

Implementando Tokens no Ethereum

Matheus kenji Nakao¹

¹Departamento Acadêmico de Computação (DACOM)
Universidade Tecnológica Federal do Paraná (UTFPR)

Abstract

Este trabalho explora a implementação de tokens na blockchain *Ethereum*, com foco no desenvolvimento de um token ERC20 personalizado chamado NKToken. Utilizando o framework Truffle e ferramentas como o *Ganache*, o projeto aborda o processo de criação, implantação e interação com contratos inteligentes que regulam o comportamento do token. O estudo detalha os aspectos técnicos da Máquina Virtual *Ethereum* (EVM), a linguagem de programação Solidity e a relevância dos padrões de tokenização, como o ERC20. Ao ilustrar o processo passo a passo, este trabalho busca fornecer um guia abrangente para desenvolvedores e pesquisadores interessados em aplicações de tokens baseadas em blockchain.

Resumo

Este trabalho explora a implementação de tokens na blockchain *Ethereum*, com foco no desenvolvimento de um token ERC20 personalizado chamado NKToken. Utilizando o framework Truffle e ferramentas como o *Ganache*, o projeto aborda o processo de criação, implantação e interação com contratos inteligentes que regulam o comportamento do token. O estudo detalha os aspectos técnicos da Máquina Virtual *Ethereum* (EVM), a linguagem de programação Solidity e a relevância dos padrões de tokenização, como o ERC20. Ao ilustrar o processo passo a passo, este trabalho busca fornecer um guia abrangente para desenvolvedores e pesquisadores interessados em aplicações de tokens baseadas em blockchain.

1 Introdução

A tecnologia blockchain se foi utilizada no lançamento do Bitcoin em 2009 por um autor conhecido como Satoshi Nakamoto, e ela é uma tecnologia de registro distribuído que funciona como um banco de dados imutável e transparente, ela revolucionou a forma como as informações podem ser registradas e compartilhadas de maneira descentralizada. Cada bloco da cadeia contém um conjunto de transações realizadas, e esses blocos são conectados de forma linear e cronológica. Os principais atributos do blockchain incluem:

- **Imutabilidade:** Uma vez registrados, os dados não podem ser alterados sem consenso da rede inteira.
- **Transparência:** As transações são visíveis para todos os participantes da rede por meio da sua identificação de hash.
- **Descentralização:** Não há uma autoridade central; a segurança e validação dependem dos nós da rede.

A blockchain transcende seu uso inicial em criptomoedas e hoje é aplicado em vastos setores distintos como finanças, saúde, cadeias de suprimentos e propriedade intelectual BASHIR. (2018).

1.1 Ethereum e Contratos Inteligentes

Ethereum é uma plataforma blockchain de código aberto que expande as funcionalidades introduzidas pelo Bitcoin ao permitir a criação e execução de contratos inteligentes. Um contrato inteligente é um programa auto executável, com regras pré-definidas escritas em código, que opera sem necessidade de intermediários. Eles são armazenados e executados na blockchain, garantindo sua segurança e integridade. Diferentemente do Bitcoin, que foi projetado principalmente como um sistema de pagamentos descentralizado, *Ethereum* permite a implementação de uma variedade de aplicações descentralizadas. Isso foi possível graças à introdução de uma linguagem de programação robusta, como Solidity, que permite o desenvolvimento de contratos que podem:

- Gerenciar ativos digitais.
- Automatizar processos empresariais.
- Criar sistemas de votação, entre outros.

1.2 Ethereum Virtual Machine

A *Ethereum Virtual Machine* ou EVM é o ambiente de execução para os contratos inteligentes no *Ethereum*. Trata-se de uma máquina virtual que permite que os desenvolvedores escrevam código em linguagens de alto nível, como Solidity, e o executem em uma rede descentralizada. O papel principal da EVM é garantir que os contratos inteligentes sejam executados exatamente como programados. Isso é feito de maneira determinística e segura. A EVM é:

- **Turing-completa:** Capaz de executar qualquer algoritmo computável, desde que haja recursos suficientes.
- **Determinística:** Garante que o mesmo código forneça o mesmo resultado em qualquer nó da rede.
- **Isolada:** Cada contrato opera em seu próprio ambiente, sem acesso direto ao sistema subjacente, prevenindo efeitos colaterais.

1.3 Tokens ou Tokenização

Tokenização é o processo de converter ativos em unidades digitais registradas na blockchain, conhecidas como tokens. Um token pode representar praticamente qualquer coisa, desde moedas digitais até bens físicos ou serviços. Essa prática democratiza o acesso a ativos, facilitando sua negociação e o seu rastreamento. No Ethereum, a tokenização é amplamente utilizada devido à padronização proporcionada por padrões como o padrão ERC20 Antonopoulos & Wood (2018). Esses padrões definem um conjunto de regras que os tokens devem seguir, promovendo interoperabilidade e facilidade de uso. Tokens são usados em diversos casos, incluindo:

- **Moedas digitais:** Como o *Ethereum* ou Bitcoin
- **Tokens de utilidade:** Que oferecem acesso a serviços, como em plataformas de crowdfunding.
- **Tokens de segurança:** Representando ações ou direitos financeiros.
- **NFTs (Tokens Não Fungíveis):** Para representar itens únicos, como arte digital e colecionáveis.

2 Contratos Inteligentes

Contratos inteligentes são programas auto executáveis que residem em redes blockchain, como *Ethereum*. Eles contêm regras pré-estabelecidas que, uma vez implementadas, garantem que as

ações sejam executadas automaticamente quando as condições definidas forem atendidas. Conceito ganhou popularidade com o *Ethereum*, que tornou sua implementação de certa forma mais prática. Os contratos inteligentes eliminam a necessidade de intermediários, garantindo que as transações sejam seguras, transparentes e confiáveis. Por exemplo, em vez de um banco atuar como um intermediário em uma transação financeira, um contrato inteligente pode gerenciar o processo automaticamente. De forma geral suas características principais são:

- **Autonomia:** Uma vez implementado, o contrato opera de forma independente, sem necessidade de intervenção humana.
- **Segurança:** O contrato é imutável após ser registrado no blockchain, protegendo-o contra alterações maliciosas.
- **Transparência:** As regras do contrato são visíveis para todos os participantes da rede.
- **Execução Automática:** Assim que as condições especificadas são atendidas, o contrato executa suas ações automaticamente.
- **Irreversibilidade:** As ações realizadas pelo contrato não podem ser revertidas.

2.1 Uso e Utilidades

Os contratos inteligentes podem ser usados para automatizar processos em diversos setores, incluindo: Finanças: Execução de empréstimos, seguros e pagamentos automáticos. Imobiliário: Automatização de contratos de aluguel e transferência de propriedade. Logística: Rastreamento de bens e gerenciamento de cadeias de suprimento. Saúde: Gerenciamento seguro de dados médicos e execução de pagamentos. Governança: Votação eletrônica e gestão de identidade digital.

2.2 Solidity

Solidity é a principal linguagem usada para desenvolver contratos inteligentes na rede *Ethereum*. Inspirada por linguagens como JavaScript e Python, ela é projetada especificamente para interagir com a já citada *Ethereum Virtual Machine (EVM)*.

3 Tokenização

Tokens são unidades digitais de valor ou utilidade que podem ser criadas, transferidas e armazenadas em redes *blockchain*. Eles não apenas representam ativos financeiros, mas também direitos, propriedades, ou até mesmo acesso a serviços. Em termos gerais, um token é um registro digital com propriedades programáveis, garantindo segurança e transparência por meio de contratos inteligentes. Os tokens ganharam popularidade graças ao *blockchain Ethereum*, que permitiu sua padronização, simplificando sua criação e integração em diversas aplicações. Eles podem ser categorizados de diversas formas, dependendo de seu uso ou função.

3.1 Padrões de Implementação

O *Ethereum* padronizou a criação de tokens através de modelos específicos que estabelecem as regras de interação. Dois dos principais padrões incluem:

1. ERC20:

- O padrão mais usado para criar tokens fungíveis.
- Define funções básicas, como transferência de tokens e consulta de saldos.
- Tokens como USDT e LINK são baseados no ERC20.

2. ERC721:

- Usado para criar tokens não fungíveis (NFTs).
- Cada token é único e possui metadados que os distinguem.
- Amplamente utilizado em jogos e colecionáveis digitais.

3.2 Funcionamento dos Tokens

Os tokens operam a partir de contratos inteligentes, que gerenciam seu comportamento e interações. Um contrato inteligente padrão para um token ERC20, por exemplo, pode conter dentre outras as seguintes funcionalidades:

- **Criação (Minting):** Gera novos tokens.
- **Transferência:** Move tokens de um endereço para outro.
- **Consulta de saldo:** Permite verificar quantos tokens estão em determinado endereço.
- **Aprovação e Transferência:** Garante que terceiros possam movimentar tokens de forma segura e autorizada.

4 Ferramentas de Desenvolvimento

Para implementar contratos inteligentes e tokens no *Ethereum*, é essencial utilizar ferramentas que simplifiquem o desenvolvimento, teste e implantação. Duas ferramentas amplamente usadas são o *Ganache* e o Truffle, que fornecem um ambiente integrado para trabalhar com redes blockchain de forma eficiente.

4.1 Ganache

*Ganache*¹ é uma ferramenta desenvolvida pelo Truffle Suite que permite criar uma blockchain local para teste de contratos inteligentes. Ele simula uma rede *Ethereum* completa em seu ambiente, fornecendo recursos úteis para desenvolvedores.

Características principais do Ganache:

1. **Blockchain Local:** Simula uma rede *Ethereum* em um ambiente local, permitindo testes rápidos e seguros.
2. **Velocidade Personalizada:** A velocidade dos blocos pode ser ajustada para testar diferentes cenários.
3. **Conta Pré-configuradas:** Oferece endereços com saldo de Ether fictício para facilitar as transações de teste.
4. **Histórico de Transações:** Armazena todas as transações realizadas durante a sessão, permitindo análise detalhada.
5. **Interface Gráfica:** Se utilizar o *Ganache UI*, ele fornece uma interface amigável para monitorar a blockchain local, contratos e transações.

4.2 Truffle

Truffle é um framework de desenvolvimento completo para *Ethereum*, projetado para ajudar em todo o ciclo de vida do desenvolvimento de contratos inteligentes, incluindo:

1. **Compilação:** Transforma código em Solidity para bytecode executável.
2. **Testes Automatizados:** Realiza testes em JavaScript ou Solidity para verificar a funcionalidade dos contratos.
3. **Interação com Contratos:** Permite a execução de funções do contrato e análise de eventos.

¹ <https://archive.trufflesuite.com/ganache/>

Características do Truffle:

- **Gerenciamento de Projetos:** Oferece uma estrutura organizada para arquivos de código, migrações e testes.
- **Testes Automatizados:** Integração com bibliotecas que facilitam testes.
- **Compatibilidade:** Funciona com diversas redes *Ethereum* como *Ganache*.
- **Integração com MetaMask:** Permite testar contratos diretamente com carteiras como MetaMask.

4.3 Open Zeppelin

A *OpenZeppelin Library*² é uma coleção amplamente reconhecida de contratos inteligentes pré-desenvolvidos e testados, criada justamente para facilitar o desenvolvimento na blockchain Ethereum visando ser segura e eficiente, ela é escrita em Solidity principalmente para padrões tokens, contratos de governancia, contratos inteligentes dentre várias outras funções, evitando assim a implementação de funções básicas do zero, como *transfer*, *approve* e *mint*.

5 Especificação e Desenvolvimento do Projeto

A máquina é um Virtualbox da Oracle, foi instalada a distribuição Ubuntu ver. 24.04.1

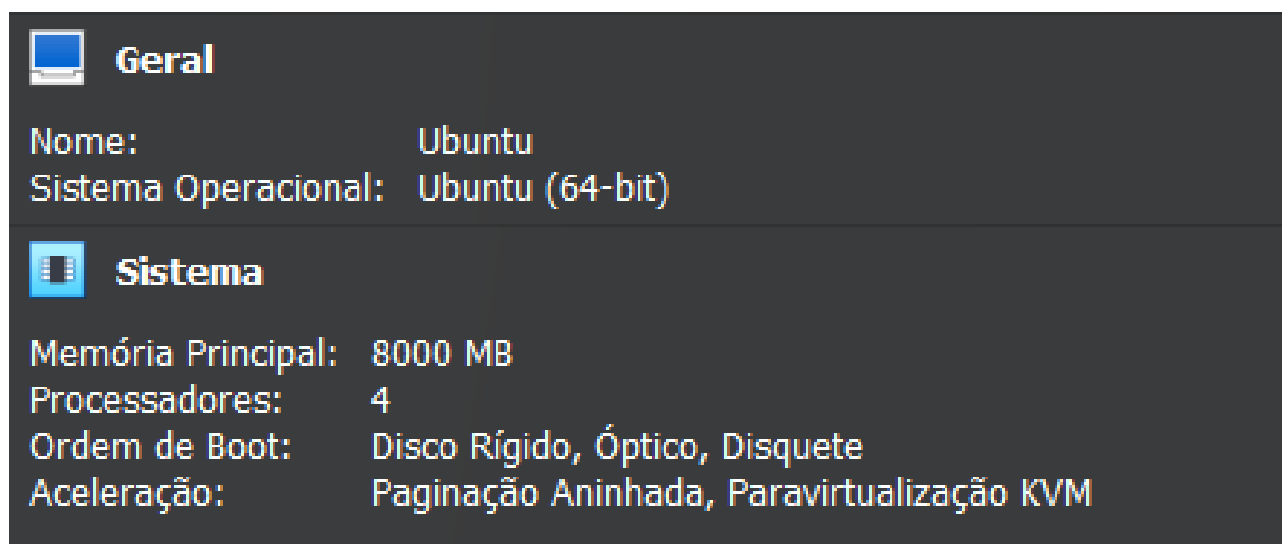


Figura 1: Especificações da Máquina

Já as ferramentas utilizadas são:

1. Truffle v5.11.5 (core: 5.11.5)
2. Ganache v7.9.1
3. Solidity v0.5.16 (solc-js)
4. Node v18.20.4
5. Web3.js v1.10.0

5.1 Criando o Token

Esse subcapítulo foi realizado com base nas instruções do capítulo 10: Tokens (Antonopoulos & Wood (2018)) Nele é criado um novo diretório, como o nosso token se chamará **Nakao Token** Abreviado para **NKT** criei um diretório **NKToken** e ele será a base para o nosso projeto. Após

² <https://www.openzeppelin.com/>

isso, foi realizado o comando 'truffle init' para inicializar o projeto truffle assim, podemos observar a estrutura do projeto conforme mostrado na Figura 2.

```
nakao@KNakao:~$ cd NKToken/
nakao@KNakao:~/NKToken$ truffle init

Starting init...
=====

> Copying project files to /home/nakao/NKToken

Init successful, sweet!

Try our scaffold commands to get started:
  $ truffle create contract YourContractName # scaffold a contract
  $ truffle create test YourTestName        # scaffold a test

http://trufflesuite.com/docs

nakao@KNakao:~/NKToken$ cd
nakao@KNakao:~$ tree NKToken/
NKToken/
├── contracts
├── migrations
├── test
└── truffle-config.js

4 directories, 1 file
nakao@KNakao:~$ S
```

Figura 2: Truffle Init e Estrutura do Projeto

Após obter o seguinte estrutura, comecei criando um simples contrato .sol chamado NKToken.sol apresentado pelo Código 1.

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.0;
4
5 import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
6
7 contract NKToken is ERC20 {
8     constructor(uint256 initialSupply) ERC20("NKToken", "NKT") {
9         _mint(msg.sender, initialSupply);
10    }
11 }
```

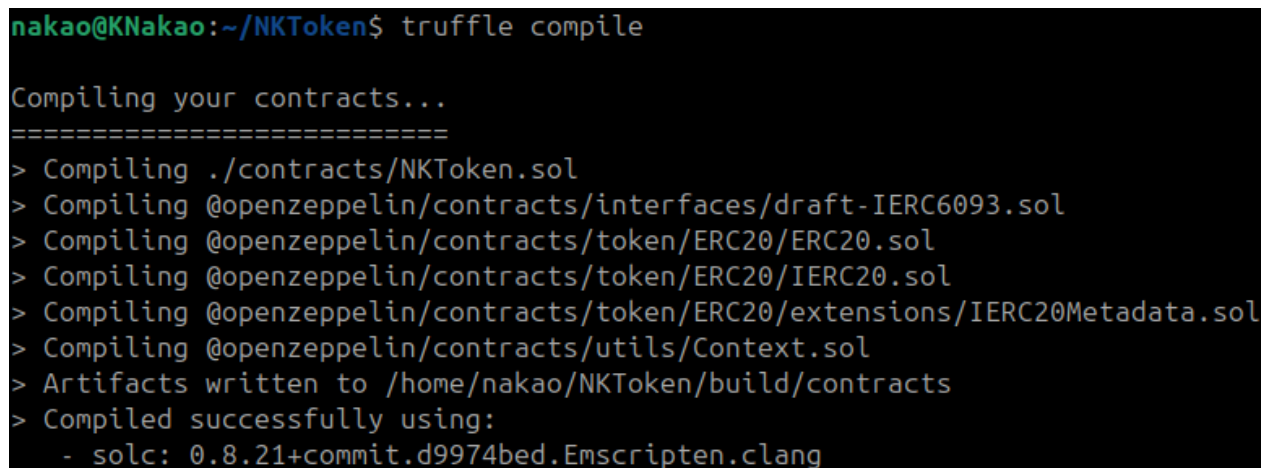
Código 1: Exemplo do NKToken.sol criado

Agora vamos configurar o arquivo de migrações que ficou conforme o Código 2.

```
1 const NKToken = artifacts.require("NKToken");
2
3 module.exports = function (deployer) {
4   const initialSupply = web3.utils.toWei("1000000", "ether"); // 1 milhão de
      tokens
5   deployer.deploy(NKToken, initialSupply);
6 };
```

Código 2: Exemplo do initial migrations .js criado

Finalizado isso podemos compilar utilizando o comando 'truffle compile' conforme a Figura 3 mostra.



```
nakao@KNakao:~/NKToken$ truffle compile

Compiling your contracts...
=====
> Compiling ./contracts/NKToken.sol
> Compiling @openzeppelin/contracts/interfaces/draft-IERC6093.sol
> Compiling @openzeppelin/contracts/token/ERC20/ERC20.sol
> Compiling @openzeppelin/contracts/token/ERC20/IERC20.sol
> Compiling @openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol
> Compiling @openzeppelin/contracts/utils/Context.sol
> Artifacts written to /home/nakao/NKToken/build/contracts
> Compiled successfully using:
   - solc: 0.8.21+commit.d9974bed.Emscripten.clang
```

Figura 3: Resultado para o comando Truffle Compile

Agora podemos configurar a rede para deploy via *ganache*, no 'truffle.config.js' podemos modificar o arquivo que o 'truffle init' gerou com o seguinte código Código 3. O arquivo foi configurado segundo os dados do próprio *ganache* client conforme a Figura 4 mostra

```
1 module.exports = {
2
3   networks: {
4     development: {
5       host: "127.0.0.1",
6       port: 7545,
7       network_id: 5777,
8     },
9   },
10
11   mocha: {
12   },
13
14   compilers: {
15     solc: {
16       version: "0.8.21",
17     },
18   },
19 };
```

Código 3: Exemplo do truffle config .js criado

```

ganache v7.9.2 (@ganache/cli: 0.10.2, @ganache/core: 0.10.2)
Starting RPC server

Available Accounts
=====
(0) 0xbc8114e6fCf281d62bE9e63ac5aF561B33f0d9B4 (1000 ETH)
(1) 0xf5e0D1Fc8c1ebcdd8AE5d54530BeD4a1C2486812 (1000 ETH)
(2) 0xFE40Bf6EC608e134d02AC8262065888c020b955a (1000 ETH)
(3) 0x87476be8BbD3F2605654060924585291D632D5c4 (1000 ETH)
(4) 0x1B81272A2aBe6Fd3E856C5843Cd89199AF025DD1 (1000 ETH)
(5) 0x58d6036F832095cb2B374A04C01A8Cf7B76CB9d2 (1000 ETH)
(6) 0xA6662e9f33FA5e04A5eFd26E9981A965b990BfCf (1000 ETH)
(7) 0xe692ee5c5bA19B1A16fD004CA539C765a2959A46 (1000 ETH)
(8) 0x997e3d28Fc643D757368550ee6B537eE1c984184 (1000 ETH)
(9) 0xEB282846C10AA47B5E9F6f7243Df7832854e15b4 (1000 ETH)

Private Keys
=====
(0) 0xe5574fdce6986d8790adfb4ba7c2966d066a1dabafa8c020c04b86f4fbb3d49e
(1) 0xd50e6fa6c2441c2f52bfcdfca276676d96f00aa862577b45cb266bf1e3a7c2ab
(2) 0x2800371e0f7cd1298dba6f9dd02e9c1b1c87986985ae475d2144352cbab81429
(3) 0xac81216fa03ad0918f37f07ab1a13e6087abb8863ffa11c636d246fa490388dc
(4) 0x3a4730af671d7d503292519a67e916f6b188c8ab5f2ec392fb8273de8e13574f
(5) 0x96a4caeecd43f8a3131dafa1dbf4a8d06097a7588889bea1401d73516d406ed9
(6) 0xa9664d247ad3dea313df04677646cb2abcd46487ed89369d8989b08929694fea
(7) 0xef043b2283b65ad08b8d17de1cfc142be6e6473ff6c1957210bee96b2a71375b
(8) 0x253e1679c63f55d06d0b1a6027bd980589f36713b7d991cac8b02cebd9de192c
(9) 0xd96836319f4f4c15dd2bdf66c8297cda4659bceb36881a263eb56b1c951b4486

HD Wallet
=====
Mnemonic:      where police just satoshi sibling say combine injury mixture page reveal maple
Base HD Path:  m/44'/60'/0'/0/{account_index}

Default Gas Price
=====
20000000000

BlockGas Limit
=====
30000000

```

Figura 4: Ganache Client executando pelo terminal Linux

Feito a alteração do arquivo 'truffle.config.js' podemos finalmente realizar o deploy via *ganache*, utilizando o comando 'truffle migrate --network development', conforme mostra a Figura 5


```
nakao@KNakao:~/NKToken$ truffle migrate --network development

Compiling your contracts...
=====
> Everything is up to date, there is nothing to compile.

Starting migrations...
=====
> Network name:      'development'
> Network id:        1733057359676
> Block gas limit: 30000000 (0x1c9c380)

1_deploy_contracts.js
=====

  Deploying 'NKToken'
  -----
  > transaction hash: 0xb786375bae3e554adbbfdab0caec7c1cf28f5d88e952b3130041740a5998ccec
  > Blocks: 0        Seconds: 0
  > contract address: 0xFCEa7C8E38e5D271C9CC86B0BaA10822cBc4Af48
  > block number:     1
  > block timestamp:  1733057422
  > account:          0xbc8114e6fCf281d62bE9e63ac5aF561B33f0d9B4
  > balance:          999.9968014315
  > gas used:          947724 (0xe760c)
  > gas price:         3.375 gwei
  > value sent:        0 ETH
  > total cost:        0.0031985685 ETH

  > Saving artifacts
  -----
  > Total cost:        0.0031985685 ETH

Summary
=====
> Total deployments: 1
> Final cost:        0.0031985685 ETH
```

Figura 5: Comando Truffle migrate

Ele executou o contrato da pasta contracts que fizemos gerando uma transação Hash, a mesma pode ser vista pelo terminal do *Ganache* Client conforme mostra a Figura 6

```

RPC Listening on 127.0.0.1:8545
eth_blockNumber
net_version
eth_accounts
eth_getBlockByNumber
eth_accounts
net_version
eth_getBlockByNumber
eth_getBlockByNumber
net_version
eth_getBlockByNumber
eth_estimateGas
net_version
eth_blockNumber
eth_getBlockByNumber
eth_estimateGas
eth_getBlockByNumber
eth_gasPrice
eth_sendTransaction

Transaction: 0xb786375bae3e554adbbfdab0caec7c1cf28f5d88e952b3130041740a5998ccec
Contract created: 0xfcea7c8e38e5d271c9cc86b0baa10822cbc4af48
Gas usage: 947724
Block number: 1
Block time: Sun Dec 01 2024 09:50:22 GMT-0300 (Brasilia Standard Time)

eth_getTransactionReceipt
eth_getCode
eth_getTransactionByHash
eth_getBlockByNumber
eth_getBalance

```

Figura 6: Transação no terminal do *Ganache*

Para realizar uma transferencia, podemos utilizar o seguinte comando mostrado no Código 4

```

1 $ truffle console
2 > let instance = await.NKToken.deployed();
3 > let total supply = await instance.totalSuply();
4 > let accounts = await web3.eth.getAccounts();
5 > await instance.stransfer(accounts[2], 500);

```

Código 4: Comandos para Realizar transação

Os comandos do Código 4 realizam em ordem:

1. **truffle console** - Abre o console truffle
2. **let instance = await.NKToken.deployed();** - Obtém uma instância do contrato inteligente NKToken que foi implantado na rede
3. **let total supply = await instance.totalSuply();** - Recupera o fornecimento total de tokens, podendo ser consultado pelo comando `'console.log(totalSupply.toString());'`.
4. **let accounts = await web3.eth.getAccounts();** - Recupera uma lista de todas as contas disponíveis na rede Ethereum.
5. **await instance.stransfer(accounts[2], 500);** - Chama a função *transfer* do contrato NKToken e transfere 500 tokens da conta que está executando a transação.

6 Considerações Finais

O desenvolvimento do NKToken evidenciou a robustez e a flexibilidade da plataforma *Ethereum* para aplicações descentralizadas. A experiência prática de implementar e interagir com contratos inteligentes utilizando ferramentas modernas como *Truffle* e *Ganache* permitiu aprofundar o conhecimento sobre a tokenização e suas aplicações práticas. Este estudo reafirma o potencial dos padrões ERC, como o ERC20, em promover interoperabilidade e padronização no ecossistema blockchain.

Referências

- Antonopoulos, A.M. & G. Wood. 2018. *Mastering ethereum: Building smart contracts and dapps*. O'Reilly. <https://books.google.com.br/books?id=SedSMQAACAAJ>.
- BASHIR., IMRAN. 2018. *Mastering blockchain - second edition*. Place of publication not identified. <https://research.ebsco.com/linkprocessor/plink?id=1f0a716c-d80f-3b75-b223-821fd4750c16>.