<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>AlgoTrade Pro - Advanced Trading Platform</title>

<style>

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

body {

font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

background: linear-gradient(135deg, #0f0f23 0%, #1a1a2e 100%);

color: #ffffff;

overflow-x: hidden;

}

.container {

max-width: 1400px;

margin: 0 auto;

padding: 20px;

}

.header {

background: rgba(255, 255, 255, 0.05);

backdrop-filter: blur(10px);

border-radius: 15px;

padding: 20px;

margin-bottom: 20px;

border: 1px solid rgba(255, 255, 255, 0.1);

}

.header h1 {

font-size: 2.5em;

background: linear-gradient(45deg, #00ff88, #00ccff);

-webkit-background-clip: text;

-webkit-text-fill-color: transparent;

text-align: center;

margin-bottom: 10px;

}

.status-bar {

display: flex;

justify-content: space-between;

align-items: center;

padding: 10px 0;

border-top: 1px solid rgba(255, 255, 255, 0.1);

margin-top: 15px;

}

.status-item {

display: flex;

align-items: center;

gap: 8px;

}

.status-dot {

width: 8px;

height: 8px;

border-radius: 50%;

background: #00ff88;

animation: pulse 2s infinite;

}

@keyframes pulse {

0% { opacity: 1; }

50% { opacity: 0.5; }

100% { opacity: 1; }

}

.main-grid {

display: grid;

grid-template-columns: 1fr 1fr;

gap: 20px;

margin-bottom: 20px;

}

.card {

background: rgba(255, 255, 255, 0.05);

backdrop-filter: blur(10px);

border-radius: 15px;

padding: 20px;

border: 1px solid rgba(255, 255, 255, 0.1);

transition: transform 0.3s ease, box-shadow 0.3s ease;

}

.card:hover {

transform: translateY(-5px);

box-shadow: 0 15px 35px rgba(0, 255, 136, 0.2);

}

.card h3 {

color: #00ff88;

margin-bottom: 15px;

font-size: 1.2em;

}

.strategy-grid {

display: grid;

grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));

gap: 15px;

margin-bottom: 20px;

}

.strategy-card {

background: rgba(255, 255, 255, 0.03);

border-radius: 10px;

padding: 15px;

border: 1px solid rgba(255, 255, 255, 0.05);

position: relative;

overflow: hidden;

}

.strategy-card::before {

content: '';

position: absolute;

top: 0;

left: 0;

right: 0;

height: 3px;

background: linear-gradient(90deg, #00ff88, #00ccff);

}

.strategy-header {

display: flex;

justify-content: space-between;

align-items: center;

margin-bottom: 10px;

}

.strategy-name {

font-weight: bold;

color: #ffffff;

}

.strategy-status {

padding: 4px 8px;

border-radius: 12px;

font-size: 0.8em;

font-weight: bold;

}

.active { background: rgba(0, 255, 136, 0.2); color: #00ff88; }

.inactive { background: rgba(255, 100, 100, 0.2); color: #ff6464; }

.metrics {

display: grid;

grid-template-columns: repeat(3, 1fr);

gap: 10px;

margin-top: 10px;

}

.metric {

text-align: center;

padding: 8px;

background: rgba(255, 255, 255, 0.05);

border-radius: 8px;

}

.metric-value {

font-size: 1.1em;

font-weight: bold;

color: #00ff88;

}

.metric-label {

font-size: 0.8em;

color: #cccccc;

}

.controls {

display: flex;

gap: 10px;

margin-top: 15px;

}

.btn {

padding: 8px 16px;

border: none;

border-radius: 8px;

cursor: pointer;

font-weight: bold;

transition: all 0.3s ease;

flex: 1;

}

.btn-primary {

background: linear-gradient(45deg, #00ff88, #00ccff);

color: #000;

}

.btn-secondary {

background: rgba(255, 255, 255, 0.1);

color: #fff;

border: 1px solid rgba(255, 255, 255, 0.2);

}

.btn:hover {

transform: translateY(-2px);

box-shadow: 0 5px 15px rgba(0, 255, 136, 0.3);

}

.portfolio-summary {

display: grid;

grid-template-columns: repeat(4, 1fr);

gap: 15px;

margin-bottom: 20px;

}

.portfolio-item {

background: rgba(255, 255, 255, 0.05);

padding: 15px;

border-radius: 10px;

text-align: center;

border: 1px solid rgba(255, 255, 255, 0.1);

}

.portfolio-value {

font-size: 1.5em;

font-weight: bold;

color: #00ff88;

margin-bottom: 5px;

}

.portfolio-label {

color: #cccccc;

font-size: 0.9em;

}

.trade-log {

max-height: 300px;

overflow-y: auto;

border: 1px solid rgba(255, 255, 255, 0.1);

border-radius: 8px;

padding: 10px;

}

.trade-entry {

display: flex;

justify-content: space-between;

padding: 8px 0;

border-bottom: 1px solid rgba(255, 255, 255, 0.05);

}

.trade-entry:last-child {

border-bottom: none;

}

.trade-symbol {

font-weight: bold;

color: #00ccff;

}

.trade-profit {

font-weight: bold;

}

.profit-positive { color: #00ff88; }

.profit-negative { color: #ff6464; }

.settings-panel {

background: rgba(255, 255, 255, 0.03);

border-radius: 10px;

padding: 15px;

margin-top: 15px;

}

.setting-group {

margin-bottom: 15px;

}

.setting-label {

display: block;

margin-bottom: 5px;

color: #cccccc;

font-size: 0.9em;

}

.setting-input {

width: 100%;

padding: 8px 12px;

background: rgba(255, 255, 255, 0.1);

border: 1px solid rgba(255, 255, 255, 0.2);

border-radius: 6px;

color: #fff;

font-size: 0.9em;

}

.setting-input:focus {

outline: none;

border-color: #00ff88;

box-shadow: 0 0 0 2px rgba(0, 255, 136, 0.2);

}

.market-data {

background: rgba(255, 255, 255, 0.03);

border-radius: 10px;

padding: 15px;

margin-top: 15px;

}

.market-item {

display: flex;

justify-content: space-between;

align-items: center;

padding: 8px 0;

border-bottom: 1px solid rgba(255, 255, 255, 0.05);

}

.market-item:last-child {

border-bottom: none;

}

.price-change {

font-weight: bold;

}

.footer {

text-align: center;

margin-top: 30px;

padding: 20px;

color: #888;

border-top: 1px solid rgba(255, 255, 255, 0.1);

}

@media (max-width: 768px) {

.main-grid {

grid-template-columns: 1fr;

}

.portfolio-summary {

grid-template-columns: repeat(2, 1fr);

}

.strategy-grid {

grid-template-columns: 1fr;

}

}

</style>

</head>

<body>

<div class="container">

<div class="header">

<h1>AlgoTrade Pro</h1>

<div class="status-bar">

<div class="status-item">

<div class="status-dot"></div>

<span>Market Open</span>

</div>

<div class="status-item">

<span id="current-time"></span>

</div>

<div class="status-item">

<span>Latency: <span id="latency">0.2ms</span></span>

</div>

</div>

</div>

<div class="portfolio-summary">

<div class="portfolio-item">

<div class="portfolio-value" id="total-value">$125,847</div>

<div class="portfolio-label">Total Portfolio</div>

</div>

<div class="portfolio-item">

<div class="portfolio-value profit-positive" id="daily-pnl">+$3,247</div>

<div class="portfolio-label">Daily P&L</div>

</div>

<div class="portfolio-item">

<div class="portfolio-value" id="active-trades">12</div>

<div class="portfolio-label">Active Trades</div>

</div>

<div class="portfolio-item">

<div class="portfolio-value profit-positive" id="win-rate">68.4%</div>

<div class="portfolio-label">Win Rate</div>

</div>

</div>

<div class="main-grid">

<div class="card">

<h3>Trading Strategies</h3>

<div class="strategy-grid">

<div class="strategy-card">

<div class="strategy-header">

<span class="strategy-name">Momentum Scalping</span>

<span class="strategy-status active">ACTIVE</span>

</div>

<div class="metrics">

<div class="metric">

<div class="metric-value">+$1,247</div>

<div class="metric-label">Today P&L</div>

</div>

<div class="metric">

<div class="metric-value">34</div>

<div class="metric-label">Trades</div>

</div>

<div class="metric">

<div class="metric-value">72%</div>

<div class="metric-label">Win Rate</div>

</div>

</div>

<div class="controls">

<button class="btn btn-secondary" onclick="pauseStrategy('momentum')">Pause</button>

<button class="btn btn-primary" onclick="optimizeStrategy('momentum')">Optimize</button>

</div>

</div>

<div class="strategy-card">

<div class="strategy-header">

<span class="strategy-name">Arbitrage Hunter</span>

<span class="strategy-status active">ACTIVE</span>

</div>

<div class="metrics">

<div class="metric">

<div class="metric-value">+$892</div>

<div class="metric-label">Today P&L</div>

</div>

<div class="metric">

<div class="metric-value">18</div>

<div class="metric-label">Trades</div>

</div>

<div class="metric">

<div class="metric-value">89%</div>

<div class="metric-label">Win Rate</div>

</div>

</div>

<div class="controls">

<button class="btn btn-secondary" onclick="pauseStrategy('arbitrage')">Pause</button>

<button class="btn btn-primary" onclick="optimizeStrategy('arbitrage')">Optimize</button>

</div>

</div>

<div class="strategy-card">

<div class="strategy-header">

<span class="strategy-name">Mean Reversion</span>

<span class="strategy-status active">ACTIVE</span>

</div>

<div class="metrics">

<div class="metric">

<div class="metric-value">+$1,108</div>

<div class="metric-label">Today P&L</div>

</div>

<div class="metric">

<div class="metric-value">22</div>

<div class="metric-label">Trades</div>

</div>

<div class="metric">

<div class="metric-value">64%</div>

<div class="metric-label">Win Rate</div>

</div>

</div>

<div class="controls">

<button class="btn btn-secondary" onclick="pauseStrategy('meanrev')">Pause</button>

<button class="btn btn-primary" onclick="optimizeStrategy('meanrev')">Optimize</button>

</div>

</div>

<div class="strategy-card">

<div class="strategy-header">

<span class="strategy-name">Statistical Arbitrage</span>

<span class="strategy-status inactive">PAUSED</span>

</div>

<div class="metrics">

<div class="metric">

<div class="metric-value">$0</div>

<div class="metric-label">Today P&L</div>

</div>

<div class="metric">

<div class="metric-value">0</div>

<div class="metric-label">Trades</div>

</div>

<div class="metric">

<div class="metric-value">--</div>

<div class="metric-label">Win Rate</div>

</div>

</div>

<div class="controls">

<button class="btn btn-primary" onclick="startStrategy('statarb')">Start</button>

<button class="btn btn-secondary" onclick="configureStrategy('statarb')">Config</button>

</div>

</div>

</div>

<div class="settings-panel">

<h4 style="color: #00ff88; margin-bottom: 15px;">Risk Management</h4>

<div class="setting-group">

<label class="setting-label">Max Position Size</label>

<input type="number" class="setting-input" value="10000" id="max-position">

</div>

<div class="setting-group">

<label class="setting-label">Daily Loss Limit</label>

<input type="number" class="setting-input" value="5000" id="loss-limit">

</div>

<div class="setting-group">

<label class="setting-label">Stop Loss %</label>

<input type="number" class="setting-input" value="2" step="0.1" id="stop-loss">

</div>

</div>

</div>

<div class="card">

<h3>Live Trading Feed</h3>

<div class="trade-log" id="trade-log">

<div class="trade-entry">

<span class="trade-symbol">AAPL</span>

<span>BUY 100 @ $175.23</span>

<span class="trade-profit profit-positive">+$127</span>

</div>

<div class="trade-entry">

<span class="trade-symbol">MSFT</span>

<span>SELL 50 @ $412.88</span>

<span class="trade-profit profit-positive">+$89</span>

</div>

<div class="trade-entry">

<span class="trade-symbol">GOOGL</span>

<span>BUY 25 @ $2,847.12</span>

<span class="trade-profit profit-negative">-$43</span>

</div>

<div class="trade-entry">

<span class="trade-symbol">TSLA</span>

<span>SELL 75 @ $267.45</span>

<span class="trade-profit profit-positive">+$234</span>

</div>

<div class="trade-entry">

<span class="trade-symbol">NVDA</span>

<span>BUY 30 @ $1,234.67</span>

<span class="trade-profit profit-positive">+$156</span>

</div>

</div>

<div class="market-data">

<h4 style="color: #00ff88; margin-bottom: 15px;">Market Watch</h4>

<div class="market-item">

<span>SPY</span>

<span>$428.94</span>

<span class="price-change profit-positive">+0.8%</span>

</div>

<div class="market-item">

<span>QQQ</span>

<span>$374.82</span>

<span class="price-change profit-positive">+1.2%</span>

</div>

<div class="market-item">

<span>IWM</span>

<span>$198.45</span>

<span class="price-change profit-negative">-0.3%</span>

</div>

<div class="market-item">

<span>VIX</span>

<span>$16.23</span>

<span class="price-change profit-negative">-2.1%</span>

</div>

</div>

</div>

</div>

<div class="footer">

<p>AlgoTrade Pro v2.0 | Advanced Algorithmic Trading Platform</p>

<p>⚠️ For educational and demonstration purposes. Not financial advice.</p>

</div>

</div>

<script>

// Trading strategies and their configurations

const strategies = {

momentum: {

name: "Momentum Scalping",

active: true,

pnl: 1247,

trades: 34,

winRate: 0.72,

params: {

lookback: 20,

threshold: 0.02,

maxHoldTime: 300

}

},

arbitrage: {

name: "Arbitrage Hunter",

active: true,

pnl: 892,

trades: 18,

winRate: 0.89,

params: {

minSpread: 0.005,

maxLatency: 50,

exchanges: ['NYSE', 'NASDAQ', 'BATS']

}

},

meanrev: {

name: "Mean Reversion",

active: true,

pnl: 1108,

trades: 22,

winRate: 0.64,

params: {

bollinger\_period: 20,

rsi\_threshold: 30,

profit\_target: 0.015

}

},

statarb: {

name: "Statistical Arbitrage",

active: false,

pnl: 0,

trades: 0,

winRate: 0,

params: {

pairs: ['SPY-QQQ', 'XLF-XLK'],

zscore\_entry: 2.0,

zscore\_exit: 0.5

}

}

};

// Portfolio state

const portfolio = {

totalValue: 125847,

dailyPnL: 3247,

activeTrades: 12,

winRate: 0.684,

cash: 25000,

positions: {}

};

// Mock market data

const marketData = {

'SPY': { price: 428.94, change: 0.008 },

'QQQ': { price: 374.82, change: 0.012 },

'IWM': { price: 198.45, change: -0.003 },

'VIX': { price: 16.23, change: -0.021 }

};

// Initialize the application

function initApp() {

updateTime();

startSimulation();

updateLatency();

setInterval(updateTime, 1000);

setInterval(simulateTrade, 2000);

setInterval(updateLatency, 5000);

setInterval(updateMarketData, 3000);

}

// Update current time with error handling

function updateTime() {

try {

const now = new Date();

const timeString = now.toLocaleTimeString('en-US', {

hour12: false,

hour: '2-digit',

minute: '2-digit',

second: '2-digit'

});

safeUpdateText('current-time', timeString);

} catch (error) {

console.warn('Time update error:', error.message);

safeUpdateText('current-time', '--:--:--');

}

}

// Update latency display with error handling

function updateLatency() {

try {

const latency = (Math.random() \* 2 + 0.1).toFixed(1);

safeUpdateText('latency', latency + 'ms');

} catch (error) {

console.warn('Latency update error:', error.message);

safeUpdateText('latency', '--ms');

}

}

// Enhanced trade simulation with error handling

function simulateTrade() {

try {

const symbols = ['AAPL', 'MSFT', 'GOOGL', 'TSLA', 'NVDA', 'AMD', 'META', 'NFLX'];

const actions = ['BUY', 'SELL'];

const symbol = symbols[Math.floor(Math.random() \* symbols.length)];

const action = actions[Math.floor(Math.random() \* actions.length)];

const quantity = Math.floor(Math.random() \* 100) + 1;

const price = (Math.random() \* 2000 + 100).toFixed(2);

const profit = (Math.random() \* 500 - 100).toFixed(0);

const tradeLog = safeGetElement('trade-log');

if (!tradeLog) {

console.warn('Trade log element not found');

return;

}

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

if (!tradeEntry) {

console.warn('Failed to create trade entry element');

return;

}

tradeEntry.className = 'trade-entry';

tradeEntry.style.opacity = '0';

tradeEntry.style.transform = 'translateY(-10px)';

const profitClass = profit >= 0 ? 'profit-positive' : 'profit-negative';

const profitSign = profit >= 0 ? '+' : '';

tradeEntry.innerHTML = `

<span class="trade-symbol">${symbol}</span>

<span>${action} ${quantity} @ ${price}</span>

<span class="trade-profit ${profitClass}">${profitSign}${profit}</span>

`;

tradeLog.insertBefore(tradeEntry, tradeLog.firstChild);

// Animate entry with error handling

setTimeout(() => {

try {

tradeEntry.style.transition = 'all 0.5s ease';

tradeEntry.style.opacity = '1';

tradeEntry.style.transform = 'translateY(0)';

} catch (animError) {

console.warn('Animation error:', animError.message);

}

}, 100);

// Remove old entries safely

try {

while (tradeLog.children.length > 10) {

const lastChild = tradeLog.lastChild;

if (lastChild) {

tradeLog.removeChild(lastChild);

} else {

break;

}

}

} catch (cleanupError) {

console.warn('Cleanup error:', cleanupError.message);

}

// Update portfolio values

updatePortfolioValues(parseInt(profit) || 0);

} catch (error) {

console.warn('Trade simulation error:', error.message);

}

}

// Update portfolio display values with error handling

function updatePortfolioValues(tradeProfit) {

try {

if (typeof tradeProfit !== 'number' || isNaN(tradeProfit)) {

tradeProfit = 0;

}

portfolio.dailyPnL += tradeProfit;

portfolio.totalValue += tradeProfit;

// Safely update daily P&L

const dailyPnLText = (portfolio.dailyPnL >= 0 ? '+' : '') + '

// Update strategy display

function updateStrategyDisplay() {

// This would update the strategy cards based on current state

// For demo purposes, we'll just show a notification

}

// Show notification

function showNotification(message) {

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

notification.style.cssText = `

position: fixed;

top: 20px;

right: 20px;

background: rgba(0, 255, 136, 0.9);

color: #000;

padding: 15px 20px;

border-radius: 8px;

font-weight: bold;

z-index: 1000;

transform: translateX(300px);

transition: transform 0.3s ease;

`;

notification.textContent = message;

document.body.appendChild(notification);

setTimeout(() => {

notification.style.transform = 'translateX(0)';

}, 100);

setTimeout(() => {

notification.style.transform = 'translateX(300px)';

setTimeout(() => {

document.body.removeChild(notification);

}, 300);

}, 3000);

}

// Advanced trading algorithms

class MomentumStrategy {

constructor(params) {

this.lookback = params.lookback || 20;

this.threshold = params.threshold || 0.02;

this.maxHoldTime = params.maxHoldTime || 300;

this.positions = new Map();

}

analyze(priceData) {

const momentum = this.calculateMomentum(priceData);

return {

signal: momentum > this.threshold ? 'BUY' : momentum < -this.threshold ? 'SELL' : 'HOLD',

confidence: Math.abs(momentum) / this.threshold,

momentum: momentum

};

}

calculateMomentum(prices) {

if (prices.length < this.lookback) return 0;

const recent = prices.slice(-this.lookback);

const sma = recent.reduce((a, b) => a + b) / recent.length;

const currentPrice = prices[prices.length - 1];

return (currentPrice - sma) / sma;

}

}

class ArbitrageStrategy {

constructor(params) {

this.minSpread = params.minSpread || 0.005;

this.maxLatency = params.maxLatency || 50;

this.exchanges = params.exchanges || ['NYSE', 'NASDAQ'];

}

findOpportunities(marketData) {

const opportunities = [];

// Simulate cross-exchange arbitrage detection

for (let symbol in marketData) {

const spread = Math.random() \* 0.01;

if (spread > this.minSpread) {

opportunities.push({

symbol: symbol,

spread: spread,

profit: spread \* marketData[symbol].price,

exchanges: this.exchanges

});

}

}

return opportunities;

}

}

class MeanReversionStrategy {

constructor(params) {

this.bollingerPeriod = params.bollinger\_period || 20;

this.rsiThreshold = params.rsi\_threshold || 30;

this.profitTarget = params.profit\_target || 0.015;

}

analyze(priceData) {

const rsi = this.calculateRSI(priceData);

const bollinger = this.calculateBollinger(priceData);

const currentPrice = priceData[priceData.length - 1];

let signal = 'HOLD';

if (rsi < this.rsiThreshold && currentPrice < bollinger.lower) {

signal = 'BUY';

} else if (rsi > (100 - this.rsiThreshold) && currentPrice > bollinger.upper) {

signal = 'SELL';

}

return {

signal: signal,

rsi: rsi,

bollinger: bollinger,

confidence: Math.abs(50 - rsi) / 50

};

}

calculateRSI(prices, period = 14) {

if (prices.length < period + 1) return 50;

let gains = 0, losses = 0;

for (let i = prices.length - period; i < prices.length; i++) {

const change = prices[i] - prices[i - 1];

if (change > 0) gains += change;

else losses += Math.abs(change);

}

const avgGain = gains / period;

const avgLoss = losses / period;

const rs = avgGain / avgLoss;

return 100 - (100 / (1 + rs));

}

calculateBollinger(prices) {

if (prices.length < this.bollingerPeriod) {

return { upper: 0, middle: 0, lower: 0 };

}

const recent = prices.slice(-this.bollingerPeriod);

const sma = recent.reduce((a, b) => a + b) / recent.length;

const variance = recent.reduce((sum, price) => sum + Math.pow(price - sma, 2), 0) / recent.length;

const stdDev = Math.sqrt(variance);

return {

upper: sma + (2 \* stdDev),

middle: sma,

lower: sma - (2 \* stdDev)

};

}

}

class StatisticalArbitrageStrategy {

constructor(params) {

this.pairs = params.pairs || ['SPY-QQQ'];

this.zscoreEntry = params.zscore\_entry || 2.0;

this.zscoreExit = params.zscore\_exit || 0.5;

this.lookback = 252; // 1 year of trading days

}

analyze(pairData) {

const signals = [];

this.pairs.forEach(pair => {

const [asset1, asset2] = pair.split('-');

if (pairData[asset1] && pairData[asset2]) {

const zscore = this.calculateZScore(pairData[asset1], pairData[asset2]);

let signal = 'HOLD';

if (zscore > this.zscoreEntry) {

signal = 'SELL\_SPREAD'; // Sell asset1, buy asset2

} else if (zscore < -this.zscoreEntry) {

signal = 'BUY\_SPREAD'; // Buy asset1, sell asset2

}

signals.push({

pair: pair,

signal: signal,

zscore: zscore,

confidence: Math.abs(zscore) / this.zscoreEntry

});

}

});

return signals;

}

calculateZScore(prices1, prices2) {

if (prices1.length < this.lookback || prices2.length < this.lookback) return 0;

const recent1 = prices1.slice(-this.lookback);

const recent2 = prices2.slice(-this.lookback);

const spread = recent1.map((price, i) => price - recent2[i]);

const mean = spread.reduce((a, b) => a + b) / spread.length;

const variance = spread.reduce((sum, val) => sum + Math.pow(val - mean, 2), 0) / spread.length;

const stdDev = Math.sqrt(variance);

const currentSpread = prices1[prices1.length - 1] - prices2[prices2.length - 1];

return (currentSpread - mean) / stdDev;

}

}

// Risk management system

class RiskManager {

constructor() {

this.maxPositionSize = 10000;

this.dailyLossLimit = 5000;

this.stopLossPercent = 0.02;

this.currentDrawdown = 0;

this.dailyPnL = 0;

}

checkRisk(trade) {

// Position size check

if (Math.abs(trade.quantity \* trade.price) > this.maxPositionSize) {

return { allowed: false, reason: 'Position size too large' };

}

// Daily loss limit check

if (this.dailyPnL < -this.dailyLossLimit) {

return { allowed: false, reason: 'Daily loss limit reached' };

}

// Stop loss check

if (trade.unrealizedPnL && trade.unrealizedPnL < -this.stopLossPercent \* trade.cost) {

return { allowed: false, reason: 'Stop loss triggered' };

}

return { allowed: true };

}

updateRisk(pnl) {

this.dailyPnL += pnl;

this.currentDrawdown = Math.min(0, this.dailyPnL);

}

}

// Initialize strategies

const momentumStrategy = new MomentumStrategy(strategies.momentum.params);

const arbitrageStrategy = new ArbitrageStrategy(strategies.arbitrage.params);

const meanRevStrategy = new MeanReversionStrategy(strategies.meanrev.params);

const statArbStrategy = new StatisticalArbitrageStrategy(strategies.statarb.params);

const riskManager = new RiskManager();

// Start simulation

function startSimulation() {

console.log('AlgoTrade Pro initialized successfully');

console.log('Strategies loaded:', Object.keys(strategies));

console.log('Risk management active');

}

// Error handling wrapper

function safeExecute(fn, errorMsg = 'Operation failed') {

try {

return fn();

} catch (error) {

console.warn(`${errorMsg}:`, error.message);

return null;

}

}

// Safe DOM element access

function safeGetElement(id) {

try {

const element = document.getElementById(id);

if (!element) {

console.warn(`Element with id '${id}' not found`);

return null;

}

return element;

} catch (error) {

console.warn(`Error accessing element '${id}':`, error.message);

return null;

}

}

// Safe text content update

function safeUpdateText(elementId, text) {

const element = safeGetElement(elementId);

if (element) {

element.textContent = text;

return true;

}

return false;

}

// Initialize the application when page loads

function initAppSafe() {

safeExecute(() => {

updateTime();

startSimulation();

updateLatency();

// Set up intervals with error handling

setInterval(() => safeExecute(updateTime, 'Time update failed'), 1000);

setInterval(() => safeExecute(simulateTrade, 'Trade simulation failed'), 2000);

setInterval(() => safeExecute(updateLatency, 'Latency update failed'), 5000);

setInterval(() => safeExecute(updateMarketData, 'Market data update failed'), 3000);

}, 'App initialization failed');

}

// Initialize when DOM is ready

if (document.readyState === 'loading') {

document.addEventListener('DOMContentLoaded', initAppSafe);

} else {

initAppSafe();

} + Math.abs(portfolio.dailyPnL).toLocaleString();

safeUpdateText('daily-pnl', dailyPnLText);

// Safely update total value

const totalValueText = '

// Update strategy display

function updateStrategyDisplay() {

// This would update the strategy cards based on current state

// For demo purposes, we'll just show a notification

}

// Show notification

function showNotification(message) {

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

notification.style.cssText = `

position: fixed;

top: 20px;

right: 20px;

background: rgba(0, 255, 136, 0.9);

color: #000;

padding: 15px 20px;

border-radius: 8px;

font-weight: bold;

z-index: 1000;

transform: translateX(300px);

transition: transform 0.3s ease;

`;

notification.textContent = message;

document.body.appendChild(notification);

setTimeout(() => {

notification.style.transform = 'translateX(0)';

}, 100);

setTimeout(() => {

notification.style.transform = 'translateX(300px)';

setTimeout(() => {

document.body.removeChild(notification);

}, 300);

}, 3000);

}

// Advanced trading algorithms

class MomentumStrategy {

constructor(params) {

this.lookback = params.lookback || 20;

this.threshold = params.threshold || 0.02;

this.maxHoldTime = params.maxHoldTime || 300;

this.positions = new Map();

}

analyze(priceData) {

const momentum = this.calculateMomentum(priceData);

return {

signal: momentum > this.threshold ? 'BUY' : momentum < -this.threshold ? 'SELL' : 'HOLD',

confidence: Math.abs(momentum) / this.threshold,

momentum: momentum

};

}

calculateMomentum(prices) {

if (prices.length < this.lookback) return 0;

const recent = prices.slice(-this.lookback);

const sma = recent.reduce((a, b) => a + b) / recent.length;

const currentPrice = prices[prices.length - 1];

return (currentPrice - sma) / sma;

}

}

class ArbitrageStrategy {

constructor(params) {

this.minSpread = params.minSpread || 0.005;

this.maxLatency = params.maxLatency || 50;

this.exchanges = params.exchanges || ['NYSE', 'NASDAQ'];

}

findOpportunities(marketData) {

const opportunities = [];

// Simulate cross-exchange arbitrage detection

for (let symbol in marketData) {

const spread = Math.random() \* 0.01;

if (spread > this.minSpread) {

opportunities.push({

symbol: symbol,

spread: spread,

profit: spread \* marketData[symbol].price,

exchanges: this.exchanges

});

}

}

return opportunities;

}

}

class MeanReversionStrategy {

constructor(params) {

this.bollingerPeriod = params.bollinger\_period || 20;

this.rsiThreshold = params.rsi\_threshold || 30;

this.profitTarget = params.profit\_target || 0.015;

}

analyze(priceData) {

const rsi = this.calculateRSI(priceData);

const bollinger = this.calculateBollinger(priceData);

const currentPrice = priceData[priceData.length - 1];

let signal = 'HOLD';

if (rsi < this.rsiThreshold && currentPrice < bollinger.lower) {

signal = 'BUY';

} else if (rsi > (100 - this.rsiThreshold) && currentPrice > bollinger.upper) {

signal = 'SELL';

}

return {

signal: signal,

rsi: rsi,

bollinger: bollinger,

confidence: Math.abs(50 - rsi) / 50

};

}

calculateRSI(prices, period = 14) {

if (prices.length < period + 1) return 50;

let gains = 0, losses = 0;

for (let i = prices.length - period; i < prices.length; i++) {

const change = prices[i] - prices[i - 1];

if (change > 0) gains += change;

else losses += Math.abs(change);

}

const avgGain = gains / period;

const avgLoss = losses / period;

const rs = avgGain / avgLoss;

return 100 - (100 / (1 + rs));

}

calculateBollinger(prices) {

if (prices.length < this.bollingerPeriod) {

return { upper: 0, middle: 0, lower: 0 };

}

const recent = prices.slice(-this.bollingerPeriod);

const sma = recent.reduce((a, b) => a + b) / recent.length;

const variance = recent.reduce((sum, price) => sum + Math.pow(price - sma, 2), 0) / recent.length;

const stdDev = Math.sqrt(variance);

return {

upper: sma + (2 \* stdDev),

middle: sma,

lower: sma - (2 \* stdDev)

};

}

}

class StatisticalArbitrageStrategy {

constructor(params) {

this.pairs = params.pairs || ['SPY-QQQ'];

this.zscoreEntry = params.zscore\_entry || 2.0;

this.zscoreExit = params.zscore\_exit || 0.5;

this.lookback = 252; // 1 year of trading days

}

analyze(pairData) {

const signals = [];

this.pairs.forEach(pair => {

const [asset1, asset2] = pair.split('-');

if (pairData[asset1] && pairData[asset2]) {

const zscore = this.calculateZScore(pairData[asset1], pairData[asset2]);

let signal = 'HOLD';

if (zscore > this.zscoreEntry) {

signal = 'SELL\_SPREAD'; // Sell asset1, buy asset2

} else if (zscore < -this.zscoreEntry) {

signal = 'BUY\_SPREAD'; // Buy asset1, sell asset2

}

signals.push({

pair: pair,

signal: signal,

zscore: zscore,

confidence: Math.abs(zscore) / this.zscoreEntry

});

}

});

return signals;

}

calculateZScore(prices1, prices2) {

if (prices1.length < this.lookback || prices2.length < this.lookback) return 0;

const recent1 = prices1.slice(-this.lookback);

const recent2 = prices2.slice(-this.lookback);

const spread = recent1.map((price, i) => price - recent2[i]);

const mean = spread.reduce((a, b) => a + b) / spread.length;

const variance = spread.reduce((sum, val) => sum + Math.pow(val - mean, 2), 0) / spread.length;

const stdDev = Math.sqrt(variance);

const currentSpread = prices1[prices1.length - 1] - prices2[prices2.length - 1];

return (currentSpread - mean) / stdDev;

}

}

// Risk management system

class RiskManager {

constructor() {

this.maxPositionSize = 10000;

this.dailyLossLimit = 5000;

this.stopLossPercent = 0.02;

this.currentDrawdown = 0;

this.dailyPnL = 0;

}

checkRisk(trade) {

// Position size check

if (Math.abs(trade.quantity \* trade.price) > this.maxPositionSize) {

return { allowed: false, reason: 'Position size too large' };

}

// Daily loss limit check

if (this.dailyPnL < -this.dailyLossLimit) {

return { allowed: false, reason: 'Daily loss limit reached' };

}

// Stop loss check

if (trade.unrealizedPnL && trade.unrealizedPnL < -this.stopLossPercent \* trade.cost) {

return { allowed: false, reason: 'Stop loss triggered' };

}

return { allowed: true };

}

updateRisk(pnl) {

this.dailyPnL += pnl;

this.currentDrawdown = Math.min(0, this.dailyPnL);

}

}

// Initialize strategies

const momentumStrategy = new MomentumStrategy(strategies.momentum.params);

const arbitrageStrategy = new ArbitrageStrategy(strategies.arbitrage.params);

const meanRevStrategy = new MeanReversionStrategy(strategies.meanrev.params);

const statArbStrategy = new StatisticalArbitrageStrategy(strategies.statarb.params);

const riskManager = new RiskManager();

// Start simulation

function startSimulation() {

console.log('AlgoTrade Pro initialized successfully');

console.log('Strategies loaded:', Object.keys(strategies));

console.log('Risk management active');

}

// Error handling wrapper

function safeExecute(fn, errorMsg = 'Operation failed') {

try {

return fn();

} catch (error) {

console.warn(`${errorMsg}:`, error.message);

return null;

}

}

// Safe DOM element access

function safeGetElement(id) {

try {

const element = document.getElementById(id);

if (!element) {

console.warn(`Element with id '${id}' not found`);

return null;

}

return element;

} catch (error) {

console.warn(`Error accessing element '${id}':`, error.message);

return null;

}

}

// Safe text content update

function safeUpdateText(elementId, text) {

const element = safeGetElement(elementId);

if (element) {

element.textContent = text;

return true;

}

return false;

}

// Initialize the application when page loads

function initAppSafe() {

safeExecute(() => {

updateTime();

startSimulation();

updateLatency();

// Set up intervals with error handling

setInterval(() => safeExecute(updateTime, 'Time update failed'), 1000);

setInterval(() => safeExecute(simulateTrade, 'Trade simulation failed'), 2000);

setInterval(() => safeExecute(updateLatency, 'Latency update failed'), 5000);

setInterval(() => safeExecute(updateMarketData, 'Market data update failed'), 3000);

}, 'App initialization failed');

}

// Initialize when DOM is ready

if (document.readyState === 'loading') {

document.addEventListener('DOMContentLoaded', initAppSafe);

} else {

initAppSafe();

} + Math.abs(portfolio.totalValue).toLocaleString();

safeUpdateText('total-value', totalValueText);

// Update P&L color safely

const pnlElement = safeGetElement('daily-pnl');

if (pnlElement) {

pnlElement.className = portfolio.dailyPnL >= 0 ? 'portfolio-value profit-positive' : 'portfolio-value profit-negative';

}

} catch (error) {

console.warn('Portfolio update error:', error.message);

}

}

// Update market data with error handling

function updateMarketData() {

try {

Object.keys(marketData).forEach(symbol => {

try {

const change = (Math.random() - 0.5) \* 0.02;

if (marketData[symbol] && typeof marketData[symbol].price === 'number') {

marketData[symbol].price \*= (1 + change);

marketData[symbol].change = change;

}

} catch (symbolError) {

console.warn(`Market data update error for ${symbol}:`, symbolError.message);

}

});

} catch (error) {

console.warn('Market data update error:', error.message);

}

}

// Enhanced strategy control functions with error handling

function pauseStrategy(strategyId) {

try {

if (strategies[strategyId]) {

strategies[strategyId].active = false;

updateStrategyDisplay();

showNotification(`${strategies[strategyId].name} paused`);

} else {

console.warn(`Strategy ${strategyId} not found`);

}

} catch (error) {

console.warn('Pause strategy error:', error.message);

showNotification('Error pausing strategy');

}

}

function startStrategy(strategyId) {

try {

if (strategies[strategyId]) {

strategies[strategyId].active = true;

updateStrategyDisplay();

showNotification(`${strategies[strategyId].name} started`);

} else {

console.warn(`Strategy ${strategyId} not found`);

}

} catch (error) {

console.warn('Start strategy error:', error.message);

showNotification('Error starting strategy');

}

}

function optimizeStrategy(strategyId) {

try {

if (strategies[strategyId]) {

showNotification(`Optimizing ${strategies[strategyId].name}...`);

setTimeout(() => {

showNotification(`${strategies[strategyId].name} optimized successfully`);

}, 2000);

} else {

console.warn(`Strategy ${strategyId} not found`);

}

} catch (error) {

console.warn('Optimize strategy error:', error.message);

showNotification('Error optimizing strategy');

}

}

function configureStrategy(strategyId) {

try {

if (strategies[strategyId]) {

showNotification(`Opening configuration for ${strategies[strategyId].name}`);

} else {

console.warn(`Strategy ${strategyId} not found`);

}

} catch (error) {

console.warn('Configure strategy error:', error.message);

showNotification('Error configuring strategy');

}

}

// Update strategy display

function updateStrategyDisplay() {

// This would update the strategy cards based on current state

// For demo purposes, we'll just show a notification

}

// Show notification

function showNotification(message) {

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

notification.style.cssText = `

position: fixed;

top: 20px;

right: 20px;

background: rgba(0, 255, 136, 0.9);

color: #000;

padding: 15px 20px;

border-radius: 8px;

font-weight: bold;

z-index: 1000;

transform: translateX(300px);

transition: transform 0.3s ease;

`;

notification.textContent = message;

document.body.appendChild(notification);

setTimeout(() => {

notification.style.transform = 'translateX(0)';

}, 100);

setTimeout(() => {

notification.style.transform = 'translateX(300px)';

setTimeout(() => {

document.body.removeChild(notification);

}, 300);

}, 3000);

}

// Advanced trading algorithms

class MomentumStrategy {

constructor(params) {

this.lookback = params.lookback || 20;

this.threshold = params.threshold || 0.02;

this.maxHoldTime = params.maxHoldTime || 300;

this.positions = new Map();

}

analyze(priceData) {

const momentum = this.calculateMomentum(priceData);

return {

signal: momentum > this.threshold ? 'BUY' : momentum < -this.threshold ? 'SELL' : 'HOLD',

confidence: Math.abs(momentum) / this.threshold,

momentum: momentum

};

}

calculateMomentum(prices) {

if (prices.length < this.lookback) return 0;

const recent = prices.slice(-this.lookback);

const sma = recent.reduce((a, b) => a + b) / recent.length;

const currentPrice = prices[prices.length - 1];

return (currentPrice - sma) / sma;

}

}

class ArbitrageStrategy {

constructor(params) {

this.minSpread = params.minSpread || 0.005;

this.maxLatency = params.maxLatency || 50;

this.exchanges = params.exchanges || ['NYSE', 'NASDAQ'];

}

findOpportunities(marketData) {

const opportunities = [];

// Simulate cross-exchange arbitrage detection

for (let symbol in marketData) {

const spread = Math.random() \* 0.01;

if (spread > this.minSpread) {

opportunities.push({

symbol: symbol,

spread: spread,

profit: spread \* marketData[symbol].price,

exchanges: this.exchanges

});

}

}

return opportunities;

}

}

class MeanReversionStrategy {

constructor(params) {

this.bollingerPeriod = params.bollinger\_period || 20;

this.rsiThreshold = params.rsi\_threshold || 30;

this.profitTarget = params.profit\_target || 0.015;

}

analyze(priceData) {

const rsi = this.calculateRSI(priceData);

const bollinger = this.calculateBollinger(priceData);

const currentPrice = priceData[priceData.length - 1];

let signal = 'HOLD';

if (rsi < this.rsiThreshold && currentPrice < bollinger.lower) {

signal = 'BUY';

} else if (rsi > (100 - this.rsiThreshold) && currentPrice > bollinger.upper) {

signal = 'SELL';

}

return {

signal: signal,

rsi: rsi,

bollinger: bollinger,

confidence: Math.abs(50 - rsi) / 50

};

}

calculateRSI(prices, period = 14) {

if (prices.length < period + 1) return 50;

let gains = 0, losses = 0;

for (let i = prices.length - period; i < prices.length; i++) {

const change = prices[i] - prices[i - 1];

if (change > 0) gains += change;

else losses += Math.abs(change);

}

const avgGain = gains / period;

const avgLoss = losses / period;

const rs = avgGain / avgLoss;

return 100 - (100 / (1 + rs));

}

calculateBollinger(prices) {

if (prices.length < this.bollingerPeriod) {

return { upper: 0, middle: 0, lower: 0 };

}

const recent = prices.slice(-this.bollingerPeriod);

const sma = recent.reduce((a, b) => a + b) / recent.length;

const variance = recent.reduce((sum, price) => sum + Math.pow(price - sma, 2), 0) / recent.length;

const stdDev = Math.sqrt(variance);

return {

upper: sma + (2 \* stdDev),

middle: sma,

lower: sma - (2 \* stdDev)

};

}

}

class StatisticalArbitrageStrategy {

constructor(params) {

this.pairs = params.pairs || ['SPY-QQQ'];

this.zscoreEntry = params.zscore\_entry || 2.0;

this.zscoreExit = params.zscore\_exit || 0.5;

this.lookback = 252; // 1 year of trading days

}

analyze(pairData) {

const signals = [];

this.pairs.forEach(pair => {

const [asset1, asset2] = pair.split('-');

if (pairData[asset1] && pairData[asset2]) {

const zscore = this.calculateZScore(pairData[asset1], pairData[asset2]);

let signal = 'HOLD';

if (zscore > this.zscoreEntry) {

signal = 'SELL\_SPREAD'; // Sell asset1, buy asset2

} else if (zscore < -this.zscoreEntry) {

signal = 'BUY\_SPREAD'; // Buy asset1, sell asset2

}

signals.push({

pair: pair,

signal: signal,

zscore: zscore,

confidence: Math.abs(zscore) / this.zscoreEntry

});

}

});

return signals;

}

calculateZScore(prices1, prices2) {

if (prices1.length < this.lookback || prices2.length < this.lookback) return 0;

const recent1 = prices1.slice(-this.lookback);

const recent2 = prices2.slice(-this.lookback);

const spread = recent1.map((price, i) => price - recent2[i]);

const mean = spread.reduce((a, b) => a + b) / spread.length;

const variance = spread.reduce((sum, val) => sum + Math.pow(val - mean, 2), 0) / spread.length;

const stdDev = Math.sqrt(variance);

const currentSpread = prices1[prices1.length - 1] - prices2[prices2.length - 1];

return (currentSpread - mean) / stdDev;

}

}

// Risk management system

class RiskManager {

constructor() {

this.maxPositionSize = 10000;

this.dailyLossLimit = 5000;

this.stopLossPercent = 0.02;

this.currentDrawdown = 0;

this.dailyPnL = 0;

}

checkRisk(trade) {

// Position size check

if (Math.abs(trade.quantity \* trade.price) > this.maxPositionSize) {

return { allowed: false, reason: 'Position size too large' };

}

// Daily loss limit check

if (this.dailyPnL < -this.dailyLossLimit) {

return { allowed: false, reason: 'Daily loss limit reached' };

}

// Stop loss check

if (trade.unrealizedPnL && trade.unrealizedPnL < -this.stopLossPercent \* trade.cost) {

return { allowed: false, reason: 'Stop loss triggered' };

}

return { allowed: true };

}

updateRisk(pnl) {

this.dailyPnL += pnl;

this.currentDrawdown = Math.min(0, this.dailyPnL);

}

}

// Initialize strategies

const momentumStrategy = new MomentumStrategy(strategies.momentum.params);

const arbitrageStrategy = new ArbitrageStrategy(strategies.arbitrage.params);

const meanRevStrategy = new MeanReversionStrategy(strategies.meanrev.params);

const statArbStrategy = new StatisticalArbitrageStrategy(strategies.statarb.params);

const riskManager = new RiskManager();

// Start simulation

function startSimulation() {

console.log('AlgoTrade Pro initialized successfully');

console.log('Strategies loaded:', Object.keys(strategies));

console.log('Risk management active');

}

// Error handling wrapper

function safeExecute(fn, errorMsg = 'Operation failed') {

try {

return fn();

} catch (error) {

console.warn(`${errorMsg}:`, error.message);

return null;

}

}

// Safe DOM element access

function safeGetElement(id) {

try {

const element = document.getElementById(id);

if (!element) {

console.warn(`Element with id '${id}' not found`);

return null;

}

return element;

} catch (error) {

console.warn(`Error accessing element '${id}':`, error.message);

return null;

}

}

// Safe text content update

function safeUpdateText(elementId, text) {

const element = safeGetElement(elementId);

if (element) {

element.textContent = text;

return true;

}

return false;

}

// Initialize the application when page loads

function initAppSafe() {

safeExecute(() => {

updateTime();

startSimulation();

updateLatency();

// Set up intervals with error handling

setInterval(() => safeExecute(updateTime, 'Time update failed'), 1000);

setInterval(() => safeExecute(simulateTrade, 'Trade simulation failed'), 2000);

setInterval(() => safeExecute(updateLatency, 'Latency update failed'), 5000);

setInterval(() => safeExecute(updateMarketData, 'Market data update failed'), 3000);

}, 'App initialization failed');

}

// Initialize when DOM is ready

if (document.readyState === 'loading') {

document.addEventListener('DOMContentLoaded', initAppSafe);

} else {

initAppSafe();

}