

Implementação de Token Fungíveis na Ethereum Blockchain: Um caso de uso reproduzível explorando Solidity, Ganache CLI, ERC-20 e ferramentas de desenvolvimento

Julio Cesar Nogueira¹

¹Departamento Acadêmico de Computação (DACOM)

Programa de Pós Graduação em Ciência da Computação

Universidade Tecnológica Federal do Paraná (UTFPR)

Abstract

This report aim to present some concepts about cryptocurrency and its technology, such as Ethereum, EVM, Blockchain and Solidity. Cryptocurrencies are digital currencies that rely on blockchain technology to ensure secure, transparent, and immutable transactions. Among these, Ethereum stands out as a decentralized platform that expands blockchain functionality with smart contracts and decentralized applications (dApps). This is enabled by the Ethereum Virtual Machine (EVM), a decentralized computational environment ensuring contract execution integrity. Solidity, a high-level programming language, facilitates the creation of smart contracts, fostering innovation in decentralized finance (DeFi), NFTs, and governance. Together, blockchain, EVM, and Solidity have reshaped application development by promoting trust, transparency, and efficiency across various industries.

Resumo

Esse relatório tem como objetivos apresentar conceitos sobre criptomoedas, suas tecnologias incluindo Ethereum, EVM, Blockchain e Solidity. Criptomoedas são moedas digitais que utilizam a tecnologia blockchain para garantir transações seguras, transparentes e imutáveis. O Ethereum é uma das plataformas mais proeminentes, permitindo a criação de contratos inteligentes e aplicativos descentralizados (dApps). Essa funcionalidade é possível graças à Ethereum Virtual Machine (EVM), um ambiente computacional descentralizado que assegura a execução confiável dos contratos. A linguagem Solidity possibilita a programação desses contratos, viabilizando inovações em finanças descentralizadas (DeFi), NFTs e governança. A integração entre blockchain, EVM e Solidity revoluciona o desenvolvimento de aplicações, promovendo confiança, transparência e eficiência.

1 Introdução

As tecnologias blockchain emergiram como uma solução inovadora para registros digitais descentralizados e imutáveis. Uma blockchain é uma estrutura distribuída que organiza transações em blocos encadeados por criptografia, promovendo transparência e resistência a adulterações Yaga et al. (2020). Além de possibilitar transações financeiras descentralizadas, como o Bitcoin, a tecnologia também abriu espaço para o desenvolvimento de plataformas que suportam contratos inteligentes, como o Ethereum.

O Ethereum se destaca ao expandir os usos das blockchains, possibilitando a execução de contratos inteligentes - programas executados de forma autônoma em redes descentralizadas. Esses contratos são processados pela Ethereum Virtual Machine (EVM), que garante a execução segura e confiável (Hoffman 2024). Dentro do ecossistema Ethereum, a tokenização é um dos conceitos mais significativos. Ela permite que ativos digitais e físicos sejam representados em tokens na blockchain, promovendo maior liquidez e acessibilidade Brunner (2024).

A tokenização possibilita criar e gerenciar tokens que representam ações, bens imóveis ou mesmo propriedade intelectual, trazendo novas perspectivas para a economia e governança. Este trabalho terá como foco explorar o Ethereum como base para aplicações práticas de tokenização, aproveitando sua infraestrutura robusta e comunidade ativa de desenvolvedores.

2 Ethereum: Uma Introdução

Ethereum é uma plataforma de blockchain descentralizada que permite o desenvolvimento de aplicativos descentralizados (dApps) e a execução de contratos inteligentes. Criada em 2015, expande os conceitos do Bitcoin ao introduzir uma linguagem de programação Turing completa, permitindo o desenvolvimento de aplicações mais complexas Buterin (2015).

2.1 Ethereum Virtual Machine (EVM)

No núcleo do Ethereum está a Ethereum Virtual Machine (EVM), um ambiente de execução que processa contratos inteligentes de forma segura e determinística. A EVM opera como um computador descentralizado, distribuído entre os nós da rede, garantindo que os contratos sejam executados de maneira consistente, independentemente da localização do nó Foundation (2024h). A EVM é compatível com diversas linguagens de programação, como Solidity e Vyper, facilitando a criação de soluções robustas e escaláveis Team (2024d).

2.2 Redes: Principal e Testes

A rede principal do Ethereum, conhecida como *mainnet*, é onde as transações e contratos têm valor real, utilizando Ether (ETH) como moeda nativa. Além da *mainnet*, existem redes de teste, como Goerli, Sepolia e Rinkeby, que permitem aos desenvolvedores experimentar contratos e aplicações sem custos financeiros ou riscos. Essas redes de teste utilizam moedas de teste (test Ether) e replicam as condições da *mainnet*, sendo essenciais para o desenvolvimento e validação de novos projetos Foundation (2024e).

O Ethereum continua a evoluir, sendo a atualização Ethereum 2.0 um marco importante, com a transição para o modelo de consenso *Proof of Stake*, melhorando a escalabilidade e a eficiência energética Foundation (2024a).

3 Contratos Inteligentes: Definição e Aplicações

Contratos inteligentes são programas de computador que executam automaticamente os termos de um contrato quando as condições predefinidas são atendidas. O conceito foi introduzido por Nick Szabo em 1997, que descreveu contratos inteligentes como “protocolos de computador que facilitam, verificam ou fazem cumprir a negociação ou execução de um contrato” Szabo (1997). Esses contratos podem ser usados em uma ampla gama de contextos, desde transações financeiras até a automação de processos em setores como seguros, logística e governança corporativa.

O principal benefício dos contratos inteligentes é a eliminação de intermediários, como advogados ou bancos, para garantir que as condições do contrato sejam cumpridas. Isso permite

transações mais rápidas, seguras e de baixo custo. A utilidade dos contratos inteligentes também está relacionada à sua transparência, já que os termos do contrato são imutáveis e visíveis para todos os participantes da rede, o que aumenta a confiança no sistema Wood (2014).

Na prática, a tecnologia de contratos inteligentes é mais amplamente associada a plataformas blockchain, como o Ethereum, onde contratos podem ser escritos e executados de maneira descentralizada Buterin (2014). A tokenização de ativos e o uso de contratos inteligentes em finanças descentralizadas (DeFi) são exemplos de como essa tecnologia pode ser aplicada em escalas maiores, promovendo novos modelos de negócios e economias digitais Foundation (2024b).

4 Solidity e Implementação de Contratos Inteligentes

Solidity é a principal linguagem de programação usada para escrever contratos inteligentes na blockchain Ethereum. Ela foi projetada para ser uma linguagem de alto nível, orientada a objetos, que permite a criação de contratos que podem interagir com o Ethereum Virtual Machine (EVM). Essa linguagem é inspirada em JavaScript, Python e C++, e foi projetada para ser fácil de aprender, segura e eficiente para contratos inteligentes Team (2024d).

A implementação de contratos inteligentes com Solidity envolve a criação de código que define a lógica de um contrato, como transferências de tokens, governança descentralizada ou acordos financeiros. Quando o código é escrito, ele é compilado em bytecode, que pode ser executado pela EVM. Isso permite que os contratos sejam executados de forma descentralizada e sem a necessidade de intermediários, garantindo a confiança e a transparência no processo Foundation (2024g).

A implementação de contratos inteligentes no Ethereum é um processo de três etapas principais: desenvolvimento, testes e implantação. O primeiro passo é escrever o código Solidity, utilizando ferramentas como o Remix IDE ou frameworks como Truffle para facilitar o desenvolvimento Team (2024f). Após a escrita do contrato, é fundamental testá-lo em redes de teste, como a Sepolia, para garantir que o código se comporta conforme esperado antes de ser implantado na rede principal do Ethereum. Por fim, uma vez validado, o contrato é implantado na blockchain Ethereum, onde suas interações podem ser acessadas por outros contratos ou usuários Foundation (2024c).

A segurança dos contratos inteligentes é uma preocupação central no uso de Solidity, já que erros no código podem resultar em perdas financeiras ou falhas no contrato. Por isso, práticas como auditorias de código e o uso de ferramentas automatizadas para detectar vulnerabilidades são essenciais para garantir a confiabilidade dos contratos implementados Foundation (2024f).

5 Ferramentas de Desenvolvimento

O desenvolvimento de contratos inteligentes no Ethereum envolve várias ferramentas essenciais para garantir eficiência, segurança e automação no processo de compilação, teste e deploy. Entre as ferramentas mais populares estão o Remix IDE, Ganache CLI e Hardhat. Estas ferramentas ajudam no ciclo de vida de contratos inteligentes, desde a escrita até a implantação na blockchain.

5.1 Remix IDE

O Remix IDE é uma ferramenta online de desenvolvimento para a criação e testes de contratos inteligentes em Ethereum. Ele oferece um ambiente de desenvolvimento completo com suporte para a linguagem Solidity, além de funcionalidades de compilação e depuração de contratos Team (2024c). O Remix permite que os desenvolvedores escrevam código, testem transações e realizem a depuração diretamente no navegador, sem necessidade de configuração adicional. A

sua interface gráfica é intuitiva, tornando-a ideal tanto para iniciantes quanto para desenvolvedores experientes. O [Remix](#) é frequentemente usado em conjunto com contratos da [OpenZeppelin](#), permitindo a fácil integração de bibliotecas seguras e testadas [OpenZeppelin](#) (2024e).

5.2 Ganache CLI

Ganache CLI é uma ferramenta que simula uma blockchain local para o Ethereum. Ele permite aos desenvolvedores executar e testar contratos inteligentes de forma rápida e isolada, sem precisar de uma rede pública. Utilizando Ganache, é possível realizar transações, verificar saldos e fazer deploy de contratos em uma blockchain local, com total controle sobre os blocos e transações Team (2024e). Isso facilita o desenvolvimento, pois permite que os contratos sejam testados em um ambiente controlado antes de serem enviados para a rede principal ou de teste. Ganache também é compatível com a [OpenZeppelin](#), permitindo que os contratos inteligentes criados com seus contratos base sejam facilmente testados e implementados OpenZeppelin (2024b).

5.3 Hardhat

Hardhat é uma das ferramentas mais poderosas e flexíveis para o desenvolvimento de contratos inteligentes. Ele oferece um ambiente de desenvolvimento que inclui uma blockchain local (Hardhat Network), integração com ferramentas como Ganache e Truffle, além de uma extensa gama de plugins que podem ser utilizados para automatizar várias tarefas Team (2024b). A principal vantagem do Hardhat é sua capacidade de personalização, permitindo que os desenvolvedores configurem scripts para compilar, testar e implantar contratos inteligentes de forma automatizada. Hardhat também integra-se bem com bibliotecas de contratos, como a [OpenZeppelin](#), fornecendo suporte robusto para contratos seguros e auditados OpenZeppelin (2024d).

5.4 OpenZeppelin

[OpenZeppelin](#) é uma biblioteca de contratos inteligentes amplamente utilizada, que fornece contratos pré-auditados e testados para funcionalidades comuns, como tokens ERC20, contratos de governança e outros componentes essenciais para a criação de aplicativos descentralizados seguros OpenZeppelin (2024c).

Utilizar a [OpenZeppelin](#) simplifica o desenvolvimento, reduzindo tempo e esforço na criação de contratos robustos e confiáveis. A biblioteca abrange diversas funcionalidades, incluindo gerenciamento de tokens (ERC20, ERC721, etc.), controle de acesso (Ownable, AccessControl) e utilitários matemáticos.

A combinação da [OpenZeppelin](#) com o padrão ERC20 contribui para a segurança e confiabilidade dos contratos inteligentes, reduzindo a probabilidade de erros e vulnerabilidades, protegendo ativos digitais e a integridade do sistema.

O padrão ERC20 Fabian Vogelsteller (2015) define as funções e eventos para tokens fungíveis no Ethereum. A conformidade com o ERC20 garante interoperabilidade com diversas carteiras, exchanges e dApps. A [OpenZeppelin](#) oferece um contrato ERC20 pré-construído, totalmente compatível com o padrão, incluindo funções para transferência, aprovação, consulta de saldos e emissão de tokens. Utilizar este contrato pré-construído é altamente recomendado devido às auditorias de segurança rigorosas, minimizando riscos de vulnerabilidades.

5.4.1 Métodos Obrigatórios no Padrão ERC20

O padrão ERC20 define uma interface padrão para a criação de tokens fungíveis na blockchain Ethereum. Ele estabelece um conjunto de métodos e eventos que devem ser implementados para garantir a compatibilidade com carteiras, exchanges e outros contratos inteligentes. Esses métodos são obrigatórios para que o token seja reconhecido e utilizável dentro do ecossistema Ethereum Vogelsteller & Buterin (2015).

5.4.2 Métodos Obrigatórios

A especificação ERC20 inclui os seguintes métodos obrigatórios:

- **totalSupply(): uint256**
Retorna o número total de tokens em circulação.
- **balanceOf(address owner): uint256**
Retorna o saldo de tokens de um endereço específico.
- **transfer(address to, uint256 value): bool**
Transfere uma quantidade específica de tokens do remetente para outro endereço.
- **transferFrom(address from, address to, uint256 value): bool**
Permite que um contrato ou endereço transfira tokens em nome de outro, desde que autorizado previamente.
- **approve(address spender, uint256 value): bool**
Autoriza um endereço (geralmente um contrato) a gastar até um valor específico de tokens do proprietário.
- **allowance(address owner, address spender): uint256**
Retorna o valor máximo de tokens que um endereço pode gastar em nome de outro.

5.4.3 Eventos Obrigatórios

Além dos métodos, o ERC20 também define os seguintes eventos que devem ser emitidos durante as operações:

- **Transfer(address indexed from, address indexed to, uint256 value)**
Emitido sempre que tokens são transferidos, indicando os endereços de origem e destino, bem como o valor transferido.
- **Approval(address indexed owner, address indexed spender, uint256 value)**
Emitido quando o proprietário aprova um endereço para gastar tokens em seu nome, indicando o valor aprovado.

5.4.4 Importância dos Métodos

Esses métodos e eventos garantem que qualquer token que implemente o padrão ERC20 possa ser integrado facilmente em diferentes aplicações, como carteiras digitais e plataformas de negociação. Além disso, a implementação rigorosa do padrão ajuda a evitar inconsistências e comportamentos inesperados Foundation (2024d).

Ao integrar a OpenZeppelin com ferramentas como Remix, Ganache CLI e Hardhat, os desenvolvedores podem acelerar o processo de desenvolvimento, reduzindo o risco de vulnerabilidades e garantindo que os contratos atendam aos melhores padrões de segurança.

5.5 Expedition: Explorador de Blockchain para Desenvolvimento de Contratos Inteligentes

O **Expedition** é um explorador de blockchain leve, projetado para redes compatíveis com Ethereum, incluindo ambientes de desenvolvimento local. Ele fornece uma interface intuitiva que facilita a inspeção e análise de transações, blocos, contratos inteligentes e contas. Essa ferramenta é particularmente útil para desenvolvedores que utilizam ambientes locais, como o **Ganache**, ou redes de teste para verificar a execução de contratos inteligentes Team (2024a).

5.5.1 Recursos Principais

O **Expedition** oferece uma ampla gama de recursos que tornam o desenvolvimento de contratos inteligentes mais acessível e eficiente:

- **Visualização de Transações:** Permite monitorar transações em tempo real, exibindo informações detalhadas como remetente, destinatário, quantidade de gás utilizado e estado da transação.
- **Exploração de Blocos:** Fornece acesso detalhado aos dados dos blocos, incluindo número do bloco, *hash*, transações incluídas e recompensas do minerador.
- **Análise de Contratos:** Permite inspecionar contratos inteligentes implantados, incluindo *bytecode*, métodos disponíveis e eventos emitidos.
- **Supporte a Redes Locais e Públicas:** Compatível com redes de teste como Rinkeby e Goerli, além de permitir a integração com ambientes locais configurados com ferramentas como Ganache Team (2024g).
- **Personalização:** Os desenvolvedores podem personalizar a interface do **Expedition** para se adequar às suas necessidades específicas, adicionando funcionalidades extras ou ajustando parâmetros para redes privadas.

5.5.2 Utilidade em Ambientes de Desenvolvimento

No contexto de desenvolvimento de contratos inteligentes, o **Expedition** desempenha um papel crucial ao fornecer uma visão detalhada da execução e estado das transações. Quando combinado com ferramentas como **Ganache** e **Hardhat**, ele permite:

- Depuração em tempo real de contratos inteligentes.
- Validação de dados emitidos por eventos durante a execução do contrato.
- Rastreamento de fluxos de transações para entender melhor como os contratos interagem.

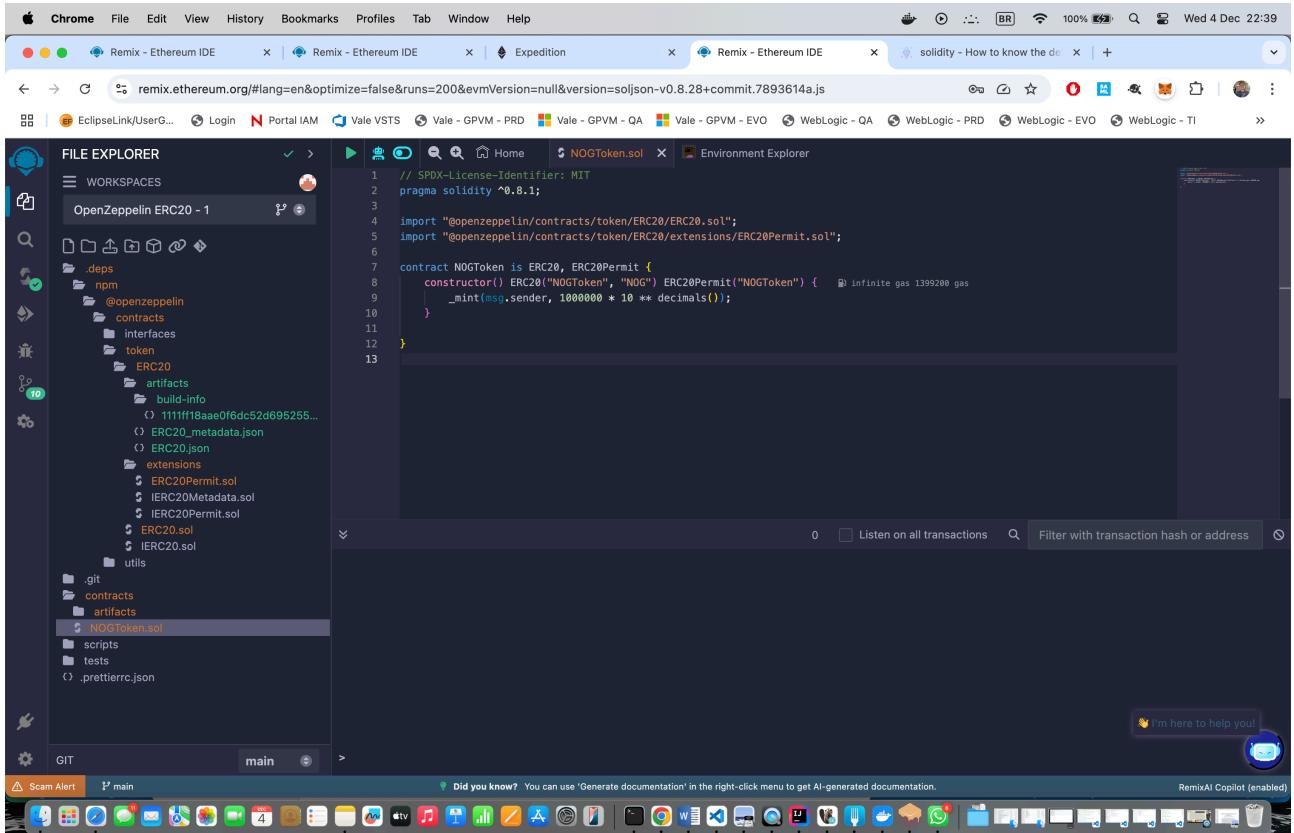
Essa integração direta com ferramentas de desenvolvimento permite uma abordagem mais robusta ao teste e implantação de contratos inteligentes, reduzindo significativamente o risco de erros e inconsistências.

5.6 Automatização do Processo de Compilação e Deploy

A automatização do processo de compilação e deploy de contratos inteligentes é crucial para aumentar a eficiência no ciclo de desenvolvimento. **Hardhat**, por exemplo, permite a configuração de scripts que automatizam a compilação e o deploy de contratos na blockchain, enquanto **Ganache Cli** fornece uma rede local para simular transações antes da implantação na **mainnet**. A **OpenZeppelin** oferece contratos robustos e seguros que podem ser facilmente integrados nesses fluxos de trabalho automatizados, garantindo que os desenvolvedores tenham acesso a componentes seguros e comprovados OpenZeppelin (2024a).

6 Atividade Prática - Desenvolvimento Token e Interação

Para a atividade, foi utilizado o ambiente de desenvolvimento Remix IDE, tendo em vista a ampla gama de plugins para se conectar às redes como Ganache e Geth. Como pode ser observado na imagem Figura 1 utilizamos a biblioteca do OpenZeppelin, que fornece interfaces ERC20 para a criação dos tokens.



```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.1;

import "openzeppelin/contracts/token/ERC20/ERC20.sol";
import "openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol";

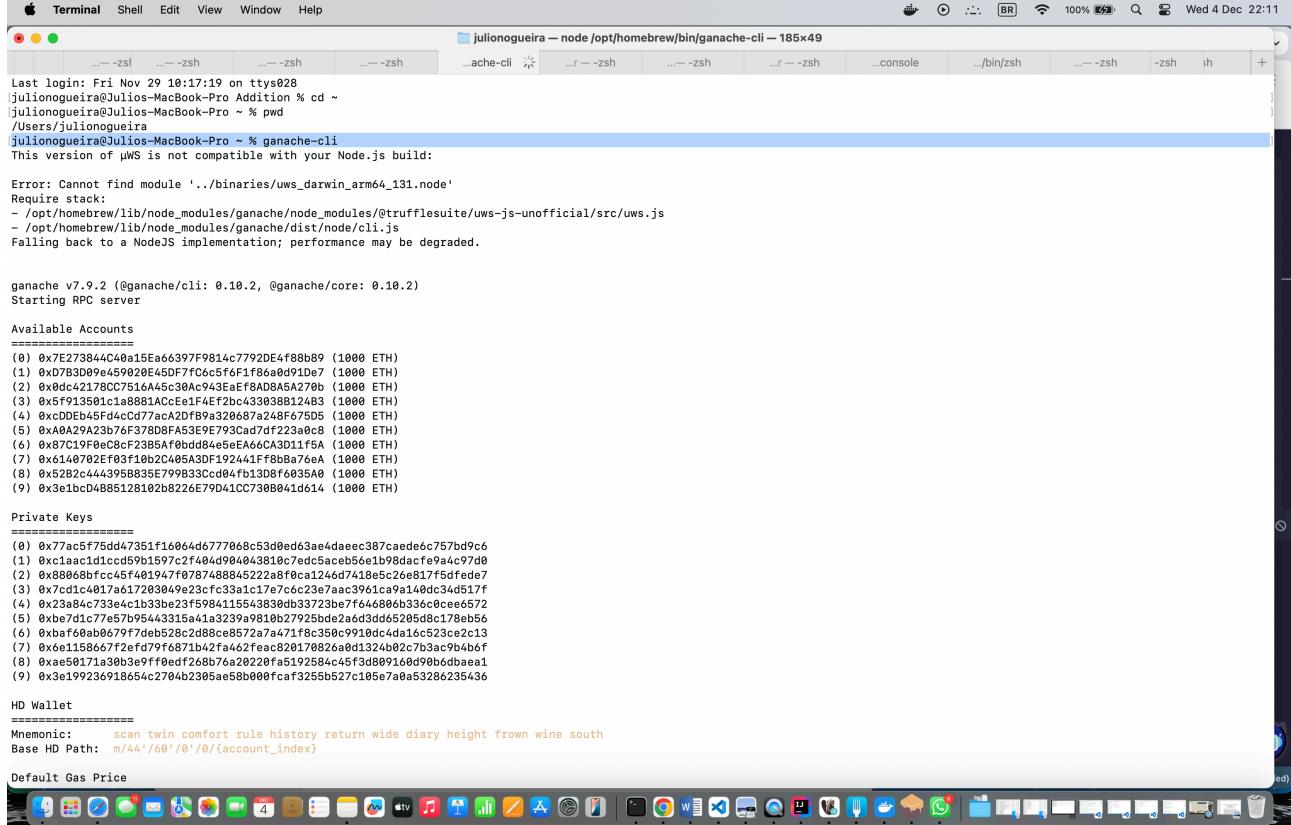
contract NOGToken is ERC20, ERC20Permit {
    constructor() ERC20("NOGToken", "NOG") ERC20Permit("NOGToken") {
        _mint(msg.sender, 100000 * 10 ** decimals());
    }
}
```

Figura 1: Criação do contrato NOGToken.sol

Estamos utilizando o Ganache Cli para simular a nossa blockchain local, ela quem receberá o deploy do contrato que estamos criando e vamos utilizar. Como pode ser observado, ele já cria por padrão 10 contas com saldo e também fornece as informações da rede, as quais iremos usar posteriormente.

Implementação de Token Fungíveis na Ethereum Blockchain: Um caso de uso reprodutível

8 explorando Solidity, Ganache CLI, ERC-20 e ferramentas de desenvolvimento



```
julionogueira - node /opt/homebrew/bin/ganache-cli - 185x49
julionogueira@Julios-MacBook-Pro ~ % cd ~
julionogueira@Julios-MacBook-Pro ~ % pwd
/Users/julionogueira
julionogueira@Julios-MacBook-Pro ~ % ganache-cli
This version of μWS is not compatible with your Node.js build:

Error: Cannot find module '../binaries/uws_darwin_arm64_131.node'
Require stack:
- /opt/homebrew/lib/node_modules/ganache/node_modules/trufflesuite/uws-js-unofficial/src/uws.js
- /opt/homebrew/lib/node_modules/ganache/dist/node/cli.js
Falling back to a NodeJS implementation; performance may be degraded.

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

Available Accounts
=====
(0) 0x7E273844C40a15Ea66397F9814c7792DE4f88b89 (1000 ETH)
(1) 0xD7B3D9e459020E45DF7fc6c5f6F1f86a0d91De7 (1000 ETH)
(2) 0x0dc42178CC7516A4d530Ac943EaF78d85A276b (1000 ETH)
(3) 0x5f913501c1a8881Acce1F4f72bc433038812483 (1000 ETH)
(4) 0xcDB45f4cd77aC2Dfb9a326867a248f67505 (1000 ETH)
(5) 0xA0A29A23b7f6378D8F5A53E9E793Cd7df223a0e8 (1000 ETH)
(6) 0x87C19F0eC8Fc23B5Af0bd84e5eA66CA3D11f5A (1000 ETH)
(7) 0x6140702Ef03f10b22C405A3Df192441Ff8bbBa76eA (1000 ETH)
(8) 0x52B2c444395B8335E799B33Cc0d4fb130Bf0035A0 (1000 ETH)
(9) 0x3ebc4b85128102b8226Ef79d41CC730B041d614 (1000 ETH)

Private Keys
=====
(0) 0x77ac5f75dd47351f16064d6777068c53d0ed63ae4daecc387caede6c757bd9c6
(1) 0xc1aa1c1d1cccd59b1597c2f494d994a43810c7edc5aceb56e1b98dacf9a4c97d0
(2) 0x88068ffc45f401947f0787488845222a8f0ca1246d7418e5c26e817f5dfede7
(3) 0x7cd1c4017a617203049e23fc33a1c17e76c23e7aac3961ca9a140dc34d517f
(4) 0x23a84c733e4c1b33be23f5984115543830db33723be7f6468b6b336c0cee6572
(5) 0xbe7d1c77e57b95443315a41a3239a981b27925bde2a6d3dd65205d8c178e5b6
(6) 0xbaf60ab679f7deb528c2d88ce8572a7471f8350c9910dc4d4a16c523c2e213
(7) 0xe6e1158667f2ef79f6871b42fa462feac820179826a0d1324b2c7b3ac9b4b6f
(8) 0xae50171a3b03e9ff0e0ff268b76a29220f5a192584c45f13d88916bd90b6dabae1
(9) 0x3e19236918654c2704b2305a85b000fcacf3255b527c105e7a0a53286235436

HD Wallet
=====
Mnemonic: scan twin comfort rule history return wide diary height frown wine south
Base HD Path: m/44'/60'/0'/0/{account_index}

Default Gas Price
```

Figura 2: Inicialização Ganache Cli

Com o ambiente de desenvolvimento do Remix e com o Ganache Cli, agora podemos conectar o Remix a partir do provider fornecido pela IDE, ao nosso ambiente local. Na aba de Deploy, podemos verificar os providers disponíveis pela IDE e selecionarmos a Dev - Ganache Provider.

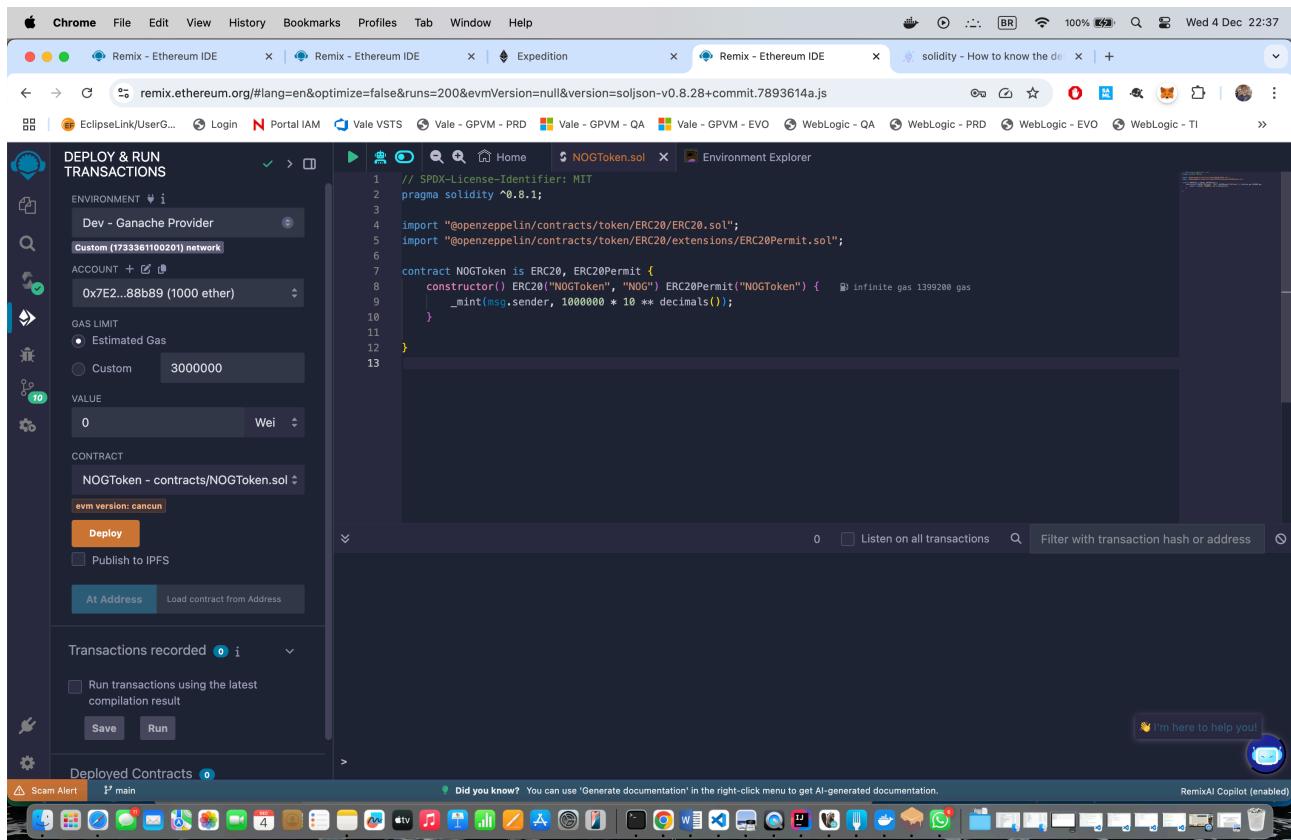


Figura 3: Aba de deploy Remix

Automaticamente é exibido um *prompt* para que seja fornecido o endereço que o Ganache Cli está rodando no ambiente local, por padrão o endereço é <http://localhost:8545> como podemos observar na imagem Figura 4

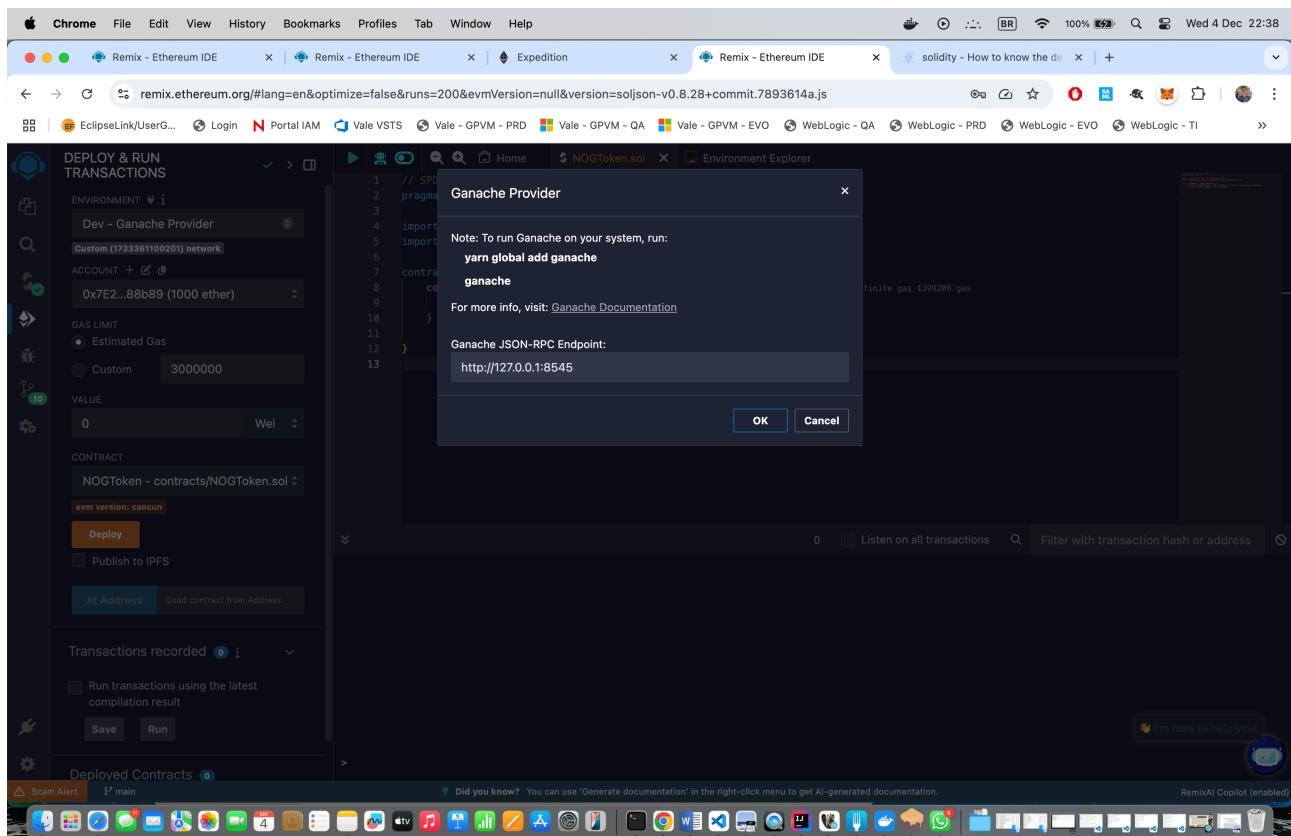


Figura 4: Conectando o Remix ao Ganache Cli

Caso a conexão ocorra com sucesso, após configurar a sua conexão do Remix com o Ganache Cli, serão exibidas as contas e os saldos das mesmas disponíveis, como pode ser observado na imagem Figura 5.

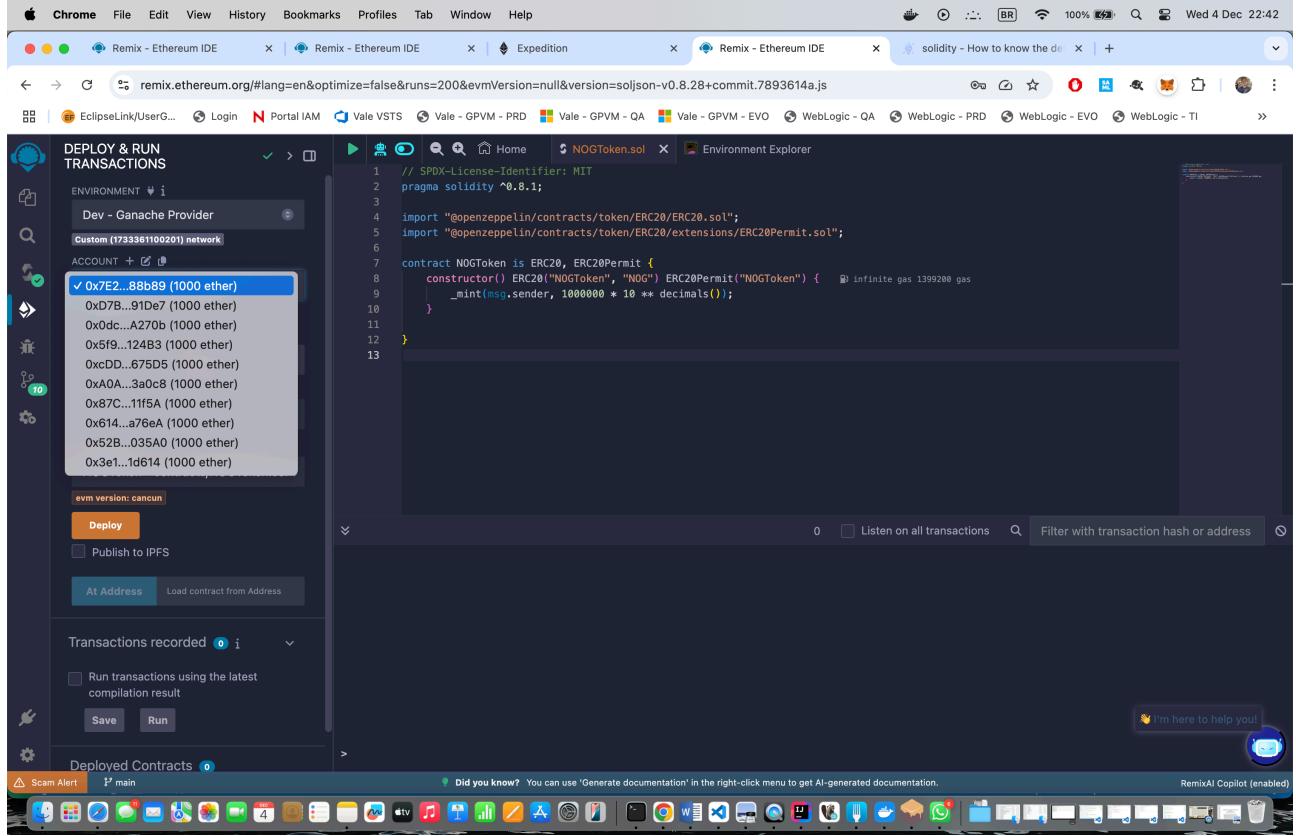


Figura 5: Contas disponíveis no Ganache Cli

Com isso, agora podemos compilar o nosso contrato criado e verificar os detalhes como ABI e Bytecode do mesmo a partir do botão Compilation Details.

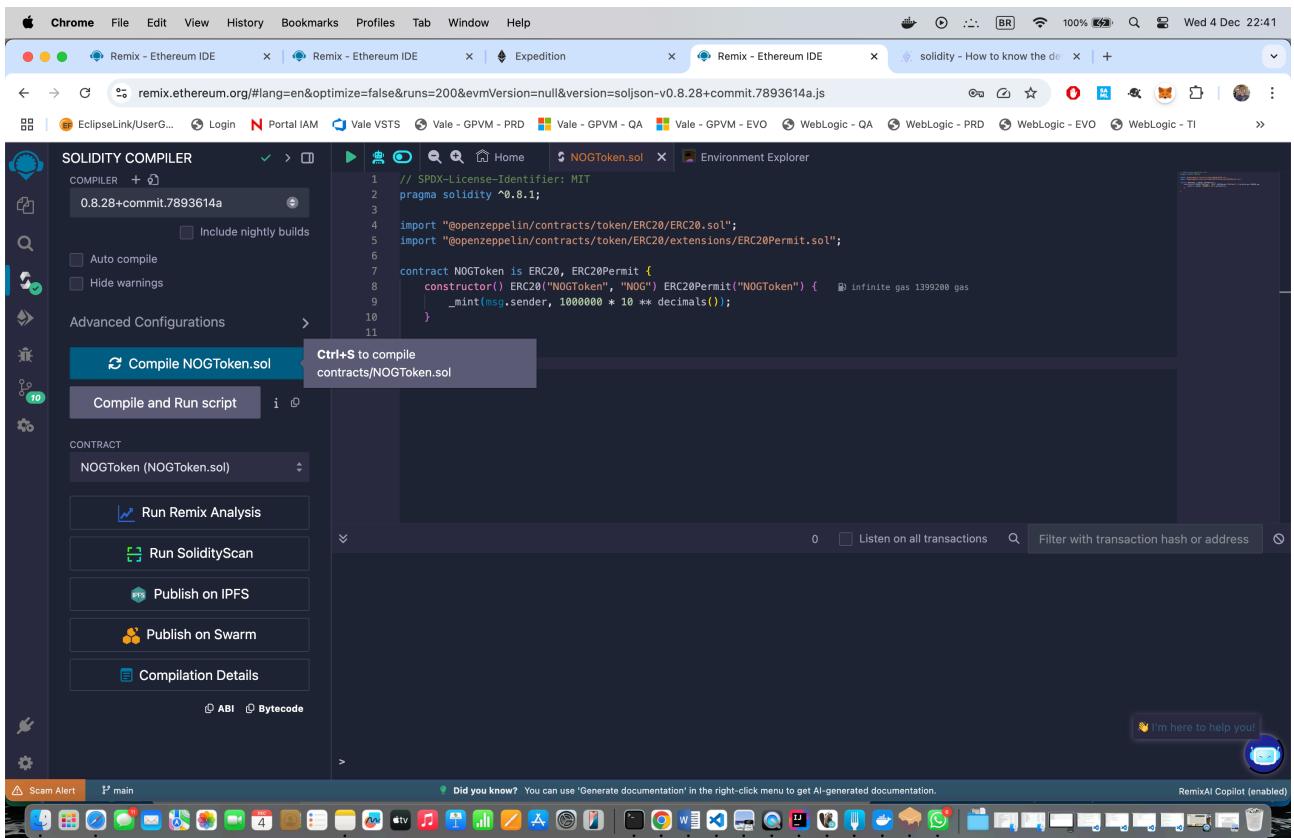


Figura 6: Compilação contrato e detalhes

Com nosso contrato compilado e sem erros, podemos verificar que o mesmo agora se encontra disponível a ser criado via deploy no Ganache. Selecionei a primeira conta Account 0 do Ganache para ser o dono `sender` dele. Também são exibidas algumas informações, como a versão da EVM que está sendo utilizada e o GAS estimado para a transação.

Implementação de Token Fungíveis na Ethereum Blockchain: Um caso de uso reprodutível

12 explorando Solidity, Ganache CLI, ERC-20 e ferramentas de desenvolvimento

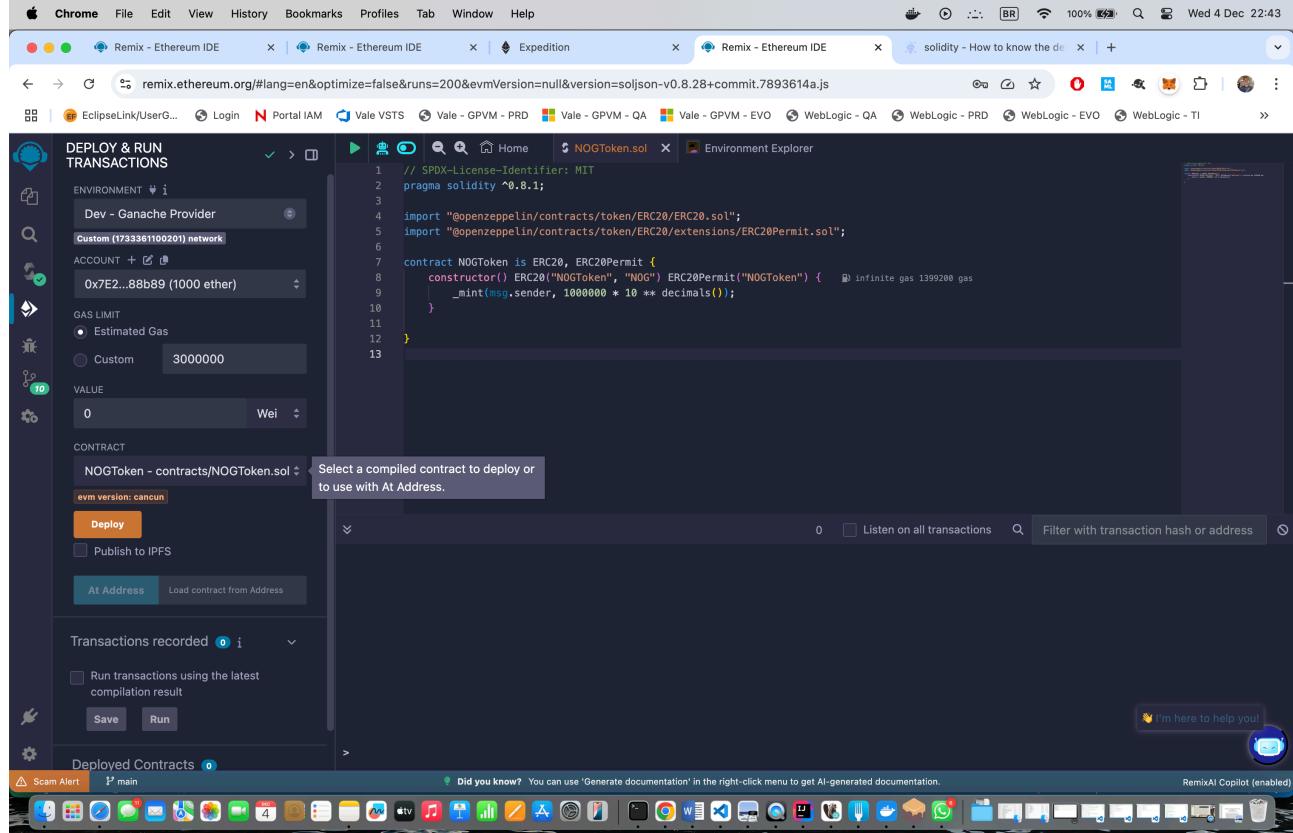


Figura 7: Deploy contrato

Ao clicarmos em Deploy, podemos observar no console do Remix, a criação do bloco 1 e mais algumas informações como o transaction hash, endereço do contrato contract address, quem enviou from e o estado status que se encontra a transação de criação do contrato, informações importantes, pois na criação do contrato já definimos um saldo, como pode ser observado na imagem Figura 1. A pessoa que o submeter será o dono dele.

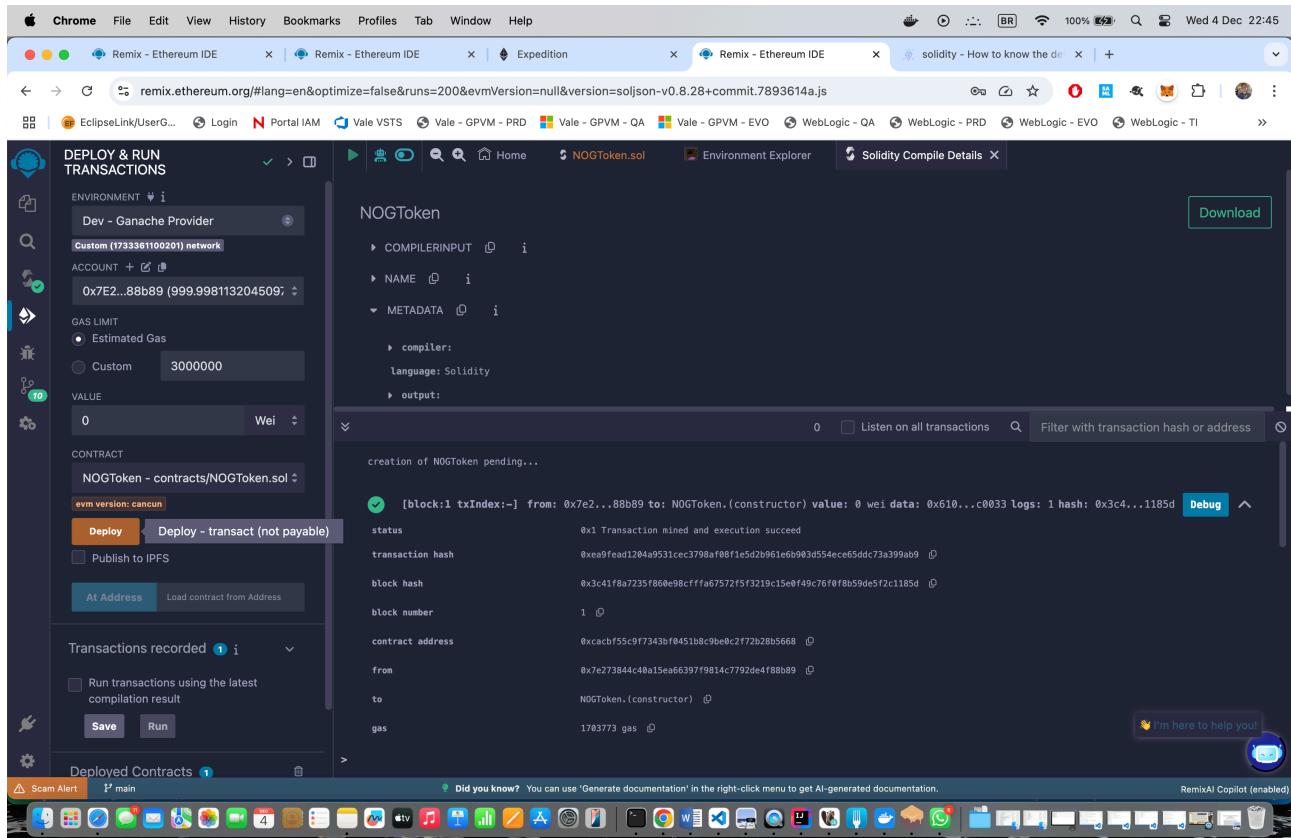


Figura 8: Efetivando o deploy do contrato no Ganache

Nós também podemos verificar as informações de criação do contrato a partir da console do Ganache Cli, como o `transaction hash`, endereço do contrato, total de GAS utilizado, bloco da criação do contrato e data e hora de criação.

Figura 9: Console Ganache Cli

Como explorador dos blocos, vou utilizar o **Expedition**, com ele podemos visualizar as informações pelo navegador, tornando a usabilidade mais amigável ao se conectar no **Ganache Cli**, já podemos observar que ele traz os blocos que possuímos na nossa blockchain.

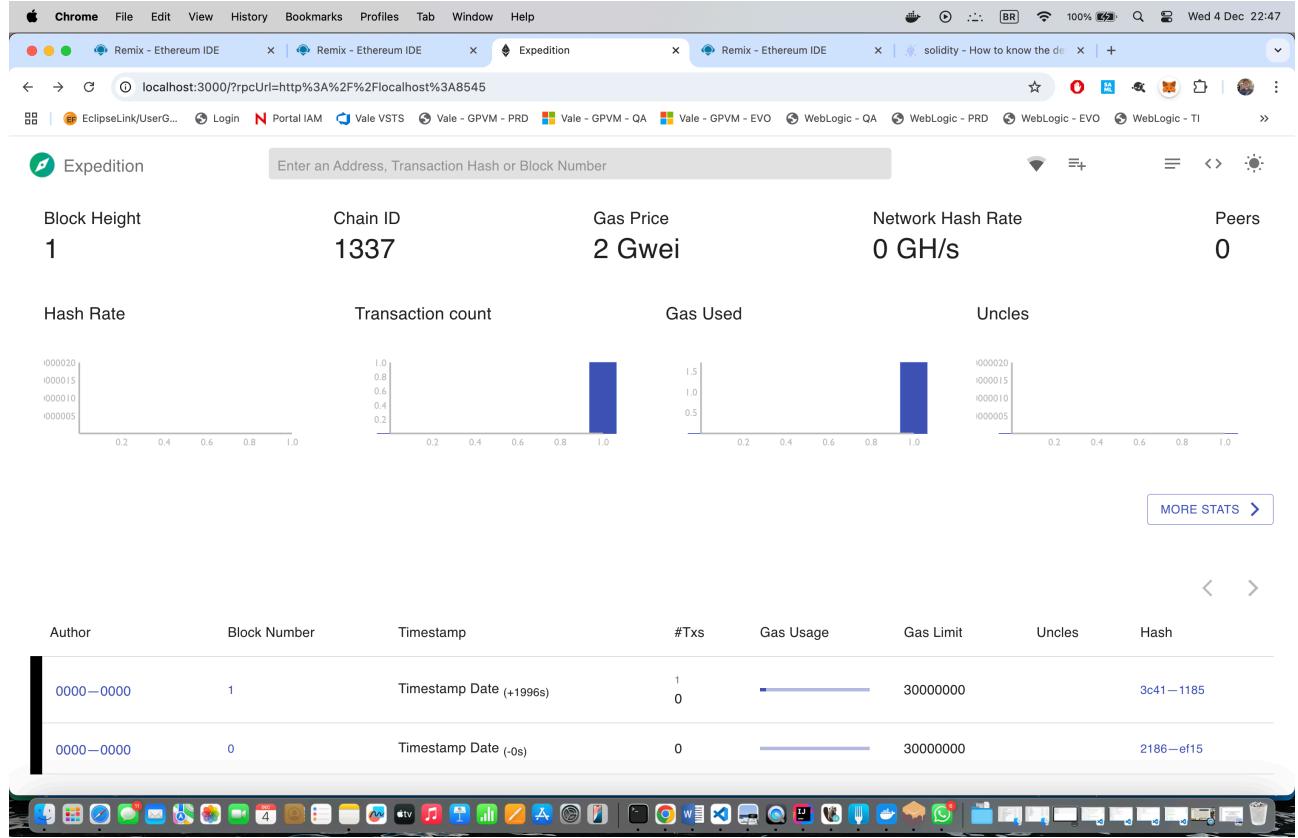


Figura 10: Expedition - Explorador de blocos

Ao analisar o bloco 1, podemos observar os detalhes da transação de criação do contrato.

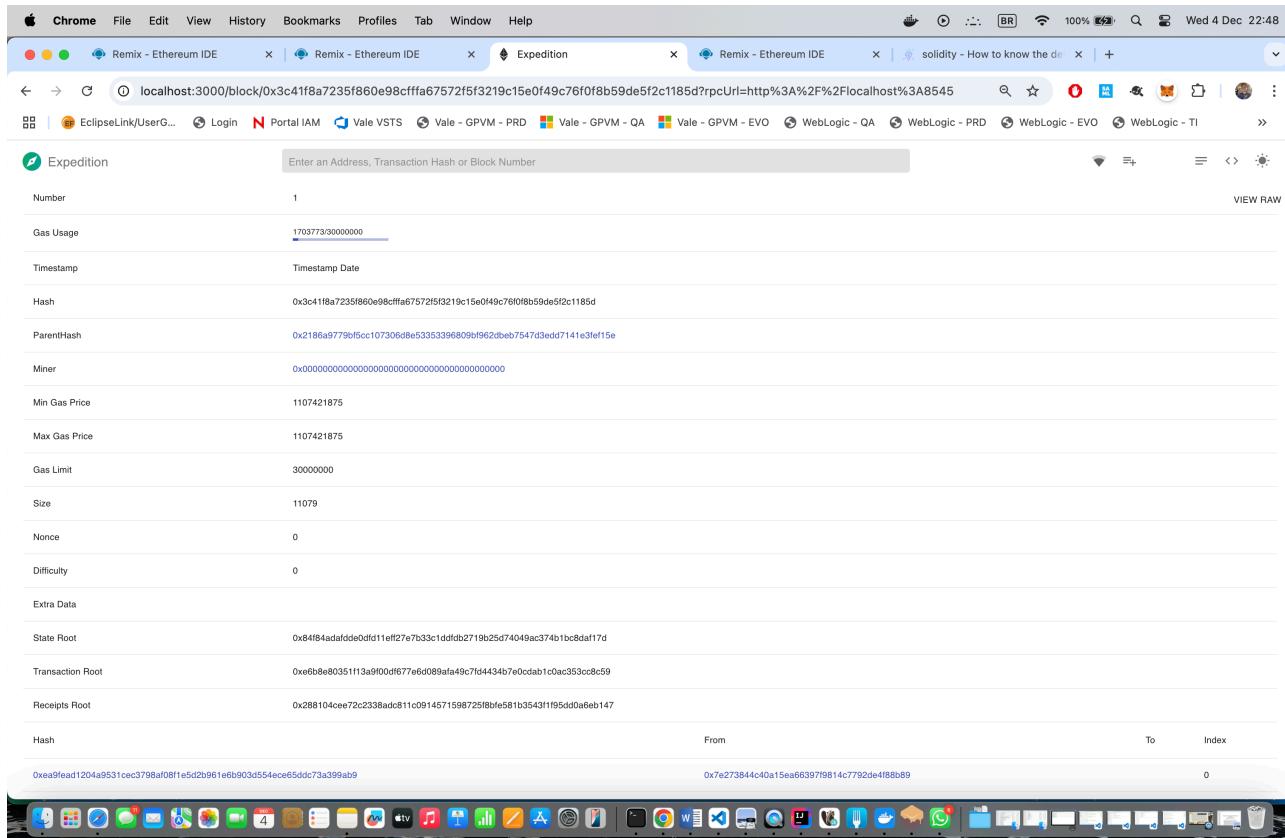


Figura 11: Análise do bloco 2, criação do contrato

Analisando o recibo da transação pelo `transaction hash` que é fornecido após a criação do contrato.

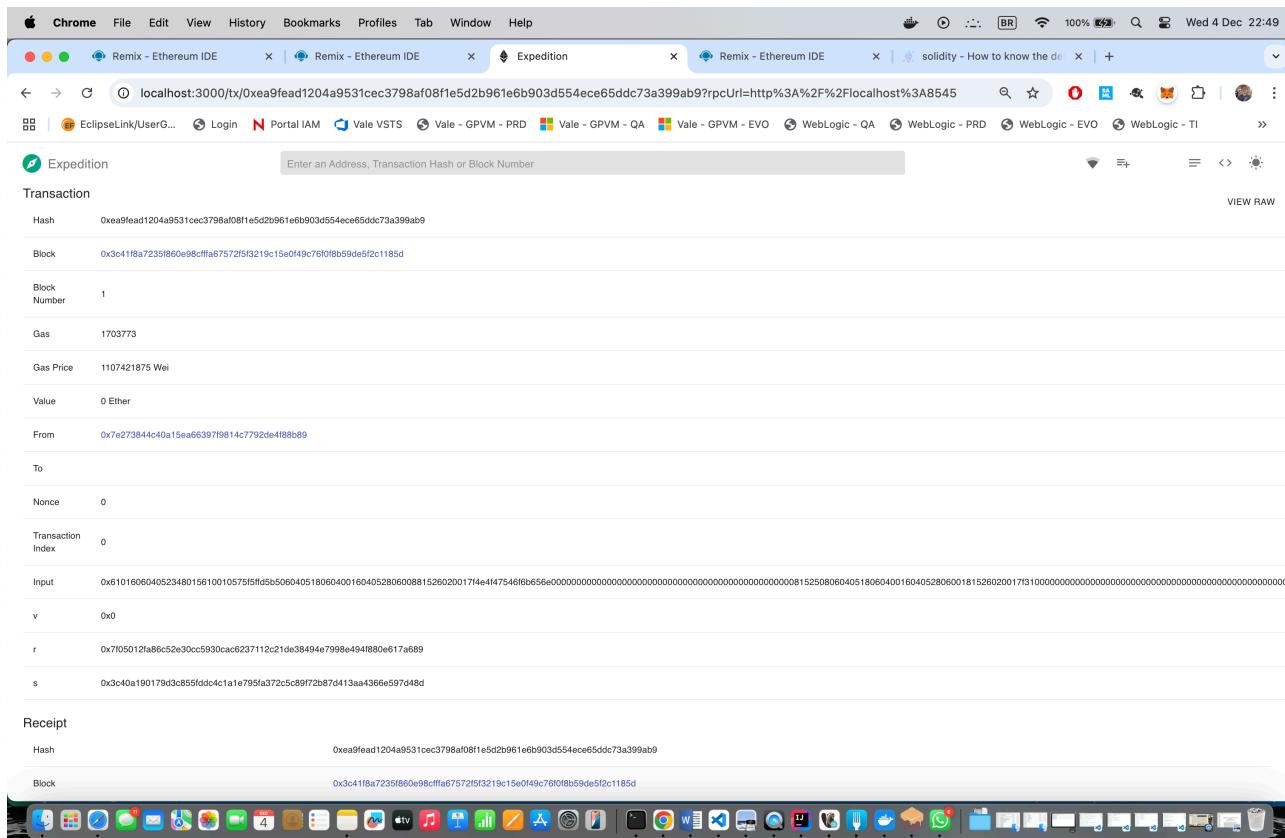


Figura 12: Analisando o recibo da transação pelo Transaction Hash

Se clicarmos no endereço fornecido em Contract address, podemos observar os detalhes do contrato.

Implementação de Token Fungíveis na Ethereum Blockchain: Um caso de uso reprodutível

16 explorando Solidity, Ganache CLI, ERC-20 e ferramentas de desenvolvimento

contrato.

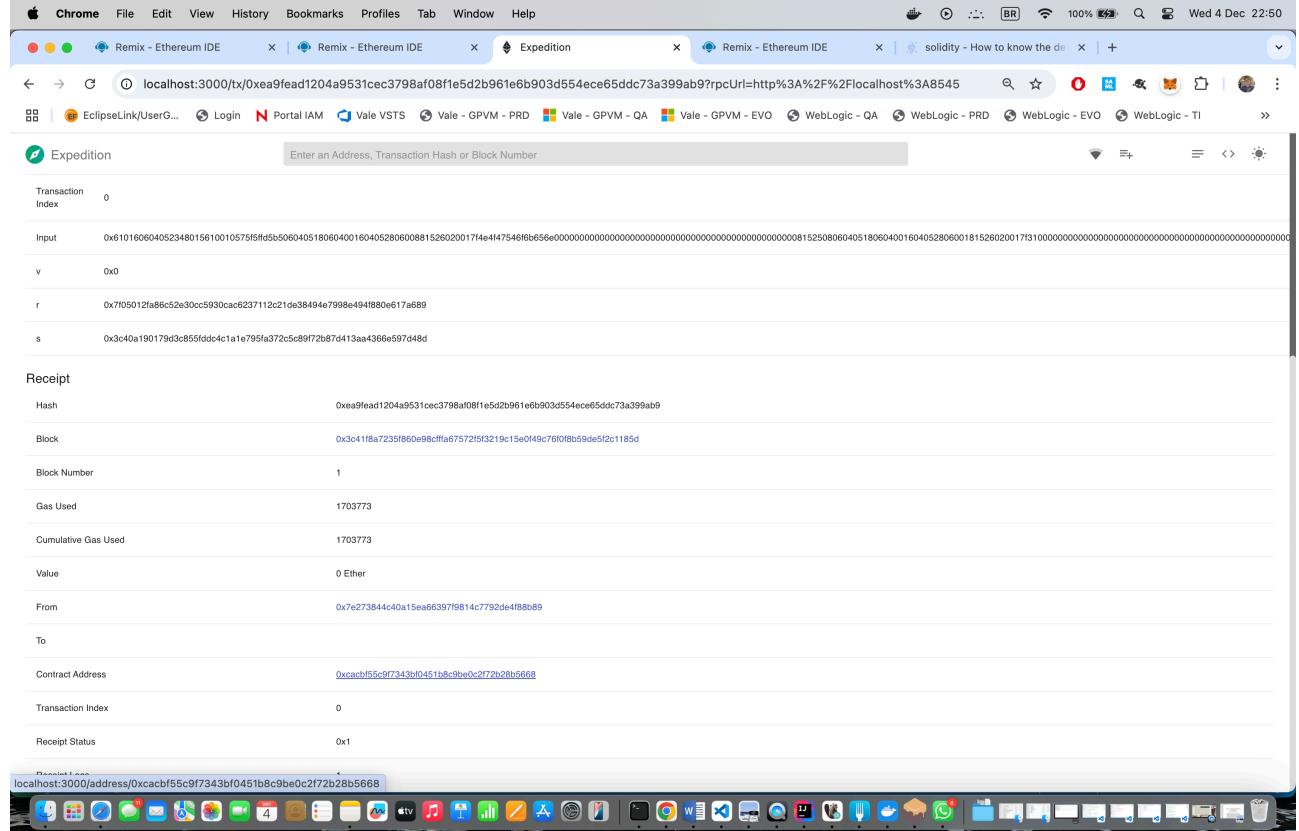


Figura 13: Endereço do contrato

Com o nosso contrato criado, saldo de token NOG em uma das contas, podemos iniciar a configuração do Metamask, como carteira digital, para efetuarmos transações entre as contas.

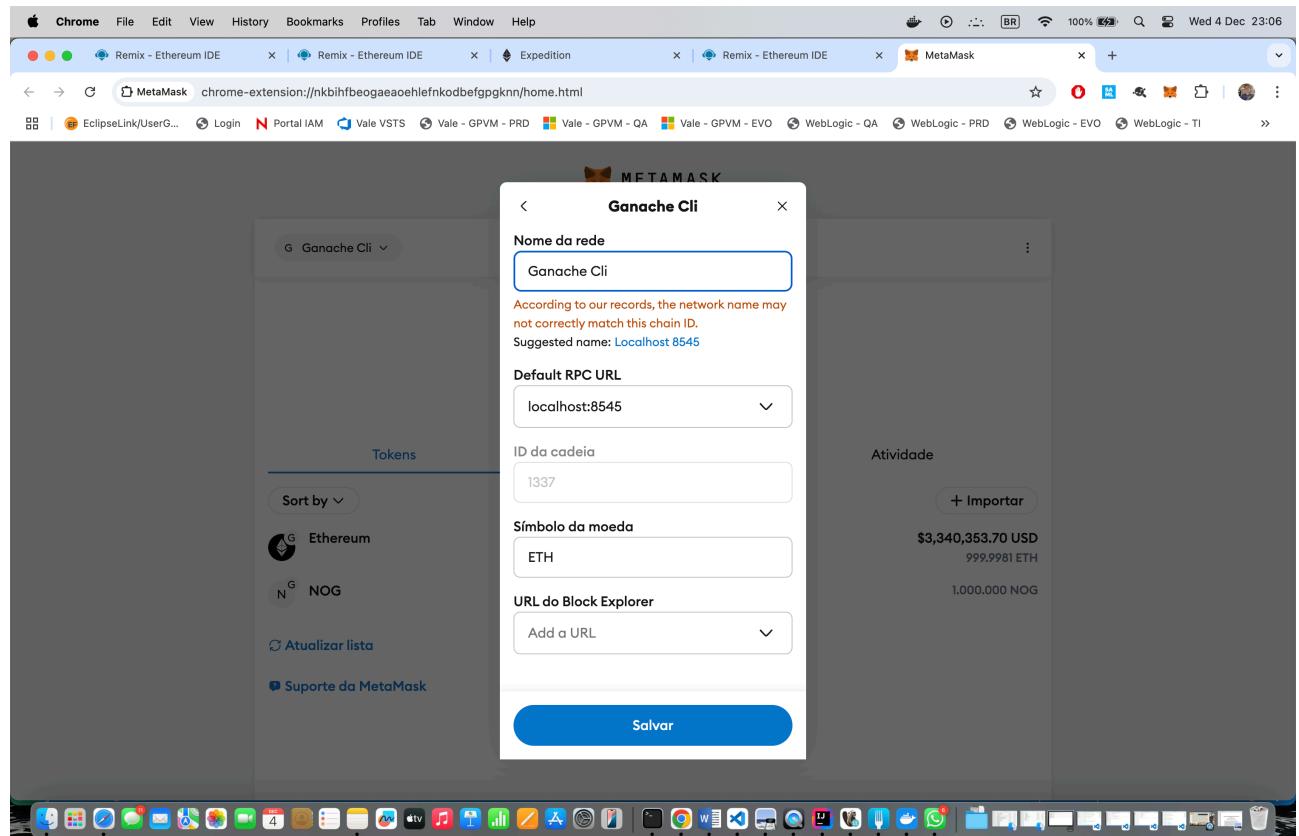


Figura 14: Conexão com o Metamask e Ganache Cli

Após criarmos a nossa conexão com a nossa blockchain, precisamos importar a `private key` das nossas contas. Estou importando da conta 1, responsável pela criação do NOGToken.

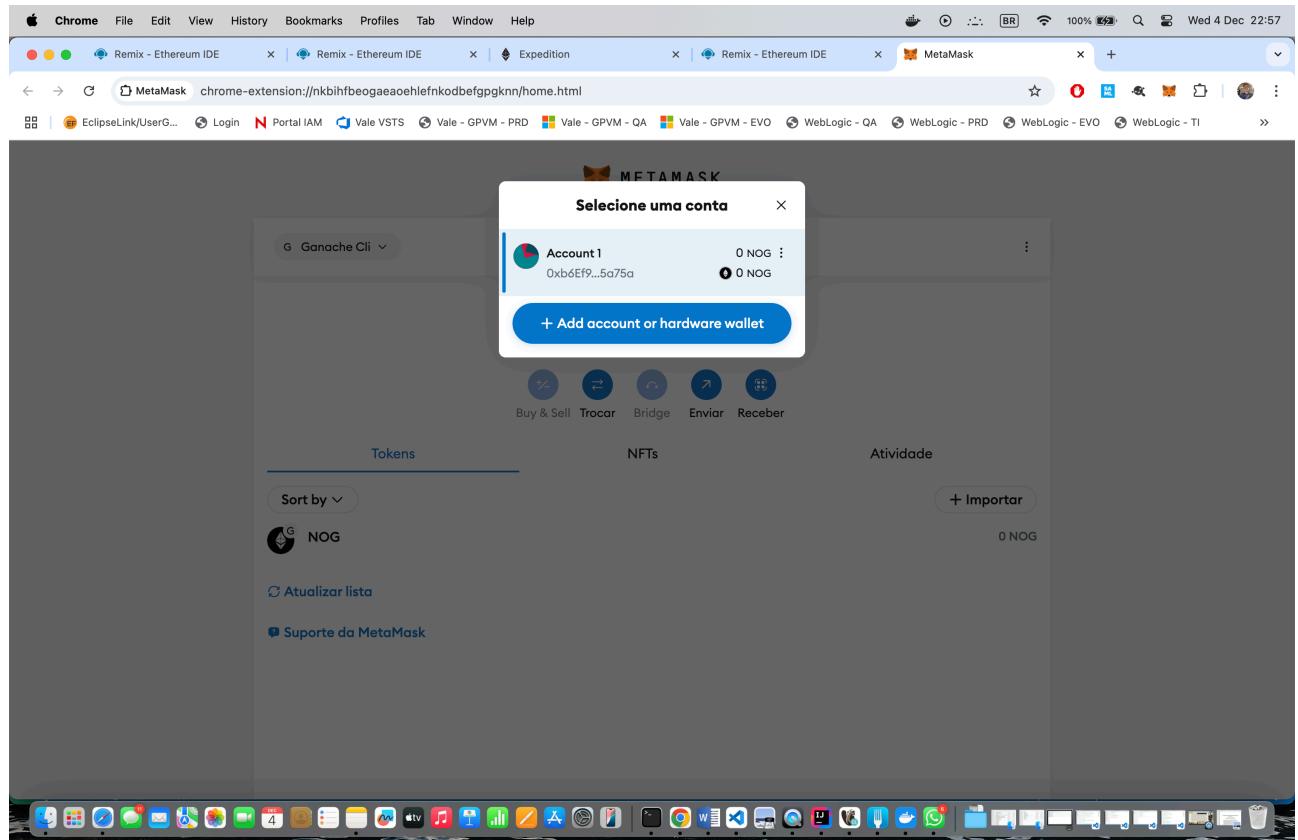


Figura 15: Importando conta no Metamask

Recuperando o valor da `private key` a partir da console do Ganache.

```
apple2: ~ julionogueira$ node /opt/homebrew/bin/ganache-cli
julionogueira -> node /opt/homebrew/bin/ganache-cli - 185x49
...
ganache v7.9.2 (ganache/cli: 0.10.2, @ganache/core: 0.10.2)
Starting RPC server
Available Accounts
=====
(0) 0x7e273844C40a15Ea66397F9814c7792DE4f88b89 (1000 ETH)
(1) 0xD7B3D9e459820E45DF7FcCc5f6f1f86a0d91De7 (1000 ETH)
(2) 0x0dc42178CC7516A4c530Ac943EeF8AD8A27b (1000 ETH)
(3) 0x5f913501c1a8881AcCe1F4Ef2bc4330388b12483 (1000 ETH)
(4) 0xcdE845f4cd77acA2DfB9a326867a248f67505 (1000 ETH)
(5) 0xA0A29A23b76f378DBFA53E9E793Ca7d7df223a0c8 (1000 ETH)
(6) 0x87C19f0eC8cF23B5Af0bd4e5eEa66CA3D11f5A (1000 ETH)
(7) 0x6140702f03f10b2c2405A3Df192441f1fbba76eA (1000 ETH)
(8) 0x52B2c4443958835E799B833Cc04fb130Bf0350A (1000 ETH)
(9) 0x3e1bcD4B85128102b8226E79D41CC730B041d614 (1000 ETH)

Private Keys
=====
(0) 0x77ac5f75dd47351f16064d6777968c53d0ed63ae4daeeec387cae0e6c757bd9c6
(1) 0x1caa1c1d1cc5d9b1597c2f484a9884a3838c0e7ed5aceb5e01b98dacfe9a4c97d0
(2) 0x880688fc45f401947f078748894522a8f0ea1246d7418e5c26e817f5fdfe07
(3) 0x7cd1c4017a617203049e239fcf33a1c17e7c6c237aae3961ca9a149dc34d517f
(4) 0x23a84c733e4c1b33be23f5984115543830eb33723be7f646806b3369ce6572
(5) 0xbef7d1c77e57b95443315ea1a3239a981b027925bde2a6d3dd6520d68c178eb56
(6) 0xbaf600b0b679f7debf528cd488e86572a7471f8c350e9910dc4da149523ca2c13
(7) 0x6eef59667f2ef79f6871b2fa462feac820178826a0d1324b02c7b3a99046cf
(8) 0xae59171a3083e9ff4df2468b76a26220fa519258ac45f3d889168d98b4d0a51
(9) 0x3e199236918654c2704b2305ae58b000fcfa7325b52/c105e/a0a53286235436

HD Wallet
=====
Mnemonic: scan twin comfort rule history return wide diary height frown wine south
Base HD Path: m/44'/60'/0'/0/{account_index}

Default Gas Price
=====
2000000000

BlockGas Limit
=====
30000000

Call Gas Limit
=====
50000000

Chain
=====
Hardfork: shanghai
```

Figura 16: Recuperando private key Ganache

Com o valor da chave, podemos importar direto no formulário de adicionar contas.

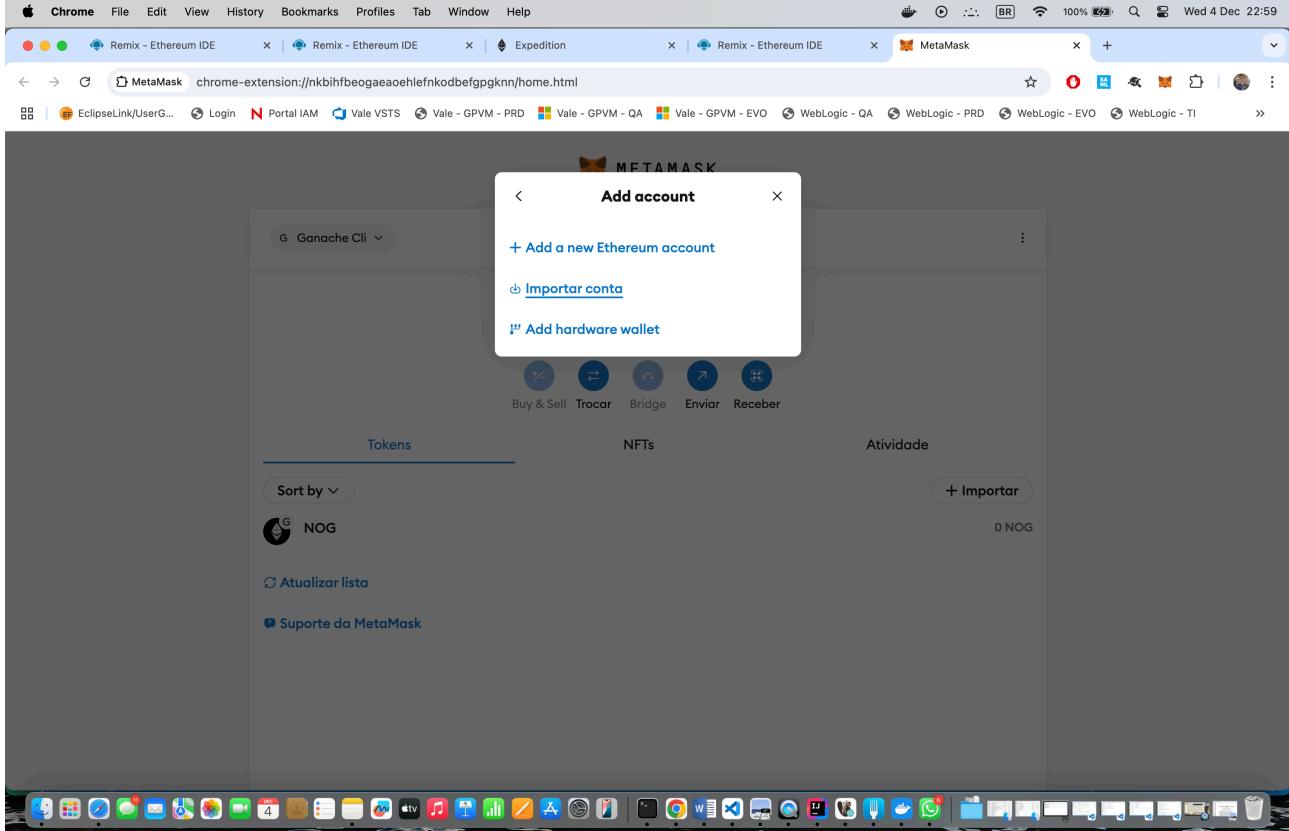


Figura 17: Importando no Metamask a private key

Insira o valor e clique em importar para que sejam carregadas as informações de saldo da conta.

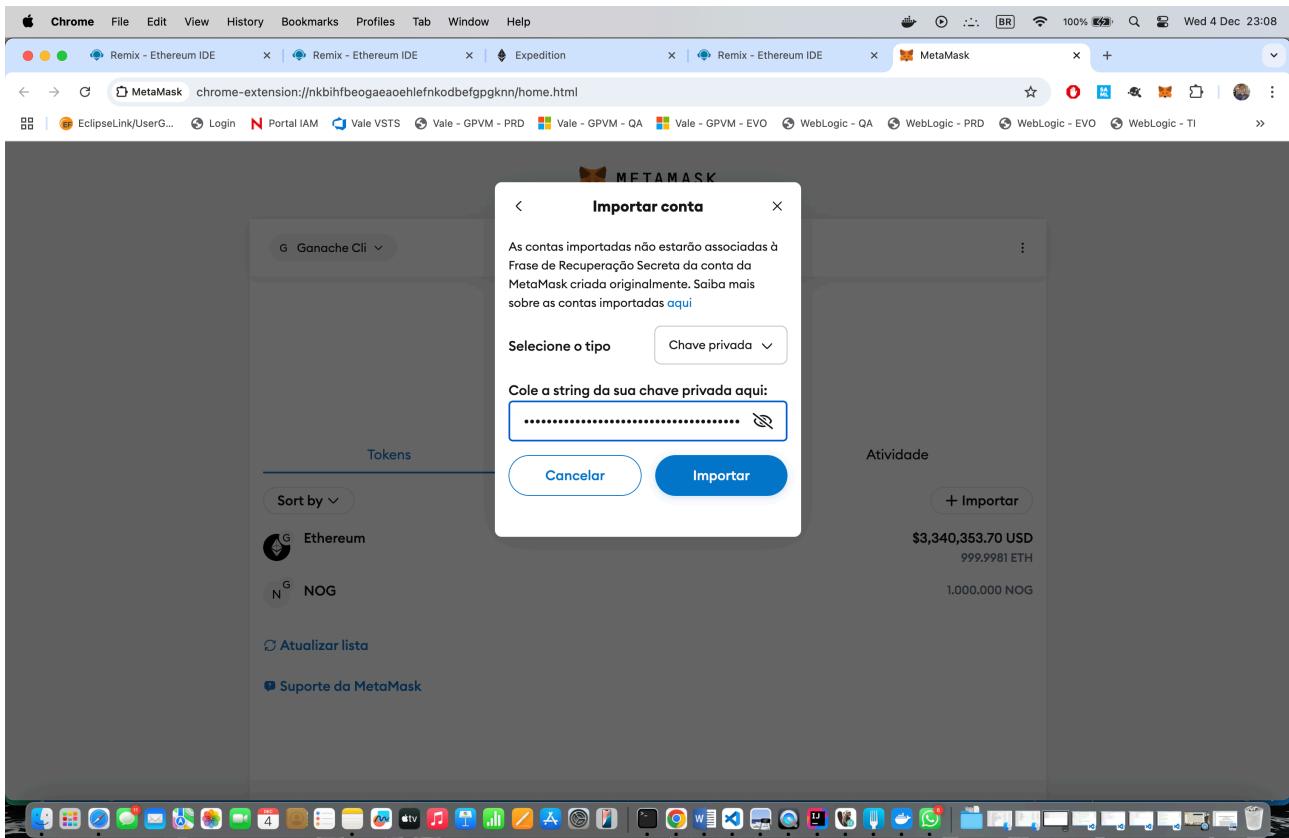


Figura 18: Importando conta 1 do Ganache, pela private key

Após importar, ele já exibe as informações de saldo e a moeda da blockchain. Como estamos usando a Ethereum, ele está exibindo o saldo atual.

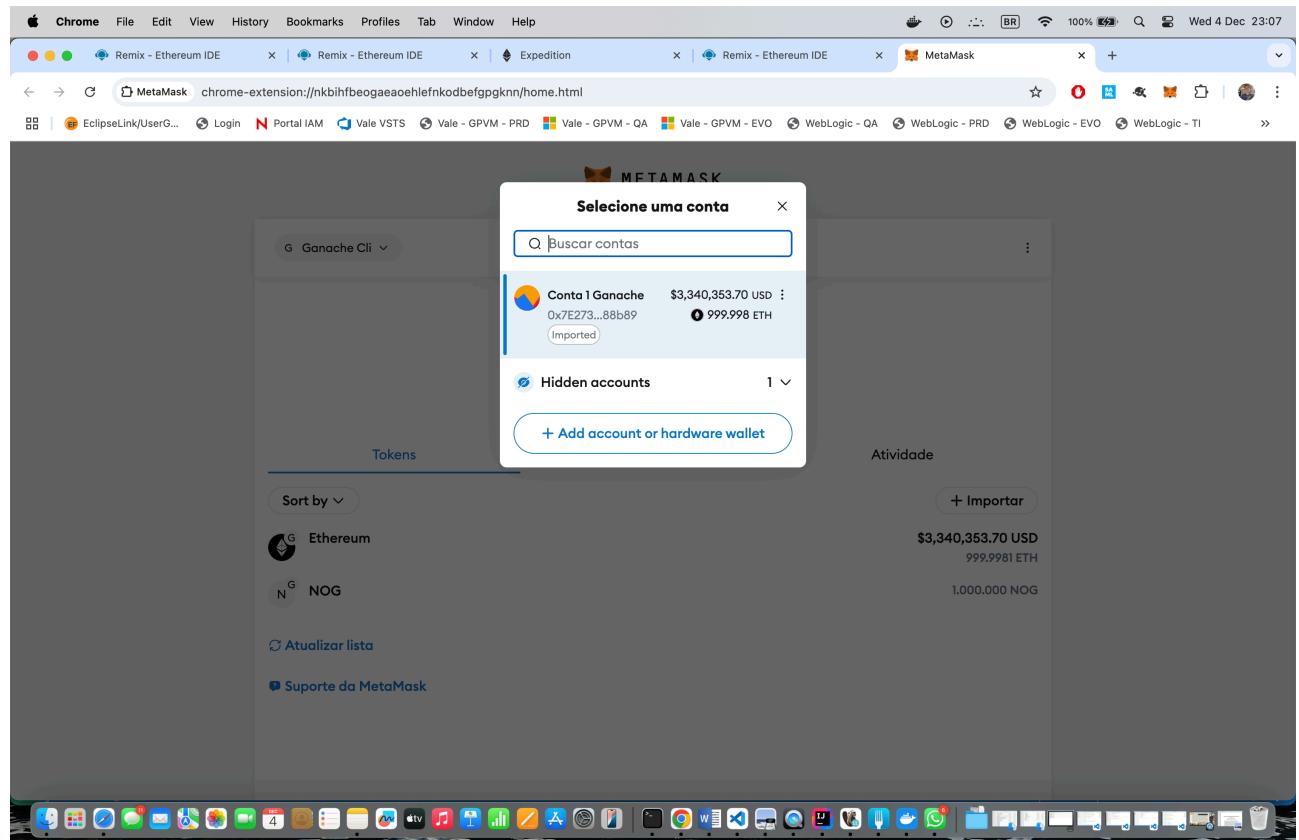


Figura 19: Conta Ganache

Na aba de token, após a importação do token criado, fornecendo o endereço do contrato, como podemos recuperar, ou pela console do Ganache Cli ou do Expedition, observando o recibo da transação, será carregada a nossa criptomoeda criada e o saldo dela na conta do dono do contrato.

Implementação de Token Fungíveis na Ethereum Blockchain: Um caso de uso reprodutível

20 explorando Solidity, Ganache CLI, ERC-20 e ferramentas de desenvolvimento

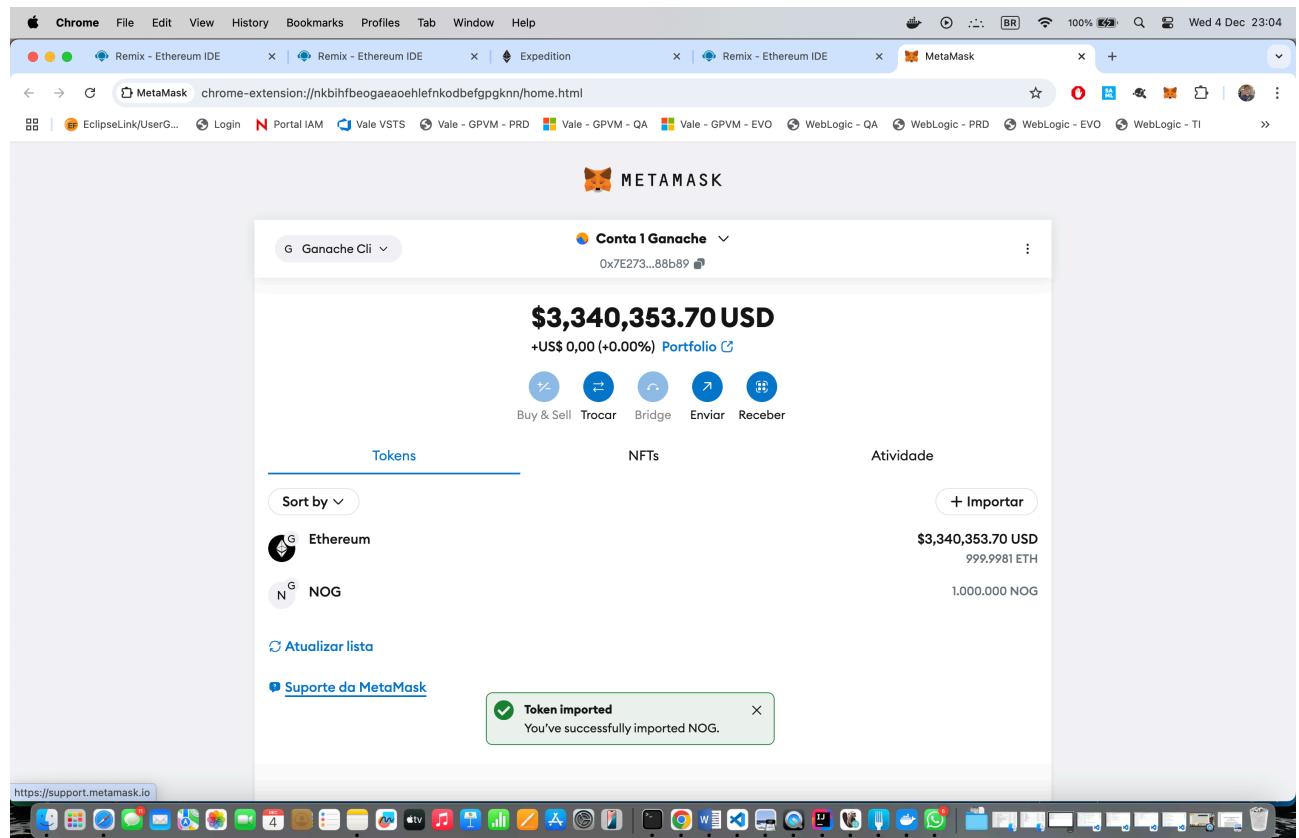


Figura 20: Endereço do contrato

Agora iremos importar uma segunda conta que será usada para transferência de token.

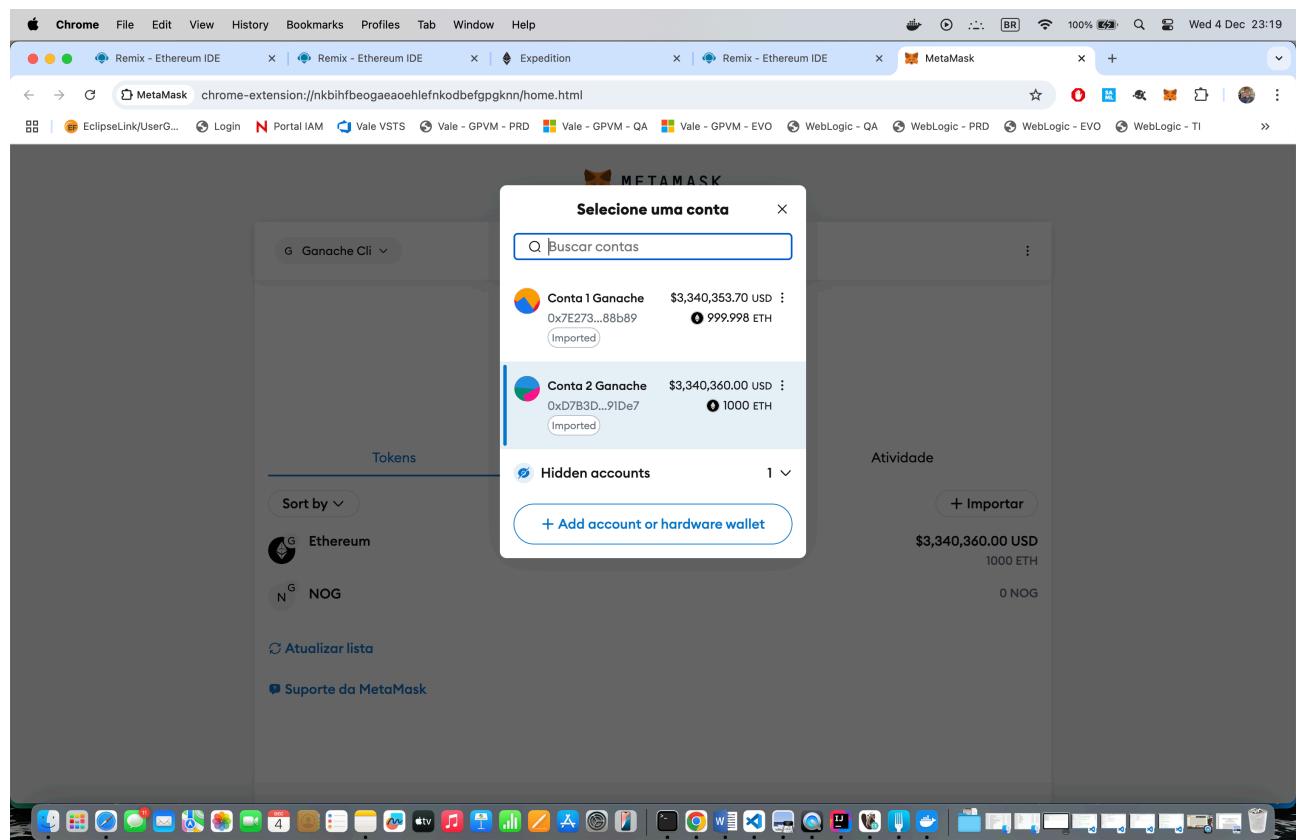


Figura 21: Segunda conta Ganache

Ao clicar em enviar, podemos configurar a nossa transação em De:/Para:, sendo o dono do contrato a conta a enviar, pois apenas ela possui saldo de NOG, e a segunda conta quem irá receber

os valores.

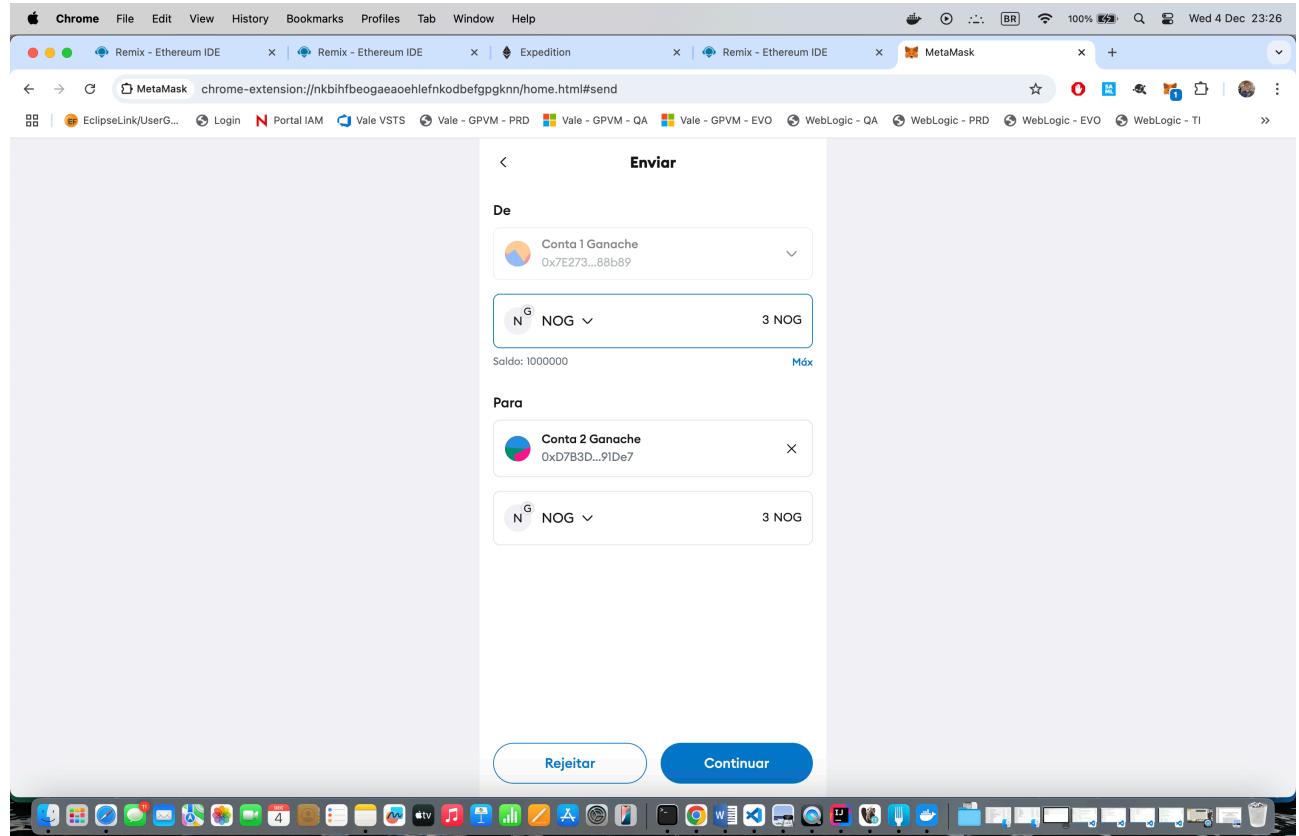


Figura 22: Transação

Revisão o envio de NOG token, após configurado o valor da transação, precisamos estimar o mínimo de GAS para que a transação seja efetuada.

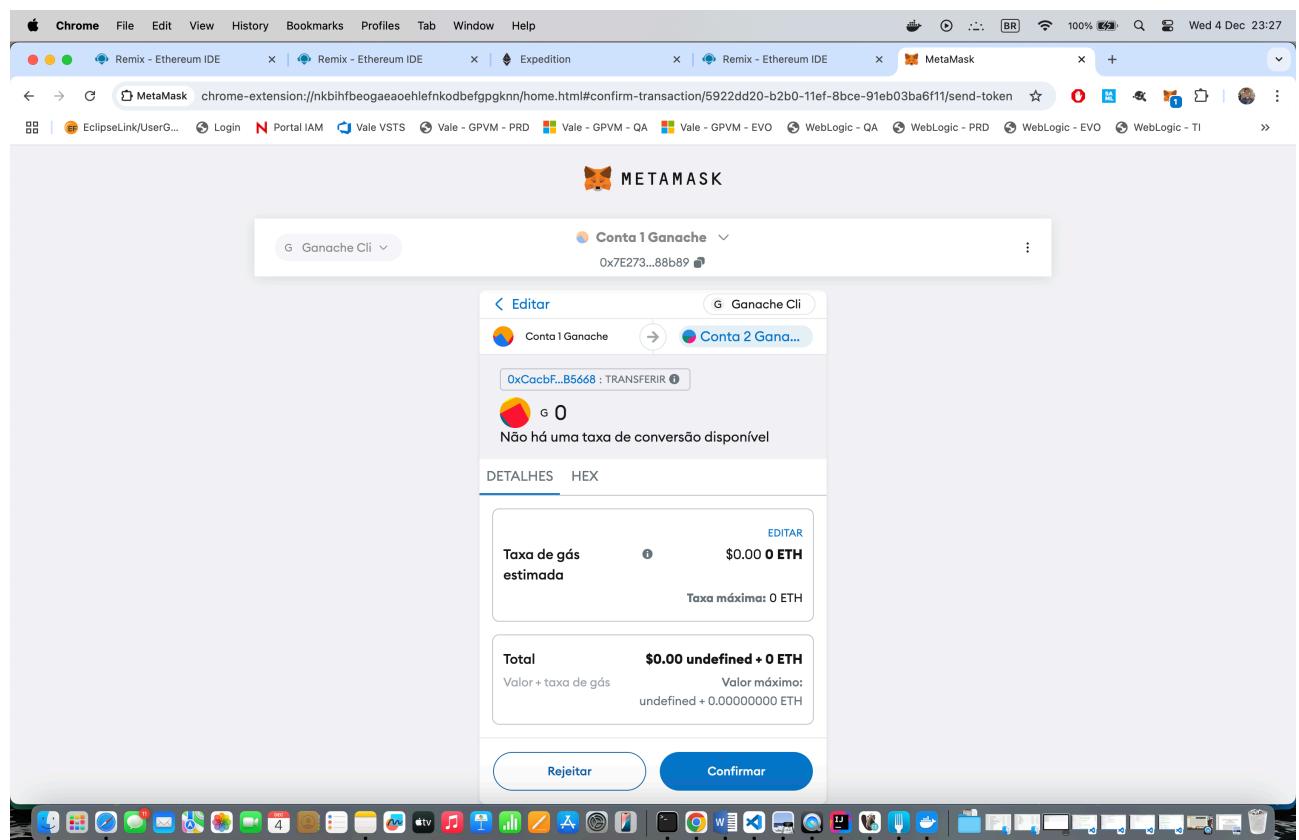


Figura 23: Revisão da transação

Adicionar o mínimo de 01 GAS wei para que seja possível à blockchain minerar e efetivar a transação.

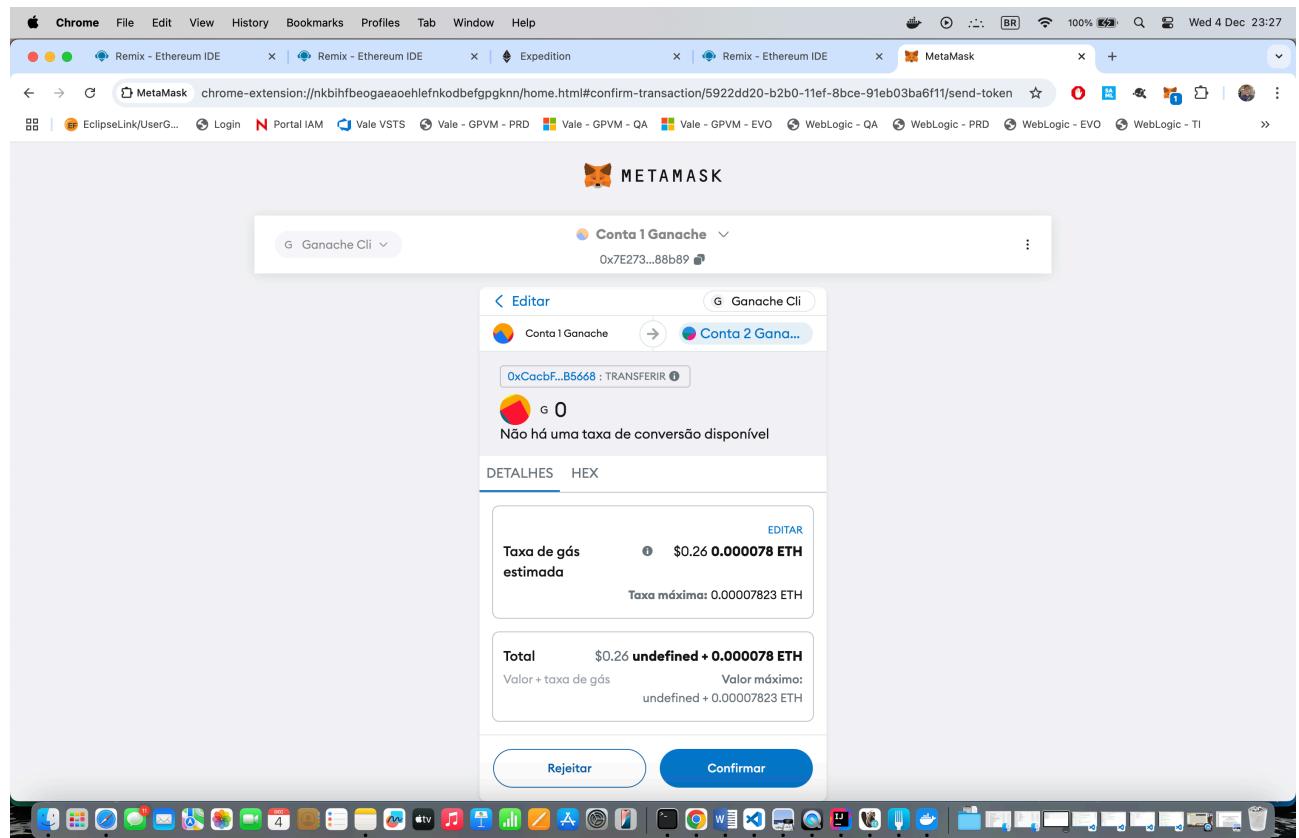


Figura 24: Configurando o mínimo de 01 GAS wei

Após se confirmar a transação, ela ainda fica no estado de pendente, aguardando que seja minerada e efetivada.

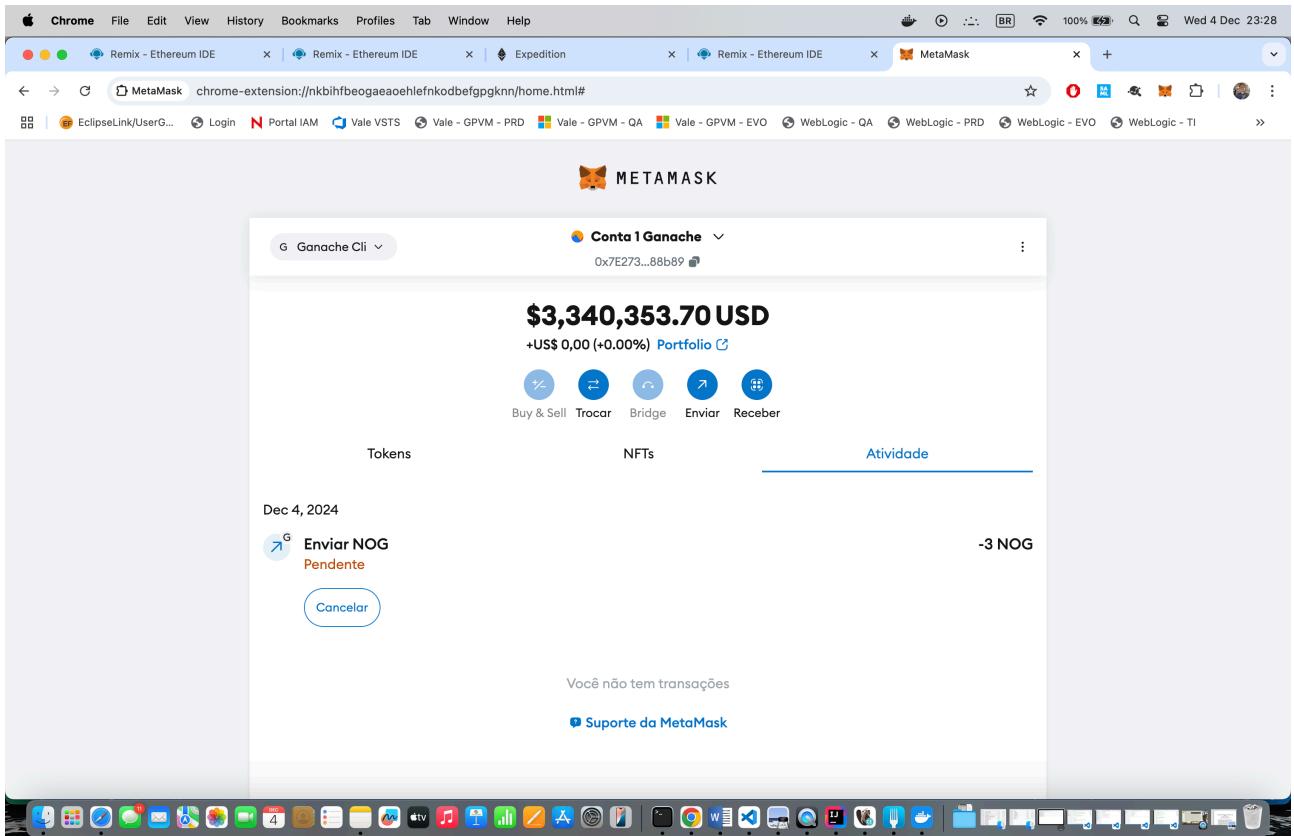


Figura 25: Transação aguardando mineração

Alguns segundos após a transação, será efetivada.

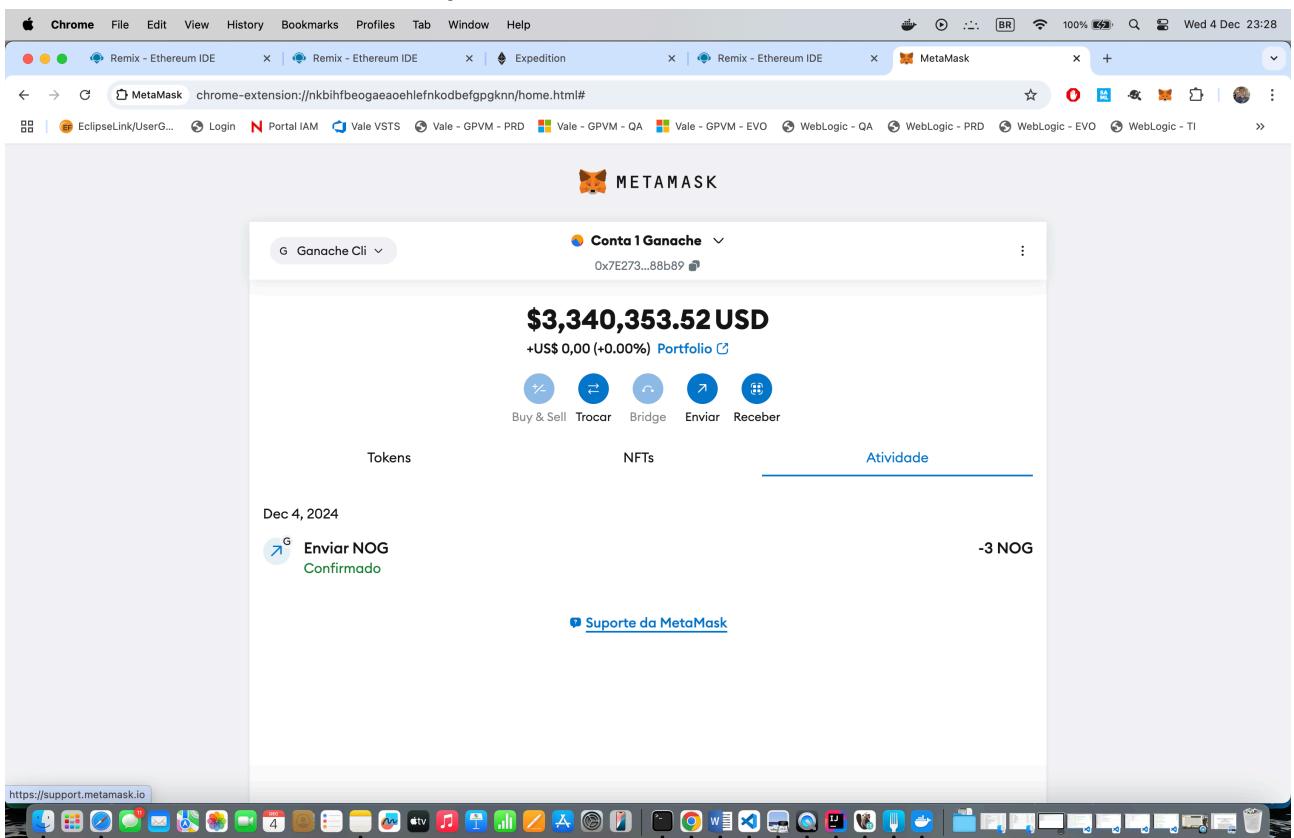


Figura 26: Transação confirmada

Também pode ser confirmado na console do Ganache Cli a transação, tal como o endereço do recibo dela e a data e hora da efetivação.

Figura 27: Console Ganache Cli, bloco

Expedition			
Enter an Address, Transaction Hash or Block Number			
Number	2		
Gas Usage	52155/30000000		
Timestamp	Timestamp Date		
Hash	0xd5848490bd2dd88205a6577fa4ce5087fc4b0c6f58fde1bcaab06ea243272e3		
ParentHash	0x3c41fb7235f860e98cffa7572f53219c15e0f49c76ffbf59de5f2c1185d		
Miner	0x000		
Max Gas Price	1000000000		
Gas Limit	30000000		
Size	724		
Nonce	0		
Difficulty	0		
Extra Data			
State Root	0x0b851dccffba4f3a15614d99fb1db1f9919a53351c5279d8043c6bbf5e24da		
Transaction Root	0x4cac2b848178aaee99ead90e127979b5a6c01cbf8d44829bc2993661456095		
Receipts Root	0x3ae7f8712b3f3e69dd7e03a5b33614d8dbbb4d0d87f4564daee877dd5544c91f		
Hash	From	To	Index
0x446700d50c22d5752d9f30159ff0d2659bb73191237cf8607ee03c7db396554c	0xe273844c40a15ea663979b814c7792de4f88b89	0xacabf55c9f7343b0451b8c9be0c2f72b2b5668	0

Figura 28: Bloco da transação que efetuou a transação de NOG Token

Usando o **Expedition**, podemos observar os detalhes da transação, com os endereços da primeira conta **From:** e o da segunda conta **To:**. Também podemos observar que não são utilizados ou enviados valores de **ETH**, pois o token usado foi o **NOG** criado para a atividade.

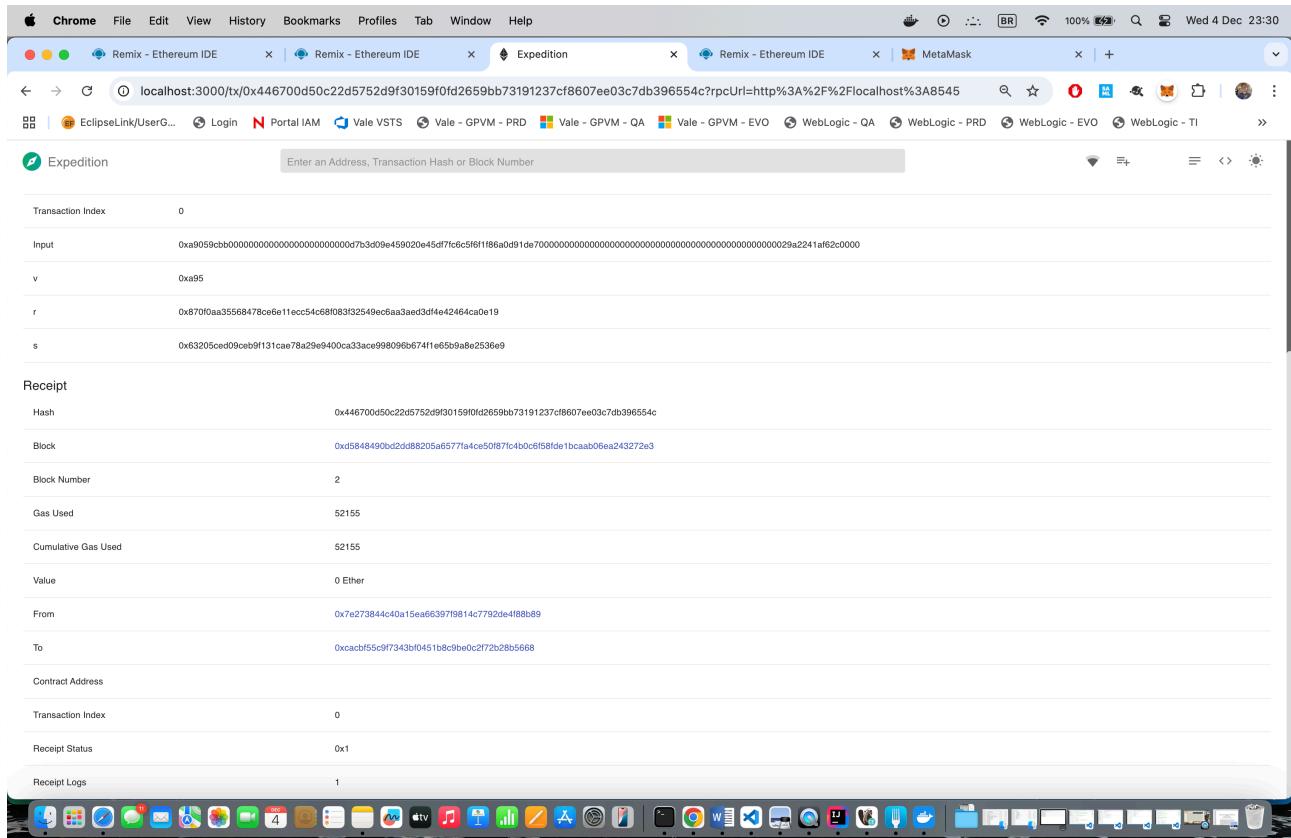


Figura 29: Bloco da transação que efetivou a transação de NOG Token

Podemos observar no saldo atual da primeira conta, dona do NØG token, o saldo atualizado sem o valor enviado na transação, conforme imagem Figura 26.

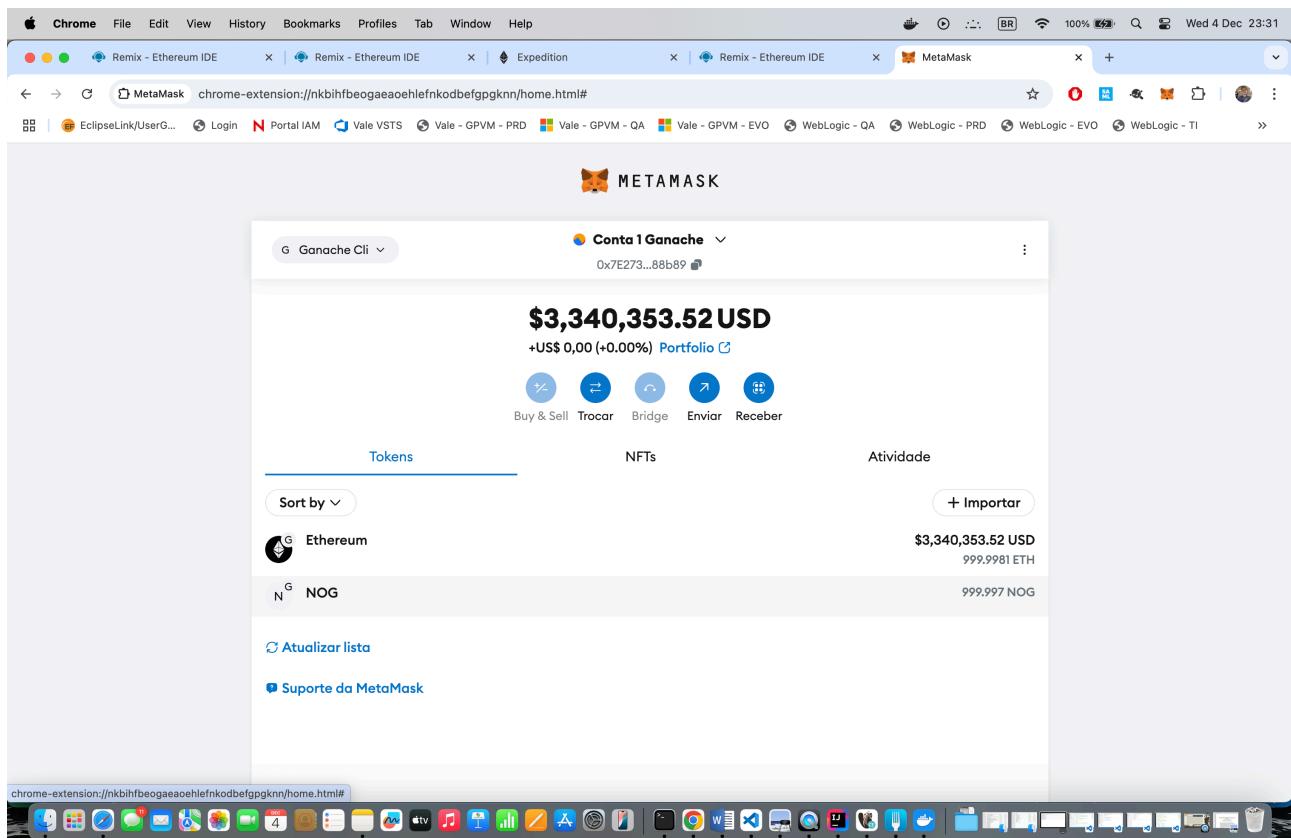


Figura 30: Saldo atualizado do envio de NOG Token

Na segunda conta agora podemos observar o saldo de NOG token atualizado com o valor que

foi enviado na transação.

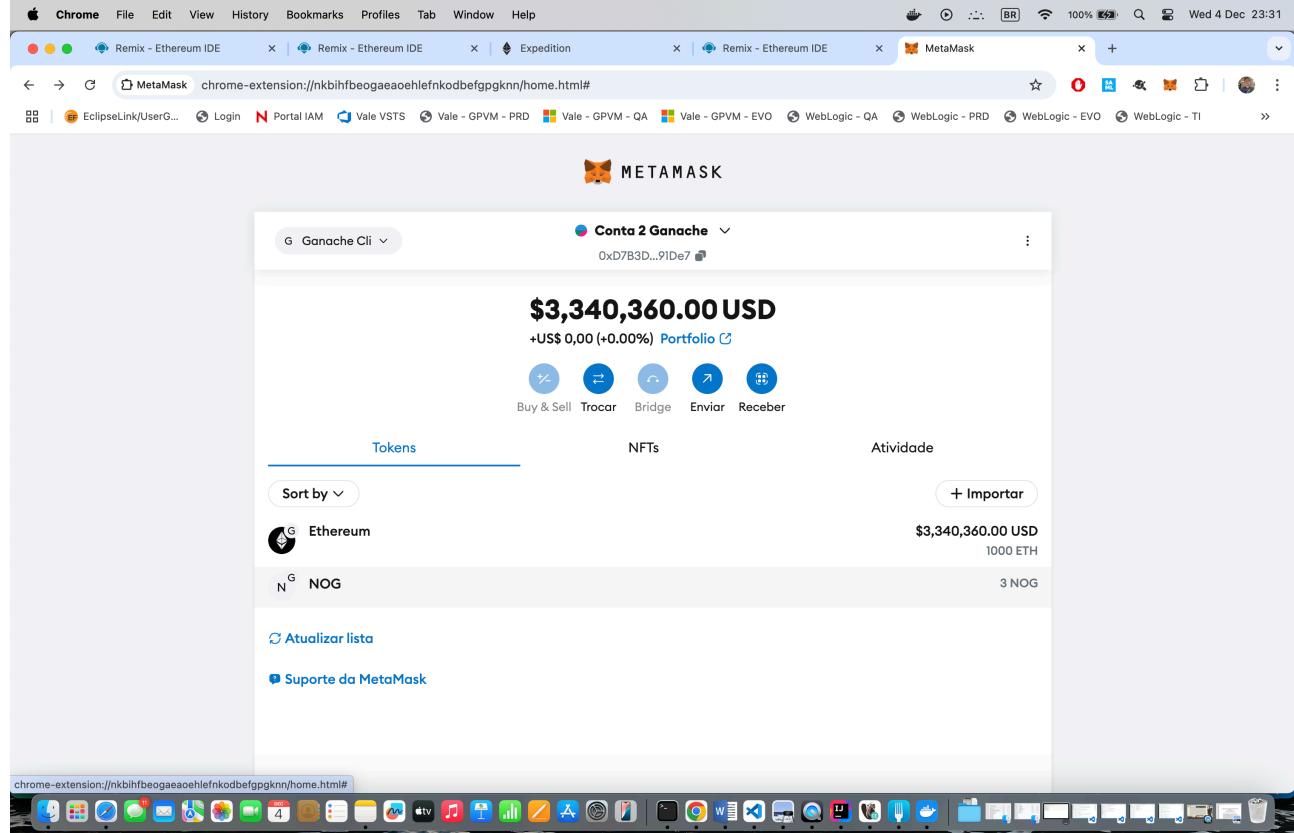


Figura 31: Recebimento de 3 NOG token

7 Códigos

O Código 1 apresenta o contrato do token NOGToken NOG criado usando a biblioteca OpenZeppelin, usando o padrão ERC20. O mesmo também se encontra versionado no link <https://github.com/nogueirjr/NOGToken>

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.1;
3
4 import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
5 import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol";
6
7 contract NOGToken is ERC20, ERC20Permit {
8     constructor() ERC20("NOGToken", "NOG") ERC20Permit("NOGToken") {
9         _mint(msg.sender, 1000000 * 10 ** decimals());
10    }
11 }
12 }
```

Código 1: Código criação contrato NOGToken

Referências

- Brunner, Robert. 2024. Blockchains, tokens, and the decentralized future. <https://www.coursera.org/learn/blockchains-tokens>.
- Buterin, Vitalik. 2014. Ethereum white paper. Accessed: 2024-12-03. <https://ethereum.org/en/whitepaper/>.
- Buterin, Vitalik. 2015. Ethereum white paper. Accessed: 2024-12-03. <https://ethereum.org/en/whitepaper/>.
- Fabian Vogelsteller, Vitalik Buteri. 2015. Erc-20 token standard. <https://eips.ethereum.org/EIPS/eip-20>. Acessado em [data de acesso].
- Foundation, Ethereum. 2024a. Ethereum 2.0 and proof of stake. Accessed: 2024-12-03. <https://ethereum.org/en/eth2/>.
- Foundation, Ethereum. 2024b. Ethereum: A platform for smart contracts and decentralized applications. Accessed: 2024-12-03. <https://ethereum.org/en/>.
- Foundation, Ethereum. 2024c. Ethereum documentation. Accessed: 2024-12-03. <https://ethereum.org/en/developers/docs/>.
- Foundation, Ethereum. 2024d. Ethereum documentation: Erc-20 standard. Accessed: 2024-12-03. <https://ethereum.org/en/developers/docs/standards/tokens/erc-20/>.
- Foundation, Ethereum. 2024e. Ethereum networks. Accessed: 2024-12-03. <https://ethereum.org/en/developers/docs/networks/>.
- Foundation, Ethereum. 2024f. Ethereum security best practices. *Ethereum Blog* <https://blog.ethereum.org/2024/01/07/ethereum-security-best-practices/>. Accessed: 2024-12-03.
- Foundation, Ethereum. 2024g. Solidity compiler: Understanding bytecode and deployment. *Ethereum Documentation* <https://ethereum.org/en/developers/docs/solidity/>. Accessed: 2024-12-03.
- Foundation, Ethereum. 2024h. What is the ethereum virtual machine (evm)? Accessed: 2024-12-03. <https://ethereum.org/en/developers/docs/evm/>.
- Hoffman, Paul. 2024. Introduction to blockchain technologies. Tech. Rep. OCTO-040 ICANN. <https://www.icann.org/en/system/files/files/octo-040-17oct24-en.pdf>.
- OpenZeppelin. 2024a. Automation with openzeppelin. Accessed: 2024-12-03. <https://openzeppelin.com/>.
- OpenZeppelin. 2024b. Openzeppelin and ganache cli. Accessed: 2024-12-03. <https://docs.openzeppelin.com/cli/2.8/getting-started>.
- OpenZeppelin. 2024c. Openzeppelin docs. Accessed: 2024-12-03. <https://docs.openzeppelin.com/>.
- OpenZeppelin. 2024d. Openzeppelin hardhat plugin. Accessed: 2024-12-03. <https://docs.openzeppelin.com/upgrades-plugins/1.x/hardhat-upgrades>.
- OpenZeppelin. 2024e. Openzeppelin remix plugin. Accessed: 2024-12-03. <https://docs.openzeppelin.com/defender/remix-plugin>.
- Szabo, Nick. 1997. The idea of smart contracts. *Nick Szabo's Research* <http://www.fon.hum.uva.nl/rob/courses/ai/lectures/contract.pdf>. Accessed: 2024-12-03.
- Team, Expedition. 2024a. Expedition documentation. Accessed: 2024-12-03. <https://expedition.dev/>.
- Team, Hardhat. 2024b. Hardhat: Ethereum development environment. Accessed: 2024-12-03. <https://hardhat.org/>.
- Team, Remix. 2024c. Remix ide. Accessed: 2024-12-03. <https://remix.ethereum.org/>.
- Team, Solidity. 2024d. Solidity documentation. Accessed: 2024-12-03. <https://soliditylang.org/>.
- Team, Truffle. 2024e. Ganache cli. Accessed: 2024-12-03. <https://www.trufflesuite.com/>

ganache.

- Team, Truffle. 2024f. *Truffle suite: Smart contract development framework*. Truffle Inc. <https://www.trufflesuite.com/>. Accessed: 2024-12-03.
- Team, Truffle Suite. 2024g. Ganache cli documentation. Accessed: 2024-12-03. <https://trufflesuite.com/ganache/>.
- Vogelsteller, Fabian & Vitalik Buterin. 2015. Eip-20: Token standard. Accessed: 2024-12-03. <https://eips.ethereum.org/EIPS/eip-20>.
- Wood, Gavin. 2014. Ethereum: A secure decentralised generalised transaction ledger. Accessed: 2024-12-03. <https://ethereum.github.io/yellowpaper/paper.pdf>.
- Yaga, Dylan, Peter Mell, Nik Roby & Karen Scarfone. 2020. Blockchain technology overview. Tech. Rep. NISTIR 8202 National Institute of Standards and Technology (NIST). <https://doi.org/10.6028/NIST.IR.8202>.