

<Button

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

className="bg-blue-600 hover:bg-blue-700"

>

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

Refresh Dashboard

</Button>

<Button

onClick={() => this.setState({ hasError: false })}

variant="outline"

className="border-slate-600 text-slate-300"

>

Try Again

</Button>

</div>

{process.env.NODE\_ENV === 'development' && this.state.error && (

<details className="mt-8 text-left">

<summary className="text-slate-500 cursor-pointer mb-2">

Error Details (Development)

</summary>

<pre className="text-xs text-red-400 bg-slate-900 p-4 rounded overflow-auto">

{this.state.error.stack}

</pre>

</details>

)}

</div>

</div>

)

}

return this.props.children

}

}

// Component-level error boundaries

const SafeMetricCard = ({ children, fallback }: { children: React.ReactNode, fallback?: React.ReactNode }) => (

<ErrorBoundary fallback={fallback || <MetricCardSkeleton />}>

{children}

</ErrorBoundary>

)

const SafeChart = ({ children }: { children: React.ReactNode }) => (

<ErrorBoundary fallback={

<div className="flex items-center justify-center h-64 bg-slate-800/50 rounded-lg">

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

<BarChart3 className="h-8 w-8 mx-auto mb-2" />

<p>Chart temporarily unavailable</p>

</div>

</div>

}>

{children}

</ErrorBoundary>

)

**Accessibility Implementation**

// ARIA labels and semantic HTML

const AccessibleDashboard = () => (

<main role="main" aria-labelledby="dashboard-title">

<h1 id="dashboard-title" className="sr-only">Trading Dashboard</h1>

{/\* Skip link for keyboard users \*/}

<a

href="#main-content"

className="sr-only focus:not-sr-only focus:absolute focus:top-0 focus:left-0 bg-blue-600 text-white p-2 z-50"

>

Skip to main content

</a>

{/\* Metrics section \*/}

<section aria-labelledby="metrics-heading" id="main-content">

<h2 id="metrics-heading" className="sr-only">Key Performance Metrics</h2>

<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6">

{metrics.map((metric, index) => (

<MetricCard

key={metric.id}

{...metric}

aria-describedby={`metric-${index}-description`}

tabIndex={0}

/>

))}

</div>

</section>

{/\* Charts section \*/}

<section aria-labelledby="charts-heading">

<h2 id="charts-heading" className="sr-only">Performance Charts</h2>

<div className="grid grid-cols-1 lg:grid-cols-5 gap-8">

<div className="lg:col-span-2" role="img" aria-labelledby="winrate-chart-title">

<h3 id="winrate-chart-title" className="sr-only">Win Rate Chart</h3>

<WinRateGauge winRate={61} totalTrades={127} />

</div>

<div className="lg:col-span-3" role="img" aria-labelledby="portfolio-chart-title">

<h3 id="portfolio-chart-title" className="sr-only">Portfolio Performance Chart</h3>

<PortfolioPerformanceChart />

</div>

</div>

</section>

{/\* Signal heatmap \*/}

<section aria-labelledby="signals-heading">

<h2 id="signals-heading" className="sr-only">Trading Signals Heatmap</h2>

<SignalHeatmap onOpenSignalModal={handleOpenSignalModal} />

</section>

</main>

)

// Keyboard navigation for signal table

const useKeyboardNavigation = () => {

const [focusedRow, setFocusedRow] = useState(0)

useEffect(() => {

const handleKeyDown = (event: KeyboardEvent) => {

switch (event.key) {

case 'ArrowDown':

event.preventDefault()

setFocusedRow(prev => Math.min(prev + 1, signalCount - 1))

break

case 'ArrowUp':

event.preventDefault()

setFocusedRow(prev => Math.max(prev - 1, 0))

break

case 'Enter':

case ' ':

event.preventDefault()

// Open signal modal for focused row

break

}

}

document.addEventListener('keydown', handleKeyDown)

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

}, [signalCount])

return focusedRow

}

// Screen reader announcements

const useScreenReaderAnnouncements = () => {

const announce = useCallback((message: string) => {

const announcement = document.createElement('div')

announcement.setAttribute('aria-live', 'polite')

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

announcement.className = 'sr-only'

announcement.textContent = message

document.body.appendChild(announcement)

setTimeout(() => document.body.removeChild(announcement), 1000)

}, [])

return announce

}

**Animation & Transition Requirements**

// CSS transitions and animations

const animationClasses = {

'metric-card-hover': 'hover:bg-slate-900/70 transition-all duration-300',

'chart-entrance': 'animate-in slide-in-from-bottom-4 duration-500',

'modal-entrance': 'animate-in zoom-in-50 duration-200',

'number-counter': 'transition-all duration-1000 ease-out',

'filter-change': 'transition-colors duration-200',

'loading-pulse': 'animate-pulse',

'success-bounce': 'animate-bounce'

}

// Animated number counter

const AnimatedNumber = ({ value, duration = 1000 }: { value: number, duration?: number }) => {

const [displayValue, setDisplayValue] = useState(0)

useEffect(() => {

let startTime: number

let startValue = displayValue

const animate = (currentTime: number) => {

if (!startTime) startTime = currentTime

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

const easeOutQuart = 1 - Math.pow(1 - progress, 4)

setDisplayValue(Math.floor(startValue + (value - startValue) \* easeOutQuart))

if (progress < 1) {

requestAnimationFrame(animate)

}

}

requestAnimationFrame(animate)

}, [value, duration, displayValue])

return <span>{displayValue.toLocaleString()}</span>

}

// Staggered animation for metric cards

const useStaggeredAnimation = (itemCount: number, delay: number = 100) => {

const [visibleItems, setVisibleItems] = useState(0)

useEffect(() => {

const timer = setInterval(() => {

setVisibleItems(prev => {

if (prev >= itemCount) {

clearInterval(timer)

return prev

}

return prev + 1

})

}, delay)

return () => clearInterval(timer)

}, [itemCount, delay])

return visibleItems

}

**7. Integration Points**

**Navigation & Routing Integration**

// Dashboard route configuration

const dashboardRoutes = {

'/dashboard': {

component: Dashboard,

preload: ['metrics', 'signals', 'portfolio']

},

'/dashboard/settings': DashboardSettings,

'/signals': SignalsPage,

'/open-positions': OpenPositionsPage,

'/orders-history': OrdersHistoryPage

}

// Navigation state management

const useDashboardNavigation = () => {

const navigate = useNavigate()

const location = useLocation()

const navigateToSignals = useCallback((filters?: SignalFilters) => {

navigate('/signals', {

state: {

filters,

source: 'dashboard'

}

})

}, [navigate])

const navigateToTrade = useCallback((signal: Signal) => {

navigate('/open-positions', {

state: {

prefilledTrade: signal,

source: 'dashboard'

}

})

}, [navigate])

const navigateToHistory = useCallback(() => {

navigate('/orders-history', {

state: { source: 'dashboard' }

})

}, [navigate])

return {

navigateToSignals,

navigateToTrade,

navigateToHistory,

currentPath: location.pathname

}

}

// Route protection and user state

const ProtectedDashboard = () => {

const { user, loading } = useAuth()

const navigate = useNavigate()

useEffect(() => {

if (!loading && !user) {

navigate('/', { replace: true })

}

}, [user, loading, navigate])

if (loading) return <DashboardSkeleton />

if (!user) return null

return (

<DashboardErrorBoundary userId={user.id}>

<Dashboard />

</DashboardErrorBoundary>

)

}

**Cross-Component State Sharing**

// Signal data sharing between components

const useSignalDataSync = () => {

const { signals, updateSignalsData } = useDashboardStore()

const { setFilters } = useSignalsStore()

// Sync dashboard filters with signals page

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

setFilters(filters)

localStorage.setItem('signalFilters', JSON.stringify(filters))

}, [setFilters])

// Update dashboard when signals change

useEffect(() => {

const handleSignalsUpdate = (event: CustomEvent) => {

updateSignalsData()

}

window.addEventListener('signals-updated', handleSignalsUpdate)

return () => window.removeEventListener('signals-updated', handleSignalsUpdate)

}, [updateSignalsData])

return { signals, syncFiltersToSignalsPage }

}

// Portfolio data sharing

const usePortfolioSync = () => {

const { portfolio } = useDashboardStore()

const { updateBalance } = usePortfolioStore()

// Sync portfolio balance changes

useEffect(() => {

if (portfolio.balance) {

updateBalance(portfolio.balance)

}

}, [portfolio.balance, updateBalance])

return portfolio

}

// Trade execution integration

const useTradeExecution = () => {

const { executePaperTrade } = useDashboardStore()

const { addTrade } = useTradesStore()

const navigate = useNavigate()

const executeTradeFromDashboard = useCallback(async (tradeData: any) => {

try {

// Execute on dashboard

await executePaperTrade(tradeData)

// Add to trades store

addTrade(tradeData)

// Navigate to open positions

navigate('/open-positions', {

state: {

newTrade: tradeData,

source: 'dashboard'

}

})

// Broadcast trade execution

window.dispatchEvent(new CustomEvent('trade-executed', {

detail: tradeData

}))

} catch (error) {

console.error('Failed to execute trade:', error)

throw error

}

}, [executePaperTrade, addTrade, navigate])

return { executeTradeFromDashboard }

}

**Event Handling & User Flows**

// Dashboard-specific event handlers

const useDashboardEvents = () => {

const { openSignalModal, closeSignalModal } = useDashboardStore()

const { executeTradeFromDashboard } = useTradeExecution()

const { announce } = useScreenReaderAnnouncements()

const handleSignalClick = useCallback((signal: Signal) => {

openSignalModal(signal)

announce(`Opened details for ${signal.ticker}`)

}, [openSignalModal, announce])

const handleTradeExecution = useCallback(async (tradeData: any) => {

try {

await executeTradeFromDashboard(tradeData)

announce(`Paper trade executed for ${tradeData.symbol}`)

closeSignalModal()

} catch (error) {

announce(`Trade execution failed: ${error.message}`)

}

}, [executeTradeFromDashboard, announce, closeSignalModal])

const handleMetricClick = useCallback((metricType: string) => {

switch (metricType) {

case 'signals':

navigate('/signals')

break

case 'trades':

navigate('/open-positions')

break

case 'history':

navigate('/orders-history')

break

}

}, [navigate])

return {

handleSignalClick,

handleTradeExecution,

handleMetricClick

}

}

// Welcome flow integration

const useWelcomeFlow = () => {

const [showWelcome, setShowWelcome] = useState(false)

const [selectedPlan, setSelectedPlan] = useState<any>(null)

const navigate = useNavigate()

useEffect(() => {

// Check for welcome flow triggers

const shouldShowWelcome = localStorage.getItem('showWelcome')

const savedPlan = localStorage.getItem('selectedPlan')

if (shouldShowWelcome === 'true') {

setShowWelcome(true)

if (savedPlan) {

try {

setSelectedPlan(JSON.parse(savedPlan))

} catch (error) {

console.error('Error parsing selected plan:', error)

}

}

}

}, [])

const dismissWelcome = useCallback(() => {

setShowWelcome(false)

localStorage.removeItem('showWelcome')

localStorage.removeItem('selectedPlan')

}, [])

const exploreFeatures = useCallback(() => {

navigate('/signals')

dismissWelcome()

}, [navigate, dismissWelcome])

const completeSetup = useCallback(() => {

navigate('/settings')

dismissWelcome()

}, [navigate, dismissWelcome])

return {

showWelcome,

selectedPlan,

dismissWelcome,

exploreFeatures,

completeSetup

}

}

**8. Testing Strategy**

**Unit Tests**

describe('Dashboard Component', () => {

const mockUser = {

id: 'user-123',

name: 'John Doe',

email: 'john@example.com'

}

beforeEach(() => {

jest.clearAllMocks()

// Mock API responses

global.fetch = jest.fn()

})

it('renders dashboard with user data', async () => {

const { getByText, getByTestId } = render(

<AuthProvider value={{ user: mockUser, loading: false }}>

<Dashboard />

</AuthProvider>

)

expect(getByText('Welcome, John Doe!')).toBeInTheDocument()

expect(getByTestId('metrics-grid')).toBeInTheDocument()

expect(getByTestId('charts-section')).toBeInTheDocument()

expect(getByTestId('signal-heatmap')).toBeInTheDocument()

})

it('redirects unauthenticated users', () => {

const mockNavigate = jest.fn()

jest.mocked(useNavigate).mockReturnValue(mockNavigate)

render(

<AuthProvider value={{ user: null, loading: false }}>

<Dashboard />

</AuthProvider>

)

expect(mockNavigate).toHaveBeenCalledWith('/')

})

it('shows welcome banner for new users', () => {

localStorage.setItem('showWelcome', 'true')

localStorage.setItem('selectedPlan', JSON.stringify({ name: 'Professional' }))

const { getByTestId } = render(

<AuthProvider value={{ user: mockUser, loading: false }}>

<Dashboard />

</AuthProvider>

)

expect(getByTestId('welcome-banner')).toBeInTheDocument()

})

it('opens signal modal when signal is clicked', async () => {

const { getByTestId, getAllByText } = render(

<AuthProvider value={{ user: mockUser, loading: false }}>

<Dashboard />

</AuthProvider>

)

// Mock signal data

const mockSignal = {

symbol: 'AAPL',

name: 'Apple Inc.',

price: 185.23,

change: 2.45,

signalScore: 92

}

// Click on a signal in the heatmap

const viewButton = getAllByText('View')[0]

fireEvent.click(viewButton)

await waitFor(() => {

expect(getByTestId('signal-modal')).toBeInTheDocument()

})

})

})

describe('SignalHeatmap Component', () => {

const mockProps = {

onOpenSignalModal: jest.fn()

}

it('renders signal table with filtering', () => {

const { getByTestId, getByText } = render(

<SignalHeatmap {...mockProps} />

)

expect(getByTestId('signal-filters')).toBeInTheDocument()

expect(getByTestId('signal-table')).toBeInTheDocument()

expect(getByText('BUY Signal Heatmap')).toBeInTheDocument()

})

it('filters signals by score threshold', async () => {

const { getByTestId } = render(

<SignalHeatmap {...mockProps} />

)

const scoreSlider = getByTestId('score-slider')

fireEvent.change(scoreSlider, { target: { value: '85' } })

// Verify filtered results

await waitFor(() => {

expect(getByTestId('signal-table')).toBeInTheDocument()

// Additional assertions for filtered data

})

})

it('calls onOpenSignalModal when view button is clicked', () => {

const { getAllByText } = render(

<SignalHeatmap {...mockProps} />

)

const viewButton = getAllByText('View')[0]

fireEvent.click(viewButton)

expect(mockProps.onOpenSignalModal).toHaveBeenCalled()

})

})

describe('SignalModal Component', () => {

const mockSignal = {

symbol: 'AAPL',

name: 'Apple Inc.',

price: 185.23,

change: 2.45,

signalScore: 92

}

const mockProps = {

isOpen: true,

onClose: jest.fn(),

signal: mockSignal,

onExecuteTrade: jest.fn(),

existingPositions: []

}

it('displays signal information correctly', () => {

const { getByText } = render(

<SignalModal {...mockProps} />

)

expect(getByText('AAPL')).toBeInTheDocument()

expect(getByText('Apple Inc.')).toBeInTheDocument()

expect(getByText('Score: 92')).toBeInTheDocument()

})

it('calculates position sizing correctly', () => {

const { getByText } = render(

<SignalModal {...mockProps} />

)

// Verify position sizing calculations

expect(getByText(/Max Risk/)).toBeInTheDocument()

expect(getByText(/Shares to Buy/)).toBeInTheDocument()

expect(getByText(/Investment Amount/)).toBeInTheDocument()

})

it('prevents trading on existing positions', () => {

const propsWithExistingPosition = {

...mockProps,

existingPositions: ['AAPL']

}

const { getByText, getByRole } = render(

<SignalModal {...propsWithExistingPosition} />

)

expect(getByText(/already have an open position/)).toBeInTheDocument()

const executeButton = getByRole('button', { name: /Position Already Open/ })

expect(executeButton).toBeDisabled()

})

it('executes paper trade with correct data', () => {

const { getByRole } = render(

<SignalModal {...mockProps} />

)

const executeButton = getByRole('button', { name: /Execute Paper Trade/ })

fireEvent.click(executeButton)

expect(mockProps.onExecuteTrade).toHaveBeenCalledWith(

expect.objectContaining({

symbol: 'AAPL',

entryPrice: 185.23,

signalScore: 92

})

)

})

})

**Integration Tests**

describe('Dashboard Integration', () => {

beforeEach(() => {

// Mock API responses

fetchMock.mockResponse(JSON.stringify({

metrics: mockMetrics,

portfolio: mockPortfolio,

signals: mockSignals

}))

})

it('completes full signal-to-trade flow', async () => {

const { getByTestId, getAllByText, getByRole } = render(

<App />

)

// Navigate to dashboard

fireEvent.click(getByText('Dashboard'))

// Wait for dashboard to load

await waitFor(() => {

expect(getByTestId('signal-heatmap')).toBeInTheDocument()

})

// Click on a signal

const viewButton = getAllByText('View')[0]

fireEvent.click(viewButton)

// Modal should open

await waitFor(() => {

expect(getByTestId('signal-modal')).toBeInTheDocument()

})

// Execute trade

const executeButton = getByRole('button', { name: /Execute Paper Trade/ })

fireEvent.click(executeButton)

// Should navigate to open positions

await waitFor(() => {

expect(window.location.pathname).toBe('/open-positions')

})

// Trade should appear in open positions

expect(getByText('AAPL')).toBeInTheDocument()

})

it('synchronizes data across dashboard sections', async () => {

const { getByTestId } = render(<Dashboard />)

// Execute a trade

const tradeData = { symbol: 'AAPL', profit: 500 }

// Mock trade execution

await act(async () => {

await executePaperTrade(tradeData)

})

// Verify metrics update

await waitFor(() => {

expect(getByTestId('active-signals-count')).toHaveTextContent('13')

})

// Verify recent trades update

expect(getByTestId('recent-trades')).toContainElement(

getByText('AAPL')

)

})

})

**Mock Data Structures**

// Mock dashboard metrics

const mockMetrics = {

todaysSignals: 12,

activeSignals: 12,

avgSignalScore: 82,

successRate: 61,

newSignalsLastHour: 3,

bestPerformer: {

symbol: 'AAPL',

profit: 487,

percentage: 3.2

},

latestSignal: {

symbol: 'NVDA',

score: 92,

timeAgo: '15 min ago'

},

alerts: {

count: 2,

description: 'positions near target'

}

}

// Mock portfolio data

const mockPortfolio = {

balance: 8000,

performance: 87.3,

vs\_sp500: 18.9,

chartData: [

{ date: '2024-02-01', portfolio: 8000, sp500: 8000 },

{ date: '2024-03-01', portfolio: 8200, sp500: 8100 },

{ date: '2024-04-01', portfolio: 8500, sp500: 8150 },

{ date: '2024-05-01', portfolio: 8800, sp500: 8200 },

{ date: '2024-06-01', portfolio: 15000, sp500: 9500 }

],

winRate: 61,

totalTrades: 127,

winningTrades: 77,

losingTrades: 50

}

// Mock signals data

const mockSignals = [

{

ticker: 'AAPL',

name: 'Apple Inc.',

price: 185.23,

change: 2.45,

signals: { '1H': 92, '4H': 88, '1D': 95, '1W': 78 },

finalScore: 90,

sector: 'Technology',

market: 'US'

},

{

ticker: 'NVDA',

name: 'NVIDIA Corp.',

price: 750.12,

change: 15.67,

signals: { '1H': 85, '4H': 92, '1D': 89, '1W': 94 },

finalScore: 89,

sector: 'Technology',

market: 'US'

}

]

// Mock recent trades

const mockRecentTrades = [

{

id: 'trade-1',

symbol: 'AAPL',

profit: 621,

percentage: 3.2,

date: 'Jun 8',

status: 'closed'

},

{

id: 'trade-2',

symbol: 'NVDA',

profit: -96,

percentage: -1.2,

date: 'Jun 7',

status: 'closed'

},

{

id: 'trade-3',

symbol: 'MSFT',

profit: 401,

percentage: 2.8,

date: 'Jun 6',

status: 'closed'

}

]

**9. Charts & Data Visualizations**

**Chart Libraries & Configurations**

// Recharts for portfolio performance chart

import { LineChart, Line, XAxis, YAxis, CartesianGrid, Tooltip, ResponsiveContainer } from 'recharts'

const PortfolioPerformanceChart = () => {

const [timeframe, setTimeframe] = useState('3M')

const [chartData, setChartData] = useState([])

const formatTooltip = (value: number, name: string) => {

const formatValue = name === 'Portfolio'

? `$${value.toFixed(0)}`

: `${value.toFixed(1)}%`

return [formatValue, name]

}

const formatXAxisLabel = (tickItem: string) => {

const date = new Date(tickItem)

return date.toLocaleDateString('en-US', { month: 'short' })

}

return (

<Card className="bg-slate-800/50 backdrop-blur-sm border-slate-700">

<CardHeader>

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

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

<TrendingUp className="h-5 w-5 text-emerald-400" />

<span className="text-white">Portfolio Performance</span>

</div>

{/\* Timeframe selector \*/}

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

{['1M', '3M', '1Y'].map((period) => (

<button

key={period}

onClick={() => setTimeframe(period)}

className={`px-2 py-1 text-xs rounded ${

timeframe === period

? 'bg-blue-600 text-white'

: 'bg-slate-700 text-slate-300 hover:bg-slate-600'

}`}

>

{period}

</button>

))}

</div>

</CardTitle>

{/\* Performance summary \*/}

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

<div>

<span className="text-2xl font-bold text-emerald-400">+87.3%</span>

<span className="text-slate-400 text-sm ml-2">vs S&P 500: +18.9%</span>

</div>

</div>

</CardHeader>

<CardContent>

<ResponsiveContainer width="100%" height={250}>

<LineChart data={chartData} margin={{ top: 5, right: 30, left: 20, bottom: 5 }}>

<CartesianGrid strokeDasharray="3 3" stroke="#374151" />

<XAxis

dataKey="date"

stroke="#9CA3AF"

tickFormatter={formatXAxisLabel}

/>

<YAxis stroke="#9CA3AF" />

<Tooltip

contentStyle={{

backgroundColor: '#1F2937',

border: '1px solid #374151',

borderRadius: '8px'

}}

labelStyle={{ color: '#E5E7EB' }}

formatter={formatTooltip}

/>

<Line

type="monotone"

dataKey="portfolio"

stroke="#10B981"

strokeWidth={3}

dot={false}

name="Portfolio"

/>

<Line

type="monotone"

dataKey="sp500"

stroke="#3B82F6"

strokeWidth={2}

strokeDasharray="5 5"

dot={false}

name="S&P 500"

/>

</LineChart>

</ResponsiveContainer>

{/\* Chart legend \*/}

<div className="flex justify-center space-x-6 mt-4">

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

<div className="w-3 h-3 bg-emerald-500 rounded-full"></div>

<span className="text-sm text-slate-300">Portfolio</span>

</div>

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

<div className="w-3 h-1 bg-blue-500 rounded-full"></div>

<span className="text-sm text-slate-300">S&P 500</span>

</div>

</div>

</CardContent>

</Card>

)

}

**Custom Circular Progress Chart for Win Rate**

const WinRateGauge = ({ winRate, totalTrades, winningTrades }: WinRateGaugeProps) => {

const [animatedWinRate, setAnimatedWinRate] = useState(0)

const radius = 80

const strokeWidth = 12

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

const strokeDasharray = circumference

const strokeDashoffset = circumference - (animatedWinRate / 100) \* circumference

// Animate the gauge on mount

useEffect(() => {

const timer = setTimeout(() => {

setAnimatedWinRate(winRate)

}, 500)

return () => clearTimeout(timer)

}, [winRate])

const getWinRateColor = (rate: number) => {

if (rate >= 60) return '#10B981' // emerald-500

if (rate >= 50) return '#F59E0B' // amber-500

return '#EF4444' // red-500

}

return (

<Card className="bg-slate-800/50 backdrop-blur-sm border-slate-700">

<CardHeader>

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

<TrendingUp className="h-5 w-5 text-amber-400" />

<span className="text-white">Win Rate</span>

<Target className="h-4 w-4 text-amber-400" />

</CardTitle>

</CardHeader>

<CardContent>

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

<div className="relative">

{/\* Background circle \*/}

<svg width="200" height="200" className="transform -rotate-90">

<circle

cx="100"

cy="100"

r={radius}

stroke="#374151"

strokeWidth={strokeWidth}

fill="none"

/>

{/\* Progress circle \*/}

<circle

cx="100"

cy="100"

r={radius}

stroke={getWinRateColor(winRate)}

strokeWidth={strokeWidth}

fill="none"

strokeDasharray={strokeDasharray}

strokeDashoffset={strokeDashoffset}

strokeLinecap="round"

className="transition-all duration-1000 ease-out"

/>

</svg>

{/\* Center content \*/}

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

<div className="text-3xl font-bold text-amber-400">

{animatedWinRate}%

</div>

<div className="text-xs text-slate-400">≥60% Target</div>

</div>

</div>

</div>

{/\* Stats below gauge \*/}

<div className="grid grid-cols-3 gap-4 mt-6 text-center text-sm">

<div>

<div className="text-slate-400">Winning Trades</div>

<div className="text-white font-semibold">{winningTrades}</div>

</div>

<div>

<div className="text-slate-400">Total Trades</div>

<div className="text-white font-semibold">{totalTrades}</div>

</div>

<div>

<div className="text-slate-400">Losing Trades</div>

<div className="text-white font-semibold">{totalTrades - winningTrades}</div>

</div>

</div>

{/\* Target performance message \*/}

<div className="mt-4 p-3 bg-amber-500/10 border border-amber-500/30 rounded-lg">

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

<Target className="h-4 w-4 text-amber-400" />

<p className="text-amber-200 text-sm font-semibold">

You've reached the target performance level. Keep monitoring signals!

</p>

</div>

</div>

</CardContent>

</Card>

)

}

**Real-time Data Updates & Animation**

// Real-time chart updates with WebSocket

const useRealTimeChartData = () => {

const [chartData, setChartData] = useState([])

useEffect(() => {

const ws = new WebSocket(process.env.REACT\_APP\_WS\_URL)

ws.onmessage = (event) => {

const data = JSON.parse(event.data)

if (data.type === 'portfolio\_update') {

setChartData(prevData => {

const newData = [...prevData]

const lastIndex = newData.length - 1

if (lastIndex >= 0) {

newData[lastIndex] = {

...newData[lastIndex],

portfolio: data.portfolioValue,

timestamp: new Date().toISOString()

}

}

return newData

})

}

}

return () => ws.close()

}, [])

return chartData

}

// Smooth number animations

const useAnimatedNumber = (targetValue: number, duration: number = 1000) => {

const [currentValue, setCurrentValue] = useState(0)

useEffect(() => {

let startTime: number

let startValue = currentValue

const animate = (currentTime: number) => {

if (!startTime) startTime = currentTime

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

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

const newValue = startValue + (targetValue - startValue) \* easeOutCubic

setCurrentValue(Math.round(newValue))

if (progress < 1) {

requestAnimationFrame(animate)

}

}

requestAnimationFrame(animate)

}, [targetValue, duration, currentValue])

return currentValue

}

// Chart interaction patterns

const useChartInteractions = () => {

const [hoveredPoint, setHoveredPoint] = useState<any>(null)

const [selectedTimeframe, setSelectedTimeframe] = useState('3M')

const handleMouseEnter = useCallback((data: any, index: number) => {

setHoveredPoint({ data, index })

}, [])

const handleMouseLeave = useCallback(() => {

setHoveredPoint(null)

}, [])

const handleTimeframeChange = useCallback((timeframe: string) => {

setSelectedTimeframe(timeframe)

// Trigger data refetch

}, [])

return {

hoveredPoint,

selectedTimeframe,

handleMouseEnter,

handleMouseLeave,

handleTimeframeChange

}

}

**Signal Heatmap Visualization**

// Signal heatmap with color coding

const SignalTable = ({ filteredSignals, timeFilter, onViewSignal }: SignalTableProps) => {

const getScoreColor = (score: number) => {

if (score >= 90) return 'bg-emerald-600 text-white' // Strong

if (score >= 80) return 'bg-blue-600 text-white' // Valid

if (score >= 70) return 'bg-amber-600 text-white' // Weak

return 'bg-slate-600 text-slate-300' // Below threshold

}

const getScoreIcon = (score: number) => {

if (score >= 90) return '✓' // Strong

if (score >= 80) return '✓' // Valid

if (score >= 70) return '⚠' // Weak

return '✗' // Below threshold

}

return (

<div className="overflow-x-auto">

<table className="w-full">

<thead>

<tr className="border-b border-slate-700">

<th className="text-left py-3 px-4 text-slate-300 font-medium">Asset</th>

<th className="text-left py-3 px-4 text-slate-300 font-medium">Current Price</th>

<th className="text-center py-3 px-4 text-slate-300 font-medium">1H</th>

<th className="text-center py-3 px-4 text-slate-300 font-medium">4H</th>

<th className="text-center py-3 px-4 text-slate-300 font-medium">1D</th>

<th className="text-center py-3 px-4 text-slate-300 font-medium">1W</th>

<th className="text-center py-3 px-4 text-slate-300 font-medium">Final Score</th>

<th className="text-center py-3 px-4 text-slate-300 font-medium">Actions</th>

</tr>

</thead>

<tbody>

{filteredSignals.map((signal, index) => (

<tr

key={signal.ticker}

className="border-b border-slate-800 hover:bg-slate-800/30 transition-colors"

>

<td className="py-3 px-4">

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

<img

src={`https://logo.clearbit.com/${signal.ticker.toLowerCase()}.com`}

alt={signal.ticker}

className="w-6 h-6 rounded"

onError={(e) => {

e.currentTarget.style.display = 'none'

}}

/>

<div>

<div className="text-white font-semibold">{signal.ticker}</div>

<div className="text-slate-400 text-sm">{signal.name}</div>

</div>

</div>

</td>

<td className="py-3 px-4">

<div className="text-white font-semibold">${signal.price.toFixed(2)}</div>

<div className={`text-sm ${signal.change >= 0 ? 'text-emerald-400' : 'text-red-400'}`}>

{signal.change >= 0 ? '+' : ''}{signal.change.toFixed(2)}%

</div>

</td>

{/\* Timeframe scores \*/}

{['1H', '4H', '1D', '1W'].map((timeframe) => (

<td key={timeframe} className="py-3 px-4 text-center">

<span className={`inline-flex items-center justify-center w-12 h-8 rounded text-sm font-semibold ${

getScoreColor(signal.signals[timeframe])

}`}>

{getScoreIcon(signal.signals[timeframe])} {signal.signals[timeframe]}

</span>

</td>

))}

{/\* Final score \*/}

<td className="py-3 px-4 text-center">

<span className={`inline-flex items-center justify-center w-16 h-8 rounded text-sm font-semibold ${

getScoreColor(calculateFinalScore(signal.signals))

}`}>

{getScoreIcon(calculateFinalScore(signal.signals))} {calculateFinalScore(signal.signals)}

</span>

</td>

{/\* Actions \*/}

<td className="py-3 px-4 text-center">

<Button

onClick={() => onViewSignal(signal, timeFilter)}

size="sm"

className="bg-emerald-600 hover:bg-emerald-700 text-white"

>

View

</Button>

</td>

</tr>

))}

</tbody>

</table>

</div>

)

}

// Summary statistics visualization

const SignalSummaryStats = ({ filteredSignals, highlightedCategory, setHighlightedCategory }: SignalSummaryStatsProps) => {

const summary = useMemo(() => {

const strong = filteredSignals.filter(s => calculateFinalScore(s.signals) >= 90).length

const valid = filteredSignals.filter(s => calculateFinalScore(s.signals) >= 80 && calculateFinalScore(s.signals) < 90).length

const weak = filteredSignals.filter(s => calculateFinalScore(s.signals) >= 70 && calculateFinalScore(s.signals) < 80).length

const total = filteredSignals.length

return { strong, valid, weak, total }

}, [filteredSignals])

return (

<div className="grid grid-cols-4 gap-6 text-center mt-6">

<div

className={`p-4 rounded-lg cursor-pointer transition-all ${

highlightedCategory === 'strong'

? 'bg-emerald-600/20 border-2 border-emerald-500'

: 'bg-slate-700/50 hover:bg-slate-700/70'

}`}

onClick={() => setHighlightedCategory(highlightedCategory === 'strong' ? null : 'strong')}

>

<div className="text-2xl font-bold text-emerald-400">{summary.strong}</div>

<div className="text-slate-300 text-sm">Strong (90+)</div>

</div>

<div

className={`p-4 rounded-lg cursor-pointer transition-all ${

highlightedCategory === 'valid'

? 'bg-blue-600/20 border-2 border-blue-500'

: 'bg-slate-700/50 hover:bg-slate-700/70'

}`}

onClick={() => setHighlightedCategory(highlightedCategory === 'valid' ? null : 'valid')}

>

<div className="text-2xl font-bold text-blue-400">{summary.valid}</div>

<div className="text-slate-300 text-sm">Valid (80-89)</div>

</div>

<div

className={`p-4 rounded-lg cursor-pointer transition-all ${

highlightedCategory === 'weak'

? 'bg-amber-600/20 border-2 border-amber-500'

: 'bg-slate-700/50 hover:bg-slate-700/70'

}`}

onClick={() => setHighlightedCategory(highlightedCategory === 'weak' ? null : 'weak')}

>

<div className="text-2xl font-bold text-amber-400">{summary.weak}</div>

<div className="text-slate-300 text-sm">Weak (70-79)</div>

</div>

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

<div className="text-2xl font-bold text-white">{summary.total}</div>

<div className="text-slate-300 text-sm">Total Signals</div>

</div>

</div>

)

}

**10. Visual Data Elements**

**Progress Indicators & Dynamic Counters**

// Animated metric cards with counters

const MetricCard = ({ title, value, subtitle, icon: Icon, iconColor, trend }: MetricCardProps) => {

const animatedValue = useAnimatedNumber(typeof value === 'number' ? value : parseInt(value), 1500)

return (

<Card className="bg-slate-900/50 backdrop-blur-sm border-blue-800/30 hover:bg-slate-900/70 transition-all duration-300 group">

<CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">

<CardTitle className="text-sm font-medium text-slate-300 group-hover:text-white transition-colors">

{title}

</CardTitle>

<Icon className={`h-4 w-4 ${iconColor} group-hover:scale-110 transition-transform`} />

</CardHeader>

<CardContent>

<div className="text-2xl font-bold text-white">

{typeof value === 'number' ? animatedValue.toLocaleString() : animatedValue}

{typeof value === 'string' && value.includes('%') && '%'}

</div>

{subtitle && (

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

{subtitle}

</p>

)}

{trend && (

<div className={`flex items-center mt-1 text-xs ${

trend > 0 ? 'text-emerald-400' : trend < 0 ? 'text-red-400' : 'text-slate-400'

}`}>

{trend > 0 ? <TrendingUp className="h-3 w-3 mr-1" /> :

trend < 0 ? <TrendingDown className="h-3 w-3 mr-1" /> :

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

{Math.abs(trend)}% from yesterday

</div>

)}

</CardContent>

</Card>

)

}

// Circular progress indicator for loading states

const CircularProgress = ({ progress, size = 40, strokeWidth = 3 }: CircularProgressProps) => {

const radius = (size - strokeWidth) / 2

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

const offset = circumference - (progress / 100) \* circumference

return (

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

<svg width={size} height={size} className="transform -rotate-90">

<circle

cx={size / 2}

cy={size / 2}

r={radius}

stroke="#374151"

strokeWidth={strokeWidth}

fill="none"

/>

<circle

cx={size / 2}

cy={size / 2}

r={radius}

stroke="#3B82F6"

strokeWidth={strokeWidth}

fill="none"

strokeDasharray={circumference}

strokeDashoffset={offset}

strokeLinecap="round"

className="transition-all duration-300"

/>

</svg>

<span className="absolute text-xs font-semibold text-white">

{Math.round(progress)}%

</span>

</div>

)

}

// Real-time status indicators

const LiveStatusIndicator = ({ status, label }: { status: 'online' | 'updating' | 'offline', label: string }) => {

const statusConfig = {

online: { color: 'bg-emerald-500', pulse: true, text: 'text-emerald-400' },

updating: { color: 'bg-amber-500', pulse: true, text: 'text-amber-400' },

offline: { color: 'bg-red-500', pulse: false, text: 'text-red-400' }

}

const config = statusConfig[status]

return (

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

<div className="relative">

<div className={`w-2 h-2 rounded-full ${config.color}`} />

{config.pulse && (

<div className={`absolute inset-0 w-2 h-2 rounded-full ${config.color} animate-ping opacity-75`} />

)}

</div>

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

{label}

</span>

</div>

)

}

**Color-coded Status Indicators**

// Signal strength color system

const SignalColors = {

strong: {

background: 'bg-emerald-600',

text: 'text-emerald-400',

border: 'border-emerald-500',

glow: 'shadow-emerald-500/20'

},

valid: {

background: 'bg-blue-600',

text: 'text-blue-400',

border: 'border-blue-500',

glow: 'shadow-blue-500/20'

},

weak: {

background: 'bg-amber-600',

text: 'text-amber-400',

border: 'border-amber-500',

glow: 'shadow-amber-500/20'

},

inactive: {

background: 'bg-slate-600',

text: 'text-slate-400',

border: 'border-slate-500',

glow: 'shadow-slate-500/20'

}

}

// Performance color indicators

const PerformanceColors = {

positive: 'text-emerald-400',

negative: 'text-red-400',

neutral: 'text-slate-400'

}

// Status badge component

const StatusBadge = ({

type,

children,

size = 'md'

}: {

type: keyof typeof SignalColors,

children: React.ReactNode,

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

}) => {

const colors = SignalColors[type]

const sizeClasses = {

sm: 'px-2 py-1 text-xs',

md: 'px-3 py-1 text-sm',

lg: 'px-4 py-2 text-base'

}

return (

<span className={`

inline-flex items-center justify-center rounded-lg font-semibold

${colors.background} ${colors.text} ${sizeClasses[size]}

${colors.glow} shadow-lg

transition-all duration-200 hover:scale-105

`}>

{children}

</span>

)

}

// Trading direction indicators

const DirectionIndicator = ({ direction, value }: { direction: 'up' | 'down' | 'neutral', value: string }) => {

const directionConfig = {

up: { icon: TrendingUp, color: 'text-emerald-400', bgColor: 'bg-emerald-500/10' },

down: { icon: TrendingDown, color: 'text-red-400', bgColor: 'bg-red-500/10' },

neutral: { icon: Minus, color: 'text-slate-400', bgColor: 'bg-slate-500/10' }

}

const config = directionConfig[direction]

const Icon = config.icon

return (

<div className={`flex items-center space-x-1 px-2 py-1 rounded ${config.bgColor}`}>

<Icon className={`h-3 w-3 ${config.color}`} />

<span className={`text-sm font-medium ${config.color}`}>

{value}

</span>

</div>

)

}

**Visual Feedback for State Changes**

// Loading state transitions

const LoadingStateTransition = ({ isLoading, children }: { isLoading: boolean, children: React.ReactNode }) => (

<div className={`transition-all duration-300 ${isLoading ? 'opacity-50 pointer-events-none' : 'opacity-100'}`}>

{isLoading && (

<div className="absolute inset-0 flex items-center justify-center z-10">

<CircularProgress progress={100} />

</div>

)}

{children}

</div>

)

// Success/error state animations

const StateChangeAnimation = ({ state, message }: { state: 'success' | 'error' | 'loading', message: string }) => {

const stateConfig = {

success: { icon: CheckCircle, color: 'text-emerald-400', bgColor: 'bg-emerald-500/10', animation: 'animate-bounce' },

error: { icon: XCircle, color: 'text-red-400', bgColor: 'bg-red-500/10', animation: 'animate-pulse' },

loading: { icon: Loader2, color: 'text-blue-400', bgColor: 'bg-blue-500/10', animation: 'animate-spin' }

}

const config = stateConfig[state]

const Icon = config.icon

return (

<div className={`flex items-center space-x-2 p-3 rounded-lg ${config.bgColor} animate-in slide-in-from-top duration-300`}>

<Icon className={`h-4 w-4 ${config.color} ${config.animation}`} />

<span className={`text-sm font-medium ${config.color}`}>

{message}

</span>

</div>

)

}

// Hover effects for interactive elements

const InteractiveCard = ({ children, onClick }: { children: React.ReactNode, onClick?: () => void }) => (

<div

className={`

bg-slate-900/50 backdrop-blur-sm border-blue-800/30 rounded-lg p-4

transition-all duration-300 cursor-pointer

hover:bg-slate-900/70 hover:border-blue-700/50 hover:scale-[1.02]

hover:shadow-lg hover:shadow-blue-500/10

active:scale-[0.98] active:transition-none

`}

onClick={onClick}

>

{children}

</div>

)

// Micro-interactions for buttons

const AnimatedButton = ({ children, variant = 'primary', ...props }: ButtonProps) => {

const variantClasses = {

primary: 'bg-blue-600 hover:bg-blue-700 text-white hover:scale-105 active:scale-95',

secondary: 'bg-slate-600 hover:bg-slate-700 text-white hover:scale-105 active:scale-95',

success: 'bg-emerald-600 hover:bg-emerald-700 text-white hover:scale-105 active:scale-95',

danger: 'bg-red-600 hover:bg-red-700 text-white hover:scale-105 active:scale-95'

}

return (

<button

className={`

px-4 py-2 rounded-lg font-semibold transition-all duration-200

${variantClasses[variant]}

focus:outline-none focus:ring-2 focus:ring-blue-500 focus:ring-offset-2 focus:ring-offset-slate-900

`}

{...props}

>

{children}

</button>

)

}

**11. Security & Validation**

**Input Validation Schemas (Zod)**

import { z } from 'zod'

// Dashboard filters validation

const signalFiltersSchema = z.object({

timeframe: z.enum(['1H', '4H', '1D', '1W']).default('1D'),

minScore: z.number().min(0).max(100).default(70),

sector: z.string().default('all'),

market: z.string().default('global'),

maxResults: z.number().min(1).max(100).default(50)

})

// Paper trading validation

const paperTradeSchema = z.object({

symbol: z.string()

.min(1, 'Symbol is required')

.max(10, 'Symbol too long')

.regex(/^[A-Z]+$/, 'Symbol must be uppercase letters only'),

name: z.string().min(1, 'Company name is required'),

entryPrice: z.number()

.positive('Entry price must be positive')

.max(100000, 'Entry price too high'),

shares: z.number()

.int('Shares must be a whole number')

.positive('Shares must be positive')

.max(10000, 'Too many shares'),

stopLoss: z.number()

.positive('Stop loss must be positive'),

takeProfit: z.number()

.positive('Take profit must be positive'),

investmentAmount: z.number()

.positive('Investment amount must be positive')

.max(1000000, 'Investment amount too high'),

riskPercentage: z.number()

.min(0.1, 'Risk percentage too low')

.max(10, 'Risk percentage too high (max 10%)'),

portfolioBalance: z.number()

.positive('Portfolio balance must be positive')

.min(100, 'Portfolio balance too low')

.max(10000000, 'Portfolio balance too high')

}).refine(data => data.stopLoss < data.entryPrice, {

message: 'Stop loss must be below entry price',

path: ['stopLoss']

}).refine(data => data.takeProfit > data.entryPrice, {

message: 'Take profit must be above entry price',

path: ['takeProfit']

}).refine(data => data.investmentAmount <= data.portfolioBalance, {

message: 'Investment amount cannot exceed portfolio balance',

path: ['investmentAmount']

})

// Dashboard preferences validation

const dashboardPreferencesSchema = z.object({

defaultTimeframe: z.enum(['1H', '4H', '1D', '1W']).default('1D'),

minSignalScore: z.number().min(50).max(100).default(70),

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

refreshIntervalSeconds: z.number().min(30).max(300).default(120),

defaultRiskPercentage: z.number().min(0.5).max(5).default(2),

maxRiskPercentage: z.number().min(1).max(10).default(5),

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

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

})

// Form validation hooks

const useFormValidation = <T>(schema: z.ZodSchema<T>) => {

const [errors, setErrors] = useState<Record<string, string>>({})

const validate = useCallback((data: any): data is T => {

try {

schema.parse(data)

setErrors({})

return true

} catch (error) {

if (error instanceof z.ZodError) {

const fieldErrors = error.errors.reduce((acc, err) => {

const path = err.path.join('.')

acc[path] = err.message

return acc

}, {} as Record<string, string>)

setErrors(fieldErrors)

}

return false

}

}, [schema])

const validateField = useCallback((field: string, value: any) => {

try {

const fieldSchema = schema.shape[field]

if (fieldSchema) {

fieldSchema.parse(value)

setErrors(prev => ({ ...prev, [field]: '' }))

return true

}

} catch (error) {

if (error instanceof z.ZodError) {

setErrors(prev => ({ ...prev, [field]: error.errors[0]?.message || 'Invalid value' }))

}

return false

}

}, [schema])

return { errors, validate, validateField, clearErrors: () => setErrors({}) }

}

**Authentication & Authorization**

// Dashboard route protection

const useDashboardAuth = () => {

const { user, loading } = useAuth()

const navigate = useNavigate()

useEffect(() => {

if (!loading && !user) {

navigate('/', { replace: true })

}

}, [user, loading, navigate])

// Check if user has access to premium features

const hasPermission = useCallback((feature: string) => {

if (!user) return false

const permissions = {

'advanced\_signals': user.subscription?.tier !== 'starter',

'historical\_data': user.subscription?.tier === 'elite',

'custom\_alerts': user.subscription?.tier !== 'starter',

'export\_data': user.subscription?.tier === 'elite'

}

return permissions[feature] || false

}, [user])

return { user, loading, hasPermission }

}

// API request authentication

const useAuthenticatedRequest = () => {

const { getToken } = useAuth()

const authenticatedFetch = useCallback(async (url: string, options: RequestInit = {}) => {

const token = await getToken()

const authHeaders = {

'Authorization': `Bearer ${token}`,

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

...options.headers

}

return fetch(url, {

...options,

headers: authHeaders

})

}, [getToken])

return { authenticatedFetch }

}

// Session validation

const useSessionValidation = () => {

const { user, signOut } = useAuth()

useEffect(() => {

if (!user) return

const validateSession = async () => {

try {

const response = await fetch('/api/auth/validate', {

headers: {

'Authorization': `Bearer ${user.token}`

}

})

if (!response.ok) {

// Session expired, log out user

await signOut()

}

} catch (error) {

console.error('Session validation failed:', error)

await signOut()

}

}

// Validate session every 5 minutes

const interval = setInterval(validateSession, 5 \* 60 \* 1000)

return () => clearInterval(interval)

}, [user, signOut])

}

**Data Sanitization & XSS Prevention**

// Input sanitization

import DOMPurify from 'isomorphic-dompurify'

const sanitizeInput = (input: string): string => {

return DOMPurify.sanitize(input, {

ALLOWED\_TAGS: [], // No HTML tags allowed

ALLOWED\_ATTR: []

})

}

// Stock symbol validation

const validateStockSymbol = (symbol: string): boolean => {

const symbolRegex = /^[A-Z]{1,5}$/

return symbolRegex.test(symbol.toUpperCase())

}

// Price validation

const validatePrice = (price: number): boolean => {

return !isNaN(price) && isFinite(price) && price > 0 && price < 100000

}

// Portfolio balance sanitization

const sanitizePortfolioBalance = (balance: string): number => {

const numericValue = parseFloat(balance.replace(/[^0-9.]/g, ''))

return Math.max(0, Math.min(10000000, numericValue))

}

// Safe JSON parsing

const safeParse = <T>(json: string, fallback: T): T => {

try {

return JSON.parse(json)

} catch (error) {

console.warn('Failed to parse JSON:', error)

return fallback

}

}

// Content Security Policy headers

const CSP\_HEADERS = {

'Content-Security-Policy': [

"default-src 'self'",

"script-src 'self' 'unsafe-inline' 'unsafe-eval' https://s3.tradingview.com",

"style-src 'self' 'unsafe-inline' https://fonts.googleapis.com",

"img-src 'self' data: https: blob:",

"connect-src 'self' wss: https:",

"font-src 'self' https://fonts.gstatic.com",

"frame-src https://s.tradingview.com",

"worker-src 'self' blob:"

].join('; ')

}

**Rate Limiting & API Protection**

// Client-side rate limiting

const useRateLimit = (maxRequests: number, windowMs: number) => {

const [requests, setRequests] = useState<number[]>([])

const canMakeRequest = useCallback(() => {

const now = Date.now()

const windowStart = now - windowMs

const recentRequests = requests.filter(time => time > windowStart)

return recentRequests.length < maxRequests

}, [requests, maxRequests, windowMs])

const recordRequest = useCallback(() => {

const now = Date.now()

setRequests(prev => [...prev.filter(time => time > now - windowMs), now])

}, [windowMs])

return { canMakeRequest, recordRequest }

}

// API request with rate limiting

const useThrottledAPI = () => {

const { canMakeRequest, recordRequest } = useRateLimit(10, 60000) // 10 requests per minute

const throttledFetch = useCallback(async (url: string, options?: RequestInit) => {

if (!canMakeRequest()) {

throw new Error('Rate limit exceeded. Please try again later.')

}

recordRequest()

return fetch(url, options)

}, [canMakeRequest, recordRequest])

return { throttledFetch }

}

// WebSocket rate limiting

const useWebSocketRateLimit = () => {

const [lastMessage, setLastMessage] = useState<number>(0)

const minInterval = 1000 // 1 second between messages

const canSendMessage = useCallback(() => {

const now = Date.now()

return now - lastMessage >= minInterval

}, [lastMessage, minInterval])

const recordMessage = useCallback(() => {

setLastMessage(Date.now())

}, [])

return { canSendMessage, recordMessage }

}

// Server-side rate limiting (for API endpoints)

const createRateLimiter = (windowMs: number, max: number) => {

const clients = new Map()

return (req: Request, res: Response, next: NextFunction) => {

const clientId = req.ip || req.connection.remoteAddress

const now = Date.now()

const windowStart = now - windowMs

if (!clients.has(clientId)) {

clients.set(clientId, [])

}

const clientRequests = clients.get(clientId)

const recentRequests = clientRequests.filter((time: number) => time > windowStart)

if (recentRequests.length >= max) {

return res.status(429).json({

error: 'Too many requests',

retryAfter: Math.ceil(windowMs / 1000)

})

}

recentRequests.push(now)

clients.set(clientId, recentRequests)

next()

}

}

**Financial Data Protection**

// Sensitive data handling

const useSecureDataHandling = () => {

const encryptSensitiveData = useCallback((data: any) => {

// Only encrypt in production

if (process.env.NODE\_ENV !== 'production') {

return data

}

// Use appropriate encryption for sensitive financial data

return encryptData(JSON.stringify(data))

}, [])

const decryptSensitiveData = useCallback((encryptedData: string) => {

if (process.env.NODE\_ENV !== 'production') {

return encryptedData

}

try {

const decrypted = decryptData(encryptedData)

return JSON.parse(decrypted)

} catch (error) {

console.error('Failed to decrypt data:', error)

return null

}

}, [])

return { encryptSensitiveData, decryptSensitiveData }

}

// Portfolio balance protection

const usePortfolioSecurity = () => {

const sanitizeBalance = useCallback((balance: number) => {

// Ensure balance is within reasonable limits

return Math.max(100, Math.min(10000000, balance))

}, [])

const validateTradeAmount = useCallback((amount: number, balance: number) => {

// Ensure trade amount doesn't exceed portfolio balance

return amount > 0 && amount <= balance

}, [])

const calculateRiskLimits = useCallback((balance: number) => {

return {

maxRiskPerTrade: balance \* 0.02, // 2% max risk per trade

maxTotalRisk: balance \* 0.10, // 10% max total risk

maxPositionSize: balance \* 0.25 // 25% max position size

}

}, [])

return { sanitizeBalance, validateTradeAmount, calculateRiskLimits }

}

// Audit logging for financial transactions

const useAuditLogging = () => {

const logTradeAction = useCallback((action: string, tradeData: any, userId: string) => {

const auditEntry = {

timestamp: new Date().toISOString(),

userId,

action,

tradeData: {

symbol: tradeData.symbol,

amount: tradeData.investmentAmount,

riskPercentage: tradeData.riskPercentage

},

ipAddress: '{{USER\_IP}}', // Would be filled by server

userAgent: navigator.userAgent

}

// Send to audit service

fetch('/api/audit/trade-actions', {

method: 'POST',

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

body: JSON.stringify(auditEntry)

}).catch(error => {

console.error('Failed to log audit entry:', error)

})

}, [])

return { logTradeAction }

}

**12. Environment & Configuration**

**Environment Variables**

# API Configuration

VITE\_API\_URL=https://api.kurzora.com

VITE\_WS\_URL=wss://ws.kurzora.com

VITE\_API\_VERSION=v1

# External Data Sources

VITE\_POLYGON\_API\_KEY=your\_polygon\_api\_key

VITE\_ALPHA\_VANTAGE\_API\_KEY=your\_alpha\_vantage\_key

VITE\_FINNHUB\_API\_KEY=your\_finnhub\_key

# TradingView Configuration

VITE\_TRADINGVIEW\_WIDGET\_ID=tradingview\_widget

VITE\_TRADINGVIEW\_THEME=dark

# Authentication

VITE\_AUTH\_DOMAIN=auth.kurzora.com

VITE\_JWT\_ISSUER=kurzora-auth

# Real-time Data

VITE\_ENABLE\_REAL\_TIME=true

VITE\_WS\_RECONNECT\_INTERVAL=5000

VITE\_DATA\_REFRESH\_INTERVAL=120000

# Feature Flags

VITE\_ENABLE\_PAPER\_TRADING=true

VITE\_ENABLE\_ADVANCED\_CHARTS=true

VITE\_ENABLE\_EXPORT\_DATA=true

VITE\_ENABLE\_CUSTOM\_ALERTS=true

VITE\_ENABLE\_PERFORMANCE\_TRACKING=true

# Dashboard Configuration

VITE\_DEFAULT\_TIMEFRAME=1D

VITE\_MIN\_SIGNAL\_SCORE=70

VITE\_MAX\_SIGNALS\_DISPLAY=50

VITE\_AUTO\_REFRESH\_DEFAULT=true

# Security

VITE\_CSP\_NONCE=true

VITE\_ENABLE\_AUDIT\_LOGGING=true

VITE\_MAX\_PORTFOLIO\_BALANCE=10000000

VITE\_MAX\_RISK\_PERCENTAGE=10

# Analytics & Monitoring

VITE\_GA\_TRACKING\_ID=G-XXXXXXXXXX

VITE\_MIXPANEL\_TOKEN=your\_mixpanel\_token

VITE\_SENTRY\_DSN=https://your-sentry-dsn@sentry.io/project

VITE\_HOTJAR\_ID=your\_hotjar\_id

# Performance

VITE\_ENABLE\_SERVICE\_WORKER=true

VITE\_CACHE\_DURATION=300000

VITE\_LAZY\_LOADING=true

# Development

VITE\_DEBUG\_MODE=false

VITE\_MOCK\_DATA=false

VITE\_SHOW\_DEV\_TOOLS=false

# Server-side Environment Variables

DATABASE\_URL=postgresql://user:password@localhost:5432/kurzora

REDIS\_URL=redis://localhost:6379

JWT\_SECRET=your-super-secret-jwt-key

ENCRYPTION\_KEY=your-encryption-key

# Email & Notifications

SENDGRID\_API\_KEY=SG.your-sendgrid-key

TELEGRAM\_BOT\_TOKEN=your-telegram-bot-token

SLACK\_WEBHOOK\_URL=https://hooks.slack.com/services/...

# External Services

STRIPE\_SECRET\_KEY=sk\_live\_your\_stripe\_key

POLYGON\_API\_KEY=your\_polygon\_api\_key

ALPHA\_VANTAGE\_API\_KEY=your\_alpha\_vantage\_key

**Feature Flags Implementation**

// Feature flag configuration

interface FeatureFlags {

enablePaperTrading: boolean

enableAdvancedCharts: boolean

enableExportData: boolean

enableCustomAlerts: boolean

enablePerformanceTracking: boolean

enableRealTimeData: boolean

enableAuditLogging: boolean

showDevTools: boolean

}

// Feature flag hook

const useFeatureFlags = (): FeatureFlags => {

return useMemo(() => ({

enablePaperTrading: import.meta.env.VITE\_ENABLE\_PAPER\_TRADING === 'true',

enableAdvancedCharts: import.meta.env.VITE\_ENABLE\_ADVANCED\_CHARTS === 'true',

enableExportData: import.meta.env.VITE\_ENABLE\_EXPORT\_DATA === 'true',

enableCustomAlerts: import.meta.env.VITE\_ENABLE\_CUSTOM\_ALERTS === 'true',

enablePerformanceTracking: import.meta.env.VITE\_ENABLE\_PERFORMANCE\_TRACKING === 'true',

enableRealTimeData: import.meta.env.VITE\_ENABLE\_REAL\_TIME === 'true',

enableAuditLogging: import.meta.env.VITE\_ENABLE\_AUDIT\_LOGGING === 'true',

showDevTools: import.meta.env.VITE\_SHOW\_DEV\_TOOLS === 'true' && process.env.NODE\_ENV === 'development'

}), [])

}

// Conditional feature rendering

const ConditionalFeature: React.FC<{

flag: keyof FeatureFlags

children: React.ReactNode

fallback?: React.ReactNode

}> = ({ flag, children, fallback = null }) => {

const featureFlags = useFeatureFlags()

return featureFlags[flag] ? <>{children}</> : <>{fallback}</>

}

// Usage in dashboard

const Dashboard = () => {

const featureFlags = useFeatureFlags()

return (

<div>

{/\* Always visible components \*/}

<MetricCards />

<SignalHeatmap />

{/\* Conditional components \*/}

<ConditionalFeature flag="enablePerformanceTracking">

<PortfolioPerformanceChart />

</ConditionalFeature>

<ConditionalFeature flag="enablePaperTrading">

<SignalModal onExecuteTrade={handleExecuteTrade} />

</ConditionalFeature>

<ConditionalFeature flag="enableAdvancedCharts">

<TradingViewWidget />

</ConditionalFeature>

{featureFlags.showDevTools && (

<DevToolsPanel />

)}

</div>

)

}

**Third-party Service Configurations**

// TradingView widget configuration

const tradingViewConfig = {

autosize: true,

symbol: "NASDAQ:AAPL",

interval: "D",

timezone: "Etc/UTC",

theme: import.meta.env.VITE\_TRADINGVIEW\_THEME || "dark",

style: "1",

locale: "en",

toolbar\_bg: "#1e293b",

enable\_publishing: false,

hide\_side\_toolbar: true,

allow\_symbol\_change: false,

container\_id: import.meta.env.VITE\_TRADINGVIEW\_WIDGET\_ID || "tradingview\_widget"

}

// Polygon.io API configuration

const polygonConfig = {

apiKey: import.meta.env.VITE\_POLYGON\_API\_KEY,

baseUrl: 'https://api.polygon.io',

websocketUrl: 'wss://socket.polygon.io',

rateLimit: {

requestsPerMinute: 5, // Free tier limit

burstLimit: 10

}

}

// Analytics configuration

const analyticsConfig = {

googleAnalytics: {

trackingId: import.meta.env.VITE\_GA\_TRACKING\_ID,

config: {

anonymize\_ip: true,

cookie\_expires: 63072000

}

},

mixpanel: {

token: import.meta.env.VITE\_MIXPANEL\_TOKEN,

config: {

debug: process.env.NODE\_ENV === 'development',

track\_pageview: true,

persistence: 'localStorage'

}

},

sentry: {

dsn: import.meta.env.VITE\_SENTRY\_DSN,

environment: process.env.NODE\_ENV,

tracesSampleRate: process.env.NODE\_ENV === 'production' ? 0.1 : 1.0

}

}

**13. Cross-Screen Data Flow**

**Data Dependencies from Other Screens/Stores**

// Cross-screen data dependencies

interface CrossScreenDataFlow {

// Data flowing INTO Dashboard

incoming: {

userProfile: 'auth/profile' // User subscription, preferences

signalFilters: 'signals/filters' // Active filters from signals page

openPositions: 'trades/positions' // Current open trades

portfolioSettings: 'portfolio/settings' // Risk preferences, balance

alertSettings: 'alerts/preferences' // Notification preferences

}

// Data flowing OUT OF Dashboard

outgoing: {

selectedSignal: 'signals/detail' // Signal clicked for detailed view

tradeExecution: 'trades/new' // Paper trade initiated

filterChanges: 'signals/filters' // Filter changes to propagate

portfolioUpdates: 'portfolio/balance' // Balance changes from trades

navigationState: 'router/state' // Navigation context

}

}

// Central data synchronization hook

const useCrossScreenSync = () => {

const dashboardStore = useDashboardStore()

const authStore = useAuthStore()

const signalsStore = useSignalsStore()

const tradesStore = useTradesStore()

const portfolioStore = usePortfolioStore()

const alertsStore = useAlertsStore()

// Sync incoming data

useEffect(() => {

// User profile updates

const userProfile = authStore.user

if (userProfile) {

dashboardStore.updateUserContext(userProfile)

}

}, [authStore.user, dashboardStore.updateUserContext])

useEffect(() => {

// Signal filters from other screens

const savedFilters = signalsStore.activeFilters

if (savedFilters && !isEqual(savedFilters, dashboardStore.signals.filters)) {

dashboardStore.setSignalFilters(savedFilters)

}

}, [signalsStore.activeFilters, dashboardStore.signals.filters])

useEffect(() => {

// Open positions from trades

const openPositions = tradesStore.openPositions

dashboardStore.updateExistingPositions(openPositions.map(t => t.symbol))

}, [tradesStore.openPositions, dashboardStore.updateExistingPositions])

useEffect(() => {

// Portfolio settings and balance

const portfolioBalance = portfolioStore.balance

if (portfolioBalance !== dashboardStore.portfolio.balance) {

dashboardStore.updatePortfolioBalance(portfolioBalance)

}

}, [portfolioStore.balance, dashboardStore.portfolio.balance])

// Sync outgoing data

const syncSignalSelection = useCallback((signal: Signal) => {

signalsStore.setSelectedSignal(signal)

signalsStore.addToRecentlyViewed(signal)

}, [signalsStore.setSelectedSignal, signalsStore.addToRecentlyViewed])

const syncTradeExecution = useCallback((tradeData: PaperTrade) => {

tradesStore.addTrade(tradeData)

portfolioStore.updateBalance(tradeData.remainingBalance)

alertsStore.addTradeAlert(tradeData)

}, [tradesStore.addTrade, portfolioStore.updateBalance, alertsStore.addTradeAlert])

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

signalsStore.updateFilters(filters)

localStorage.setItem('globalSignalFilters', JSON.stringify(filters))

}, [signalsStore.updateFilters])