📜 **Dossiê Completo – *Simple DeFi Token***  
Baseado no livro *"Building Full Stack DeFi Applications"* (Samuel Zhou, 2024)

**1. Objetivo do Projeto**

Criar, testar e executar localmente um token ERC20 usando Hardhat, com extensão de funcionalidade para incluir um mecanismo de queima automática ("auto-burn") de 10% nas transferências via função especial.

**2. Estrutura do Projeto**

**2.1 Diretórios Criados**

/src

└── /backend

├── /contracts

│ └── SimpleDeFiToken.sol

├── /scripts

│ └── deploy.js

├── /test

│ ├── SimpleDeFiToken.test.js

│ └── Utils.js

└── hardhat.config.js

**2.2 Principais Arquivos**

**SimpleDeFiToken.sol** → Contrato inteligente ERC20.

**deploy.js** → Script de deploy do contrato na rede local.

**SimpleDeFiToken.test.js** → Testes automatizados do token.

**Utils.js** → Funções auxiliares (toWei e fromWei).

**hardhat.config.js** → Configuração do Hardhat para Solidity 0.8.28 e redes.

**3. Etapas de Execução**

**3.1 Preparação do Ambiente**

Instalar Hardhat:

npm install --save-dev hardhat

Inicializar projeto:

npx hardhat

Selecionar **Create a JavaScript project**.

Instalar dependências:

npm install @openzeppelin/contracts chai ethers

npm install --save-dev @nomicfoundation/hardhat-toolbox

**3.2 Implementação do Contrato Inicial**

Arquivo **src/backend/contracts/SimpleDeFiToken.sol**:

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract SimpleDeFiToken is ERC20 {

constructor() ERC20("Simple DeFi Token", "SDFT") {

\_mint(msg.sender, 1e24); // 1 milhão de tokens (18 casas decimais)

}

}

**3.3 Scripts Utilizados**

**3.3.1 Deploy do contrato**

Arquivo **src/backend/scripts/deploy.js**:

async function main() {

const Token = await ethers.getContractFactory("SimpleDeFiToken");

const token = await Token.deploy();

await token.deployed();

console.log("SimpleDeFiToken deployed to:", token.address);

}

main().catch((error) => {

console.error(error);

process.exitCode = 1;

});

**Execução**

npx hardhat node

npx hardhat run scripts/deploy.js --network localhost

**3.4 Testes Iniciais**

Arquivo **src/backend/test/Utils.js**:

const { ethers } = require("hardhat");

function toWei(amount) {

return ethers.parseUnits(amount.toString(), 18);

}

function fromWei(amount) {

return parseFloat(ethers.formatUnits(amount, 18));

}

module.exports = { toWei, fromWei };

Arquivo **src/backend/test/SimpleDeFiToken.test.js** (versão inicial):

const { expect } = require("chai");

const { ethers } = require("hardhat");

const { toWei } = require("./Utils");

describe("SimpleDeFiToken", () => {

let deployer, addr1, addr2, token;

beforeEach(async () => {

[deployer, addr1, addr2] = await ethers.getSigners();

const tokenFactory = await ethers.getContractFactory("SimpleDeFiToken");

token = await tokenFactory.deploy();

});

it("Should have correct name, symbol and total supply", async () => {

expect(await token.name()).to.equal("Simple DeFi Token");

expect(await token.symbol()).to.equal("SDFT");

expect(await token.totalSupply()).to.equal(toWei(1000000));

});

it("Should transfer token from one to another", async () => {

expect(await token.balanceOf(deployer.address)).to.equal(toWei(1000000));

await token.connect(deployer).transfer(addr1.address, toWei(5));

expect(await token.balanceOf(addr1.address)).to.equal(toWei(5));

expect(await token.balanceOf(deployer.address)).to.equal(toWei(999995));

await expect(

token.connect(addr1).transfer(addr2.address, toWei(10))

).to.be.revertedWithCustomError(token, "ERC20InsufficientBalance");

});

});

Execução:

npx hardhat test

**3.5 Implementação do Auto-Burn**

Alteração no **SimpleDeFiToken.sol**:

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

import "hardhat/console.sol";

contract SimpleDeFiToken is ERC20 {

constructor() ERC20("Simple DeFi Token", "SDFT") {

\_mint(msg.sender, 1e24);

}

function transferWithAutoBurn(address to, uint256 amount) public {

require(balanceOf(msg.sender) >= amount, "Not enough tokens");

uint256 burnAmount = amount / 10;

console.log(

"Burning %s from %s, balance is %s",

burnAmount,

to,

balanceOf(to)

);

\_burn(msg.sender, burnAmount);

\_transfer(msg.sender, to, amount - burnAmount);

}

}

**3.6 Teste do Auto-Burn**

Alteração no **SimpleDeFiToken.test.js**:

it("Should burn token automatically when calling transferWithAutoBurn", async () => {

await token.connect(deployer).transfer(addr1.address, toWei(1));

await token.connect(addr1).transferWithAutoBurn(addr2.address, toWei(1));

const addr2Balance = await token.balanceOf(addr2.address);

expect(addr2Balance).to.equal(toWei(0.9));

});

Execução:

npx hardhat test

**3.7 Resultado Final**

Output final:

SimpleDeFiToken

√ Should have correct name, symbol and total supply

√ Should transfer token from one to another

Burning 100000000000000000 from 0x..., balance is 0

√ Should burn token automatically when calling transferWithAutoBurn

3 passing

**4. Resumo Técnico**

Linguagem: Solidity 0.8.28.

Framework: Hardhat.

Biblioteca ERC20: OpenZeppelin.

Testes: Mocha + Chai.

Rede: Hardhat local.

Novas funcionalidades: transferWithAutoBurn (10% queimados por transação especial).

Cobertura de testes: Nome, símbolo, total supply, transferências normais e transferência com auto-burn.

Se você quiser, posso agora fazer **a segunda parte**, que seria transformar este dossiê em um **guia de execução rápido** passo-a-passo que qualquer pessoa possa seguir sem precisar ler tudo na íntegra. Isso seria útil para reproduzir o projeto em minutos. Quer que eu faça isso agora?