**Passo-a-passo fiel ao projeto**

**0) Pré-requisitos**

* Node.js + npm
* VS Code (ou editor de sua preferência)
* (Opcional p/ testnet) MetaMask e uma conta em um provedor RPC (Infura/MetaMask Developer) + faucet Sepolia

**1) Criar o projeto base React**

npx create-react-app defi-apps

cd defi-apps

O front não foi usado ainda, mas este é o esqueleto do capítulo.

**2) Instalar Hardhat e dependências**

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

npm install @openzeppelin/contracts

npm install chai

**3) Inicializar o Hardhat**

npx hardhat

Escolha criar um projeto básico (pode ser o “empty hardhat.config.js”).  
O arquivo hardhat.config.js ficará no **root** do projeto.

**4) Estrutura de pastas do backend**

Crie pastas para contratos e testes (usamos essa estrutura no projeto):

src/backend/contracts

src/backend/test

scripts

A pasta **scripts** (no root) será usada para o deploy.js.

**5) Configuração do Hardhat**

Edite **hardhat.config.js** (no root) para usar os caminhos que adotamos e (opcional) Sepolia:

require("@nomicfoundation/hardhat-toolbox");

require("dotenv").config();

const SEPOLIA\_API\_URL = process.env.API\_URL; // opcional

const SEPOLIA\_PRIVATE\_KEY = process.env.PRIVATE\_KEY; // opcional

module.exports = {

solidity: "0.8.28",

paths: {

sources: "./src/backend/contracts",

artifacts: "./src/backend/artifacts",

cache: "./src/backend/cache",

tests: "./src/backend/test",

},

networks: {

// opcional (só se for fazer deploy na Sepolia)

sepolia: {

url: SEPOLIA\_API\_URL,

accounts: SEPOLIA\_PRIVATE\_KEY ? [SEPOLIA\_PRIVATE\_KEY] : [],

},

},

};

**6) Contrato Solidity**

Crie **src/backend/contracts/SimpleDeFiToken.sol**.  
Começamos simples e depois adicionamos a função de queima automática (versão final abaixo):

// 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); // 1,000,000 \* 1e18

}

// Queima automática de 10% do valor transferido

function transferWithAutoBurn(address to, uint256 amount) public {

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

uint256 burnAmount = amount / 10; // 10%

\_burn(msg.sender, burnAmount); // queima sai do remetente

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

console.log("Burning %s from %s, balance is %s",

burnAmount, to, balanceOf(to));

}

}

**7) Script de deploy (local)**

Crie **scripts/deploy.js** (no **root** do projeto):

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

async function main() {

const [deployer] = await ethers.getSigners();

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

const token = await factory.deploy();

await token.waitForDeployment();

console.log("Simple DeFi Token Contract Address:", await token.getAddress());

console.log("Deployer:", deployer.address);

console.log(

"Deployer ETH balance:",

(await ethers.provider.getBalance(deployer.address)).toString()

);

}

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

console.error(err);

process.exitCode = 1;

});

**8) Utilitários de teste**

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

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

function toWei(n) {

return ethers.parseUnits(n.toString(), "ether");

}

function fromWei(v) {

return ethers.formatUnits(v, "ether");

}

module.exports = { toWei, fromWei };

**9) Testes automatizados**

Crie **src/backend/test/SimpleDeFiToken.test.js** com os três testes que rodamos:

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

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

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

describe("SimpleDeFiToken", () => {

let deployer, user1, user2, token;

beforeEach(async () => {

[deployer, user1, user2] = await ethers.getSigners();

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

token = await factory.deploy();

await token.waitForDeployment();

});

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(1\_000\_000));

});

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

expect(await token.balanceOf(deployer.address)).to.equal(toWei(1\_000\_000));

await token.connect(deployer).transfer(user1.address, toWei(5));

expect(await token.balanceOf(user1.address)).to.equal(toWei(5));

expect(await token.balanceOf(deployer.address)).to.equal(toWei(999\_995));

await expect(

token.connect(user1).transfer(user2.address, toWei(10))

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

});

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

await token.connect(deployer).transfer(user1.address, toWei(1));

await token.connect(user1).transferWithAutoBurn(user2.address, toWei(1));

expect(fromWei(await token.balanceOf(user1.address))).to.equal("0.9");

expect(fromWei(await token.balanceOf(user2.address))).to.equal("0.9");

const expectedSupply = toWei(1\_000\_000 - 0.1);

expect(await token.totalSupply()).to.equal(expectedSupply);

});

});

**10) Rodar testes (VM embutida do Hardhat)**

npx hardhat test

Resultado esperado: **3 passing**.

**11) Rede local (node + deploy + console)**

1. Suba o node local:

npx hardhat node

1. Em **outro terminal**, faça o deploy:

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

1. Abra o console:

npx hardhat console --network localhost

No console, para interagir:

// use o endereço impresso no deploy (ex.: 0x5FbD...)

const contract = await ethers.getContractAt(

"SimpleDeFiToken",

"0xSEU\_ENDERECO\_DO\_DEPLOY"

);

await contract.name(); // "Simple DeFi Token"

await contract.symbol(); // "SDFT"

(await contract.totalSupply()).toString();

**12) (Opcional) Sepolia**

Se quiser preparar o deploy em Sepolia (como no livro), crie **.env** no root:

API\_URL=https://sepolia.infura.io/v3/SUA\_KEY

PRIVATE\_KEY=SUA\_CHAVE\_PRIVADA\_METAMASK

Depois:

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

É preciso ter **ETH de teste** (faucet). Não fizemos o deploy por falta de saldo de faucet, mas a configuração ficou pronta.