







# ETAPA 3: Bots de Trading - Society Token Project

## 🎯 Objetivos da Etapa 3

-  Bot de compra automática (BPS → SPB)
-  Bot de venda automática (SPB → BPS)
-  Sistema de simulação de volume
-  Padrões de trading realistas
-  Dashboard de monitoramento
-  Controle de spread e slippage

## 🏗️ Estrutura Expandida

```
societytoken/  
├── bots/  
│   ├── core/  
│   │   ├── TradingBot.js      # Bot principal  
│   │   ├── BuyBot.js         # Bot especializado em compras  
│   │   ├── SellBot.js        # Bot especializado em vendas  
│   │   └── VolumeSimulator.js # Simulador de volume  
│   ├── strategies/  
│   │   ├── RandomStrategy.js # Estratégia aleatória  
│   │   ├── TrendStrategy.js  # Estratégia de tendência  
│   │   └── ScalpStrategy.js   # Estratégia de scalping  
│   ├── config/  
│   │   ├── botConfig.js      # Configurações dos bots  
│   │   └── wallets.js         # Carteiras para os bots  
│   └── utils/  
│       ├── priceCalculator.js # Calculadora de preços  
│       ├── volumeAnalyzer.js  # Analisador de volume  
│       └── logger.js          # Sistema de logs  
├── dashboard/  
│   ├── index.html            # Dashboard de monitoramento  
│   ├── dashboard.js          # Lógica do dashboard  
│   └── dashboard.css         # Estilos do dashboard  
└── scripts/  
    ├── startBots.js          # Iniciar todos os bots  
    ├── stopBots.js           # Parar todos os bots  
    └── monitorBots.js        # Monitorar atividade
```

## 🤖 Bot Principal de Trading

### 1. TradingBot.js - Classe Base





```

// bots/core/TradingBot.js
const { ethers } = require('ethers');
const { EventEmitter } = require('events');

class TradingBot extends EventEmitter {
  constructor(config) {
    super();
    this.config = config;
    this.provider = new ethers.providers.JsonRpcProvider(config.rpcUrl);
    this.wallet = new ethers.Wallet(config.privateKey, this.provider);
    this.router = null;
    this.spbToken = null;
    this.bpsToken = null;
    this.isRunning = false;
    this.tradeCount = 0;
    this.totalVolume = ethers.BigNumber.from(0);
    this.lastTradeTime = 0;
  }

  async initialize() {
    console.log(`🤖 Inicializando bot: ${this.config.name}`);

    // Carregar contratos
    const addresses = this.loadAddresses();

    this.router = new ethers.Contract(
      addresses.router,
      require('../abi/UniswapV2Router02.json'),
      this.wallet
    );

    this.spbToken = new ethers.Contract(
      addresses.spbToken,
      require('../abi/SPBToken.json'),
      this.wallet
    );

    this.bpsToken = new ethers.Contract(
      addresses.bpsToken,
      require('../abi/BPSToken.json'),
      this.wallet
    );

    // Verificar saldos
    await this.checkBalances();
  }
}

```

```

        console.log(`✅ Bot ${this.config.name} inicializado com sucesso`);
    }

    async checkBalances() {
        const spbBalance = await this.spbToken.balanceOf(this.wallet.address);
        const bpsBalance = await this.bpsToken.balanceOf(this.wallet.address);
        const bnbBalance = await this.wallet.getBalance();

        console.log(`💰 Saldos do bot ${this.config.name}:`);
        console.log(`    SPB: ${ethers.utils.formatEther(spbBalance)}`);
        console.log(`    BPS: ${ethers.utils.formatEther(bpsBalance)}`);
        console.log(`    BNB: ${ethers.utils.formatEther(bnbBalance)}`);
    }

    async start() {
        if (this.isRunning) return;

        this.isRunning = true;
        console.log(`🚀 Iniciando bot: ${this.config.name}`);

        this.tradingLoop();
    }

    async stop() {
        this.isRunning = false;
        console.log(`🛑 Parando bot: ${this.config.name}`);
    }

    async tradingLoop() {
        while (this.isRunning) {
            try {
                await this.executeTrade();
                await this.sleep(this.getRandomInterval());
            } catch (error) {
                console.error(`❌ Erro no bot ${this.config.name}:`, error.message);
                await this.sleep(5000); // Esperar 5s em caso de erro
            }
        }
    }

    async executeTrade() {
        const shouldTrade = await this.shouldExecuteTrade();
        if (!shouldTrade) return;

        const tradeType = this.determineTradeType();
        const amount = this.calculateTradeAmount();
    }

```

```

    if (tradeType === 'BUY') {
        await this.buyToken(amount);
    } else {
        await this.sellToken(amount);
    }
}

async buyToken(bpsAmount) {
    try {
        console.log(`📈 ${this.config.name} comprando SPB com ${ethers.utils.formatEther(

        // Aprovar BPS
        await this.bpsToken.approve(this.router.address, bpsAmount);

        const deadline = Math.floor(Date.now() / 1000) + 60 * 10; // 10 minutos
        const path = [this.bpsToken.address, this.spbToken.address];

        const tx = await this.router.swapExactTokensForTokens(
            bpsAmount,
            0, // Accept any amount of SPB
            path,
            this.wallet.address,
            deadline,
            { gasLimit: 300000 }
        );

        await tx.wait();
        this.onTradeExecuted('BUY', bpsAmount, tx.hash);

    } catch (error) {
        console.error(`❌ Erro na compra:`, error.message);
    }
}

async sellToken(spbAmount) {
    try {
        console.log(`📉 ${this.config.name} vendendo ${ethers.utils.formatEther(

        // Aprovar SPB
        await this.spbToken.approve(this.router.address, spbAmount);

        const deadline = Math.floor(Date.now() / 1000) + 60 * 10; // 10 minutos
        const path = [this.spbToken.address, this.bpsToken.address];

        const tx = await this.router.swapExactTokensForTokens(
            spbAmount,
            0, // Accept any amount of BPS

```

```

        path,
        this.wallet.address,
        deadline,
        { gasLimit: 300000 }
    );

    await tx.wait();
    this.onTradeExecuted('SELL', spbAmount, tx.hash);

    } catch (error) {
        console.error(`❌ Erro na venda:`, error.message);
    }
}

onTradeExecuted(type, amount, txHash) {
    this.tradeCount++;
    this.totalVolume = this.totalVolume.add(amount);
    this.lastTradeTime = Date.now();

    const trade = {
        type,
        amount: ethers.utils.formatEther(amount),
        txHash,
        timestamp: new Date().toISOString(),
        bot: this.config.name
    };

    this.emit('trade', trade);
    console.log(`✅ Trade executado: ${type} ${trade.amount} - TX: ${txHash.substring(0, 10)}`);
}

// Métodos para serem sobrescritos pelas estratégias
async shouldExecuteTrade() {
    return Math.random() < 0.3; // 30% chance por ciclo
}

determineTradeType() {
    return Math.random() < 0.5 ? 'BUY' : 'SELL';
}

calculateTradeAmount() {
    const minAmount = ethers.utils.parseEther(this.config.minTradeAmount.toString());
    const maxAmount = ethers.utils.parseEther(this.config.maxTradeAmount.toString());
    const range = maxAmount.sub(minAmount);
    const randomAmount = range.mul(Math.floor(Math.random() * 100)).div(100);
    return minAmount.add(randomAmount);
}

```

```

getRandomInterval() {
    const min = this.config.minInterval * 1000;
    const max = this.config.maxInterval * 1000;
    return Math.floor(Math.random() * (max - min + 1)) + min;
}

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

loadAddresses() {
    const dexAddresses = require('../../dex-addresses.json');
    const tokenAddresses = require('../../token-addresses.json');

    return {
        router: dexAddresses.UniswapV2Router02,
        spbToken: tokenAddresses.SPBTToken,
        bpsToken: tokenAddresses.BPSToken
    };
}

getStats() {
    return {
        name: this.config.name,
        isRunning: this.isRunning,
        tradeCount: this.tradeCount,
        totalVolume: ethers.utils.formatEther(this.totalVolume),
        lastTradeTime: this.lastTradeTime
    };
}
}

module.exports = TradingBot;

```

## 2. BuyBot.js - Bot Especializado em Compras





```

// bots/core/BuyBot.js
const TradingBot = require('./TradingBot');

class BuyBot extends TradingBot {
  constructor(config) {
    super({
      ...config,
      name: config.name || 'BuyBot',
      bias: 'buy' // Tendência de compra
    });
    this.buyPressure = 0.7; // 70% das operações são compras
  }

  async shouldExecuteTrade() {
    // Verificar se tem BPS suficiente para comprar
    const bpsBalance = await this.bpsToken.balanceOf(this.wallet.address);
    const minAmount = ethers.utils.parseEther(this.config.minTradeAmount.toString());

    if (bpsBalance.lt(minAmount)) {
      console.log(`⚠️ ${this.config.name} sem BPS suficiente para comprar`);
      return false;
    }

    // Aumentar atividade durante horários de pico
    const hour = new Date().getHours();
    const isPeakHour = (hour >= 9 && hour <= 11) || (hour >= 14 && hour <= 16) |
    const baseChance = isPeakHour ? 0.4 : 0.2;

    return Math.random() < baseChance;
  }

  determineTradeType() {
    // Bot com viés de compra
    return Math.random() < this.buyPressure ? 'BUY' : 'SELL';
  }

  calculateTradeAmount() {
    // Compras ligeiramente maiores que vendas
    const baseAmount = super.calculateTradeAmount();
    const tradeType = this.determineTradeType();

    if (tradeType === 'BUY') {
      return baseAmount.mul(110).div(100); // 10% maior
    }
    return baseAmount.mul(90).div(100); // 10% menor
  }
}

```

```

async executeTrade() {
  // Verificar tendência do mercado antes de operar
  const marketTrend = await this.analyzeMarketTrend();

  if (marketTrend === 'BEARISH' && Math.random() < 0.8) {
    // 80% chance de comprar em mercado baixista (oportunidade)
    const amount = this.calculateTradeAmount();
    await this.buyToken(amount);
  } else {
    await super.executeTrade();
  }
}

async analyzeMarketTrend() {
  // Análise simples baseada em trades recentes
  const recentTrades = this.getRecentTrades();
  if (recentTrades.length < 3) return 'NEUTRAL';

  const sellCount = recentTrades.filter(t => t.type === 'SELL').length;
  const buyCount = recentTrades.filter(t => t.type === 'BUY').length;

  if (sellCount > buyCount * 1.5) return 'BEARISH';
  if (buyCount > sellCount * 1.5) return 'BULLISH';
  return 'NEUTRAL';
}

getRecentTrades() {
  // Implementar lógica para buscar trades recentes
  // Por enquanto, retorna array vazio
  return [];
}
}

module.exports = BuyBot;

```

### 3. SellBot.js - Bot Especializado em Vendas



```

// bots/core/SellBot.js
const TradingBot = require('./TradingBot');

class SellBot extends TradingBot {
  constructor(config) {
    super({
      ...config,
      name: config.name || 'SellBot',
      bias: 'sell' // Tendência de venda
    });
    this.sellPressure = 0.65; // 65% das operações são vendas
    this.profitTarget = 1.05; // 5% de lucro alvo
  }

  async shouldExecuteTrade() {
    // Verificar se tem SPB suficiente para vender
    const spbBalance = await this.spbToken.balanceOf(this.wallet.address);
    const minAmount = ethers.utils.parseEther(this.config.minTradeAmount.toString());

    if (spbBalance.lt(minAmount)) {
      console.log(`⚠️ ${this.config.name} sem SPB suficiente para vender`);
      return false;
    }

    // Vender mais durante alta volatilidade
    const volatility = await this.calculateVolatility();
    const baseChance = volatility > 0.05 ? 0.5 : 0.25;

    return Math.random() < baseChance;
  }

  determineTradeType() {
    // Bot com viés de venda
    return Math.random() < this.sellPressure ? 'SELL' : 'BUY';
  }

  async calculateVolatility() {
    // Simular cálculo de volatilidade
    // Em implementação real, analisaria preços históricos
    return Math.random() * 0.1; // 0-10%
  }

  calculateTradeAmount() {
    // Vendas em lotes variados para simular realismo
    const baseAmount = super.calculateTradeAmount();
    const multipliers = [0.8, 1.0, 1.2, 1.5, 2.0]; // Diferentes tamanhos
  }
}

```

```

    const multiplier = multipliers[Math.floor(Math.random() * multipliers.length)];

    return baseAmount.mul(Math.floor(multiplier * 100)).div(100);
  }

  async executeTrade() {
    // Estratégia de take profit
    const hasProfit = await this.checkProfitOpportunity();

    if (hasProfit) {
      console.log(`💰 ${this.config.name} realizando lucro`);
      const profitAmount = this.calculateProfitAmount();
      await this.sellToken(profitAmount);
    } else {
      await super.executeTrade();
    }
  }

  async checkProfitOpportunity() {
    // Simular verificação de oportunidade de lucro
    return Math.random() < 0.3; // 30% chance
  }

  calculateProfitAmount() {
    // Calcular quantidade para realizar lucro
    const baseAmount = this.calculateTradeAmount();
    return baseAmount.mul(120).div(100); // 20% maior para maximizar lucro
  }
}

module.exports = SellBot;

```

#### 4. VolumeSimulator.js - Simulador de Volume



```

// bots/core/VolumeSimulator.js
const TradingBot = require('./TradingBot');

class VolumeSimulator extends TradingBot {
  constructor(config) {
    super({
      ...config,
      name: config.name || 'VolumeSimulator'
    });
    this.volumeTarget = ethers.utils.parseEther(config.volumeTarget?.toString());
    this.currentDayVolume = ethers.BigNumber.from(0);
    this.lastVolumeReset = new Date().getDate();
  }

  async start() {
    console.log(`📊 Iniciando simulador de volume - Meta: ${ethers.utils.formatEther(
    await super.start();
  }

  async shouldExecuteTrade() {
    this.checkDailyVolumeReset();

    // Verificar se precisa acelerar para atingir meta
    const hour = new Date().getHours();
    const dayProgress = hour / 24;
    const volumeProgress = this.currentDayVolume.mul(100).div(this.volumeTarget)

    if (volumeProgress < dayProgress * 0.8) {
      // Acelerar se estiver abaixo de 80% da meta
      return Math.random() < 0.8;
    }

    if (volumeProgress > dayProgress * 1.2) {
      // Desacelerar se estiver acima de 120% da meta
      return Math.random() < 0.1;
    }

    return Math.random() < 0.4;
  }

  checkDailyVolumeReset() {
    const currentDay = new Date().getDate();
    if (currentDay !== this.lastVolumeReset) {
      console.log(`📈 Volume diário anterior: ${ethers.utils.formatEther(this.
      this.currentDayVolume = ethers.BigNumber.from(0);
      this.lastVolumeReset = currentDay;
    }
  }

```



```

    }
}

determineTradeType() {
    // Balanceamento para manter liquidez
    const spbBalance = this.spbToken.balanceOf(this.wallet.address);
    const bpsBalance = this.bpsToken.balanceOf(this.wallet.address);

    // Se tem muito mais de um token, tender a vender ele
    if (spbBalance > bpsBalance.mul(2)) {
        return Math.random() < 0.7 ? 'SELL' : 'BUY';
    }

    if (bpsBalance > spbBalance.div(2)) {
        return Math.random() < 0.7 ? 'BUY' : 'SELL';
    }

    return Math.random() < 0.5 ? 'BUY' : 'SELL';
}

calculateTradeAmount() {
    // Variar tamanhos para simular diferentes tipos de traders
    const traderTypes = [
        { name: 'small', weight: 0.6, multiplier: 0.5 }, // 60% pequenos
        { name: 'medium', weight: 0.3, multiplier: 1.0 }, // 30% médios
        { name: 'large', weight: 0.1, multiplier: 3.0 } // 10% grandes
    ];

    const random = Math.random();
    let selectedType = traderTypes[0];
    let cumulativeWeight = 0;

    for (const type of traderTypes) {
        cumulativeWeight += type.weight;
        if (random <= cumulativeWeight) {
            selectedType = type;
            break;
        }
    }

    const baseAmount = super.calculateTradeAmount();
    return baseAmount.mul(Math.floor(selectedType.multiplier * 100)).div(100);
}

onTradeExecuted(type, amount, txHash) {
    super.onTradeExecuted(type, amount, txHash);
    this.currentDayVolume = this.currentDayVolume.add(amount);
}

```

```

    const progress = this.currentDayVolume.mul(100).div(this.volumeTarget).toNuml
    console.log(`📊 Volume diário: ${progress.toFixed(1)}% da meta`);
  }

  getRandomInterval() {
    // Intervalos mais frequentes durante horários de pico
    const hour = new Date().getHours();
    const isPeakHour = (hour >= 9 && hour <= 11) || (hour >= 14 && hour <= 16) |

    if (isPeakHour) {
      return Math.floor(Math.random() * 30000) + 10000; // 10-40s
    } else {
      return Math.floor(Math.random() * 120000) + 60000; // 1-3min
    }
  }

  getStats() {
    const baseStats = super.getStats();
    return {
      ...baseStats,
      dailyVolume: ethers.utils.formatEther(this.currentDayVolume),
      volumeTarget: ethers.utils.formatEther(this.volumeTarget),
      volumeProgress: this.currentDayVolume.mul(100).div(this.volumeTarget).tol
    };
  }
}

module.exports = VolumeSimulator;

```

## 5. Configuração dos Bots



```
// bots/config/botConfig.js
const botConfigs = {
  buyBot1: {
    name: 'BuyBot Alpha',
    privateKey: process.env.BOT_BUYBOT1_PRIVATE_KEY,
    rpcUrl: process.env.RPC_URL || 'http://localhost:8545',
    minTradeAmount: 10,    // 10 tokens mínimo
    maxTradeAmount: 500,   // 500 tokens máximo
    minInterval: 30,       // 30 segundos mínimo
    maxInterval: 180,      // 3 minutos máximo
    enabled: true
  },

  sellBot1: {
    name: 'SellBot Alpha',
    privateKey: process.env.BOT_SELLBOT1_PRIVATE_KEY,
    rpcUrl: process.env.RPC_URL || 'http://localhost:8545',
    minTradeAmount: 15,
    maxTradeAmount: 800,
    minInterval: 45,
    maxInterval: 240,
    enabled: true
  },

  volumeBot1: {
    name: 'Volume Bot 1',
    privateKey: process.env.BOT_VOLUME1_PRIVATE_KEY,
    rpcUrl: process.env.RPC_URL || 'http://localhost:8545',
    minTradeAmount: 5,
    maxTradeAmount: 1000,
    minInterval: 20,
    maxInterval: 300,
    volumeTarget: 5000,    // 5k tokens de volume diário
    enabled: true
  },

  volumeBot2: {
    name: 'Volume Bot 2',
    privateKey: process.env.BOT_VOLUME2_PRIVATE_KEY,
    rpcUrl: process.env.RPC_URL || 'http://localhost:8545',
    minTradeAmount: 8,
    maxTradeAmount: 1200,
    minInterval: 25,
    maxInterval: 200,
    volumeTarget: 7000,    // 7k tokens de volume diário
    enabled: true
  }
}
```

```
    }  
};  
  
module.exports = botConfigs;
```

## Dashboard de Monitoramento

dashboard/index.html



```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>🤖 Society Trading Bots Dashboard</title>
  <link rel="stylesheet" href="dashboard.css">
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
<body>
  <div class="dashboard">
    <header>
      <h1>🤖 Society Trading Bots</h1>
      <div class="controls">
        <button id="startAllBots" class="btn-start">Iniciar Todos</button>
        <button id="stopAllBots" class="btn-stop">Parar Todos</button>
        <button id="refreshData" class="btn-refresh">Atualizar</button>
      </div>
    </header>

    <div class="stats-grid">
      <div class="stat-card">
        <h3>📊 Volume Total</h3>
        <div class="stat-value" id="totalVolume">0</div>
        <div class="stat-label">Tokens negociados</div>
      </div>

      <div class="stat-card">
        <h3>🔄 Trades Hoje</h3>
        <div class="stat-value" id="tradesCount">0</div>
        <div class="stat-label">Operações executadas</div>
      </div>

      <div class="stat-card">
        <h3>🎯 Bots Ativos</h3>
        <div class="stat-value" id="activeBots">0</div>
        <div class="stat-label">De 5 bots configurados</div>
      </div>

      <div class="stat-card">
        <h3>💰 Valor Médio</h3>
        <div class="stat-value" id="avgTradeValue">0</div>
        <div class="stat-label">Tokens por trade</div>
      </div>
    </div>
  </div>
</body>
</html>
```

```

<div class="charts-section">
  <div class="chart-container">
    <h3>📈 Volume por Hora</h3>
    <canvas id="volumeChart"></canvas>
  </div>

  <div class="chart-container">
    <h3>🔄 Tipos de Operação</h3>
    <canvas id="tradeTypeChart"></canvas>
  </div>
</div>

<div class="bots-section">
  <h2>🤖 Status dos Bots</h2>
  <div id="botsStatus"></div>
</div>

<div class="trades-section">
  <h2>📋 Últimas Operações</h2>
  <div class="trades-table">
    <table>
      <thead>
        <tr>
          <th>Horário</th>
          <th>Bot</th>
          <th>Tipo</th>
          <th>Quantidade</th>
          <th>TX Hash</th>
        </tr>
      </thead>
      <tbody id="tradesTableBody">
        <!-- Trades serão inseridos aqui -->
      </tbody>
    </table>
  </div>
</div>
</div>

<script src="dashboard.js"></script>
</body>
</html>

```



## Scripts de Controle

scripts/startBots.js





```

// scripts/startBots.js
const BuyBot = require('../bots/core/BuyBot');
const SellBot = require('../bots/core/SellBot');
const VolumeSimulator = require('../bots/core/VolumeSimulator');
const botConfigs = require('../bots/config/botConfig');

class BotManager {
  constructor() {
    this.bots = new Map();
    this.trades = [];
    this.isRunning = false;
  }

  async startAllBots() {
    console.log('🚀 Iniciando todos os bots...');

    for (const [botId, config] of Object.entries(botConfigs)) {
      if (!config.enabled) continue;

      try {
        let bot;

        if (botId.includes('buy')) {
          bot = new BuyBot(config);
        } else if (botId.includes('sell')) {
          bot = new SellBot(config);
        } else if (botId.includes('volume')) {
          bot = new VolumeSimulator(config);
        }

        await bot.initialize();

        // Escutar eventos de trade
        bot.on('trade', (trade) => {
          this.trades.push(trade);
          this.logTrade(trade);
        });

        this.bots.set(botId, bot);
        await bot.start();

        console.log(`✅ Bot ${config.name} iniciado com sucesso`);
      } catch (error) {
        console.error(`❌ Erro ao iniciar bot ${config.name}:`, error.message);
      }
    }
  }
}

```

```

    this.isRunning = true;
    this.startMonitoring();

    console.log(`🎯 ${this.bots.size} bots iniciados com sucesso!`);
}

async stopAllBots() {
    console.log('🛑 Parando todos os bots...');

    for (const [botId, bot] of this.bots.entries()) {
        await bot.stop();
        console.log(`🛑 Bot ${bot.config.name} parado`);
    }

    this.isRunning = false;
    this.bots.clear();
    console.log('✅ Todos os bots foram parados');
}

startMonitoring() {
    setInterval(() => {
        this.printStats();
    }, 60000); // A cada minuto
}

printStats() {
    console.log(`\n📊 ===== ESTATÍSTICAS DOS BOTS =

```