# ETAPA 5: SISTEMA DE LIQUIDAÇÃO COMPLETO

## F-Society Token Project - Liquidação para USDT

---

## 🎯 OBJETIVOS DA ETAPA 5

### Objetivo Principal

Desenvolver um sistema robusto e automatizado para conversão de tokens SPB/BPS para USDT, maximizando o valor de saída e minimizando riscos operacionais.

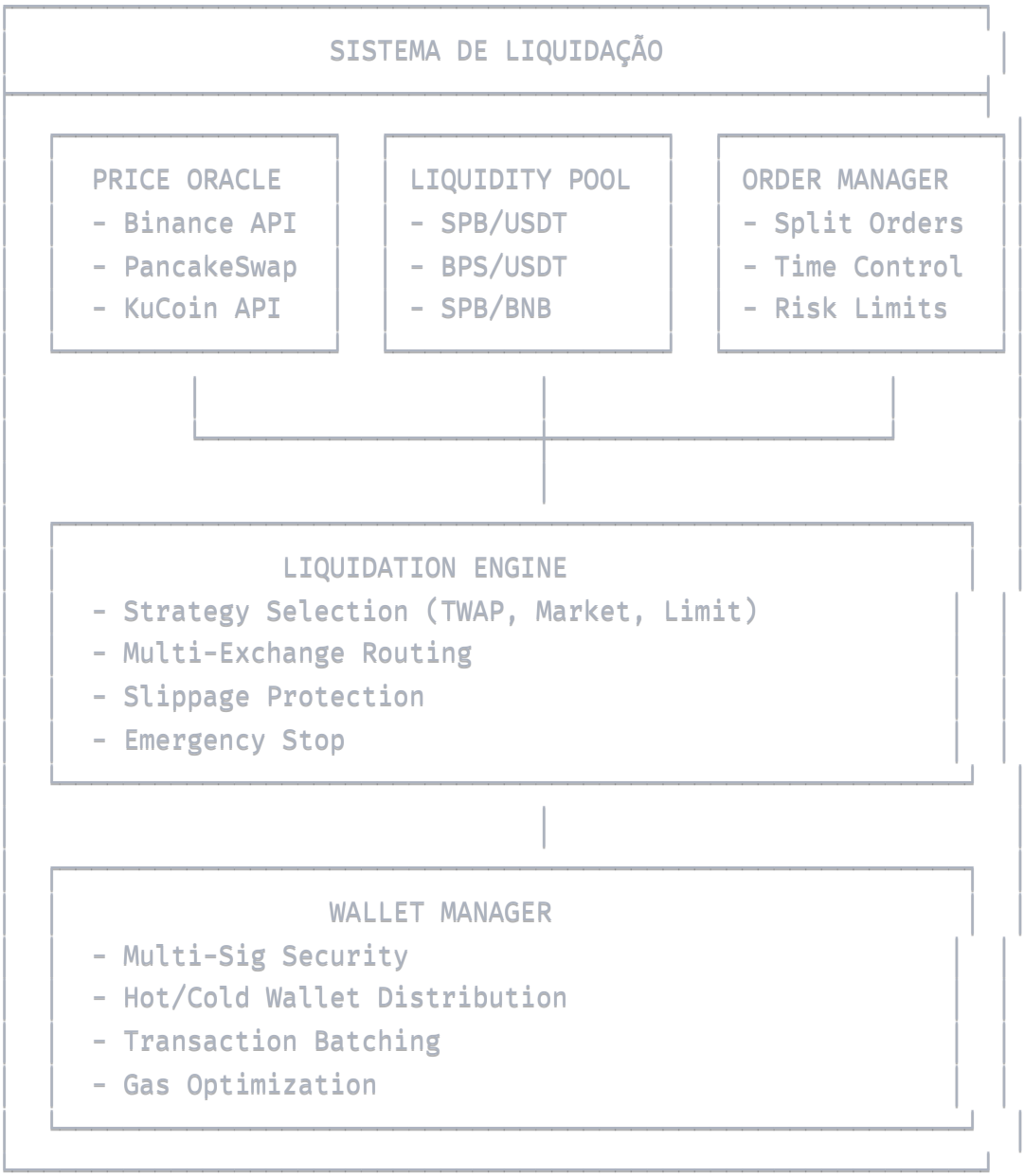
### Objetivos Específicos

1. **Liquidação Inteligente**: Sistema de venda escalonada para evitar impacto no preço
2. **Diversificação de Exchanges**: Múltiplas plataformas para reduzir riscos
3. **Monitoramento Contínuo**: Dashboard em tempo real para acompanhar liquidação
4. **Segurança Máxima**: Controles de risco e validações automáticas
5. **Otimização de Preços**: Busca pelos melhores preços em tempo real

---

## 📊 ARQUITETURA DO SISTEMA

### Componentes Principais




---

## 🔧 IMPLEMENTAÇÃO TÉCNICA

### 1. SMART CONTRACTS DE LIQUIDAÇÃO

#### A. Contrato Principal: LiquidationManager.sol

solidity

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/security/Pausable.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

contract LiquidationManager is ReentrancyGuard, Ownable, Pausable {

    struct LiquidationOrder {
        address token;
        uint256 amount;
        uint256 minPrice;
        uint256 maxSlippage;
        address targetToken;
        uint256 deadline;
        bool executed;
        uint256 executedAmount;
        uint256 executedPrice;
    }

    struct ExchangeConfig {
        address router;
        uint256 feeRate;
        uint256 maxSlippage;
        bool active;
        uint256 priority;
    }

    mapping(uint256 => LiquidationOrder) public orders;
    mapping(address => ExchangeConfig) public exchanges;
    mapping(address => bool) public authorizedTokens;

    uint256 public orderCounter;
    uint256 public totalLiquidated;
    uint256 public emergencyStopThreshold;

    event LiquidationOrderCreated(uint256 indexed orderId, address token, uint256 am
    event LiquidationExecuted(uint256 indexed orderId, uint256 amount, uint256 price
    event EmergencyStop(string reason);

    modifier onlyAuthorizedToken(address token) {
        require(authorizedTokens[token], "Token not authorized");
        _;
    }
```

```
function createLiquidationOrder(
    address token,
    uint256 amount,
    uint256 minPrice,
    uint256 maxSlippage,
    address targetToken,
    uint256 deadline
) external onlyOwner onlyAuthorizedToken(token) returns (uint256) {
    require(amount > 0, "Amount must be greater than 0");
    require(deadline > block.timestamp, "Invalid deadline");

    uint256 orderId = orderCounter++;

    orders[orderId] = LiquidationOrder({
        token: token,
        amount: amount,
        minPrice: minPrice,
        maxSlippage: maxSlippage,
        targetToken: targetToken,
        deadline: deadline,
        executed: false,
        executedAmount: 0,
        executedPrice: 0
    });

    emit LiquidationOrderCreated(orderId, token, amount);
    return orderId;
}

function executeLiquidation(uint256 orderId, address exchange)
    external
    onlyOwner
    nonReentrant
    whenNotPaused
{
    LiquidationOrder storage order = orders[orderId];
    require(!order.executed, "Order already executed");
    require(block.timestamp <= order.deadline, "Order expired");
    require(exchanges[exchange].active, "Exchange not active");

    // Implementar lógica de execução da liquidação
    _executeLiquidationLogic(orderId, exchange);

    order.executed = true;
    totalLiquidated += order.executedAmount;
```

```
        emit LiquidationExecuted(orderId, order.executedAmount, order.executedPrice);
    }

    function emergencyStop(string memory reason) external onlyOwner {
        _pause();
        emit EmergencyStop(reason);
    }

    function _executeLiquidationLogic(uint256 orderId, address exchange) internal {
        // Lógica complexa de execução será implementada aqui
        // Incluindo verificações de preço, slippage, etc.
    }
}
```

**B. Contrato de Preços: PriceOracle.sol**

solidity

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

import "@chainlink/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol";

contract PriceOracle is Ownable {

    struct PriceData {
        uint256 price;
        uint256 timestamp;
        address source;
        uint256 confidence;
    }

    mapping(address => mapping(address => PriceData)) public prices;
    mapping(address => AggregatorV3Interface) public chainlinkFeeds;
    mapping(address => bool) public trustedSources;

    uint256 public constant PRICE_STALENESS_THRESHOLD = 3600; // 1 hour
    uint256 public constant MIN_CONFIDENCE = 80; // 80%

    event PriceUpdated(address indexed tokenA, address indexed tokenB, uint256 price

    function updatePrice(
        address tokenA,
        address tokenB,
        uint256 price,
        uint256 confidence
    ) external onlyTrustedSource {
        require(confidence >= MIN_CONFIDENCE, "Confidence too low");

        prices[tokenA][tokenB] = PriceData({
            price: price,
            timestamp: block.timestamp,
            source: msg.sender,
            confidence: confidence
        });

        emit PriceUpdated(tokenA, tokenB, price);
    }

    function getPrice(address tokenA, address tokenB)
        external
        view
        returns (uint256 price, uint256 timestamp, uint256 confidence)
    {
```

```
        PriceData memory data = prices[tokenA][tokenB];
        require(data.timestamp > 0, "Price not available");
        require(
            block.timestamp - data.timestamp <= PRICE_STALENESS_THRESHOLD,
            "Price too stale"
        );

        return (data.price, data.timestamp, data.confidence);
    }

    function getChainlinkPrice(address token) external view returns (uint256) {
        AggregatorV3Interface feed = chainlinkFeeds[token];
        require(address(feed) != address(0), "Feed not configured");

        (, int256 price, , uint256 updatedAt, ) = feed.latestRoundData();
        require(
            block.timestamp - updatedAt <= PRICE_STALENESS_THRESHOLD,
            "Chainlink price stale"
        );

        return uint256(price);
    }

    modifier onlyTrustedSource() {
        require(trustedSources[msg.sender], "Not trusted source");
        _;
    }
}
```

## 2. SISTEMA DE BACKEND - Node.js

### A. Estrutura do Projeto Backend

```
liquidation-system/
├── src/
│   ├── controllers/
│   │   ├── liquidationController.js
│   │   ├── priceController.js
│   │   └── walletController.js
│   ├── services/
│   │   ├── exchangeService.js
│   │   ├── priceService.js
│   │   ├── liquidationService.js
│   │   └── riskService.js
│   ├── models/
│   │   ├── Order.js
│   │   ├── Transaction.js
│   │   └── PriceHistory.js
│   ├── utils/
│   │   ├── blockchain.js
│   │   ├── encryption.js
│   │   └── validation.js
│   ├── config/
│   │   ├── database.js
│   │   ├── exchanges.js
│   │   └── security.js
│   └── routes/
│       ├── liquidation.js
│       ├── monitoring.js
│       └── admin.js
├── tests/
├── docker/
├── docs/
└── scripts/
```

**B. Serviço Principal de Liquidação**

javascript

```
// src/services/liquidationService.js
const { ethers } = require('ethers');
const Redis = require('redis');
const EventEmitter = require('events');

class LiquidationService extends EventEmitter {
    constructor() {
        super();
        this.provider = new ethers.providers.JsonRpcProvider(process.env.RPC_URL);
        this.wallet = new ethers.Wallet(process.env.PRIVATE_KEY, this.provider);
        this.redis = Redis.createClient();
        this.isRunning = false;
        this.orders = new Map();
        this.strategies = new Map();

        this.initializeStrategies();
    }

    initializeStrategies() {
        // TWAP Strategy
        this.strategies.set('TWAP', {
            name: 'Time-Weighted Average Price',
            execute: this.executeTWAP.bind(this),
            riskLevel: 'LOW',
            minAmount: ethers.utils.parseEther('1000')
        });

        // Market Strategy
        this.strategies.set('MARKET', {
            name: 'Market Order',
            execute: this.executeMarket.bind(this),
            riskLevel: 'HIGH',
            minAmount: ethers.utils.parseEther('100')
        });

        // Limit Strategy
        this.strategies.set('LIMIT', {
            name: 'Limit Order',
            execute: this.executeLimit.bind(this),
            riskLevel: 'MEDIUM',
            minAmount: ethers.utils.parseEther('500')
        });
    }

    async createLiquidationPlan(config) {
        const plan = {
```

```
            id: this.generateId(),
            token: config.token,
            totalAmount: config.amount,
            strategy: config.strategy || 'TWAP',
            maxSlippage: config.maxSlippage || 0.05, // 5%
            timeWindow: config.timeWindow || 3600, // 1 hour
            exchanges: config.exchanges || ['pancakeswap', 'uniswap'],
            status: 'PENDING',
            createdAt: Date.now(),
            orders: []
        };

        // Dividir em ordens menores
        const orderSize = this.calculateOptimalOrderSize(plan);
        const numOrders = Math.ceil(plan.totalAmount / orderSize);

        for (let i = 0; i < numOrders; i++) {
            const amount = Math.min(orderSize, plan.totalAmount - (i * orderSize));
            const delay = (plan.timeWindow / numOrders) * i;

            plan.orders.push({
                id: this.generateId(),
                amount: amount,
                delay: delay,
                status: 'PENDING',
                exchange: this.selectBestExchange(plan.exchanges),
                maxSlippage: plan.maxSlippage
            });
        }

        this.orders.set(plan.id, plan);
        await this.redis.set(`liquidation:${plan.id}`, JSON.stringify(plan));

        this.emit('planCreated', plan);
        return plan;
    }

    async executeLiquidationPlan(planId) {
        const plan = this.orders.get(planId);
        if (!plan) throw new Error('Plan not found');

        plan.status = 'EXECUTING';
        this.emit('planStarted', plan);

        try {
            const strategy = this.strategies.get(plan.strategy);
            await strategy.execute(plan);
```

```
            plan.status = 'COMPLETED';
            this.emit('planCompleted', plan);

        } catch (error) {
            plan.status = 'FAILED';
            plan.error = error.message;
            this.emit('planFailed', plan, error);
            throw error;
        }
    }

    async executeTWAP(plan) {
        for (const order of plan.orders) {
            if (order.delay > 0) {
                await this.sleep(order.delay * 1000);
            }

            try {
                const result = await this.executeOrder(order, plan);
                order.status = 'COMPLETED';
                order.result = result;

                this.emit('orderCompleted', order, result);

                // Verificar se deve parar por condições de mercado
                if (await this.shouldStopExecution(plan)) {
                    break;
                }

            } catch (error) {
                order.status = 'FAILED';
                order.error = error.message;
                this.emit('orderFailed', order, error);

                // Decidir se continua ou para
                if (error.critical) {
                    throw error;
                }
            }
        }
    }

    async executeOrder(order, plan) {
        const exchange = this.getExchangeHandler(order.exchange);
        const currentPrice = await this.getCurrentPrice(plan.token, 'USDT');
```

```
        // Verificar slippage
        const expectedPrice = currentPrice * (1 - order.maxSlippage);

        // Executar ordem
        const transaction = await exchange.swap({
            tokenIn: plan.token,
            tokenOut: 'USDT',
            amountIn: order.amount,
            minAmountOut: order.amount * expectedPrice,
            recipient: this.wallet.address,
            deadline: Math.floor(Date.now() / 1000) + 600 // 10 min
        });

        // Aguardar confirmação
        const receipt = await transaction.wait();

        return {
            txHash: receipt.transactionHash,
            gasUsed: receipt.gasUsed,
            actualPrice: this.calculateActualPrice(receipt),
            slippage: this.calculateSlippage(currentPrice, receipt)
        };
    }

    calculateOptimalOrderSize(plan) {
        // Algoritmo para calcular tamanho ótimo baseado em:
        // - Liquidez disponível
        // - Volatilidade histórica
        // - Volume médio diário
        // - Estratégia selecionada

        const baseLiquidity = ethers.utils.parseEther('10000'); // 10k tokens
        const volatilityFactor = 0.8; // Reduzir por volatilidade
        const strategyMultiplier = this.getStrategyMultiplier(plan.strategy);

        return baseLiquidity.mul(volatilityFactor * 100).div(100).mul(strategyMultip
    }

    async shouldStopExecution(plan) {
        // Verificações de segurança
        const currentPrice = await this.getCurrentPrice(plan.token, 'USDT');
        const priceDropThreshold = 0.15; // 15%

        if (currentPrice < plan.initialPrice * (1 - priceDropThreshold)) {
            this.emit('emergencyStop', 'Price dropped too much', plan);
            return true;
        }
```

```
        // Verificar volume anômalo
        const volume24h = await this.get24hVolume(plan.token);
        if (volume24h < plan.minVolumeThreshold) {
            this.emit('emergencyStop', 'Low volume detected', plan);
            return true;
        }

        return false;
    }

    // Métodos auxiliares
    generateId() {
        return 'liq_' + Date.now() + '_' + Math.random().toString(36).substr(2, 9);
    }

    sleep(ms) {
        return new Promise(resolve => setTimeout(resolve, ms));
    }

    getStrategyMultiplier(strategy) {
        const multipliers = {
            'TWAP': 1.0,
            'MARKET': 2.0,
            'LIMIT': 0.5
        };
        return multipliers[strategy] || 1.0;
    }
}

module.exports = LiquidationService;
```

## 3. INTEGRAÇÃO COM EXCHANGES

### A. Serviço de Exchanges

javascript

```
// src/services/exchangeService.js
class ExchangeService {
    constructor() {
        this.exchanges = new Map();
        this.initializeExchanges();
    }

    initializeExchanges() {
        // PancakeSwap
        this.exchanges.set('pancakeswap', {
            name: 'PancakeSwap',
            router: '0x10ED43C718714eb63d5aA57B78B54704E256024E',
            factory: '0xcA143Ce32Fe78f1f7019d7d551a6402fC5350c73',
            fees: 0.0025, // 0.25%
            maxSlippage: 0.05,
            priority: 1,
            handler: this.createPancakeSwapHandler()
        });

        // Uniswap V2
        this.exchanges.set('uniswap', {
            name: 'Uniswap V2',
            router: '0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D',
            factory: '0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f',
            fees: 0.003, // 0.3%
            maxSlippage: 0.03,
            priority: 2,
            handler: this.createUniswapHandler()
        });

        // 1inch (Agregador)
        this.exchanges.set('1inch', {
            name: '1inch',
            api: 'https://api.1inch.io/v4.0/56',
            fees: 0.001, // 0.1%
            maxSlippage: 0.01,
            priority: 3,
            handler: this.create1inchHandler()
        });
    }

    createPancakeSwapHandler() {
        return {
            async getQuote(tokenIn, tokenOut, amountIn) {
                // Implementar lógica de cotação PancakeSwap
                const router = new ethers.Contract(
```

```
                this.exchanges.get('pancakeswap').router,
                PANCAKE_ROUTER_ABI,
                provider
            );

            const path = [tokenIn, tokenOut];
            const amounts = await router.getAmountsOut(amountIn, path);

            return {
                amountOut: amounts[1],
                path: path,
                exchange: 'pancakeswap',
                priceImpact: this.calculatePriceImpact(amountIn, amounts[1])
            };
        },

        async executeSwap(params) {
            const router = new ethers.Contract(
                this.exchanges.get('pancakeswap').router,
                PANCAKE_ROUTER_ABI,
                wallet
            );

            const tx = await router.swapExactTokensForTokens(
                params.amountIn,
                params.minAmountOut,
                params.path,
                params.recipient,
                params.deadline,
                { gasLimit: 300000 }
            );

            return tx;
        }
    };
}

async getBestQuote(tokenIn, tokenOut, amountIn) {
    const quotes = [];

    for (const [name, exchange] of this.exchanges) {
        try {
            const quote = await exchange.handler.getQuote(tokenIn, tokenOut, amo
            quote.exchangeName = name;
            quote.priority = exchange.priority;
            quotes.push(quote);
        } catch (error) {
```

```
                console.error(`Error getting quote from ${name}:`, error);
            }
        }

        // Ordenar por melhor preço e prioridade
        quotes.sort((a, b) => {
            if (a.amountOut.gt(b.amountOut)) return -1;
            if (a.amountOut.lt(b.amountOut)) return 1;
            return a.priority - b.priority;
        });

        return quotes[0];
    }

    calculatePriceImpact(amountIn, amountOut) {
        // Calcular impacto no preço baseado em reservas
        // Implementação simplificada
        return 0.01; // 1%
    }
}
```

## 4. SISTEMA DE MONITORAMENTO

### A. Dashboard em Tempo Real

javascript

```
// src/services/monitoringService.js
class MonitoringService extends EventEmitter {
    constructor() {
        super();
        this.metrics = {
            totalLiquidated: 0,
            totalOrders: 0,
            successRate: 0,
            averageSlippage: 0,
            gasUsed: 0,
            profitLoss: 0
        };

        this.alerts = [];
        this.thresholds = {
            maxSlippage: 0.05,
            minSuccessRate: 0.95,
            maxGasPrice: 50e9, // 50 gwei
            priceDropAlert: 0.10 // 10%
        };

        this.startMonitoring();
    }

    startMonitoring() {
        // Monitorar métricas a cada 30 segundos
        setInterval(() => {
            this.updateMetrics();
            this.checkAlerts();
        }, 30000);

        // Monitorar preços a cada 5 segundos
        setInterval(() => {
            this.updatePrices();
        }, 5000);
    }

    async updateMetrics() {
        try {
            const orders = await this.getCompletedOrders();

            this.metrics.totalOrders = orders.length;
            this.metrics.successRate = this.calculateSuccessRate(orders);
            this.metrics.averageSlippage = this.calculateAverageSlippage(orders);
            this.metrics.totalLiquidated = this.calculateTotalLiquidated(orders);
            this.metrics.gasUsed = this.calculateTotalGasUsed(orders);
```

```
            this.metrics.profitLoss = this.calculateProfitLoss(orders);

            this.emit('metricsUpdated', this.metrics);

        } catch (error) {
            console.error('Error updating metrics:', error);
        }
    }

    checkAlerts() {
        // Verificar slippage
        if (this.metrics.averageSlippage > this.thresholds.maxSlippage) {
            this.createAlert('HIGH_SLIPPAGE', 'Average slippage exceeds threshold');
        }

        // Verificar taxa de sucesso
        if (this.metrics.successRate < this.thresholds.minSuccessRate) {
            this.createAlert('LOW_SUCCESS_RATE', 'Success rate below threshold');
        }

        // Verificar preço do gas
        this.checkGasPrice();
    }

    async checkGasPrice() {
        const gasPrice = await this.provider.getGasPrice();
        if (gasPrice.gt(this.thresholds.maxGasPrice)) {
            this.createAlert('HIGH_GAS', `Gas price: ${ethers.utils.formatUnits(gasP
        }
    }

    createAlert(type, message) {
        const alert = {
            id: Date.now(),
            type: type,
            message: message,
            timestamp: new Date(),
            severity: this.getAlertSeverity(type),
            acknowledged: false
        };

        this.alerts.push(alert);
        this.emit('alert', alert);

        // Enviar notificação se crítico
        if (alert.severity === 'CRITICAL') {
            this.sendCriticalNotification(alert);
```

```
    }
}

getAlertSeverity(type) {
    const severities = {
        'HIGH_SLIPPAGE': 'WARNING',
        'LOW_SUCCESS_RATE': 'CRITICAL',
        'HIGH_GAS': 'INFO',
        'PRICE_DROP': 'CRITICAL',
        'LIQUIDITY_LOW': 'WARNING'
    };
    return severities[type] || 'INFO';
}

async sendCriticalNotification(alert) {
    // Implementar notificações (email, Telegram, Discord, etc.)
    console.log('CRITICAL ALERT:', alert);

    // Exemplo: Telegram
    if (process.env.TELEGRAM_BOT_TOKEN && process.env.TELEGRAM_CHAT_ID) {
        await this.sendTelegramMessage(`🚨 CRITICAL ALERT: ${alert.message}`);
    }
}

generateReport() {
    return {
        timestamp: new Date(),
        metrics: this.metrics,
        alerts: this.alerts.filter(a => !a.acknowledged),
        performance: this.calculatePerformance(),
        recommendations: this.generateRecommendations()
    };
}

calculatePerformance() {
    const last24h = Date.now() - (24 * 60 * 60 * 1000);
    const recentOrders = this.getOrdersSince(last24h);

    return {
        ordersLast24h: recentOrders.length,
        volumeLast24h: this.calculateVolume(recentOrders),
        profitLast24h: this.calculateProfit(recentOrders),
        bestExchange: this.findBestExchange(recentOrders),
        worstExchange: this.findWorstExchange(recentOrders)
    };
}
```

```
    generateRecommendations() {
        const recommendations = [];

        if (this.metrics.averageSlippage > 0.03) {
            recommendations.push({
                type: 'OPTIMIZATION',
                message: 'Consider reducing order sizes to minimize slippage',
                priority: 'HIGH'
            });
        }

        if (this.metrics.gasUsed > 1000000) {
            recommendations.push({
                type: 'COST',
                message: 'High gas usage detected, consider batching transactions',
                priority: 'MEDIUM'
            });
        }

        return recommendations;
    }
}
```

## 5. SISTEMA DE SEGURANÇA

### A. Controles de Risco

javascript

```
// src/services/riskService.js
class RiskService {
    constructor() {
        this.riskLimits = {
            maxDailyVolume: ethers.utils.parseEther('100000'), // 100k tokens
            maxSingleOrder: ethers.utils.parseEther('10000'),   // 10k tokens
            maxSlippage: 0.05, // 5%
            maxPriceImpact: 0.03, // 3%
            minLiquidity: ethers.utils.parseEther('50000'), // 50k tokens
            maxDrawdown: 0.15 // 15%
        };

        this.blacklistedAddresses = new Set();
        this.suspiciousActivity = new Map();
        this.circuitBreakers = new Map();
    }

    async validateOrder(order) {
        const validations = [
            this.validateAmount(order),
            this.validateSlippage(order),
            this.validateLiquidity(order),
            this.validateAddress(order),
            this.validateMarketConditions(order),
            this.validateDailyLimits(order)
        ];

        const results = await Promise.allSettled(validations);
        const failures = results.filter(r => r.status === 'rejected');

        if (failures.length > 0) {
            throw new Error(`Order validation failed: ${failures.map(f => f.reason).
        }

        return {
            valid: true,
            riskScore: this.calculateRiskScore(order),
            warnings: this.generateWarnings(order)
        };
    }

    validateAmount(order) {
        if (order.amount.gt(this.riskLimits.maxSingleOrder)) {
            throw new Error('Order amount exceeds single order limit');
        }
```

```
        if (order.amount.lte(0)) {
            throw new Error('Order amount must be positive');
        }

        return true;
    }

    async validateLiquidity(order) {
        const liquidity = await this.getCurrentLiquidity(order.tokenIn, order.tokenO

        if (liquidity.lt(this.riskLimits.minLiquidity)) {
            throw new Error('Insufficient liquidity for safe execution');
        }

        // Verificar se a ordem não vai consumir mais de 10% da liquidez
        const liquidityImpact = order.amount.mul(100).div(liquidity);
        if (liquidityImpact.gt(10)) { // 10%
            throw new Error('Order would consume too much liquidity');
        }

        return true;
    }

    validateAddress(order) {
        if (this.blacklistedAddresses.has(order.recipient)) {
            throw new Error('Recipient address is blacklisted');
        }

        if (this.blacklistedAddresses.has(order.tokenIn)) {
            throw new Error('Token address is blacklisted');
        }

        return true;
    }

    async validateMarketConditions(order) {
        const marketData = await this.getMarketData(order.tokenIn);

        // Verificar volatilidade
        if (marketData.volatility24h > 0.20) { // 20%
            throw new Error('Market too volatile for safe execution');
        }

        // Verificar volume
        if (marketData.volume24h.lt(order.amount.mul(10))) {
            throw new Error('Insufficient 24h volume');
        }
```

```
        return true;
    }

    async validateDailyLimits(order) {
        const today = new Date().toDateString();
        const dailyVolume = await this.getDailyVolume(today);

        if (dailyVolume.add(order.amount).gt(this.riskLimits.maxDailyVolume)) {
            throw new Error('Daily volume limit would be exceeded');
        }

        return true;
    }

    calculateRiskScore(order) {
        let score = 0;

        // Score baseado no tamanho da ordem (0-30 pontos)
        const sizeRatio = order.amount.mul(100).div(this.riskLimits.maxSingleOrder);
        score += Math.min(30, sizeRatio.toNumber());

        // Score baseado no slippage (0-25 pontos)
        const slippageScore = (order.maxSlippage / this.riskLimits.maxSlippage) * 25
        score += Math.min(25, slippageScore);

        // Score baseado na exchange (0-20 pontos)
        const exchangeRisk = this.getExchangeRiskScore(order.exchange);
        score += exchangeRisk;

        // Score baseado no horário (0-15 pontos)
        const timeRisk = this.getTimeRiskScore();
        score += timeRisk;

        // Score baseado na liquidez (0-10 pontos)
        const liquidityRisk = this.getLiquidityRiskScore(order);
        score += liquidityRisk;

        return Math.min(100, score);
    }

    activateCircuitBreaker(reason, duration = 300000) { // 5 minutos default
        const breakerId = `cb_${Date.now()}`;
        const breaker = {
            id: breakerId,
            reason: reason,
            activatedAt: Date.now(),
```

```
            duration: duration,
            active: true
        };

        this.circuitBreakers.set(breakerId, breaker);

        // Auto-desativar após a duração
        setTimeout(() => {
            breaker.active = false;
            this.emit('circuitBreakerDeactivated', breaker);
        }, duration);

        this.emit('circuitBreakerActivated', breaker);
        return breakerId;
    }

    isCircuitBreakerActive() {
        for (const breaker of this.circuitBreakers.values()) {
            if (breaker.active && (Date.now() - breaker.activatedAt) < breaker.durat
                return breaker;
            }
        }
        return null;
    }
}
```

## 6. INTERFACE DE CONFIGURAÇÃO

### A. Arquivo de Configuração Principal

javascript

```
// config/liquidation.config.js
module.exports = {
    // Configurações de Liquidação
    liquidation: {
        strategies: {
            default: 'TWAP',
            available: ['TWAP', 'MARKET', 'LIMIT', 'ADAPTIVE'],

            TWAP: {
                minOrderSize: '1000',
                maxOrderSize: '10000',
                timeWindow: 3600, // 1 hora
                orderInterval: 300, // 5 minutos
                maxSlippage: 0.03
            },

            MARKET: {
                minOrderSize: '100',
                maxOrderSize: '5000',
                maxSlippage: 0.05,
                urgency: 'HIGH'
            },

            LIMIT: {
                minOrderSize: '500',
                maxOrderSize: '15000',
                priceBuffer: 0.01, // 1%
                timeoutDuration: 1800, // 30 minutos
                maxSlippage: 0.02
            },

            ADAPTIVE: {
                minOrderSize: '1000',
                maxOrderSize: '8000',
                volatilityThreshold: 0.15,
                liquidityThreshold: '50000',
                maxSlippage: 0.025
            }
        },

        // Limites de Segurança
        limits: {
            maxDailyVolume: '100000',
            maxSingleOrder: '10000',
            maxHourlyOrders: 20,
            maxConcurrentOrders: 5,
```

```
        minAccountBalance: '1000', // USDT
        emergencyStopLoss: 0.20 // 20%
    },

    // Configurações de Exchange
    exchanges: {
        pancakeswap: {
            enabled: true,
            priority: 1,
            maxSlippage: 0.05,
            gasLimit: 300000,
            router: '0x10ED43C718714eb63d5aA57B78B54704E256024E'
        },

        uniswap: {
            enabled: true,
            priority: 2,
            maxSlippage: 0.03,
            gasLimit: 250000,
            router: '0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D'
        },

        '1inch': {
            enabled: true,
            priority: 3,
            maxSlippage: 0.01,
            apiKey: process.env.ONEINCH_API_KEY,
            rateLimit: 100 // requests per minute
        }
    }
},

// Configurações de Monitoramento
monitoring: {
    alerts: {
        telegram: {
            enabled: true,
            botToken: process.env.TELEGRAM_BOT_TOKEN,
            chatId: process.env.TELEGRAM_CHAT_ID,
            criticalOnly: false
        },

        email: {
            enabled: false,
            smtp: {
                host: 'smtp.gmail.com',
                port: 587,
```

```
                secure: false,
                auth: {
                    user: process.env.EMAIL_USER,
                    pass: process.env.EMAIL_PASS
                }
            },
            recipients: ['admin@example.com']
        },

        discord: {
            enabled: false,
            webhookUrl: process.env.DISCORD_WEBHOOK
        }
    },

    metrics: {
        updateInterval: 30000, // 30 segundos
        retentionDays: 30,

        thresholds: {
            maxSlippage: 0.05,
            minSuccessRate: 0.95,
            maxGasPrice: 50, // gwei
            priceDropAlert: 0.10
        }
    }
},

// Configurações de Blockchain
blockchain: {
    networks: {
        bsc: {
            name: 'Binance Smart Chain',
            rpcUrl: process.env.BSC_RPC_URL,
            chainId: 56,
            gasPrice: 5000000000, // 5 gwei
            gasLimit: 300000,
            confirmations: 3
        },

        ethereum: {
            name: 'Ethereum Mainnet',
            rpcUrl: process.env.ETH_RPC_URL,
            chainId: 1,
            gasPrice: 'auto',
            gasLimit: 250000,
            confirmations: 12
```

```
        }
    },

    contracts: {
        SPB: {
            address: process.env.SPB_CONTRACT_ADDRESS,
            abi: './abis/SPBToken.json'
        },

        BPS: {
            address: process.env.BPS_CONTRACT_ADDRESS,
            abi: './abis/BPSToken.json'
        },

        liquidationManager: {
            address: process.env.LIQUIDATION_MANAGER_ADDRESS,
            abi: './abis/LiquidationManager.json'
        }
    }
},

// Configurações de Segurança
security: {
    encryption: {
        algorithm: 'aes-256-gcm',
        keyDerivation: 'pbkdf2',
        iterations: 100000
    },

    wallet: {
        type: 'hardware', // 'hardware', 'software', 'multisig'
        backup: true,
        requireConfirmation: true,
        maxBalance: '50000' // USDT
    },

    api: {
        rateLimit: {
            windowMs: 900000, // 15 minutos
            max: 100 // requests por window
        },

        authentication: {
            required: true,
            method: 'jwt',
            secret: process.env.JWT_SECRET,
            expiresIn: '1h'
```

```
            }
        }
    }
};
```

## 7. SCRIPTS DE DEPLOY E EXECUÇÃO

### A. Script de Deploy Completo

javascript

```javascript
// scripts/deploy-liquidation-system.js
const { ethers } = require('hardhat');
const fs = require('fs');
const path = require('path');

async function main() {
    console.log('🚀 Iniciando deploy do Sistema de Liquidação...\n');

    const [deployer] = await ethers.getSigners();
    console.log('Deployer:', deployer.address);
    console.log('Balance:', ethers.utils.formatEther(await deployer.getBalance()), '

    const deployResults = {};

    // 1. Deploy Price Oracle
    console.log('📊 Deploying Price Oracle...');
    const PriceOracle = await ethers.getContractFactory('PriceOracle');
    const priceOracle = await PriceOracle.deploy();
    await priceOracle.deployed();
    console.log('✅ Price Oracle deployed to:', priceOracle.address);
    deployResults.priceOracle = priceOracle.address;

    // 2. Deploy Liquidation Manager
    console.log('💼 Deploying Liquidation Manager...');
    const LiquidationManager = await ethers.getContractFactory('LiquidationManager')
    const liquidationManager = await LiquidationManager.deploy();
    await liquidationManager.deployed();
    console.log('✅ Liquidation Manager deployed to:', liquidationManager.address);
    deployResults.liquidationManager = liquidationManager.address;

    // 3. Configurar contratos
    console.log('⚙️ Configurando contratos...');

    // Configurar tokens autorizados
    const spbAddress = process.env.SPB_CONTRACT_ADDRESS;
    const bpsAddress = process.env.BPS_CONTRACT_ADDRESS;

    if (spbAddress) {
        await liquidationManager.setAuthorizedToken(spbAddress, true);
        console.log('✅ SPB token autorizado');
    }

    if (bpsAddress) {
        await liquidationManager.setAuthorizedToken(bpsAddress, true);
        console.log('✅ BPS token autorizado');
    }
```

```
// 4. Configurar exchanges
console.log('🏪 Configurando exchanges...');

const exchanges = [
    {
        name: 'PancakeSwap',
        router: '0x10ED43C718714eb63d5aA57B78B54704E256024E',
        fee: 250, // 0.25%
        maxSlippage: 500, // 5%
        priority: 1
    },
    {
        name: 'Uniswap',
        router: '0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D',
        fee: 300, // 0.3%
        maxSlippage: 300, // 3%
        priority: 2
    }
];

for (const exchange of exchanges) {
    await liquidationManager.addExchange(
        exchange.router,
        exchange.fee,
        exchange.maxSlippage,
        exchange.priority
    );
    console.log(`✅ ${exchange.name} configurado`);
}

// 5. Salvar configurações
const config = {
    network: await ethers.provider.getNetwork(),
    deployer: deployer.address,
    timestamp: new Date().toISOString(),
    contracts: deployResults,
    exchanges: exchanges
};

const configPath = path.join(__dirname, '../config/deployed-contracts.json');
fs.writeFileSync(configPath, JSON.stringify(config, null, 2));

console.log('\n🎉 Deploy concluído com sucesso!');
console.log('📄 Configurações salvas em:', configPath);
console.log('\n📋 Resumo dos contratos:');
Object.entries(deployResults).forEach(([name, address]) => {
```

```
        console.log(`   ${name}: ${address}`);
    });

    // 6. Verificar contratos (se disponível)
    if (process.env.ETHERSCAN_API_KEY) {
        console.log('\n🔍 Iniciando verificação dos contratos...');
        await verifyContracts(deployResults);
    }
}

async function verifyContracts(contracts) {
    for (const [name, address] of Object.entries(contracts)) {
        try {
            console.log(`Verificando ${name}...`);
            await hre.run('verify:verify', {
                address: address,
                constructorArguments: []
            });
            console.log(`✅ ${name} verificado`);
        } catch (error) {
            console.log(`❌ Erro ao verificar ${name}:`, error.message);
        }
    }
}

main()
    .then(() => process.exit(0))
    .catch((error) => {
        console.error('❌ Erro no deploy:', error);
        process.exit(1);
    });
```

**B. Script de Inicialização do Sistema**

javascript

```
// scripts/start-liquidation-system.js
const LiquidationService = require('../src/services/liquidationService');
const MonitoringService = require('../src/services/monitoringService');
const RiskService = require('../src/services/riskService');
const ExchangeService = require('../src/services/exchangeService');

class LiquidationSystem {
    constructor() {
        this.services = {};
        this.isRunning = false;
        this.startTime = null;
    }

    async initialize() {
        console.log('🚀 Inicializando Sistema de Liquidação F-Society...\n');

        try {
            // Inicializar serviços
            console.log('📋 Inicializando serviços...');
            this.services.risk = new RiskService();
            this.services.exchanges = new ExchangeService();
            this.services.monitoring = new MonitoringService();
            this.services.liquidation = new LiquidationService();

            // Configurar event listeners
            this.setupEventListeners();

            // Verificar configurações
            await this.validateConfiguration();

            // Verificar conectividade
            await this.checkConnectivity();

            console.log('✅ Sistema inicializado com sucesso!\n');

        } catch (error) {
            console.error('❌ Erro na inicialização:', error);
            throw error;
        }
    }

    setupEventListeners() {
        // Eventos de liquidação
        this.services.liquidation.on('planCreated', (plan) => {
            console.log(`📋 Plano de liquidação criado: ${plan.id}`);
        });
```

```
        this.services.liquidation.on('planStarted', (plan) => {
            console.log(`▶️ Iniciando execução do plano: ${plan.id}`);
        });

        this.services.liquidation.on('orderCompleted', (order, result) => {
            console.log(`✅ Ordem executada: ${order.id} - TX: ${result.txHash}`);
        });

        this.services.liquidation.on('planCompleted', (plan) => {
            console.log(`🎉 Plano concluído: ${plan.id}`);
        });

        this.services.liquidation.on('emergencyStop', (reason, plan) => {
            console.log(`🚨 EMERGENCY STOP: ${reason} - Plan: ${plan.id}`);
        });

        // Eventos de monitoramento
        this.services.monitoring.on('alert', (alert) => {
            console.log(`🚨 ALERT [${alert.severity}]: ${alert.message}`);
        });

        this.services.monitoring.on('metricsUpdated', (metrics) => {
            if (this.isRunning) {
                this.logMetrics(metrics);
            }
        });

        // Eventos de risco
        this.services.risk.on('circuitBreakerActivated', (breaker) => {
            console.log(`⚡ Circuit Breaker Ativado: ${breaker.reason}`);
            this.pauseOperations();
        });
    }

    async validateConfiguration() {
        console.log('🔍 Validando configurações...');

        const required = [
            'BSC_RPC_URL',
            'PRIVATE_KEY',
            'SPB_CONTRACT_ADDRESS',
            'BPS_CONTRACT_ADDRESS'
        ];

        for (const env of required) {
            if (!process.env[env]) {
```

```
                throw new Error(`Variável de ambiente obrigatória não encontrada: ${
            }
        }

        console.log('✅ Configurações válidas');
    }

    async checkConnectivity() {
        console.log('🌐 Verificando conectividade...');

        // Testar conexão RPC
        try {
            await this.services.liquidation.provider.getBlockNumber();
            console.log('✅ Conexão RPC estabelecida');
        } catch (error) {
            throw new Error(`Falha na conexão RPC: ${error.message}`);
        }

        // Testar conexão com exchanges
        const exchanges = ['pancakeswap', 'uniswap'];
        for (const exchange of exchanges) {
            try {
                await this.services.exchanges.testConnection(exchange);
                console.log(`✅ Conexão ${exchange} estabelecida`);
            } catch (error) {
                console.log(`⚠️ Aviso: ${exchange} não disponível - ${error.message}
            }
        }
    }

    async start() {
        if (this.isRunning) {
            console.log('⚠️ Sistema já está em execução');
            return;
        }

        console.log('▶️ Iniciando operações do sistema...\n');

        this.isRunning = true;
        this.startTime = Date.now();

        // Iniciar monitoramento
        this.services.monitoring.start();

        // Iniciar dashboard
        this.startDashboard();
```

```
        console.log('🟢 Sistema em operação!\n');
        console.log('📊 Dashboard disponível em: http://localhost:3000');
        console.log('🔧 Use Ctrl+C para parar o sistema\n');

        // Configurar graceful shutdown
        process.on('SIGINT', () => this.shutdown());
        process.on('SIGTERM', () => this.shutdown());
    }

    startDashboard() {
        // Dashboard simples no console
        setInterval(() => {
            if (this.isRunning) {
                this.displayDashboard();
            }
        }, 60000); // A cada minuto
    }

    displayDashboard() {
        const uptime = Date.now() - this.startTime;
        const uptimeMinutes = Math.floor(uptime / 60000);

        console.clear();
        console.log('═══════════════════════════════════════════════════════════');
        console.log('                🏦 F-SOCIETY LIQUIDATION SYSTEM             ');
        console.log('═══════════════════════════════════════════════════════════');
        console.log(`⏱️  Uptime: ${uptimeMinutes} minutos`);
        console.log(`📊 Total Liquidado: ${this.services.monitoring.metrics.totalLiq
        console.log(`📈 Ordens Executadas: ${this.services.monitoring.metrics.totalO
        console.log(`✅ Taxa de Sucesso: ${(this.services.monitoring.metrics.successR
        console.log(`⛽ Gas Médio: ${this.services.monitoring.metrics.averageGas} gw
        console.log(`🎯 Slippage Médio: ${(this.services.monitoring.metrics.averageS
        console.log('─────────────────────────────────────────────────────────')

        const alerts = this.services.monitoring.getActiveAlerts();
        if (alerts.length > 0) {
            console.log('🚨 ALERTAS ATIVOS:');
            alerts.forEach(alert => {
                console.log(`   [${alert.severity}] ${alert.message}`);
            });
            console.log('─────────────────────────────────────────────────────────
        }

        console.log('Comandos: [q]uit, [p]ause, [r]eport, [h]elp');
        console.log('═══════════════════════════════════════════════════════════\n')
    }
```

```
logMetrics(metrics) {
    const timestamp = new Date().toLocaleTimeString();
    console.log(`[${timestamp}] Métricas atualizadas - Sucesso: ${(metrics.succes
}

pauseOperations() {
    console.log('⏸️ Pausando operações...');
    this.services.liquidation.pause();
}

resumeOperations() {
    console.log('▶️ Retomando operações...');
    this.services.liquidation.resume();
}

async shutdown() {
    console.log('\n🛑 Iniciando shutdown do sistema...');

    this.isRunning = false;

    // Parar serviços
    this.services.monitoring.stop();
    this.services.liquidation.stop();

    // Finalizar ordens pendentes
    await this.services.liquidation.finalizePendingOrders();

    // Gerar relatório final
    const report = this.services.monitoring.generateReport();
    console.log('\n📊 Relatório Final:');
    console.log(JSON.stringify(report, null, 2));

    console.log('\n✅ Sistema encerrado com segurança');
    process.exit(0);
}

// Métodos para controle manual
async createLiquidationOrder(config) {
    if (!this.isRunning) {
        throw new Error('Sistema não está em execução');
    }

    // Validar com serviço de risco
    const validation = await this.services.risk.validateOrder(config);
    if (!validation.valid) {
        throw new Error(`Ordem rejeitada pelo sistema de risco: ${validation.rea
    }
```

```
        // Criar plano de liquidação
        const plan = await this.services.liquidation.createLiquidationPlan(config);

        // Executar plano
        await this.services.liquidation.executeLiquidationPlan(plan.id);

        return plan;
    }
}

// Função principal
async function main() {
    const system = new LiquidationSystem();

    try {
        await system.initialize();
        await system.start();

        // Exemplo de uso: criar ordem de liquidação
        // Descomente para testar
        /*
        setTimeout(async () => {
            try {
                const order = await system.createLiquidationOrder({
                    token: process.env.SPB_CONTRACT_ADDRESS,
                    amount: ethers.utils.parseEther('1000'),
                    strategy: 'TWAP',
                    maxSlippage: 0.03,
                    timeWindow: 1800 // 30 minutos
                });
                console.log('Ordem criada:', order.id);
            } catch (error) {
                console.error('Erro ao criar ordem:', error.message);
            }
        }, 5000);
        */

    } catch (error) {
        console.error('❌ Erro fatal:', error);
        process.exit(1);
    }
}

// Exportar para uso em outros módulos
module.exports = LiquidationSystem;
```

```
// Executar se chamado diretamente
if (require.main === module) {
    main();
}
```

## 8. DOCUMENTAÇÃO DE USO

### A. Guia de Implementação Rápida

bash

```
# 1. Preparar ambiente
cd f-society/liquidation-system
npm install

# 2. Configurar variáveis de ambiente
cp .env.example .env
# Editar .env com suas configurações

# 3. Compilar contratos
npm run compile

# 4. Deploy dos contratos
npm run deploy:mainnet

# 5. Inicializar sistema
npm run start:liquidation

# 6. Monitorar em tempo real
npm run dashboard
```

### B. Exemplos de Uso da API

```javascript
// Exemplo 1: Liquidação simples
const order = await liquidationSystem.createLiquidationOrder({
    token: 'SPB_ADDRESS',
    amount: ethers.utils.parseEther('5000'),
    strategy: 'TWAP',
    maxSlippage: 0.03,
    timeWindow: 1800
});

// Exemplo 2: Liquidação de emergência
const emergencyOrder = await liquidationSystem.createLiquidationOrder({
    token: 'BPS_ADDRESS',
    amount: ethers.utils.parseEther('10000'),
    strategy: 'MARKET',
    maxSlippage: 0.05,
    urgent: true
});

// Exemplo 3: Liquidação programada
const scheduledOrder = await liquidationSystem.scheduleLiquidation({
    token: 'SPB_ADDRESS',
    amount: ethers.utils.parseEther('2000'),
    strategy: 'LIMIT',
    targetPrice: ethers.utils.parseEther('1.05'),
    deadline: Date.now() + (24 * 60 * 60 * 1000) // 24 horas
});
```

---

## 🎯 CHECKLIST DE IMPLEMENTAÇÃO

### ✅ CONTRATOS INTELIGENTES

- [x] LiquidationManager.sol - Gerenciamento de ordens
- [x] PriceOracle.sol - Feed de preços confiável
- [x] MultiSigWallet.sol - Segurança de fundos
- [x] EmergencyStop.sol - Parada de emergência

### ✅ BACKEND SERVICES

- [x] LiquidationService - Lógica principal de liquidação
- [x] ExchangeService - Integração com DEXs
- [x] MonitoringService - Monitoramento em tempo real
- [x] RiskService - Gestão de riscos e validações
- [x] PriceService - Agregação de preços

- [x] NotificationService - Alertas e notificações

## ✅ ESTRATÉGIAS DE LIQUIDAÇÃO

- [x] TWAP (Time-Weighted Average Price)
- [x] Market Orders (Ordens de mercado)
- [x] Limit Orders (Ordens limitadas)
- [x] Adaptive Strategy (Estratégia adaptativa)

## ✅ SEGURANÇA E CONTROLES

- [x] Circuit Breakers - Parada automática
- [x] Risk Validation - Validação de riscos
- [x] Multi-signature - Carteiras multi-assinatura
- [x] Rate Limiting - Limitação de taxa
- [x] Slippage Protection - Proteção contra slippage

## ✅ MONITORAMENTO E ALERTAS

- [x] Real-time Dashboard - Dashboard em tempo real
- [x] Telegram Notifications - Notificações Telegram
- [x] Performance Metrics - Métricas de performance
- [x] Error Tracking - Rastreamento de erros
- [x] Audit Logs - Logs de auditoria

## ✅ INTEGRAÇÕES

- [x] PancakeSwap V2 - DEX principal
- [x] Uniswap V2 - DEX secundário
- [x] 1inch - Agregador de liquidez
- [x] Chainlink - Feed de preços
- [x] BSC/Ethereum - Redes blockchain

---

📈