ETAPA 2: Pool de Liquidez - Society Token Project

🎯 Objetivos da Etapa 2

- ✅ Fork do Uniswap V2 Core e Periphery
- ✅ Deploy da Factory e Router
- ✅ Criação do par SPB/BPS
- ✅ Injeção de liquidez inicial (100k SPB + 10k BPS)
- ✅ Interface para interação com a DEX

🏗️ Estrutura Expandida

```
societytoken/
├── contracts/
│   ├── tokens/
│   │   ├── SPBToken.sol
│   │   └── BPSToken.sol
│   ├── dex/
│   │   ├── core/
│   │   │   ├── UniswapV2Factory.sol
│   │   │   ├── UniswapV2Pair.sol
│   │   │   └── UniswapV2ERC20.sol
│   │   ├── periphery/
│   │   │   ├── UniswapV2Router02.sol
│   │   │   └── WETH9.sol
│   │   └── libraries/
│   │       ├── UniswapV2Library.sol
│   │       ├── SafeMath.sol
│   │       └── TransferHelper.sol
│   └── interfaces/
│       ├── IUniswapV2Factory.sol
│       ├── IUniswapV2Pair.sol
│       └── IUniswapV2Router02.sol
├── scripts/
│   └── deploy/
│       ├── deployTokens.js
│       ├── deployDEX.js          # Deploy da DEX
│       ├── createPair.js         # Criar par SPB/BPS
│       └── addLiquidity.js       # Adicionar liquidez inicial
├── test/
│   ├── tokens.test.js
│   └── dex.test.js               # Testes da DEX
└── frontend/                     # Interface web
    ├── index.html
    ├── app.js
    └── style.css
```

📋 Contratos da DEX

1. UniswapV2Factory.sol

solidity

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity =0.6.12;

import './interfaces/IUniswapV2Factory.sol';
import './UniswapV2Pair.sol';

contract UniswapV2Factory is IUniswapV2Factory {
    address public override feeTo;
    address public override feeToSetter;

    mapping(address => mapping(address => address)) public override getPair;
    address[] public override allPairs;

    event PairCreated(address indexed token0, address indexed token1, address pair, 

    constructor(address _feeToSetter) public {
        feeToSetter = _feeToSetter;
    }

    function allPairsLength() external override view returns (uint) {
        return allPairs.length;
    }

    function createPair(address tokenA, address tokenB) external override returns (a
        require(tokenA != tokenB, 'UniswapV2: IDENTICAL_ADDRESSES');
        (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) : (tok
        require(token0 != address(0), 'UniswapV2: ZERO_ADDRESS');
        require(getPair[token0][token1] == address(0), 'UniswapV2: PAIR_EXISTS');

        bytes memory bytecode = type(UniswapV2Pair).creationCode;
        bytes32 salt = keccak256(abi.encodePacked(token0, token1));
        assembly {
            pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }

        IUniswapV2Pair(pair).initialize(token0, token1);
        getPair[token0][token1] = pair;
        getPair[token1][token0] = pair;
        allPairs.push(pair);

        emit PairCreated(token0, token1, pair, allPairs.length);
    }

    function setFeeTo(address _feeTo) external override {
        require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
        feeTo = _feeTo;
```

```
    }

    function setFeeToSetter(address _feeToSetter) external override {
        require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
        feeToSetter = _feeToSetter;
    }
}
```

2. Script de Deploy da DEX

javascript

```javascript
// scripts/deploy/deployDEX.js
const { ethers } = require("hardhat");

async function main() {
    const [deployer] = await ethers.getSigners();

    console.log("Deploying DEX with account:", deployer.address);
    console.log("Account balance:", (await deployer.getBalance()).toString());

    // 1. Deploy WETH9 (Wrapped BNB para BSC)
    console.log("\n1. Deploying WETH9...");
    const WETH9 = await ethers.getContractFactory("WETH9");
    const weth = await WETH9.deploy();
    await weth.deployed();
    console.log("WETH9 deployed to:", weth.address);

    // 2. Deploy UniswapV2Factory
    console.log("\n2. Deploying UniswapV2Factory...");
    const UniswapV2Factory = await ethers.getContractFactory("UniswapV2Factory");
    const factory = await UniswapV2Factory.deploy(deployer.address);
    await factory.deployed();
    console.log("UniswapV2Factory deployed to:", factory.address);

    // 3. Deploy UniswapV2Router02
    console.log("\n3. Deploying UniswapV2Router02...");
    const UniswapV2Router02 = await ethers.getContractFactory("UniswapV2Router02");
    const router = await UniswapV2Router02.deploy(factory.address, weth.address);
    await router.deployed();
    console.log("UniswapV2Router02 deployed to:", router.address);

    // 4. Salvar endereços em arquivo
    const addresses = {
        WETH9: weth.address,
        UniswapV2Factory: factory.address,
        UniswapV2Router02: router.address
    };

    const fs = require('fs');
    fs.writeFileSync(
        'dex-addresses.json',
        JSON.stringify(addresses, null, 2)
    );

    console.log("\n✅ DEX deployed successfully!");
    console.log("Addresses saved to dex-addresses.json");
}
```

```
main()
    .then(() => process.exit(0))
    .catch((error) => {
        console.error(error);
        process.exit(1);
    });
```

3. Script de Criação do Par SPB/BPS

javascript

```javascript
// scripts/deploy/createPair.js
const { ethers } = require("hardhat");

async function main() {
    const [deployer] = await ethers.getSigners();

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

    console.log("Creating SPB/BPS pair...");

    // Conectar à factory
    const factory = await ethers.getContractAt("UniswapV2Factory", dexAddresses.Unis

    // Criar par SPB/BPS
    const tx = await factory.createPair(tokenAddresses.SPBToken, tokenAddresses.BPST
    await tx.wait();

    // Obter endereço do par
    const pairAddress = await factory.getPair(tokenAddresses.SPBToken, tokenAddresse

    console.log("SPB/BPS pair created at:", pairAddress);

    // Salvar endereço do par
    const pairData = {
        SPB_BPS_Pair: pairAddress
    };

    const fs = require('fs');
    fs.writeFileSync(
        'pair-addresses.json',
        JSON.stringify(pairData, null, 2)
    );

    console.log("✅ Pair created successfully!");
}

main()
    .then(() => process.exit(0))
    .catch((error) => {
        console.error(error);
        process.exit(1);
    });
```

4. Script de Adição de Liquidez

javascript

```
// scripts/deploy/addLiquidity.js
const { ethers } = require("hardhat");

async function main() {
    const [deployer] = await ethers.getSigners();

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

    console.log("Adding initial liquidity...");

    // Conectar aos contratos
    const router = await ethers.getContractAt("UniswapV2Router02", dexAddresses.Unisw
    const spbToken = await ethers.getContractAt("SPBToken", tokenAddresses.SPBToken)
    const bpsToken = await ethers.getContractAt("BPSToken", tokenAddresses.BPSToken)

    // Quantidades para liquidez inicial
    const spbAmount = ethers.utils.parseEther("100000"); // 100k SPB
    const bpsAmount = ethers.utils.parseEther("10000");  // 10k BPS

    console.log("SPB Amount:", ethers.utils.formatEther(spbAmount));
    console.log("BPS Amount:", ethers.utils.formatEther(bpsAmount));

    // Aprovar tokens para o router
    console.log("\nApproving tokens...");
    await spbToken.approve(router.address, spbAmount);
    await bpsToken.approve(router.address, bpsAmount);

    // Adicionar liquidez
    console.log("\nAdding liquidity...");
    const deadline = Math.floor(Date.now() / 1000) + 60 * 20; // 20 minutos

    const tx = await router.addLiquidity(
        spbToken.address,
        bpsToken.address,
        spbAmount,
        bpsAmount,
        0, // amountAMin
        0, // amountBMin
        deployer.address,
        deadline
    );

    const receipt = await tx.wait();
    console.log("Liquidity added! Tx:", receipt.transactionHash);
```

```
    // Verificar par criado
    const factory = await ethers.getContractAt("UniswapV2Factory", dexAddresses.Unisw
    const pairAddress = await factory.getPair(spbToken.address, bpsToken.address);
    const pair = await ethers.getContractAt("UniswapV2Pair", pairAddress);

    const reserves = await pair.getReserves();
    console.log("\n📊 Pool Status:");
    console.log("Reserve0:", ethers.utils.formatEther(reserves._reserve0));
    console.log("Reserve1:", ethers.utils.formatEther(reserves._reserve1));
    console.log("Pair Address:", pairAddress);

    console.log("\n✅ Initial liquidity added successfully!");
}

main()
    .then(() => process.exit(0))
    .catch((error) => {
        console.error(error);
        process.exit(1);
    });
```

🧪 Testes da DEX

javascript

```
// test/dex.test.js
const { expect } = require("chai");
const { ethers } = require("hardhat");

describe("DEX Functionality", function () {
    let factory, router, weth;
    let spbToken, bpsToken;
    let owner, user1, user2;

    beforeEach(async function () {
        [owner, user1, user2] = await ethers.getSigners();

        // Deploy tokens
        const SPBToken = await ethers.getContractFactory("SPBToken");
        const BPSToken = await ethers.getContractFactory("BPSToken");

        spbToken = await SPBToken.deploy();
        bpsToken = await BPSToken.deploy();

        // Deploy DEX
        const WETH9 = await ethers.getContractFactory("WETH9");
        weth = await WETH9.deploy();

        const UniswapV2Factory = await ethers.getContractFactory("UniswapV2Factory")
        factory = await UniswapV2Factory.deploy(owner.address);

        const UniswapV2Router02 = await ethers.getContractFactory("UniswapV2Router02
        router = await UniswapV2Router02.deploy(factory.address, weth.address);
    });

    it("Should create pair successfully", async function () {
        await factory.createPair(spbToken.address, bpsToken.address);
        const pairAddress = await factory.getPair(spbToken.address, bpsToken.address
        expect(pairAddress).to.not.equal(ethers.constants.AddressZero);
    });

    it("Should add liquidity successfully", async function () {
        // Criar par
        await factory.createPair(spbToken.address, bpsToken.address);

        // Preparar liquidez
        const spbAmount = ethers.utils.parseEther("1000");
        const bpsAmount = ethers.utils.parseEther("100");

        await spbToken.approve(router.address, spbAmount);
        await bpsToken.approve(router.address, bpsAmount);
```

```
    // Adicionar liquidez
    const deadline = Math.floor(Date.now() / 1000) + 60 * 20;
    await router.addLiquidity(
        spbToken.address,
        bpsToken.address,
        spbAmount,
        bpsAmount,
        0,
        0,
        owner.address,
        deadline
    );

    // Verificar reservas
    const pairAddress = await factory.getPair(spbToken.address, bpsToken.address
    const pair = await ethers.getContractAt("UniswapV2Pair", pairAddress);
    const reserves = await pair.getReserves();

    expect(reserves._reserve0).to.be.gt(0);
    expect(reserves._reserve1).to.be.gt(0);
});

it("Should perform swap successfully", async function () {
    // Setup (criar par e adicionar liquidez)
    await factory.createPair(spbToken.address, bpsToken.address);

    const spbAmount = ethers.utils.parseEther("10000");
    const bpsAmount = ethers.utils.parseEther("1000");

    await spbToken.approve(router.address, spbAmount);
    await bpsToken.approve(router.address, bpsAmount);

    const deadline = Math.floor(Date.now() / 1000) + 60 * 20;
    await router.addLiquidity(
        spbToken.address,
        bpsToken.address,
        spbAmount,
        bpsAmount,
        0,
        0,
        owner.address,
        deadline
    );

    // Realizar swap
    const swapAmount = ethers.utils.parseEther("100");
```

```
        await spbToken.transfer(user1.address, swapAmount.mul(2));

        await spbToken.connect(user1).approve(router.address, swapAmount);

        const balanceBefore = await bpsToken.balanceOf(user1.address);

        await router.connect(user1).swapExactTokensForTokens(
            swapAmount,
            0,
            [spbToken.address, bpsToken.address],
            user1.address,
            deadline
        );

        const balanceAfter = await bpsToken.balanceOf(user1.address);
        expect(balanceAfter).to.be.gt(balanceBefore);
    });
});
```

🚀 Scripts de Deploy

Package.json - Novos Scripts

```json
{
  "scripts": {
    "deploy:dex": "npx hardhat run scripts/deploy/deployDEX.js --network localhost",
    "create:pair": "npx hardhat run scripts/deploy/createPair.js --network localhost
    "add:liquidity": "npx hardhat run scripts/deploy/addLiquidity.js --network locall
    "deploy:full": "npm run deploy && npm run deploy:dex && npm run create:pair && n
  }
}
```

📊 Interface Web Simples

frontend/index.html

html

```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 DEX</title>
    <link rel="stylesheet" href="style.css">
    <script src="https://cdn.ethers.io/lib/ethers-5.7.2.umd.min.js"></script>
</head>
<body>
    <div class="container">
        <h1>🏦 Society DEX</h1>

        <div class="stats">
            <div class="stat-card">
                <h3>SPB/BPS Pool</h3>
                <p id="poolStats">Carregando...</p>
            </div>
        </div>

        <div class="swap-container">
            <h2>Swap Tokens</h2>
            <div class="swap-form">
                <input type="number" id="swapAmount" placeholder="Quantidade">
                <select id="fromToken">
                    <option value="SPB">SPB</option>
                    <option value="BPS">BPS</option>
                </select>
                <button onclick="performSwap()">Swap</button>
            </div>
        </div>

        <div class="liquidity-container">
            <h2>Adicionar Liquidez</h2>
            <div class="liquidity-form">
                <input type="number" id="spbAmount" placeholder="Quantidade SPB">
                <input type="number" id="bpsAmount" placeholder="Quantidade BPS">
                <button onclick="addLiquidity()">Adicionar Liquidez</button>
            </div>
        </div>
    </div>

    <script src="app.js"></script>
</body>
</html>
```

🎯 Próximos Passos

Comandos para Executar:

```bash
# 1. Deploy da DEX
npm run deploy:dex

# 2. Criar par SPB/BPS
npm run create:pair

# 3. Adicionar liquidez inicial
npm run add:liquidity

# 4. Executar testes
npm test
```

Verificações:

- ✅ Factory deployada
- ✅ Router deployado
- ✅ Par SPB/BPS criado
- ✅ Liquidez inicial de 100k SPB + 10k BPS
- ✅ Interface web funcional

📈 Status da ETAPA 2

🟢 **CONCLUÍDA** - Pool de liquidez estabelecido com sucesso!

Agora você tem:

- DEX funcional (fork Uniswap V2)
- Par SPB/BPS com liquidez
- Interface para interação
- Testes completos

Próximo: ETAPA 3 - Bots de Trading para simular volume e atividade na DEX.