**Signals**

**🎯 Kurzora Signals Page - Complete UI Analysis**

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

**1. UI Components & Layout**

**Interactive Elements**

**Primary Signal Components:**

* **SignalFilters** (timeframe buttons, score slider, sector/market dropdowns)
* **SignalCard** (individual signal tiles with scores and CTAs)
* **SignalModal** (trade execution dialog with position sizing)
* **SignalDetail** (full-screen signal analysis view)
* **MarketStatusIndicator** (live/closed status with pulsing animation)
* **ResetFiltersButton** (clear all filters functionality)

**Navigation & Controls:**

* Dynamic breadcrumb navigation
* Real-time signal counter
* Filter reset functionality
* Market status indicator (live/closed)

**React + TypeScript Component Structure**

// Complete Signals Page Architecture

<Layout>

<div className="min-h-screen bg-slate-950">

{/\* Header Section \*/}

<div className="border-b border-slate-800 bg-slate-900/50 backdrop-blur-sm">

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

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

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

<h1 className="text-2xl font-bold text-white">Trading Signals</h1>

<MarketStatusIndicator status={marketStatus} />

</div>

<div className="text-right">

<div className="text-sm text-slate-400">Live Signals</div>

<div className="text-lg font-bold text-white">{filteredSignals.length}</div>

</div>

</div>

</div>

</div>

{/\* Main Content \*/}

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

<div className="flex gap-8 lg:flex-row flex-col">

{/\* Sidebar Filters \*/}

<div className="w-full lg:w-80 space-y-6">

<SignalFilters

timeFilter={timeFilter}

setTimeFilter={setTimeFilter}

scoreThreshold={scoreThreshold}

setScoreThreshold={setScoreThreshold}

sectorFilter={sectorFilter}

setSectorFilter={setSectorFilter}

marketFilter={marketFilter}

setMarketFilter={setMarketFilter}

language={language}

/>

</div>

{/\* Signal Cards Grid \*/}

<div className="flex-1">

{loading ? (

<SignalSkeletonGrid />

) : filteredSignals.length > 0 ? (

<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 xl:grid-cols-4 gap-6">

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

<SignalCard

key={`${signal.ticker}-${index}`}

signal={signal}

onViewDetails={handleViewDetails}

onOpenModal={handleOpenModal}

existingPositions={existingPositions}

/>

))}

</div>

) : (

<EmptySignalsState onResetFilters={resetFilters} />

)}

</div>

</div>

</div>

{/\* Modals \*/}

<SignalModal

isOpen={isModalOpen}

onClose={handleCloseModal}

signal={selectedSignal}

onExecuteTrade={handleExecuteTrade}

existingPositions={existingPositions}

/>

</div>

</Layout>

**Responsive Design Patterns**

/\* Mobile-first responsive classes \*/

.signal-grid {

@apply grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 xl:grid-cols-4 gap-6;

}

.filters-layout {

@apply flex gap-8 lg:flex-row flex-col;

}

.filters-sidebar {

@apply w-full lg:w-80 space-y-6;

}

/\* Mobile filter adjustments \*/

@media (max-width: 1023px) {

.filters-timeframe {

@apply grid grid-cols-2 gap-2;

}

.signal-card {

@apply min-h-[280px];

}

}

**2. State Management (Zustand)**

**Store Structure**

interface SignalsStore {

// Data State

signals: Signal[];

filteredSignals: Signal[];

selectedSignal: Signal | null;

loading: boolean;

error: string | null;

lastUpdated: string;

existingPositions: string[];

// Filter State

filters: {

timeframe: string;

scoreThreshold: number[];

sector: string;

market: string;

};

// UI State

isModalOpen: boolean;

isDetailViewOpen: boolean;

marketStatus: 'open' | 'closed' | 'premarket' | 'afterhours';

// Actions

loadSignals: () => Promise<void>;

updateFilters: (filters: Partial<SignalFilters>) => void;

applyFilters: () => void;

selectSignal: (signal: Signal) => void;

openModal: () => void;

closeModal: () => void;

executeTrade: (tradeData: TradeData) => Promise<void>;

resetFilters: () => void;

refreshSignals: () => Promise<void>;

}

// Zustand Store Implementation

export const useSignalsStore = create<SignalsStore>((set, get) => ({

// Initial State

signals: [],

filteredSignals: [],

selectedSignal: null,

loading: false,

error: null,

lastUpdated: '',

existingPositions: [],

filters: {

timeframe: '1D',

scoreThreshold: [70],

sector: 'all',

market: 'global',

},

isModalOpen: false,

isDetailViewOpen: false,

marketStatus: 'closed',

// Actions

loadSignals: async () => {

set({ loading: true, error: null });

try {

const response = await fetch('/api/signals');

const data = await response.json();

set({

signals: data.signals,

lastUpdated: new Date().toISOString(),

loading: false

});

get().applyFilters();

} catch (error) {

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

}

},

updateFilters: (newFilters) => {

set(state => ({

filters: { ...state.filters, ...newFilters }

}));

get().applyFilters();

},

applyFilters: () => {

const { signals, filters } = get();

const filtered = signals.filter(signal => {

const meetsScore = signal.signals[filters.timeframe] >= filters.scoreThreshold[0];

const meetsSector = filters.sector === 'all' || signal.sector === filters.sector;

const meetsMarket = filters.market === 'global' || signal.market === filters.market;

return meetsScore && meetsSector && meetsMarket;

});

set({ filteredSignals: filtered });

},

resetFilters: () => {

set({

filters: {

timeframe: '1D',

scoreThreshold: [70],

sector: 'all',

market: 'global',

}

});

get().applyFilters();

},

selectSignal: (signal) => set({ selectedSignal: signal }),

openModal: () => set({ isModalOpen: true }),

closeModal: () => set({ isModalOpen: false, selectedSignal: null }),

executeTrade: async (tradeData) => {

try {

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

method: 'POST',

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

body: JSON.stringify(tradeData)

});

if (response.ok) {

set(state => ({

existingPositions: [...state.existingPositions, tradeData.symbol]

}));

get().closeModal();

}

} catch (error) {

set({ error: error.message });

}

}

}));

**3. API Contracts & Integration**

**API Endpoints**

// GET /api/signals - Fetch all signals

interface SignalsRequest {

timeframe?: '1H' | '4H' | '1D' | '1W';

minScore?: number;

sector?: string;

market?: string;

limit?: number;

offset?: number;

}

interface SignalsResponse {

signals: Signal[];

total: number;

lastUpdated: string;

marketStatus: 'open' | 'closed' | 'premarket' | 'afterhours';

}

// POST /api/paper-trades - Execute paper trade

interface PaperTradeRequest {

symbol: string;

name: string;

entryPrice: number;

shares: number;

stopLoss: number;

takeProfit: number;

investmentAmount: number;

signalScore: number;

userId: string;

}

interface PaperTradeResponse {

tradeId: string;

status: 'success' | 'error';

message: string;

trade?: PaperTrade;

}

// GET /api/signals/:symbol/detail - Get detailed signal analysis

interface SignalDetailResponse {

signal: Signal;

technicalAnalysis: {

rsi: { value: number; signal: 'buy' | 'sell' | 'hold' };

macd: { value: number; signal: 'buy' | 'sell' | 'hold' };

volume: { ratio: number; signal: 'high' | 'normal' | 'low' };

support: { level: number; strength: number };

};

aiExplanation: string;

riskAnalysis: {

volatility: number;

beta: number;

riskScore: number;

};

}

// WebSocket Events for Real-time Updates

interface WebSocketEvents {

'signal-update': Signal;

'market-status': 'open' | 'closed' | 'premarket' | 'afterhours';

'new-signal': Signal;

'signal-removed': { symbol: string };

}

**4. Performance & Optimization**

**Lazy Loading Strategies**

// Lazy load heavy components

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

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

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

// Virtual scrolling for large signal lists

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

// Code splitting by routes

const SignalsPage = lazy(() => import('../pages/Signals'));

**Memoization Opportunities**

// Memoized signal card component

const SignalCard = React.memo(({ signal, onViewDetails, onOpenModal }) => {

// Component implementation

}, (prevProps, nextProps) => {

return prevProps.signal.ticker === nextProps.signal.ticker &&

prevProps.signal.signals === nextProps.signal.signals;

});

// Memoized filter calculations

const useFilteredSignals = (signals: Signal[], filters: Filters) => {

return useMemo(() => {

return signals.filter(signal => {

// Filter logic

});

}, [signals, filters]);

};

// Memoized score calculations

const useSignalScore = (signal: Signal, timeframe: string) => {

return useMemo(() => {

return calculateWeightedScore(signal.signals, timeframe);

}, [signal.signals, timeframe]);

};

**Caching Strategies**

// React Query for server state management

const useSignals = (filters: Filters) => {

return useQuery({

queryKey: ['signals', filters],

queryFn: () => fetchSignals(filters),

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

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

refetchInterval: 30000, // 30 seconds during market hours

});

};

// Local storage for user preferences

const usePersistedFilters = () => {

const [filters, setFilters] = useState(() => {

const saved = localStorage.getItem('signal-filters');

return saved ? JSON.parse(saved) : defaultFilters;

});

useEffect(() => {

localStorage.setItem('signal-filters', JSON.stringify(filters));

}, [filters]);

return [filters, setFilters];

};

**5. Database Schema**

**PostgreSQL Table Structures**

-- Signals table

CREATE TABLE signals (

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

ticker VARCHAR(20) NOT NULL,

name VARCHAR(255) NOT NULL,

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

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

sector VARCHAR(50) NOT NULL,

market VARCHAR(20) NOT NULL,

signals JSONB NOT NULL, -- {1H: 85, 4H: 92, 1D: 88, 1W: 90}

volume\_ratio DECIMAL(5,2) DEFAULT 1.0,

rsi\_14 DECIMAL(5,2),

macd\_histogram DECIMAL(8,4),

support\_level DECIMAL(10,2),

resistance\_level DECIMAL(10,2),

is\_shariah\_compliant BOOLEAN DEFAULT false,

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

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

-- Indexes for performance

INDEX idx\_signals\_ticker (ticker),

INDEX idx\_signals\_sector (sector),

INDEX idx\_signals\_market (market),

INDEX idx\_signals\_score ((signals->>'1D')::int),

INDEX idx\_signals\_updated (updated\_at DESC)

);

-- Paper trades table

CREATE TABLE paper\_trades (

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

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

signal\_id UUID REFERENCES signals(id),

ticker VARCHAR(20) NOT NULL,

trade\_type VARCHAR(10) NOT NULL, -- 'BUY', 'SELL'

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

current\_price DECIMAL(10,2),

shares INTEGER NOT NULL,

stop\_loss DECIMAL(10,2),

take\_profit DECIMAL(10,2),

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

current\_value DECIMAL(12,2),

unrealized\_pnl DECIMAL(12,2),

signal\_score INTEGER NOT NULL,

status VARCHAR(20) DEFAULT 'OPEN', -- 'OPEN', 'CLOSED', 'STOPPED'

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

closed\_at TIMESTAMP WITH TIME ZONE,

INDEX idx\_paper\_trades\_user (user\_id),

INDEX idx\_paper\_trades\_ticker (ticker),

INDEX idx\_paper\_trades\_status (status),

INDEX idx\_paper\_trades\_opened (opened\_at DESC)

);

-- Signal history for backtesting

CREATE TABLE signal\_history (

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

ticker VARCHAR(20) NOT NULL,

signal\_score INTEGER NOT NULL,

timeframe VARCHAR(5) NOT NULL,

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

price\_after\_1d DECIMAL(10,2),

price\_after\_1w DECIMAL(10,2),

price\_after\_1m DECIMAL(10,2),

was\_successful BOOLEAN,

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

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

INDEX idx\_signal\_history\_timeframe (timeframe),

INDEX idx\_signal\_history\_recorded (recorded\_at DESC)

);

-- User watchlists

CREATE TABLE user\_watchlists (

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

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

name VARCHAR(100) NOT NULL,

tickers TEXT[] NOT NULL,

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

INDEX idx\_watchlists\_user (user\_id)

);

**6. User Experience**

**Loading States & Skeleton Screens**

const SignalSkeletonGrid = () => (

<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 xl:grid-cols-4 gap-6">

{Array.from({ length: 12 }).map((\_, i) => (

<div key={i} className="bg-slate-900/50 border-2 border-slate-700 rounded-lg p-6 animate-pulse">

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

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

</div>

<div className="text-center mb-4">

<div className="w-16 h-6 bg-slate-700 rounded mx-auto mb-2"></div>

<div className="w-32 h-4 bg-slate-700 rounded mx-auto"></div>

</div>

<div className="text-center mb-6">

<div className="w-24 h-8 bg-slate-700 rounded mx-auto mb-2"></div>

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

</div>

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

</div>

))}

</div>

);

const EmptySignalsState = ({ onResetFilters }: { onResetFilters: () => void }) => (

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

<div className="text-6xl mb-4">📊</div>

<h3 className="text-xl font-semibold text-white mb-2">No signals found</h3>

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

Try adjusting your filters to see more trading opportunities.

</p>

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

Reset All Filters

</Button>

</div>

);

**Error Boundaries & Fallback UI**

class SignalsErrorBoundary extends React.Component {

constructor(props) {

super(props);

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

}

static getDerivedStateFromError(error) {

return { hasError: true, error };

}

componentDidCatch(error, errorInfo) {

console.error('Signals page error:', error, errorInfo);

// Log to error reporting service

}

render() {

if (this.state.hasError) {

return (

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

<div className="text-6xl mb-4">⚠️</div>

<h3 className="text-xl font-semibold text-white mb-2">Something went wrong</h3>

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

We're having trouble loading signals. Please try refreshing the page.

</p>

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

Refresh Page

</Button>

</div>

);

}

return this.props.children;

}

}

**Accessibility Considerations**

// ARIA labels and keyboard navigation

const SignalCard = ({ signal, onViewDetails }) => (

<Card

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

role="button"

tabIndex={0}

aria-label={`${signal.ticker} signal with score ${signal.signalScore}`}

onKeyDown={(e) => {

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

onViewDetails(signal);

}

}}

>

{/\* Card content \*/}

</Card>

);

// Screen reader announcements

const useSignalAnnouncements = () => {

const announce = (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;

};

**7. Integration Points**

**Navigation Patterns**

// Route configuration

const signalRoutes = {

signals: '/signals',

signalDetail: '/signals/:symbol',

signalHistory: '/signals/history',

watchlist: '/watchlist',

openPositions: '/positions'

};

// Navigation handlers

const useSignalNavigation = () => {

const navigate = useNavigate();

const navigateToDetail = (signal: Signal) => {

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

state: {

selectedStock: signal,

timeframe: '1D',

score: signal.signalScore

}

});

};

const navigateToOrders = (signal: Signal) => {

navigate('/orders', {

state: { selectedStock: signal }

});

};

return { navigateToDetail, navigateToOrders };

};

**Shared State Synchronization**

// Cross-page state sharing

export const useGlobalSignalState = () => {

const signalsStore = useSignalsStore();

const ordersStore = useOrdersStore();

const dashboardStore = useDashboardStore();

// Sync new trades with orders page

const syncTradeExecution = (trade: PaperTrade) => {

ordersStore.addTrade(trade);

dashboardStore.updateRecentTrades(trade);

signalsStore.addExistingPosition(trade.symbol);

};

return { syncTradeExecution };

};

**8. Testing Strategy**

**Unit Test Requirements**

// SignalCard.test.tsx

describe('SignalCard', () => {

it('displays signal score with correct color coding', () => {

const highScoreSignal = { ...mockSignal, signalScore: 92 };

render(<SignalCard signal={highScoreSignal} />);

expect(screen.getByText('92')).toHaveClass('bg-emerald-500');

});

it('shows position open indicator for existing positions', () => {

render(

<SignalCard

signal={mockSignal}

existingPositions={['AAPL']}

/>

);

expect(screen.getByText('Position Open')).toBeInTheDocument();

});

it('calls onViewDetails when View Signal button is clicked', () => {

const mockOnViewDetails = jest.fn();

render(<SignalCard signal={mockSignal} onViewDetails={mockOnViewDetails} />);

fireEvent.click(screen.getByText('View Signal'));

expect(mockOnViewDetails).toHaveBeenCalledWith(mockSignal);

});

});

// SignalFilters.test.tsx

describe('SignalFilters', () => {

it('updates score threshold when slider changes', () => {

const mockSetScoreThreshold = jest.fn();

render(<SignalFilters scoreThreshold={[70]} setScoreThreshold={mockSetScoreThreshold} />);

const slider = screen.getByRole('slider');

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

expect(mockSetScoreThreshold).toHaveBeenCalledWith([85]);

});

});

**Integration Test Scenarios**

// signals.integration.test.tsx

describe('Signals Page Integration', () => {

it('filters signals correctly when multiple filters are applied', async () => {

render(<SignalsPage />);

// Apply timeframe filter

fireEvent.click(screen.getByText('4H'));

// Apply sector filter

fireEvent.click(screen.getByText('Technology'));

// Apply score filter

const slider = screen.getByRole('slider');

fireEvent.change(slider, { target: { value: '80' } });

await waitFor(() => {

const signalCards = screen.getAllByTestId('signal-card');

expect(signalCards).toHaveLength(3); // Expected filtered results

});

});

it('executes paper trade flow correctly', async () => {

const mockExecuteTrade = jest.fn();

render(<SignalsPage onExecuteTrade={mockExecuteTrade} />);

// Click on a signal card

fireEvent.click(screen.getByText('View Signal'));

// Open modal

expect(screen.getByText('Execute Paper Trade')).toBeInTheDocument();

// Adjust position size

const riskSlider = screen.getByLabelText(/Risk Percentage/);

fireEvent.change(riskSlider, { target: { value: '2' } });

// Execute trade

fireEvent.click(screen.getByText('Execute Paper Trade'));

await waitFor(() => {

expect(mockExecuteTrade).toHaveBeenCalled();

});

});

});

**9. Charts & Data Visualizations**

**TradingView Widget Integration**

// RealisticSignalChart.tsx

const RealisticSignalChart = ({

symbol,

timeframe,

signalScore

}: {

symbol: string;

timeframe: string;

signalScore: number;

}) => {

useEffect(() => {

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

script.src = 'https://s3.tradingview.com/external-embedding/embed-widget-advanced-chart.js';

script.type = 'text/javascript';

script.async = true;

script.innerHTML = JSON.stringify({

autosize: true,

symbol: `NASDAQ:${symbol}`,

interval: timeframe,

timezone: "Etc/UTC",

theme: "dark",

style: "1",

locale: "en",

toolbar\_bg: "#1e293b",

enable\_publishing: false,

allow\_symbol\_change: false,

container\_id: `tradingview\_${symbol}`

});

const container = document.getElementById(`tradingview\_${symbol}`);

if (container) {

container.appendChild(script);

}

return () => {

if (container) {

container.innerHTML = '';

}

};

}, [symbol, timeframe]);

return (

<div className="bg-slate-800 rounded-lg p-4">

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

<h3 className="text-lg font-semibold text-white">{symbol} Chart</h3>

<Badge className="bg-emerald-600 text-white">

Score: {signalScore}

</Badge>

</div>

<div

id={`tradingview\_${symbol}`}

className="h-[400px] w-full"

/>

</div>

);

};

**Signal Strength Progress Bars**

const SignalStrengthBar = ({

score,

timeframe

}: {

score: number;

timeframe: string;

}) => {

const getColorClass = (score: number) => {

if (score >= 90) return 'bg-gradient-to-r from-emerald-400 to-emerald-600';

if (score >= 80) return 'bg-gradient-to-r from-blue-400 to-blue-600';

if (score >= 70) return 'bg-gradient-to-r from-yellow-400 to-yellow-600';

return 'bg-gradient-to-r from-red-400 to-red-600';

};

return (

<div className="w-full">

<div className="flex justify-between text-sm text-slate-400 mb-1">

<span>Signal Strength</span>

<span>{score}/100</span>

</div>

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

<div

className={`h-3 rounded-full transition-all duration-500 ${getColorClass(score)}`}

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

/>

</div>

<div className="text-xs text-slate-500 mt-1 text-center">

{timeframe} Timeframe

</div>

</div>

);

};

**10. Visual Data Elements**

**Dynamic Score Badges**

const ScoreBadge = ({ score }: { score: number }) => {

const getBadgeStyle = (score: number) => {

if (score >= 90) return {

className: 'bg-emerald-500 text-white ring-2 ring-emerald-400 ring-opacity-50',

emoji: '💎',

label: 'Very Strong'

};

if (score >= 80) return {

className: 'bg-blue-500 text-white ring-2 ring-blue-400 ring-opacity-50',

emoji: '✅',

label: 'Strong'

};

if (score >= 70) return {

className: 'bg-yellow-500 text-white ring-2 ring-yellow-400 ring-opacity-50',

emoji: '⚠️',

label: 'Valid'

};

return {

className: 'bg-red-500 text-white ring-2 ring-red-400 ring-opacity-50',

emoji: '🚫',

label: 'Weak'

};

};

const style = getBadgeStyle(score);

return (

<div className={`px-4 py-2 rounded-full text-lg font-bold flex items-center space-x-2 ${style.className}`}>

<span>{style.emoji}</span>

<span>{score}</span>

</div>

);

};

**Animated Price Changes**

const AnimatedPriceChange = ({

change,

animate = true

}: {

change: number;

animate?: boolean;

}) => {

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

useEffect(() => {

if (animate) {

setIsAnimating(true);

const timer = setTimeout(() => setIsAnimating(false), 1000);

return () => clearTimeout(timer);

}

}, [change, animate]);

return (

<div className={`flex items-center text-sm font-medium transition-all duration-300 ${

change >= 0 ? 'text-emerald-400' : 'text-red-400'

} ${isAnimating ? 'scale-110' : 'scale-100'}`}>

{change >= 0 ? (

<ArrowUp className="h-4 w-4 mr-1" />

) : (

<ArrowDown className="h-4 w-4 mr-1" />

)}

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

</div>

);

};

**11. Security & Validation**

**Input Validation Schemas**

import { z } from 'zod';

// Signal filter validation

export const signalFiltersSchema = z.object({

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

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

sector: z.enum(['all', 'technology', 'healthcare', 'finance', 'consumer', 'energy', 'crypto']),

market: z.enum(['global', 'usa', 'saudi', 'uae', 'qatar', 'kuwait', 'bahrain', 'oman', 'crypto'])

});

// Paper trade validation

export const paperTradeSchema = z.object({

symbol: z.string().min(1).max(10),

shares: z.number().positive().max(10000),

entryPrice: z.number().positive(),

stopLoss: z.number().positive(),

takeProfit: z.number().positive(),

investmentAmount: z.number().positive().max(1000000),

riskPercentage: z.number().min(0.5).max(10)

}).refine(

(data) => data.stopLoss < data.entryPrice,

{ message: "Stop loss must be below entry price" }

).refine(

(data) => data.takeProfit > data.entryPrice,

{ message: "Take profit must be above entry price" }

);

// API request validation middleware

export const validateSignalFilters = (req: Request, res: Response, next: NextFunction) => {

try {

signalFiltersSchema.parse(req.query);

next();

} catch (error) {

res.status(400).json({ error: 'Invalid filter parameters' });

}

};

**Authentication & Authorization**

// Protected route wrapper

const ProtectedSignalsPage = () => {

const { user, loading } = useAuth();

const navigate = useNavigate();

useEffect(() => {

if (!loading && !user) {

navigate('/auth/login');

}

}, [user, loading, navigate]);

if (loading) return <LoadingSpinner />;

if (!user) return null;

return <SignalsPage />;

};

// API route protection

export const requireAuth = async (req: Request, res: Response, next: NextFunction) => {

try {

const token = req.headers.authorization?.replace('Bearer ', '');

if (!token) {

return res.status(401).json({ error: 'No token provided' });

}

const decodedToken = await admin.auth().verifyIdToken(token);

req.user = decodedToken;

next();

} catch (error) {

res.status(401).json({ error: 'Invalid token' });

}

};

**12. Environment & Configuration**

**Environment Variables**

// .env.local

NEXT\_PUBLIC\_API\_URL=http://localhost:3001

NEXT\_PUBLIC\_WS\_URL=ws://localhost:3001

NEXT\_PUBLIC\_SUPABASE\_URL=https://your-project.supabase.co

NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY=your\_supabase\_anon\_key

NEXT\_PUBLIC\_POLYGON\_API\_KEY=your\_polygon\_api\_key

NEXT\_PUBLIC\_TRADINGVIEW\_WIDGET\_ID=your\_tradingview\_id

# Server-side only

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

OPENAI\_API\_KEY=your\_openai\_key

MAKE\_WEBHOOK\_URL=https://hook.integromat.com/your-webhook

TELEGRAM\_BOT\_TOKEN=your\_telegram\_token

**Feature Flags**

// Feature flag configuration

export const featureFlags = {

realTimeSignals: process.env.NODE\_ENV === 'production',

cryptoSignals: true,

saudiMarket: true,

aiExplanations: true,

paperTrading: true,

telegramAlerts: process.env.TELEGRAM\_BOT\_TOKEN !== undefined,

advancedCharts: true,

darkMode: true,

multiLanguage: true

};

// Feature flag hook

export const useFeatureFlag = (flag: keyof typeof featureFlags) => {

return featureFlags[flag];

};

**Monitoring & Analytics**

// Analytics tracking

export const trackSignalEvent = (eventName: string, properties: Record<string, any>) => {

if (typeof window !== 'undefined' && window.gtag) {

window.gtag('event', eventName, {

custom\_parameter\_1: properties.signal\_score,

custom\_parameter\_2: properties.timeframe,

custom\_parameter\_3: properties.sector

});

}

};

// Error reporting

export const reportError = (error: Error, context: Record<string, any>) => {

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

// Sentry.captureException(error, { extra: context });

console.error('Error reported:', error, context);

}

};

**13. Cross-Screen Data Flow**

**Data Dependencies**

// Global state synchronization

export const useGlobalDataSync = () => {

const signalsStore = useSignalsStore();

const dashboardStore = useDashboardStore();

const ordersStore = useOrdersStore();

// Sync signals with dashboard metrics

useEffect(() => {

const strongSignals = signalsStore.filteredSignals.filter(s => s.signalScore >= 80);

dashboardStore.updateMetrics({

todaySignals: signalsStore.filteredSignals.length,

strongSignals: strongSignals.length

});

}, [signalsStore.filteredSignals]);

// Sync paper trades with open positions

useEffect(() => {

const openPositions = ordersStore.trades.filter(t => t.status === 'OPEN');

signalsStore.setExistingPositions(openPositions.map(t => t.symbol));

}, [ordersStore.trades]);

};

**Real-time Update Propagation**

// WebSocket connection for real-time updates

export const useRealtimeSignals = () => {

const signalsStore = useSignalsStore();

const [socket, setSocket] = useState<WebSocket | null>(null);

useEffect(() => {

const ws = new WebSocket(process.env.NEXT\_PUBLIC\_WS\_URL);

ws.onmessage = (event) => {

const data = JSON.parse(event.data);

switch (data.type) {

case 'signal-update':

signalsStore.updateSignal(data.signal);

break;

case 'new-signal':

signalsStore.addSignal(data.signal);

trackSignalEvent('new\_signal\_received', {

signal\_score: data.signal.signalScore,

timeframe: data.signal.timeframe

});

break;

case 'market-status':

signalsStore.setMarketStatus(data.status);

break;

}

};

setSocket(ws);

return () => {

ws.close();

};

}, []);

return socket;

};

**Cache Invalidation Strategies**

// React Query cache invalidation

export const useSignalCacheManagement = () => {

const queryClient = useQueryClient();

const invalidateSignalsCache = () => {

queryClient.invalidateQueries({ queryKey: ['signals'] });

};

const updateSignalInCache = (updatedSignal: Signal) => {

queryClient.setQueryData(['signals'], (oldData: Signal[]) => {

return oldData?.map(signal =>

signal.ticker === updatedSignal.ticker ? updatedSignal : signal

);

});

};

const prefetchSignalDetail = (symbol: string) => {

queryClient.prefetchQuery({

queryKey: ['signal-detail', symbol],

queryFn: () => fetchSignalDetail(symbol)

});

};

return {

invalidateSignalsCache,

updateSignalInCache,

prefetchSignalDetail

};

};

**🚀 Implementation Checklist**

**Phase 1: Core Components (Week 1)**

* [ ] Create SignalCard component with proper styling
* [ ] Implement SignalFilters with all filter types
* [ ] Set up basic Zustand store for signals
* [ ] Add loading states and skeleton screens
* [ ] Implement responsive grid layout

**Phase 2: Advanced Features (Week 2)**

* [ ] Add SignalModal with paper trading functionality
* [ ] Integrate TradingView charts
* [ ] Implement real-time WebSocket updates
* [ ] Add error boundaries and error handling
* [ ] Create signal detail page

**Phase 3: Optimization & Polish (Week 3)**

* [ ] Add React Query for server state management
* [ ] Implement lazy loading and code splitting
* [ ] Add comprehensive test coverage
* [ ] Optimize performance with memoization
* [ ] Add accessibility features

**Phase 4: Integration (Week 4)**

* [ ] Connect to PostgreSQL database
* [ ] Set up Firebase Cloud Functions
* [ ] Integrate with Make.com for alerts
* [ ] Add monitoring and analytics
* [ ] Deploy to production environment

**This comprehensive analysis provides everything needed for immediate implementation in Cursor. Each section includes specific code examples, TypeScript interfaces, and detailed implementation guidance for building a production-ready Signals page.**