

# Projeto de Implementação: Implementando Tokens no Ethereum

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ  
CÂMPUS CAMPO MOURÃO  
DIRETORIA DE GRADUAÇÃO E EDUCAÇÃO PROFISSIONAL  
CURSO DE CIÊNCIA DA COMPUTAÇÃO

Aluno: Gabriel Chiari Dias

RA: 2515571

## Resumo

O trabalho envolveu o desenvolvimento de um contrato inteligente no Ethereum utilizando o padrão ERC-20 para criar e gerenciar tokens digitais. Foram utilizadas ferramentas como Truffle, Ganache e MetaMask para compilar, implantar e testar o contrato em uma blockchain local.

## 1 Introdução

Blockchain é uma tecnologia descentralizada que armazena dados de maneira distribuída em uma rede de computadores. Cada conjunto de dados é chamado de "bloco", e esses blocos são interligados em uma sequência, formando uma cadeia (daí o nome "blockchain"). A principal vantagem do blockchain é sua segurança, pois as transações são imutáveis, transparentes e protegidas contra fraudes. Além disso, o blockchain elimina a necessidade de intermediários, permitindo transações diretas entre as partes. O Ethereum é uma plataforma blockchain que possibilita a criação e execução de contratos inteligentes (smart contracts), que são programas autônomos e autoexecutáveis, que não dependem de intermediários. O Ethereum vai além de apenas armazenar transações financeiras (como o Bitcoin) e oferece a capacidade de executar aplicativos descentralizados (dApps). Um dos conceitos mais importantes no Ethereum é a "tokenização", ou seja, a criação de ativos digitais chamados "tokens", que podem representar qualquer coisa de valor, como moedas, ações, propriedade intelectual, etc.

Contratos inteligentes são programas autoexecutáveis com regras predefinidas que podem ser executadas na blockchain. No Ethereum, esses contratos são escritos em Solidity, uma linguagem de programação de alto nível baseada em JavaScript. A principal vantagem dos contratos inteligentes é que, uma vez implantados, não podem ser alterados, o que garante confiança e segurança nas transações. A tokenização, por sua vez, refere-se à criação de tokens digitais que podem representar qualquer ativo. O Ethereum suporta diferentes padrões de tokens, sendo o mais popular o **ERC-20**, que define uma interface padrão para a criação de tokens fungíveis, ou seja, tokens que podem ser trocados por outros de mesmo valor. Um exemplo comum de token ERC-20 é o **USDT**, um stablecoin que mantém seu valor atrelado ao dólar.

## 2 Contratos Inteligentes

Solidity é a linguagem de programação principal utilizada para escrever contratos inteligentes na plataforma Ethereum. Ela foi criada especificamente para a Ethereum Virtual Machine (EVM), que é o ambiente de execução que interpreta os contratos. Solidity é fortemente tipada, e sua sintaxe é baseada em JavaScript, o que facilita para desenvolvedores com experiência em web development aprenderem.

### Funções do ERC-20 Implementadas

No desenvolvimento do contrato inteligente, optou-se por implementar o padrão ERC20, que define uma série de funções obrigatórias para criar tokens fungíveis. As principais funções do ERC-20 são:

1. `totalSupply()` : Retorna o número total de tokens emitidos.
2. `balanceOf(address)` : Retorna o saldo de tokens de uma carteira.
3. `transfer(address, uint256)` : Permite que um usuário envie tokens para outro endereço.
4. `approve(address, uint256)` : Permite que um usuário aprove outro endereço a gastar uma quantidade específica de tokens em seu nome.
5. `transferFrom(address, address, uint256)` : Permite que um endereço transfira tokens de outro endereço, caso tenha sido aprovado previamente.
6. `allowance(address, address)` : Retorna a quantidade de tokens que um endereço está autorizado a gastar de outro.

Essas funções são essenciais para garantir que os tokens possam ser transferidos, gerenciados e manipulados de maneira padronizada e segura.

### 3 Ferramentas de Desenvolvimento

O *Ganache* é uma ferramenta de desenvolvimento para Ethereum que simula uma blockchain privada local, permitindo que os desenvolvedores testem seus contratos inteligentes sem a necessidade de gastar Ether (ETH) real. Ele proporciona uma interface gráfica intuitiva para visualizar transações, blocos e contratos implantados. Durante o desenvolvimento, o Ganache foi fundamental para realizar testes e simulações de transações de forma rápida e segura.

O *Truffle* é um framework de desenvolvimento que facilita o desenvolvimento de contratos inteligentes no Ethereum. Ele oferece funcionalidades para compilar, implantar, testar e gerenciar contratos de forma simples e estruturada. O Truffle é amplamente utilizado para realizar a automação de testes de contratos inteligentes, como foi feito durante o processo de desenvolvimento do token ERC-20.

O *MetaMask* é uma carteira digital que permite aos usuários interagir com a blockchain do Ethereum diretamente pelo navegador. Ele foi usado para interagir com o contrato inteligente depois de implantado, permitindo que os usuários realizassem transações e acessassem suas carteiras de tokens ERC-20 de forma prática e segura.

### 4 Especificação e Desenvolvimento

O contrato inteligente desenvolvido segue o padrão ERC-20 e inclui as seguintes funcionalidades:

- **Emissão de tokens:** O contrato permite a criação de uma quantidade inicial de tokens que é atribuída ao criador do contrato.
- **Transferência de tokens:** Os usuários podem transferir tokens entre si, de acordo com o saldo disponível em suas carteiras.
- **Aprovação e delegação:** Usuários podem autorizar outros endereços a transferirem seus tokens em seu nome, um recurso essencial para integração com outros contratos ou exchanges descentralizadas.

A implementação foi realizada em Solidity no arquivo *MyToken.sol* (pasta contracts), e o código do contrato é apresentado no Código 1.

```

1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.28;
3
4 import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
5
6 contract MyToken is ERC20 {
7     constructor(
8         string memory name,
9         string memory symbol,
10        uint256 initialSupply
11    ) ERC20(name, symbol) {
12
13        uint256 totalSupply = initialSupply * (10 ** decimals());
14        require(totalSupply <= type(uint256).max, "Initial supply exceeds
15            uint256 limit");
16
17        _mint(msg.sender, totalSupply);
18    }
19 }

```

Código 1: MyToken.sol

## Migração

O arquivo *2-deploy-contracts.js* é um script de migração essencial no framework **Truffle**, utilizado para automatizar a implantação do contrato inteligente MyToken na blockchain. Nele, os parâmetros do token, como nome (name), símbolo (symbol) e suprimento inicial (initialSupply), são definidos. O token foi configurado com o nome "MyToken", símbolo "MTK" e um suprimento inicial de **1000 unidades**. E o código do *deploy* é mostrado no Código 2.

```

1 const MyToken = artifacts.require("MyToken");
2
3 module.exports = async function (deployer) {
4     const name = "MyToken"; // Nome do token
5     const symbol = "MTK"; // Símbolo do token
6     const initialSupply = 1000; // Suprimento inicial reduzido
7
8     await deployer.deploy(MyToken, name, symbol, initialSupply);
9 };

```

Código 2: 2-deploy-contracts.js

## Truffle Config:

O arquivo *truffle-config.js* configura o ambiente de desenvolvimento para contratos inteligentes. Define a rede de desenvolvimento local com o Ganache (host: 127.0.0.1, porta 7545) e especifica limites como gas (8.000.000) e gasPrice (20 Gwei) para transações. O compilador Solidity usa a versão 0.8.28 com otimizações ativadas para 200 execuções, equilibrando custo e tamanho do contrato. e o código do contrato é apresentado no Código 3.

```

1 module.exports = {
2   networks: {
3     // Rede de desenvolvimento usando o Ganache
4     development: {
5       host: "127.0.0.1",
6       port: 7545,
7       network_id: "*",
8       gas: 8000000,
9       gasPrice: 200000000000
10    },
11  },
12
13  compilers: {
14    solc: {
15      version: "^0.8.28",
16      settings: {
17        optimizer: {
18          enabled: true,
19          runs: 200
20        }
21      }
22    }
23  },
24
25  contracts_directory: "./contracts",
26  contracts_build_directory: "./build/contracts",
27
28
29  mocha: {
30    timeout: 100000
31  },
32
33 };

```

Código 3: truffle-config.js

### Etapas do Deploy

1. **Compilação:** O contrato foi compilado utilizando o Truffle.
2. **Deploy:** Utilizando o Ganache, o contrato foi implantado em uma blockchain local para testes.
3. **Interação:** Após o deploy, foi feita a interação com o contrato através do MetaMask.

A seguir estão capturas de tela do Ganache:

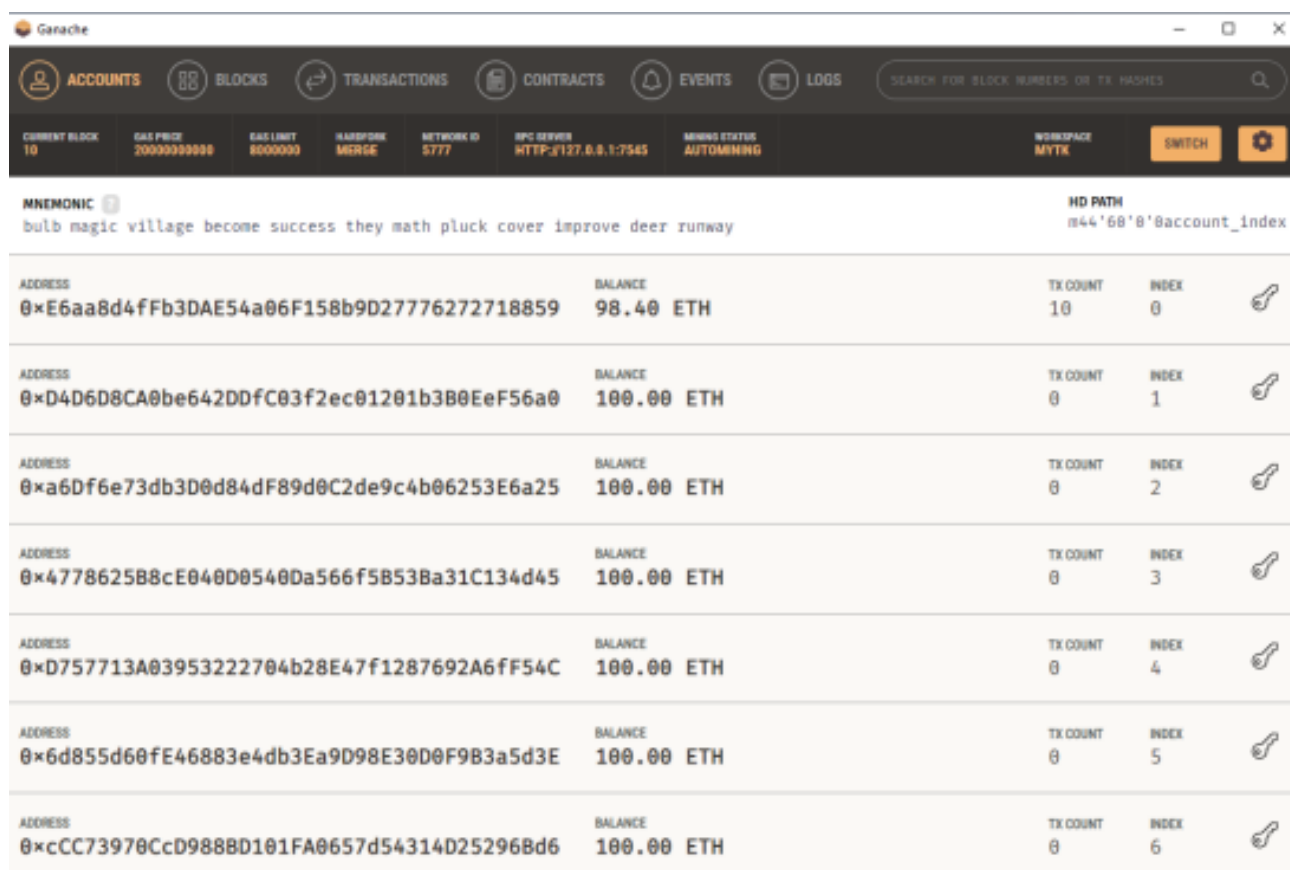


Figura 1: Tela principal do ganache

A imagem 1 mostra a interface do Ganache, exibindo contas simuladas em uma rede local Ethereum. Cada conta tem um endereço, saldo em ETH (Ether) inicial, número de transações realizadas (Tx Count) e índice.

CURRENT BLOCK	GAS PRICE	GAS LIMIT	HARDFORK	NETWORK ID	RPC SERVER	MINING STATUS	WORKSPACE
10	20000000000	8000000	MERGE	5777	HTTP://127.0.0.1:7545	AUTOMINING	MYTK

BLOCK	MINED ON	GAS USED	TRANSACTIONS
10	2024-12-02 22:13:58	8000000	1 TRANSACTION
9	2024-12-02 22:12:38	8000000	1 TRANSACTION
8	2024-12-02 22:10:19	8000000	1 TRANSACTION
7	2024-12-02 22:07:28	8000000	1 TRANSACTION
6	2024-12-02 22:07:02	8000000	1 TRANSACTION
5	2024-11-28 12:35:27	8000000	1 TRANSACTION
4	2024-11-28 12:26:37	8000000	1 TRANSACTION
3	2024-11-28 12:18:12	8000000	1 TRANSACTION
2	2024-11-28 12:15:58	8000000	1 TRANSACTION
1	2024-11-28 12:13:57	8000000	1 TRANSACTION

Figura 2: Tela de Blocks ganache

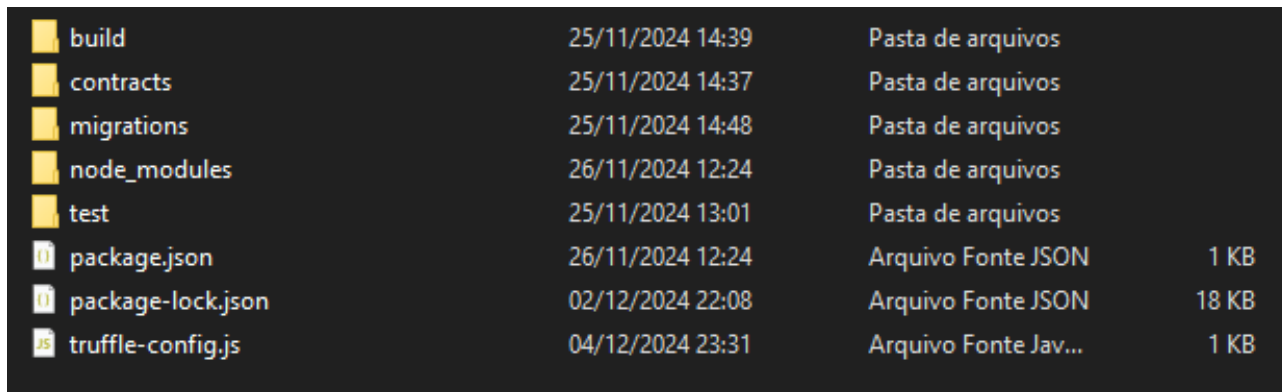
A imagem 2 mostra a aba Blocks do Ganache, onde estão listados os blocos minerados na rede local simulada. Os blocos estão organizados do mais recente (número 10) ao mais antigo (número 1). Cada bloco exibe informações como o número do bloco, a data e hora em que foi minerado, a quantidade de gás utilizada nas transações e o botão que indica a quantidade de transações incluídas, sendo 1 transação em todos os blocos mostrados.

## Pasta Do Projeto

O projeto segue a estrutura padrão do framework Truffle para contratos inteligentes na blockchain Ethereum. Abaixo, descrevo os principais diretórios e arquivos:

1. **build** Armazena os artefatos gerados após a compilação dos contratos, como o bytecode e a ABI, necessários para implantar e interagir com os contratos na blockchain.
2. **contracts** Contém os contratos inteligentes do projeto, incluindo o MyToken.sol, que define o token (nome, símbolo e suprimento inicial).
3. **migrations** Possui os scripts para implantação dos contratos. O arquivo 2-deploy-contracts.js automatiza o deploy do MyToken na rede configurada.
4. **node-modules** Contém as bibliotecas e dependências do projeto, como OpenZeppelin, essenciais para criar o token.
5. **test** Direcionada para testes unitários, que validam o funcionamento correto dos contratos inteligentes.

6. `package.json` e `package-lock.json` O `package.json` lista as informações do projeto e dependências. O `package-lock.json` garante a consistência das versões instaladas.
7. `truffle-config.js` Configura redes, compilador Solidity e otimizações para deploy e execução.



build	25/11/2024 14:39	Pasta de arquivos	
contracts	25/11/2024 14:37	Pasta de arquivos	
migrations	25/11/2024 14:48	Pasta de arquivos	
node_modules	26/11/2024 12:24	Pasta de arquivos	
test	25/11/2024 13:01	Pasta de arquivos	
package.json	26/11/2024 12:24	Arquivo Fonte JSON	1 KB
package-lock.json	02/12/2024 22:08	Arquivo Fonte JSON	18 KB
truffle-config.js	04/12/2024 23:31	Arquivo Fonte Jav...	1 KB

Figura 3: Pasta fonte do Projeto

## 5 Considerações Finais

Este projeto me ajudou muito a entender e pesquisar a fundo sobre Tokens e seus projetos, um tema que não conhecia antes e me fez aprender muitas coisas sobre o tema. Ferramentas como Truffle, openZeppelin me ajudaram a simplificar muito o processo de criação, implantação e gerenciamento de contratos inteligentes. Este projeto destacou a importância de compreender os padrões ERC-20, que garantem interoperabilidade e ampla adoção no ecossistema Ethereum. Particularmente tive um problema a mais em relação a problemas de Deploy, usando o comando “truffle compile” e em seguida “truffle migrate –reset –network development”, aparecia inúmeros erros de deploy acusando meu “Mytoken” ser inválido. Houve problemas também em relação aos “blocks” que não estavam contando e nem funcionando, mas realizando uma alteração no limite de “Gas” tudo foi resolvido e funcionando normalmente. Este projeto proporcionou uma compreensão profunda sobre o desenvolvimento de contratos inteligentes e sobre como eles podem ser utilizados para tokenizar ativos no blockchain do Ethereum.

(Antonopoulos & Wood 2018) (Imran 2018)

## Referências

- Antonopoulos, A.M. & G. Wood. 2018. *Mastering ethereum: Building smart contracts and dapps*. O’Reilly Media, Incorporated. <https://books.google.com.br/books?id=SedSMQAACAAJ>.
- Imran, Bashir. 2018. *Mastering blockchain : Distributed ledger technology, decentralization, and smart contracts explained, 2nd edition*. Packt Publishing. <https://search.ebscohost.com/login.aspx?direct=true&db=e000xww&AN=1789486&lang=pt-br&site=eds-live&scope=site>.