UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO, DE CIÊNCIAS EXATAS E EDUCAÇÃO
DEPARTAMENTO DE ENG. DE CONTROLE, AUTOMAÇÃO E COMPUTAÇÃO
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Amanda Mendes Akiau

Desenvolvimento de um *Smart Contract* para *Crowdfunding* Utilizando a rede de *Blockchain Ethereum*

Blumenau
2023

Amanda Mendes Akiau

Desenvolvimento de um *Smart Contract* para *Crowdfunding* Utilizando a rede de *Blockchain Ethereum*

Trabalho de Conclusão de Curso submetido ao curso de Engenharia de Controle e Automação do Centro Tecnológico, de Ciências Exatas e Educação da Universidade Federal de Santa Catarina como requisito parcial para a obtenção do título de Engenheira de Controle e Automação.

Orientador: Prof. Alex Fabiano Bueno, Dr.

Blumenau

2023

Akiau, Amanda Mendes
Desenvolvimento de um Smart Contract para Crowdfunding Utilizando a rede de Blockchain Ethereum / Amanda Mendes Akiau ; orientador, Alex Fabiano Bueno, 2023.
57 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Campus Blumenau, Graduação em Engenharia de Controle e Automação, Blumenau, 2023.

Inclui referências.

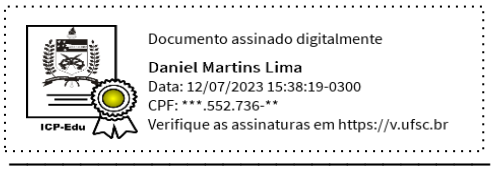
1. Engenharia de Controle e Automação. 2. Smart contract. 3. Crowdfunding. 4. Blockchain. 5. Ethereum. I. Bueno, Alex Fabiano. II. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. III. Título.

Amanda Mendes Akiau

Desenvolvimento de um Smart Contract para Crowdfunding Utilizando a rede de blockchain Ethereum

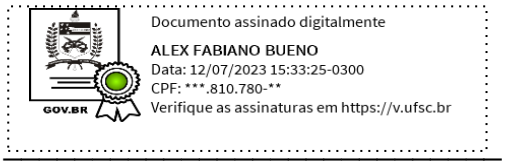
Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do título de “Engenheira de Controle e Automação” e aprovado em sua forma final pelo Curso Engenharia de Controle e Automação.

Blumenau, 06 de julho de 2023.



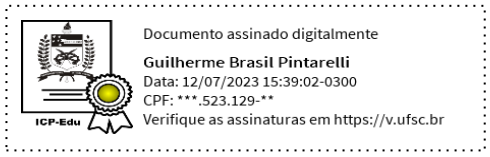
Coordenação do Curso

Banca examinadora



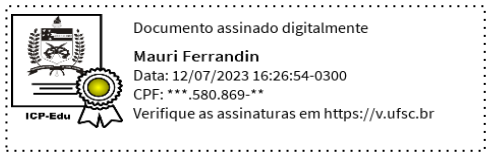
Prof. Alex Fabiano Bueno, Dr.

Orientador



Prof. Guilherme Brasil Pintarelli Dr.

Instituição Universidade Federal de Santa Catarina



Prof. Mauri Ferrandin Dr.

Instituição Universidade Federal de Santa Catarina

Blumenau, 2023.

Dedico este trabalho ao meu orientador, Alex Bueno, aos meus pais, e ao meu irmão. A todos vocês, dedico esta conquista com profundo carinho e gratidão. Sem o apoio de cada um de vocês, este trabalho não teria sido possível. Obrigado por fazerem parte de minha história.

AGRADECIMENTOS

Agradeço, em primeiro lugar, ao meu orientador, Alex Bueno, cuja motivação e crença em minhas habilidades foram fundamentais para a realização deste trabalho. A você, que acreditou em mim mesmo quando eu duvidava de mim mesma, minha sincera gratidão por seu apoio e orientação.

Em segundo lugar, aos meus pais, que estiveram ao meu lado durante toda a minha jornada acadêmica. Sua paciência, encorajamento e amor incondicional me deram a força necessária para superar os desafios e chegar até aqui.

Em terceiro lugar e não menos importante, ao meu irmão, que sempre esteve presente nos momentos mais difíceis e compartilhou comigo as alegrias e tristezas dessa caminhada. Sua amizade e solidariedade foram fundamentais para que eu nunca desistisse e continuasse em busca dos meus sonhos.

RESUMO

A *blockchain* pode transformar significativamente a maneira de se realizar um projeto *crowdfunding* devido a sua tecnologia de transparência das transações e, também, ao fato de que o controle das decisões passa a estar nas mãos dos investidores do projeto, estabelecendo um modelo de confiança maior dos investidores com os criadores dos projetos. Atualmente em plataformas tradicionais de crowdfunding, existem algumas falhas como, por exemplo, a falta de transparência e controle que os investidores têm sobre os projetos que colaboram monetariamente. Este presente trabalho propõe a utilização de um smart contract em rede blockchain para gerar maior transparência e controle aos investidores em projetos crowdfunding. Primeiramente, realizou-se um estudo aprofundado a respeito de tópicos necessários para a base de entendimento da aplicação como *blockchain, ethereum, smart contract* e *crowdfunding.* Após isso, separou-se requisitos de funcionamento da aplicação e diagramas de funcionamento do ponto de vista do administrador e dos contribuintes da aplicação. Desenvolveu-se o *smart contract* utilizando a IDE *Remix Ethereum* em linguagem de programação *Solidity*, e apresentou-se os resultados dos testes da aplicação mostrando o seu funcionamento correto segundo a lógica estabelecida no desenvolvimento.

Palavras-chave: Contratos inteligentes; Financiamento Coletivo; Solidity; IDE Remix Ethereum.

ABSTRACT

Blockchain can significantly transform the way a crowdfunding project is carried out due to its transaction transparency technology and also to the fact that control of decisions is now in the hands of project investors, establishing a greater trust model among investors with project creators. Currently in traditional crowdfunding platforms, there are some flaws such as, for example, the lack of transparency and control that investors have over the projects they collaborate monetarily. This present work proposes the use of a smart contract in a blockchain network to generate greater transparency and control for investors in crowdfunding projects. First, an in-depth study was carried out on topics necessary for the base of understanding the application such as blockchain, ethereum, smart contract and crowdfunding. After that, the application's operating requirements and operating diagrams were separated from the point of view of the administrator and the contributors of the application. The smart contract was developed using the IDE Remix Ethereum in Solidity programming language, and the test results of the application were presented, showing its correct functioning according to the logic established in the development.

Keywords: Contratos inteligentes; Financiamento Coletivo; Solidity; IDE Remix Ethereum.

LISTA DE FIGURAS

Figura 1 – Arquitetura de um sistema distribuído (à esquerda) e um centralizado (à direita). 19
Figura 2 – Esquemático do funcionamento de transação de uma moeda eletrônica. 24
Figura 3 – Esquemático do Funcionamento de um Servidor Timestamp em Rede Distribuída. 25
Figura 4 – Esquemático do sistema de prova-de-trabalho em blocos distribuídos....26
Figura 5 – Exemplo de smart contracts entre um fornecedor e um comprador. 32
Figura 6 – Benefícios dos Smart Contracts. 33
Figura 7 – Cenário ideal de um Crowdfunding. 36
Figura 8 – Cenário não ideal em um Crowdfunding. 37
Figura 9 – Cenário da Aplicação de Smart Contract em um Crowdfunding. 37
Figura 10 – Diagrama da Atividade do Usuário Como Administrador da Campanha. 41
Figura 11 – Diagrama de Atividade do Usuário como Contribuinte da Campanha. ..42
Figura 12 – Compilação do Contrato.....49
Figura 13 – Informações Básicas Referentes ao Contrato ….....49
Figura 14 – Contribuindo com o Contrato. 50
Figura 15 – Saldo da Conta 0x167 Após a Contribuição. 51
Figura 16 – Botão obterReembolso.....51
Figura 17 –Saldo da Conta 0x617... Após o Reembolso. 52
Figura 18 – Saldo do contrato após o reembolso.....52
Figura 19 – Mensagem de Retorno Após um Usuário Comum Tentar Criar uma Requisição.....53
Figura 20 – Visualização da requisição de índice 0.. 53
Figura 21 – Mensagem de Retorno Após o Administrador Tentar Votar em uma Requisição.....54
Figura 22 – Atualização de votos na requisição de índice 0.. 54
Figura 23 – Mensagem de Retorno após um Usuário Comum Tentar Realizar o Pagamento. ….....55
Figura 24 – Saldo Atual do Recebedor. 55
Figura 25 – Novo saldo do endereço 0x17F Após A Realização Do Pagamento. ..56
Figura 26 – Novo Saldo do Contrato e Status da Requisição de Índice 0.....56

Figura 27: Diagrama de classe UML do contrato..57

LISTA DE TABELAS

Tabela 1 – Unidades De Medida Do Ether.....28

LISTA DE QUADROS

Quadro 1 – Nonces para resolver um quebra-cabeça de hash. 22

LISTA DE ABREVIATURAS E SIGLAS

CD – Disco Compacto

P2P – *Peer-to-Peer*

SI – Sistema Internacional de Unidades

PoW - *Proof-of-Work*

PoS - *Proof-of-Stake*

CPU - *Central Processing Unit*

ETH – *Ether*

DApps - Aplicativos Distribuídos

EVM – Máquina Virtual Ethereum

IDE - *Integrated Development Environment*

SUMÁRIO

1 INTRODUÇÃO 15
1.1 OBJETIVOS 16
1.1.1 OBJETIVOS ESPECÍFICOS 16
1.2 ESTRUTURA DO TRABALHO 16
2 REVISÃO DA LITERATURA 18
2.1 A TECNOLOGIA BLOCKCHAIN 18
2.1.2 *HASHING* DE DADOS 20
2.1.3 SISTEMA *PROOF-OF-WORK* 22
2.1.4 SISTEMA PROOF-OF-STAKE 23
2.1.5 PROBLEMA DO GASTO DUPLO E DA REVERSIBILIDADE EM TRANSAÇÕES ELETRÔNICAS 23
2.1.6 SERVIDOR DE *TIMESTAMP* DISTRIBUÍDO NUMA BASE *PEER-TO-PEER* 25
2.1.7 INCENTIVO 26
2.1.8 *ETHEREUM* 27
2.1.9 SOLIDITY 29
2.2 SMART CONTRACTS 30
2.2.1 EXEMPLO DE CASO DE USO DE SMART CONTRACTS 31
2.2.2 VANTAGENS DOS *SMART CONTRACTS* 32
2.2.3 DESVANTAGENS DOS *SMART CONTRACTS* 33
2.3 *CROWDFUNDING* 34
2.3.1 TIPOS DE *CROWDFUNDING* 35
2.4 COMO OS SMART CONTRACTS PODEM SER APLICADOS AO CROWDFUNDING 36
3 METODOLOGIA 40
3.1 REQUERIMENTOS 40
3.2 DIAGRAMA DE FUNCIONAMENTO DO CONTRATO 40
4 DESENVOLVIMENTO DO *SMART CONTRACT* 43
4.1 CONTRIBUINDO COM O CONTATO 43
4.2 OBTENDO REEMBOLSO 45
4.3 CRIANDO UMA SOLICITAÇÃO DE GASTOS 45
4.4 VOTAÇÃO EM SOLICITAÇÕES DE GASTOS 47

4.5 REALIZANDO O PAGAMENTO DE UMA REQUISIÇÃO 48
5 RESULTADOS 49
5.1 CONTRIBUINDO COM O PROJETO 49
5.2 RECEBENDO REEMBOLSO 51
5.3 CRIANDO UMA REQUISIÇÃO DE GASTOS 53
5.4 VOTAÇÃO EM REQUISIÇÃO DE GASTO 54
5.5 REALIZANDO O PAGAMENTO DE UMA REQUISIÇÃO 54
5.6 DIAGRAMA DE CLASSE UML DO CONTRATO 57
6 CONCLUSÕES 58
REFERÊNCIAS 59

1 INTRODUÇÃO

A participação e colaboração crescentes apresentadas pelas redes na internet proporcionou o uso de sites de *crowdfunding* como forma de financiamento aos mais variados projetos. O termo *crowdfunding* teve origem a partir do processo denominado *crowdsourcing*, que é um modelo de criação e/ou produção baseado em redes de conhecimento coletivo para criar conteúdo, solucionar problemas ou inventar novos produtos de forma colaborativa.

O sistema *crowdfunding* pode ser definido como um financiamento coletivo em que, a partir de doações via internet, é possível que se atinja o valor monetário necessário para realizar um projeto. A variedade de projetos é grande, podendo ser a gravação de um CD, financiamento de campanhas políticas, eventos, shows, produtos VALIATI e TIETZMANN (2012). Contudo, este modelo de negócio possui algumas falhas como, por exemplo, os financiadores dos projetos não possuem controle e nem transparência sobre a forma como o dinheiro deles está sendo gasto no projeto. Uma possível solução para incluir a transparência nas transações financeiras de um projeto *crowdfunding* é a utilização de um *smart contract* criado em uma rede *blockchain*.

O avanço da tecnologia gerou um aumento na quantidade de transações financeiras digitais. Porém, existem algumas fragilidades inerentes deste tipo de sistema como, por exemplo, as transações são reversíveis e uma mesma moeda pode ser utilizada em duas transações diferentes em momentos diferentes (gasto duplo). Um dos motivos disso ocorrer é que neste tipo de sistema uma entidade central é responsável por controlar todas as transações. Nakamoto (2008) introduziu a *blockchain* como solução para a reversibilidade das transações e o gasto duplo utilizando um sistema distribuído.

Blockchain, resumidamente, é uma tecnologia de transações financeiras distribuídas, bastante conhecida por sua utilização na moeda criptográfica *Bitcoin*. De acordo com WUST e GERVAIS, 2018, a *blockchain* é uma inovação tecnológica que permite revolucionar a forma como a sociedade negocia e interage. Isto deve-se em relação às suas propriedades de permitir que duas pessoas troquem valor financeiro e interajam sem depender de um terceiro confiável (por exemplo, um banco). Além disso, a *blockchain* fornece proteção de integridade de armazenamento de dados e permite dar transparência ao processo. As transações em *blockchain* não estão restritas apenas a transações financeiras. A *blockchain Ethereum* permite executar códigos arbitrários dentro de contratos denominados *smart contracts*.

Um *smart contract* é um conjunto de programas que são auto verificáveis, autoexecutáveis e invioláveis, que a partir da integração com a tecnologia *blockchain* é capaz de realizar uma tarefa em tempo real com baixo custo e fornecer maior segurança (MONTANA et al, 2018). Para desenvolver um smart contract na *blockchain Ethereum* a linguagem de programação utilizada é a *Solidity*. A linguagem *Solidity* usa a abordagem “design por contrato” ou “orientada para contrato” para verificar a execução de métodos e garantir que o estado geral das estruturas de dados do contrato não seja corrompido. (JAGGI, Harish; JHA, Raj, 2019). A proposta deste trabalho é a utilização de *smart contracts* em rede *blockchain Ethereum* para solucionar o problema relacionado a falta de controle e transparência dos financiadores em projetos de *crowdfunding*. Tem-se como limitações prévias deste trabalho o desenvolvimento de uma interface com o usuário (*front-end*) e a implementação do código em ambiente real e produtivo.

1.1 OBJETIVOS

O objetivo principal deste trabalho é desenvolver um *smart contract* para *crowdfunding* utilizando a rede *blockchain Ethereum*, buscando melhorias no processo de investimento em projetos com colaboração coletiva.

1.1.1 OBJETIVOS ESPECÍFICOS

- Realizar uma busca bibliográfica sobre blockchain, smart contratcs e crowdfunding
- Realizar uma busca bibliográfica em cima de trabalhos que apresentam smart contracts para crowdfunding
- Definir requisitos para a solução;
- Elaborar diagramas da solução de *smart contract* para *crowdfunding*;
- Implementar a solução utilizando a linguagem de programação *Solidity*;
- Mostrar os resultados e discutir a solução desenvolvida.

1.2 ESTRUTURA DO TRABALHO

Este documento está organizado da seguinte maneira: no capítulo 2, aborda-se a fundamentação teórica fundamental para fornecer ao leitor a base necessária para o entendimento deste trabalho, com tópicos como: *blockchain*, *Ethereum*, *smart*

contract e *crowdfunding*. No capítulo 3, apresentam-se os requerimentos do funcionamento e diagramas da aplicação. No capítulo 4, apresenta-se o desenvolvimento da solução. No capítulo 5, apresentam-se os resultados e discussões da solução desenvolvida. Por fim, no capítulo 6, apresentam-se as conclusões e limitações deste trabalho.

2 REVISÃO DA LITERATURA

Este capítulo aborda, com a ajuda de discussões já existentes na literatura, todos os conceitos necessários para o entendimento deste documento.

2.1 A TECNOLOGIA BLOCKCHAIN

Em seu influente trabalho "*Bitcoin: A Peer to Peer Electronic Cash System*", Nakamoto (2008) apresentou o que mais tarde se tornaria conhecida como tecnologia *blockchain*. Como observa Di Pierro (2017), a proposta de Nakamoto estabeleceu uma base matemática essencial que proporcionou avanços em criptomoedas e sistemas financeiros convencionais. O impacto de sua ideia foi substancial; hoje, essa tecnologia inovadora é usada em várias aplicações além das finanças - incluindo campos emergentes, como os *smart contracts*.

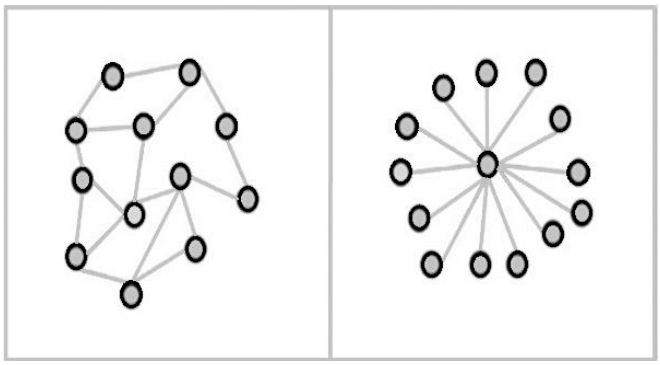
O problema que Nakamoto resolveu com a *blockchain* foi estabelecer confiança em um sistema distribuído, solucionando o problema do gasto duplo e da reversibilidade em transações eletrônicas, cujo modelo é baseado em confiança. A solução proposta por ele é o uso de um servidor de *timestamp (*tipo de dado contendo um valor combinado de data e hora, que quando impresso exibe no formato yyyy-mm-dd hh:mm:ss) distribuído *P2P* para gerar prova computacional da ordem cronológica das transações, permitindo que duas partes dispostas façam transações diretamente entre si sem a necessidade de um terceiro confiável, tornando assim o processo menos custoso e mais rápido.

A *blockchain* utiliza um mecanismo de consenso para torná-la muito trabalhosa (sistemas de *proof of work*) ou muito arriscada (sistemas de *proof of stake*) para tentar falsificar o livro-razão (registro das transações imutáveis que ocorrem na blockchain). Isso é abordado por uma relação matemática chamada *hash* e um mecanismo de consenso para verificar *hashes* (FAIRFIELD, 2022).

Uma das decisões fundamentais ao implementar um sistema diz respeito ao modo como seus componentes são organizados e se relacionam, em outras palavras, à sua arquitetura. As duas principais abordagens para a arquitetura de sistemas de software são: centralizada e distribuída (DRESCHER, 2018). Em sistemas centralizados, os componentes estão localizados em torno de um componente central e estão conectados a ele. E em sistemas distribuídos, os componentes estão conectados uns aos outros, sem que haja nenhum elemento central para coordenação

ou controle. A Figura 1 representa essas duas arquiteturas opostas (DRESCHER, 2018).

Figura 1: Arquitetura de um sistema distribuído (à esquerda) e um centralizado (à direita).



Fonte: Drescher, 2018.

De acordo com Drescher (2018), as principais vantagens de sistemas distribuídos em relação a sistemas centralizados são:

- maior capacidade de processamento: a capacidade de processamento em um sistema distribuído é o resultado da combinação de todos os computadores conectados, em contrapartida em sistemas centralizados a capacidade de processamento depende apenas do computador central;
- menor custo: os custos para instalar, operar e manter um supercomputador (em um sistema centralizado) são muito mais altos do que para fazer o mesmo em um sistema distribuído;
- maior confiabilidade: ao contrário de sistemas centralizados, um sistema distribuído não possui um único ponto de falha, então se um elemento falhar os elementos restantes poderão assumir sua função, dessa forma, há maior confiabilidade neste modelo;
- capacidade de expandir-se naturalmente: para aumentar a capacidade de processamento de todo o sistema basta conectar computadores adicionais a ele, por isso a capacidade de expandir-se naturalmente.

Em relação às desvantagens de sistemas distribuídos perante os sistemas centralizados, Drescher (2018) pontua problemas como:

- *overhead* de coordenação: devido à ausência de entidades centrais em sistemas distribuídos, a coordenação de membros deve ser feita pelos próprios participantes do sistema, este trabalho de coordenação exige esforços e capacidade de processamento;
- *overhead* de comunicação: os computadores em um sistema distribuído precisam se comunicar uns com os outros e isso exige um protocolo de comunicação, envio, recepção e processamento de mensagens que demandam esforços e capacidade de processamento;
- dependência de redes: os computadores se comunicam através de uma rede, isso pode ser um problema pois as redes possuem seus próprios desafios e adversidades que geram impacto na comunicação e na coordenação entre os computadores que compõem um sistema distribuído;
- mais complexidade nos programas: considerando as desvantagens mencionadas anteriormente, qualquer software em um sistema distribuído deve resolver problemas adicionais como comunicação, coordenação e utilização de redes, aumentando assim a complexidade dos softwares;
- problemas de segurança: enviar informações por uma rede implica em preocupações com segurança, pois entidades não confiáveis poderão utilizar indevidamente a rede a fim de acessar e explorar informações, portanto, quanto menos restritivo o acesso à rede, maiores serão as preocupações com a segurança.

2.1.2 *HASHING* DE DADOS

A criptografia é uma técnica matemática para proteger os dados quando estão sendo transferidos entre duas pessoas, evitando que uma terceira intercepte a comunicação e entenda o significado dos dados que estão sendo transferidos. Existem várias técnicas envolvidas na criptografia. Algumas das principais técnicas são *Hashing*, Criptografia e Descriptografia.”

Uma função *hash* criptográfica é uma função *hash* unidirecional que mapeia dados de tamanho arbitrário para uma cadeia de bits de tamanho fixo. A saída da função *hash* é um formato hexadecimal de tamanho fixo. Se a saída for passada novamente para a função de hash, ela gerará outro valor de *hash*. A natureza unidirecional significa que é computacionalmente inviável recriar os dados de entrada

se apenas conhecer o *hash* de saída. A única maneira de determinar uma possível entrada é verificando cada candidato para uma saída correspondente; dado que o espaço de busca é virtualmente infinito, é inviável a prática da tarefa. Mesmo que sejam encontrados alguns dados de entrada que criem um *hash* correspondente, eles podem não ser os dados de entrada originais. Encontrar dois conjuntos de dados de entrada com *hash* para a mesma saída é chamado de colisão de *hash*. Quanto melhor a função de *hash*, mais raras são as colisões de *hash* (ANTONOPOULOS, 2018; FOGANG, 2018).

As principais propriedades das funções de *hash* criptográficas são (ANTONOPOULOS, 2018):

- Determinismo: uma determinada mensagem de entrada sempre produz a mesma saída de *hash*;
- Verificabilidade: calcular o *hash* de uma mensagem é eficiente (complexidade linear);
- Não-correlação: uma pequena alteração na mensagem (por exemplo, uma alteração de 1 bit) deve alterar a saída do *hash* tão extensivamente que não pode ser correlacionada com o *hash* da mensagem original;
- Irreversibilidade: calcular a mensagem a partir de seu *hash* é computacionalmente inviável;
- Proteção contra colisão: deve ser inviável calcular duas mensagens diferentes que produzem a mesma saída de *hash.* A resistência a colisões de *hash* é particularmente importante para evitar a falsificação de assinatura digital na blockchain.

A combinação dessas propriedades torna as funções hash criptográficas úteis para uma ampla gama de aplicações de segurança, incluindo (ANTONOPOULOS, 2018).:

- Impressão digital de dados
- Integridade da mensagem (detecção de erros)
- Prova de trabalho (proof-of-work)
- Autenticação (hashing de senha e extensão de chave)
- Geradores de números pseudoaleatórios
- Compromisso de mensagem (mecanismos de confirmação-revelação)
- Identificadores exclusivos

Algumas das funções de *hash* populares são MD5, SHA1, SHA2, SHA128, SHA256 e SHA512. *Hashing* é muito usado na tecnologia *blockchain* para proteger os dados e transações seguras na rede. Criptografia e Descriptografia é outra técnica criptográfica usada em blockchain. Essa técnica envolve o uso de uma chave pública para criptografar dados que só podem ser descriptografados com a chave privada associada a essa chave pública. É usado na assinatura digital e na verificação de transações no blockchain. Chave pública/privada é um conceito muito importante na tecnologia blockchain. Os dados nas blockchains são públicos, abertos a qualquer pessoa que tenha acesso a eles. Para preservar a identidade de quem os dados pertencem, o blockchain usa diferentes técnicas de criptografia para tornar isso possível (FOGANG, 2018).

2.1.3 SISTEMA *PROOF-OF-WORK*

No contexto da *blockchain*, utiliza-se quebra-cabeças de *hash* que são muitas vezes chamados *Proof-of-Work* (PoW). Os quebra-cabeças de *hash* são quebra-cabeças computacionais que podem ser vistos como os equivalentes digitais da tarefa de abrir um cadeado de combinação por tentativa de erro. Um exemplo de quebra-cabeça de *hash* é determinar qual dado combinado com "*Hello World*" produziria um valor de *hash* com três zeros na frente (Quadro 1). Exigir que o valor de *hash* atenda a uma determinada restrição é a parte essencial do quebra-cabeça de hash, e como só podem ser resolvidos por tentativa e erro isso exige muita capacidade de processamento e, portanto, consomem muito tempo e energia (DRESCHER, 2018).

Quadro 1 - *Nonces* para resolver um quebra-cabeça de *hash*.

Nonce	Texto cujo hash será gerado	Saída
0	Hello World! 0	4EE4B77
1	Hello World! 1	3345B9A
2	Hello World! 2	7204084
3	Hello World! 3	02307DS
	...	
613	Hello World! 613	E861901
614	Hello World! 614	00068A3C
615	Hello World! 615	5EB7483

Fonte: DRESCHER, 2018.

2.1.4 SISTEMA PROOF-OF-STAKE

O mecanismo de consenso PoW possui diversas limitações, por exemplo, ineficiência energética, atraso e vulnerabilidade a ameaças de segurança. Para superar esses problemas, desenvolveu-se um mecanismo de consenso denominado *proof-of-stake* (PoS), que permite alcançar o consenso por meio da prova de participação (NGUYEN et al 2019).

O PoS é um mecanismo de consenso da *blockchain* em que os usuários devem apostar uma certa quantia de suas moedas para apresentar a chance de serem escolhidos para validar blocos de transações e obter uma recompensa por validar os blocos. Uma pessoa pode validar transações de blocos de acordo com o valor de sua aposta na rede. Quanto maior for a participação de um usuário, mais benefícios ele obterá do sistema (BAMAKAN; MOTAVALI; BONDARTI, 2020).

2.1.5 PROBLEMA DO GASTO DUPLO E DA REVERSIBILIDADE EM TRANSAÇÕES ELETRÔNICAS

Define-se uma moeda eletrônica como uma série de assinaturas digitais. Para transferir a propriedade, cada proprietário assina digitalmente um *hash* da transação anterior e a chave pública do novo proprietário, anexando-os ao final da moeda. Ao verificar as assinaturas, o destinatário pode confirmar a cadeia de propriedade conforme a Figura 2 (NAKAMOTO, 2008).

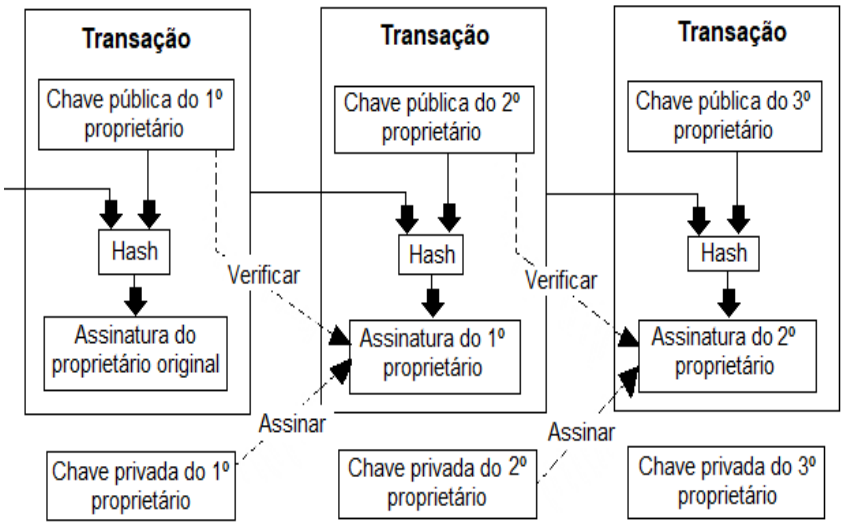
No entanto, surge um desafio, pois o destinatário não pode garantir que um dos proprietários não gastou a moeda em dobro. Uma solução comumente empregada é a introdução de uma autoridade central confiável, responsável por verificar todas as transações para evitar gastos duplos. Após cada transação, a moeda é devolvida à casa da moeda para a emissão de uma nova moeda. Apenas as moedas emitidas diretamente pela casa da moeda são consideradas confiáveis, garantindo que não sejam gastas duas vezes (NAKAMOTO, 2008).

No entanto, essa solução representa um problema, pois todo o sistema monetário se torna dependente da empresa que opera a casa da moeda. Toda transação deve passar por eles, assemelhando-se a um banco tradicional. Para resolver isso, exige-se um método para o destinatário verificar se os proprietários anteriores não assinaram nenhuma transação anterior.

Nesse contexto, a primeira transação tem importância, tornando irrelevantes as tentativas subsequentes de duplo gasto. Para confirmar a ausência de uma transação,

é necessário estar ciente de todas as transações. No modelo baseado na casa da moeda, a casa da moeda possui conhecimento de todas as transações e determina sua ordem cronológica. No entanto, para conseguir isso sem depender de uma entidade confiável, as transações devem ser anunciadas publicamente (DAI, 1998). Além disso, um mecanismo de consenso é essencial para que os participantes concordem com uma história singular que registre a ordem em que as transações foram recebidas. O destinatário precisa de prova de que, no momento de cada transação, a maioria dos nós concordou que foi o primeiro recebido (NAKAMOTO, 2008).

Figura 2: Esquemático do funcionamento de transação de uma moeda eletrônica.



Fonte: Adaptado de NAKAMOTO (2008).

Em relação à reversibilidade, tem-se que transações completamente irreversíveis não são realmente possíveis, ao se tratar de transações eletrônicas tradicionais, uma vez que as instituições financeiras não podem evitar a mediação de conflitos. O custo da mediação aumenta os custos de transação, limitando o tamanho mínimo da transação prática e eliminando a possibilidade de pequenas transações casuais, além disso há um custo mais amplo na perda da capacidade da realização de pagamentos irreversíveis, por serviços irreversíveis. Com a possibilidade de reversão, a necessidade de confiança se espalha. Os comerciantes devem ser cautelosos com seus clientes, importunando-os para obter mais informações do que de fato precisam. Uma certa porcentagem de fraude é aceita como inevitável. Esses custos e incertezas de pagamento podem ser evitados pessoalmente usando moeda

física, mas até então não existia nenhum mecanismo para fazer pagamentos por meio de um canal de comunicação sem uma parte confiável (NAKAMOTO, 2008).

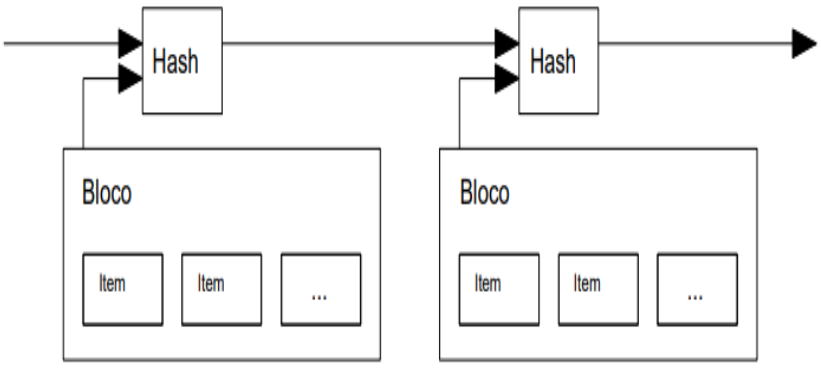
Pensando nisso, Nakamoto (2008) introduz em seu artigo sobre a moeda *Bitcoin* um sistema de pagamento eletrônico baseado em prova criptográfica em vez de confiança, permitindo que quaisquer duas partes dispostas transacionem diretamente entre si sem a necessidade de um terceiro confiável. As transações cuja reversão é computacionalmente impraticável protegem os vendedores contra fraudes, e mecanismos de custódia de rotina poderiam ser facilmente implementados para proteger os compradores.

2.1.6 SERVIDOR DE *TIMESTAMP* DISTRIBUÍDO NUMA BASE *PEER-TO-PEER*

A solução que Nakamoto (2008) propõe, começa com um servidor de *timestamp*, chamado também de carimbo de tempo. Existem duas técnicas de *timestamp*: aquelas que trabalham com uma terceira parte confiável e aquelas que são baseadas no conceito de confiança distribuída. A técnica baseada na confiança distribuída consiste em fazer documentos datados e assinados por um grande conjunto de pessoas, a fim de provar que não há a possibilidade de ter corrompido todos eles (SCHWIENBACHER; LARRALDE, 2010).

O *timestamp* prova que os dados devem ter existido na época, a fim de entrar no *hash*. Cada *timestamp* inclui o anterior em seu *hash*, formando uma cadeia, com cada *timestamp* adicional reforçando os que vieram antes dele, conforme Figura 3 (NAKAMOTO, 2008).

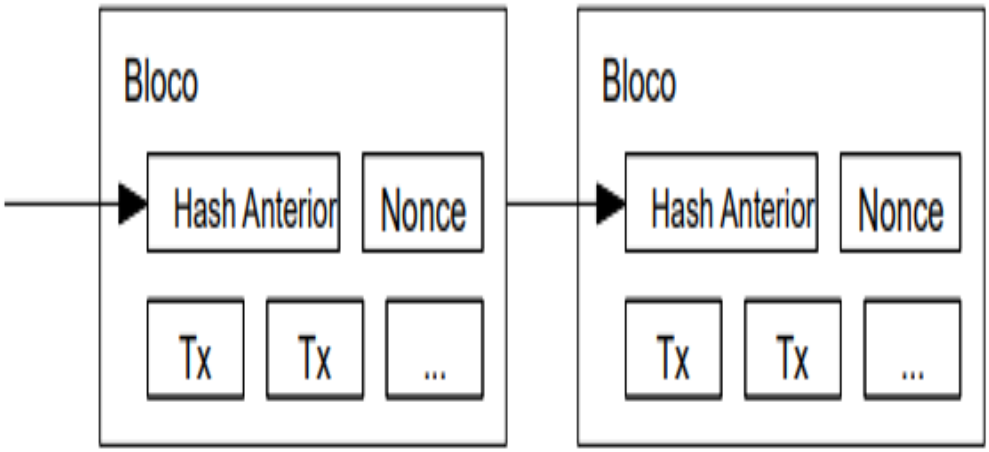
Figura 3: Esquemático do Funcionamento de um Servidor *Timestamp* em Rede Distribuída.



Fonte: NAKAMOTO (2008), Tradução PINTO.

De acordo com Nakamoto (2008), para utilizar um servidor *timestamp* distribuído numa base *P2P* é necessário usar um sistema de *proof-of-work*. Então, implementa-se a PoW incrementando um *nonce* no bloco até que seja encontrado um valor que dê ao *hash* do bloco a quantidade necessária de *bits* zero, conforme Figura 4. Uma vez que o esforço da CPU tem sido consumido para satisfazer a PoW, o bloco não pode ser alterado sem refazer o trabalho. O trabalho para mudar um bloco incluiria refazer todos os blocos após ele, tornando-se inviável computacionalmente (NAKAMOTO 2008).

Figura 4: Esquemático do sistema de prova-de-trabalho em blocos distribuídos.



Fonte: NAKAMOTO (2008), Tradução PINTO.

2.1.7 INCENTIVO

Pela convenção estabelecida, a primeira transação dentro de um bloco é uma operação especial que dá início a uma nova unidade de moeda sob a posse do criador do bloco. Essa prática funciona como um estímulo para os nós da rede, proporcionando um meio inicial de colocar as moedas em circulação, uma vez que não há uma autoridade central responsável por sua emissão. Essa adição constante de novas moedas é comparável ao trabalho dos garimpeiros, que investem recursos para aumentar a quantidade de ouro em circulação. Neste caso, esses recursos são representados pelo tempo de processamento e pelo consumo de eletricidade (NAKAMOTO, 2008).

Além disso, o estímulo também pode ser financiado por meio de taxas de transação. Se o valor da transação de saída for menor do que o valor de entrada, a

diferença se torna uma taxa de transação que é adicionada ao valor de estímulo do bloco que contém a transação. Após um número pré-determinado de moedas ter sido colocado em circulação, o estímulo pode depender exclusivamente das taxas de transação, tornando-se livre de inflação (NAKAMOTO, 2008).

O objetivo desse estímulo é encorajar os nós a permanecerem honestos. Caso um atacante seja capaz de reunir mais poder de processamento do que todos os nós honestos, ele se deparará com uma escolha: utilizar esse poder para defraudar as pessoas e roubar seus pagamentos, ou utilizá-lo para gerar novas moedas. No entanto, ele deve reconhecer que é mais lucrativo seguir as regras estabelecidas, uma vez que essas regras garantem a ele um maior número de novas moedas do que todos os outros combinados. Além disso, violar as regras prejudicaria o sistema como um todo e comprometeria a validade de sua própria riqueza (NAKAMOTO, 2008).

2.1.8 *ETHEREUM*

Ethereum, fundada em 2014 por Vitalik Buterin, é uma infraestrutura de software descentralizada e de código aberto que executa programas denominados contratos inteligentes (*smart contracts*). Ela utiliza uma *blockchain* para sincronizar e armazenar as mudanças de estado do sistema, juntamente com uma criptomoeda denominada *Ether* para medir e restringir os custos de recursos de execução (ANTONOPOULOS; WOOD, 2018).

A plataforma *Ethereum* fornece aos usuários a opção de criarem suas próprias aplicações descentralizadas. As aplicações construídas nesta plataforma são conhecidas como Aplicativos Distribuídos (DApps). Esses DApps contêm contratos inteligentes que possuem código definido pelo usuário para realizar alguma tarefa definida de uma aplicação. Esse código é implantado e executado usando a EVM (WOOD, 2017; GRISHCHENKO; MAFFEI e SCHNEIDEWIND, 2018).

Ethereum é atualmente a plataforma mais popular para o desenvolvimento de contratos inteligentes, por isso é chamada de *Blockchain* 2.0 (ALHARBY; MOORSEL, 2017). Além da *Ethereum*, existem algumas outras plataformas que podem ser utilizadas para desenvolver contratos inteligentes, como *Hyperledger Fabric*, Corda, *BigchainDB* etc.

A unidade monetária do Ethereum é chamada de ether, identificada também como “ETH” ou com os símbolos Ξ (da letra grega “Xi” que se parece com um E

maiúsculo estilizado) ou, menos usual, ♦: por exemplo, 1 ether, ou 1 ETH, ou Ξ1, ou ♦1. (ANTONOPOULOS; WOOD, 2018).

O *ether* é subdividido em unidades menores, até a menor unidade possível, que é denominada *wei*. Um *ether* é 1 quintilhão *wei* (1×10^18 ou 1.000.000.000.000.000.000) (ANTONOPOULOS; WOOD, 2018).

Tabela 1: Unidades De Medida Do *Ether*

Valor (em wei)	Expoente	Nome comum	Nome SI
1	1	Wei	Wei
1.000	10^3	Babbage	Kilowei ou femtoether
1.000.000	10^6	Lovelace	Megawei ou picoether
1.000.000.000	10^9	Shannon	Gigawei ou nanoether
1.000.000.000.000	10^{12}	Szabo	Microether ou micro
1.000.000.000.000.000	10^{15}	Finney	Milliether ou milli
1.000.000.000.000.000.000	10^{18}	Ether	Ether
1.000.000.000.000.000.000.000	10^{21}	Grand	Kiloether
1.000.000.000.000.000.000.000.000	10^{24}		Megaether

Fonte: ANTONOPOULOS; WOOD, 2018

Gas é a taxa que um usuário paga para processar uma transação na *blockchain* Ethereum. Os preços do *gas* normalmente são expressos em "*gwei*", que é uma denominação da moeda Ether (ETH). 1 *gwei* é igual a 0,000000001 ETH (PEASTER, 2020).

Quando o usuário paga a taxa de *gas* para realizar uma transação, significa que ele está pagando pela energia computacional necessária para validar essa transação na rede *Ethereum*. Como a rede *Ethereum* 1.0 utilizava o mecanismo de consenso *PoW*, essa taxa de *gas* era uma parte dos incentivos fornecidos aos mineradores que utilizavam hardwares especiais para competir pelo pedido e processamento de blocos *Ethereum* preenchidos por transações. Em 2022 a *ethereum* oficialmente mudou o mecanismo de consenso para *PoS*, essa mudança eliminou gradualmente a mineração em favor da aposta, momento em que os

apostadores que depositam ETH competem por recompensas de bloco e taxas de *gas*, extinguindo-se os mineradores (PEASTER, 2020).

Diferentes tipos de atividades executadas na rede *Ethereum* terão diferentes custos de *gas*. Por exemplo, é mais barato enviar diretamente ETH de uma carteira para outra do que realizar interações mais complexas com *smart contracts*. Em resumo, os custos do *gas* aumentam de acordo com a complexidade da atividade na rede (PEASTER, 2020).

Segundo Peaster (2020) dois equívocos comuns sobre a taxa de *gas* é que elas são definidas pelos desenvolvedores ou definidas pelos mineradores. Em vez disso, no sistema *PoS* os usuários enviavam transações com preços de *gas* solicitados e os mineradores escolhiam quais transações desejavam minerar em um bloco. Nesse sentido, os preços do *gas* são dinâmicos e são resultado de um equilíbrio alcançado entre o que os usuários oferecem e o que os mineradores aceitam de forma contínua. Quanto mais transações os usuários solicitarem em um determinado momento, mais caros serão os preços de *gas* à medida que o espaço em blocos se torna cada vez mais escasso.

O limite de *gas* é um componente chave do sistema de *gas* da *Ethereum*. No contexto das transações, o limite de *gas* é a quantidade máxima de unidades de *gas* que o usuário está disposto a gastar em uma transação. Esse teto é usado para garantir que as transações sejam executadas e, como nem sempre o usuário pagará o valor máximo, qualquer ETH não utilizado é retornado à carteira do usuário. Para transações básicas de ETH, um limite de *gas* padrão é 21.000 *wei*. Por exemplo, em uma transação hipotética cujo preço do *gas* é de 100 *gwei*, pode-se calcular o custo dessa transação da seguinte maneira segundo Peaster (2020):

Custo da transação = 21.000 ×100 ×0,000000001 ETH = 0,0021 ETH

2.1.9 SOLIDITY

Vitalik Butering, o iniciador da *Ethereum*, projetou a *Ethereum* com contratos inteligentes de forma que qualquer pessoa possa criar contratos inteligentes usando a *Ethereum Virtual Machine* (EVM). EVM é um ambiente de programação para execução de código *Solidity* em *blockchain*. EVM é um computador global distribuído onde todos os contratos inteligentes são executados, às vezes referido como um "computador mundial" onde os contratos inteligentes vivem na forma de *bytecode*

dentro do banco de dados descentralizado em todos os nós (JAGGI, Harish; JHA, Raj, 2019).

Gavin Wood e sua equipe desenvolveram a linguagem *Solidity* para permitir que os desenvolvedores escrevam contratos *Ethereum* com menos esforço e alta precisão. A sintaxe da Solidity é baseada em *JavaScript* e muitos de seus conceitos são extraídos das linguagens *Python* e C++. A linguagem *Solidity* usa a abordagem "*design* por contrato" ou "orientada para contrato" para verificar a execução de métodos e garantir que o estado geral das estruturas de dados do contrato não seja corrompido. Ela permite que os desenvolvedores definam elementos formais como pré-condições, pós-condições e invariantes para cada método de contrato para garantir a execução válida do contrato (JAGGI, Harish; JHA, Raj, 2019).

Solidity é uma linguagem orientada a objetos com suporte à composição de contratos, heranças (única, multinível, hierárquica e múltipla), encapsulamento, polimorfismo (contrato e funcional), contrato abstrato e interfaces (JAGGI, Harish; JHA, Raj, 2019).

2.2 SMART CONTRACTS

Contratos convencionais precisam ser concluídos por um terceiro confiável de maneira centralizada, resultando em longo tempo de execução e custo extra. Por outro lado, em um *smart contract* cláusulas de contrato são escritas em um programa computacional para que ocorra a execução automática do contrato quando condições predefinidas são identificadas, sem que haja a necessidade de um intermediador. Esses contratos são essencialmente armazenados, replicados e atualizados em *blockchains* distribuídos, tornando-os, portanto, contratos rastreáveis e imutáveis. A integração da tecnologia *blockchain* com contratos inteligentes torna realidade um "mercado ponto a ponto" (DRESCHER, 2018).

Goulart e Martins (2022) defendem que os smart contracts representam uma próxima etapa na progressão de *blockchains*. Segundo eles, os contratos inteligentes são peças de software que estendem a utilidade das *blockchains*, que antes eram utilizadas apenas para manter um registro de entradas de transações financeiras, e agora podem implementar automaticamente termos de acordos multipartidários sem a necessidade de um intermediário.

Portanto, *smart contracts* são contratos autoexecutáveis com os termos do acordo entre as partes interessadas. Os contratos são escritos na forma de códigos de programa que existem em uma rede *blockchain* distribuída e descentralizada (TAPSCOTT; TAPSCOTT, 2016). Existem várias *Blockchains* que suportam *smart contracts*, como por exemplo a Ethereum, que será utilizada no desenvolvimento deste trabalho.

2.2.1 EXEMPLO DE CASO DE USO DE SMART CONTRACTS

Zheng et al (2020) fornecem um exemplo do uso de *smart contracts* em um processo entre um comprador e um fornecedor, dividido em três etapas: ordem de compra, transporte e recebimento do produto, e pagamento. Primeiramente, o fornecedor envia o catálogo dos produtos para o comprador através de uma rede *blockchain*. Este catálogo inclui a descrição dos produtos (como propriedades, quantidade, preço e disponibilidade), frete e termos de pagamento, e isso tudo é armazenado e distribuído na *blockchain*. Assim o comprador consegue obter as informações do produto e verificar a autenticidade e reputação do fornecedor ao mesmo tempo. O comprador envia o pedido com a quantidade especificada e informações de pagamento através da *blockchain*. Todo este procedimento forma um contrato de compra (contrato 1), conforme ilustrado na Figura 5.

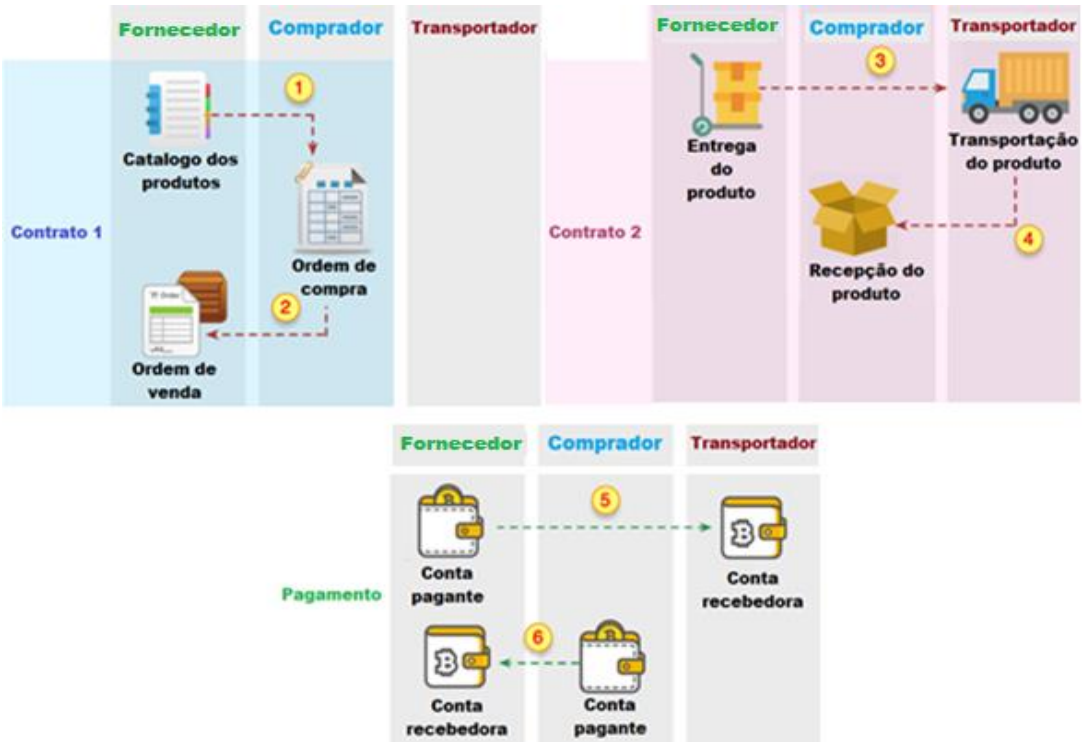
Após a finalização do contrato 1, a etapa seguinte se refere a entrega do produto ao comprador. O fornecedor irá procurar uma transportadora na *blockchain* para completar a fase de envio. Semelhante ao contrato 1, a transportadora também publica a descrição do envio (como taxas de transporte, origem, destino, capacidade e tempo de frete), bem como condições e termos de envio na *blockchain*. Se o fornecedor aceitar o contrato emitido pela transportadora, os produtos serão entregues à transportadora, que por fim despacha os produtos ao comprador. Todo este procedimento forma o contrato de entrega (contrato 2), conforme ilustrado na Figura 5.

Após a finalização do contrato 2, a próxima, e última, etapa se refere ao pagamento. Os procedimentos de pagamento (incluindo o pagamento do fornecedor ao transportador e do comprador ao fornecedor) são automatizados pelos contratos. Uma vez que o comprador confirma a recepção dos produtos, o pagamento entre o comprador e o fornecedor será acionado automaticamente conforme condição pré-definida no contrato 1 é atendida, e o pagamento entre o fornecedor e o transportado

também será acionado automaticamente conforme condição pré-definida no contrato 2 é atendida.

O processo inteiro descrito acima é realizado de forma *P2P*, ou seja, sem a intervenção de terceiros como banco. Percebe-se que, nestas condições, o tempo e o custo transacional podem ser bastante economizados.

Figura 5: Exemplo de *smart contracts* entre um fornecedor e um comprador.



Fonte: Adaptado de **Zheng *et al* (2020)**.

2.2.2 VANTAGENS DOS *SMART CONTRACTS*

Os contratos inteligentes têm as seguintes vantagens em comparação com os contratos convencionais, segundo Zheng *et al* (2020):

- **Redução de riscos:** Devido à imutabilidade do *blockchain*, os contratos inteligentes não podem ser alterados depois de emitidos. Além disso, todas as transações armazenadas na *blockchain* são rastreáveis e auditáveis. Como resultado, comportamentos maliciosos como fraudes financeiras podem ser mitigados;
- **Redução de custos administrativos e de serviços:** *Blockchains* garantem a confiança de todo o sistema por meio de mecanismos de consenso distribuídos sem passar por uma autoridade central ou um mediador. Contratos inteligentes

armazenados em *blockchains* podem ser acionados automaticamente de forma descentralizada. Consequentemente, os custos de administração e serviços devidos à intervenção de terceiros podem ser significativamente economizados;

- **Melhora a eficiência dos processos de negócios.** A eliminação da dependência do intermediário pode melhorar significativamente a eficiência do processo de negócios. O tempo de resposta entre os processos pode ser significativamente reduzido devido a automatização feita pelos contratos sem a dependência de um intermediário.

Para Maia (2022), os benefícios gerais dos *smart contracts* podem ser sintetizados em sete itens principais, conforme ilustrado na Figura 6.

Figura 6: Benefícios dos *Smart Contracts.*

Fonte: Maia (2022)

2.2.3 DESVANTAGENS DOS *SMART CONTRACTS*

Desvantagens que podem ser citadas sobre os smart contracts são:

- **Imutabilidade:** como os contratos inteligentes são escritos por código, uma vez configurados os contatos não podem ser modificados facilmente. Nos contratos tradicionais, a alteração de termos e condições é frequentemente utilizada, especialmente em contratos de longo prazo cuja execução depende da dinâmica da vida real, e as condições mudam continuamente. Devido à rigidez exibida nos contratos inteligentes depois de estabelecidos, estes últimos resultam em uma ampla gama de problemas práticos principalmente no que diz respeito à facilidade de modificar os termos do contrato dependendo

de várias situações. Isso é corroborado por Huckle et al. (2016), que explica que os contratos convencionais possuem disposições que permitem a anulação, incorporação e modificação de contratos;

- **Sigilo contratual:** a tecnologia *blockchain* envolve o compartilhamento de contrato inteligente em todos os nós da rede *blockchain*, uma vez que toda a transação é registrada no livro-razão. A tecnologia *blockchain* envolve o uso do anonimato, em que todos os participantes de uma rede *blockchain* são anônimos e protegidos. No entanto, não há segurança na execução do contrato. Isso ocorre porque, embora os nós sejam anônimos em suas operações, o livro-razão é mantido público e, portanto, as transações são visíveis e não há segurança em relação a isso. Buterin explica que essa é uma área que precisa ser focada porque, apesar dos nós serem anônimos, a manutenção de um livro público no ambiente distribuído resulta em um lapso de privacidade (BUTERIN, 2014).
- **Alto consumo de energia:** como os *smart contracts* utilizam a *blockchain* como infraestrutura, deve-se levar em consideração que a principal desvantagem da *blockchain* é o alto consumo de energia. O consumo de energia é necessário para manter um registro em tempo real de todas as transações. Os mineradores da rede utilizam quantidades substanciais de energia do computador para validar diversas transações por segundo na *blockchain* (SONG et al, 2016).

2.3 *CROWDFUNDING*

O *crowdfunding* é um método muito utilizado para financiar novos empreendimentos, permitindo que fundadores de projetos sociais, culturais ou com fins lucrativos adquiram apoio financeiro de muitos investidores individuais através da Internet, geralmente proporcionando em troca para eles produtos futuros ou até mesmo um percentual de *equity* (SCHWIENBACHER; LARRALDE, 2010). *Crowdfunding* é um fenômeno baseado na Internet que utiliza plataformas para facilitar transações entre quem solicita fundos (*fundraisers*) e aqueles que têm fundos para investir ou simplesmente doar (financiadores). O objetivo das plataformas de *crowdfunding* é conectar potenciais investidores como pessoas físicas diretamente com projetos ou empresas aprovadas pela plataforma (CERVANTES-ZACARÉS et al, 2023).

Uma maneira de facilitar a compreensão do conceito é desmembrar o nome. *Crowd*, em inglês, significa “multidão”; e "*funding*” significa “financiamento”. Ou seja, pode-se traduzir como financiamento coletivo. O objetivo é arrecadar uma quantia para realizar um projeto. Em outras palavras, em vez de levantar dinheiro a partir de um grupo pequeno de investidores (bancos, investidores anjos, venture capital etc.), que é muito mais difícil e burocrático, a ideia é obter o dinheiro a partir de um grande público, em que cada indivíduo contribui com um valor pequeno (SCHWIENBACHER; LARRALDE, 2010; BELLEFLAMME; LAMBERT; SCHWIENBACHER, 2012; BELLEFLAMME, LAMBERT; SCHWIENBACHER 2013).

O financiamento coletivo começou em 2008. Foi impulsionado tanto pela inovação tecnológica quanto pela crise financeira de 2008, que reduziu os empréstimos dos bancos. Esta combinação de fatores criou uma oportunidade para novos tipos de financiamento (KIRBY; WORNER, 2014).

2.3.1 TIPOS DE *CROWDFUNDING*

Existem muitas formas de financiamento coletivo, incluindo *crowdfunding* de consumidores e *crowdfunding* de pré-venda. *Crowdfunding* de consumidores significa que, com base nos produtos existentes, os produtores mostram seus produtos e serviços a consumidores em potencial por meio de plataformas de *crowdfunding* para atrair consumidores em potencial a participar das atividades de *crowdfunding* e coletar a demanda dispersa do mercado (CHEN; LIU, 2023). *Crowdfunding* pré-venda refere-se a empreendedores que atraem consumidores para pré-encomendar o produto para obter os fundos necessários para a produção do produto (BELLEFLAMME, LAMBERT; SCHWIENBACHER 2013). Ambas as formas visam levantar os recursos necessários para a produção do produto, mas o *crowdfunding* de consumidores se concentra em atividades com base no produto existente. Enquanto o *crowdfunding* de pré-venda se concentra em atividades sem produtos para obter todos os fundos necessários para a produção do produto (WAN et al, 2023).

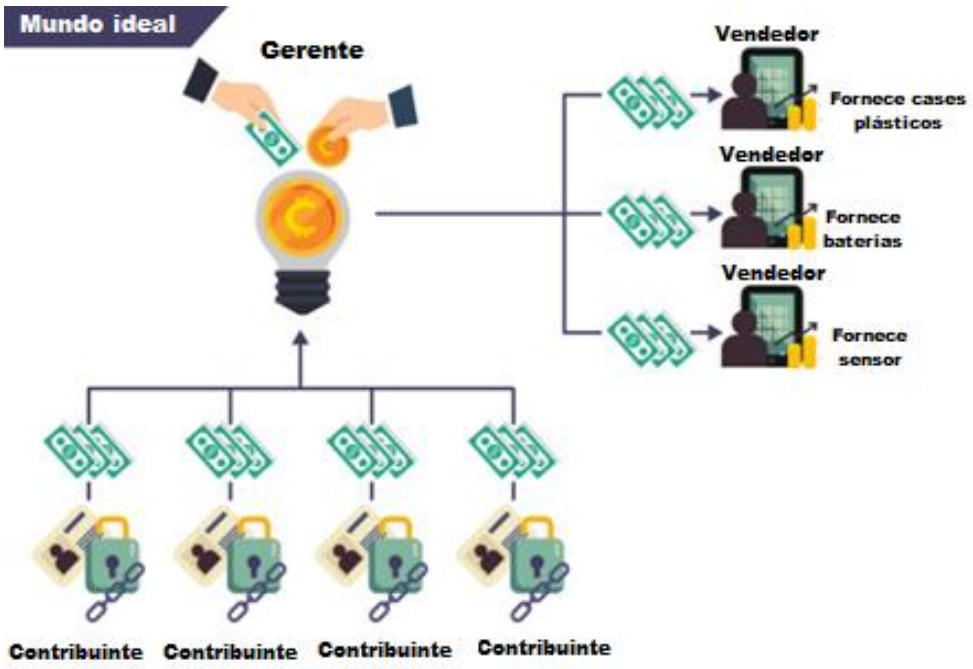
Além dos vários tipos de investimentos, também existem três modelos gerais de investimento em *crowdfunding*: investimento passivo, investimento ativo e doação. O investimento passivo se refere a uma forma de investimento em que a empresa investidora não busca influência sobre as atividades dentro da empresa em que está investindo. Um empréstimo bancário regular é um exemplo de investimento passivo: o banco não será ativo nas decisões tomadas pelo tomador do empréstimo. Os

empreendedores que buscam investidores passivos querem apenas arrecadar dinheiro. Eles não querem seus investidores como participantes ativos em suas operações e decisões subsequentes (SCHWIENBACHER; LARRALDE, 2010). Por outro lado, os investimentos ativos permitem que os investidores participem do processo subsequente e afetam diretamente o resultado dos projetos do empreendedor. Um exemplo seria uma compra de ações em que o comprador adquire uma parcela considerável da empresa (RUBINTON, 2011). O último modelo é o investimento de doação em que os investidores doam seu dinheiro a projetos, causas, produtos etc. através de uma plataforma de *crowdfunding*.

2.4 COMO OS SMART CONTRACTS PODEM SER APLICADOS AO CROWDFUNDING

Para maior entendimento de como os smart contracts podem ser utilizados de forma positiva em um *crowdfunding*, Maia (2022) ilustra a partir da Figura 7 como seria uma situação ideal em que o *crowdfunding* aconteceria, onde a pessoa que criou o projeto é honesta e realmente utiliza o dinheiro dos investidores para o desenvolvimento do que foi prometido, transferindo os fundos para fornecedores de componentes a fim de colocar o projeto em prática.

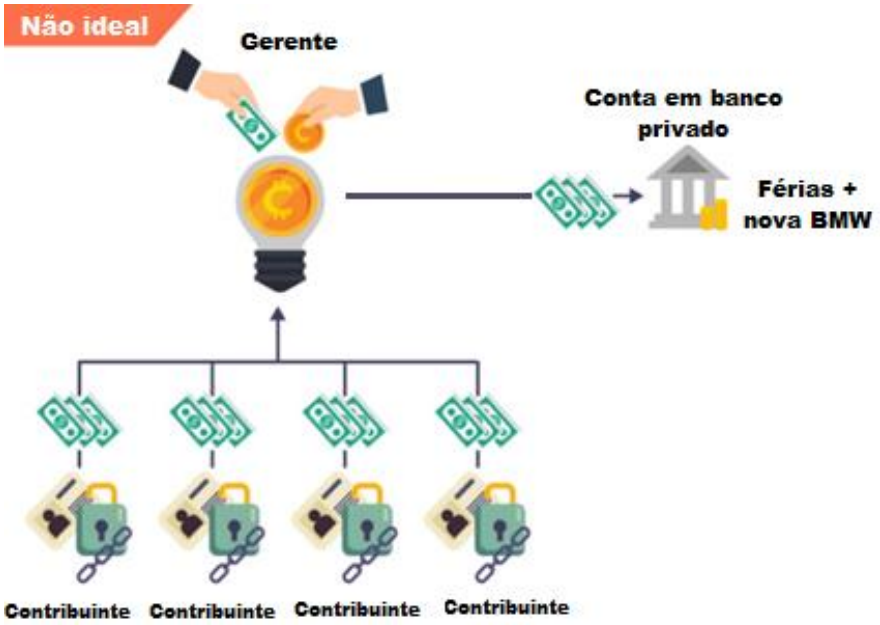
Figura 7: Cenário ideal de um *Crowdfunding*.



Fonte: MAIA (2022)

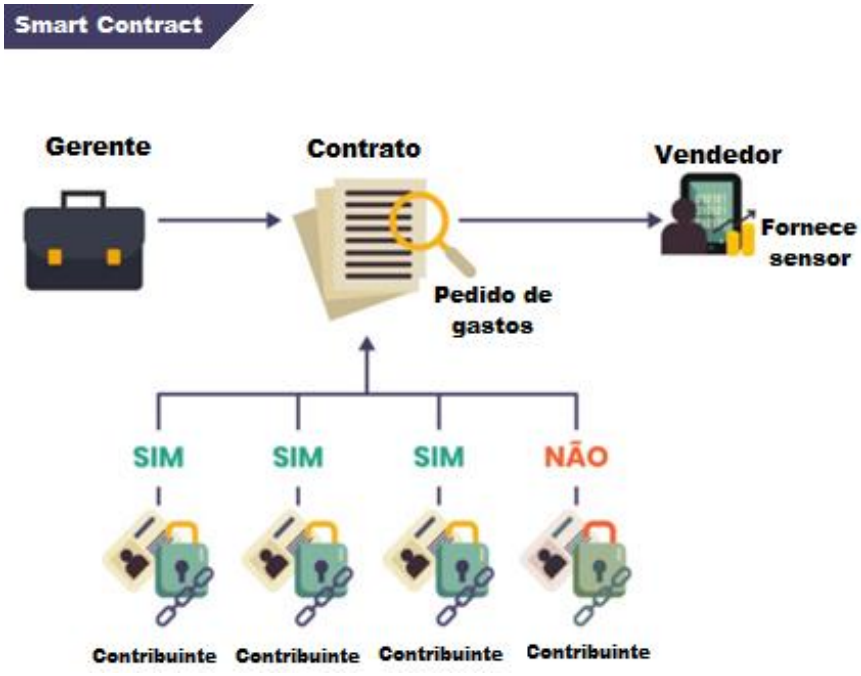
Porém, pode acontecer de os criadores dessas campanhas receberem o dinheiro dos investidores e não darem seguimento ao projeto como previsto e simplesmente desaparecem com todo o dinheiro arrecadado, como ilustrado na Figura 8 (MAIA, 2022).

Figura 8: Cenário não ideal em um *Crowdfunding*.



Fonte: MAIA (2022)

Figura 9: Cenário da Aplicação de *Smart Contract* em um *Crowdfunding*.



Fonte: MAIA (2022)

Em uma aplicação onde o gerente do projeto deve criar uma solicitação de compra indicando o valor e o endereço da carteira digital do fornecedor, e somente consegue transferir os fundos caso haja a aprovação de 50% dos financiadores, é mais difícil para aqueles que apenas pretendem se aproveitar do dinheiro. Com acesso ao histórico de requisições criadas na campanha, os investidores conseguem assim ter mais transparência e auxílio na rastreabilidade do dinheiro arrecadado. A Figura 9 ilustra como passa a ser o caminho do dinheiro saindo do contrato para a carteira de algum fornecedor (MAIA, 2022).

De acordo com Maia (2022), uma das principais diferenças entre sites tradicionais de *crowdfunding* e a aplicação proposta por ele, *crowdfunding* feito com *smart contract* é: a existência de um *plugin* para a autorização das transações, o tempo para a validação de uma transação, os investidores passarem a ter o poder de julgar gastos feitos para o desenvolvimento dos projetos e a existência de uma auditoria pública contendo todos os gastos do projeto. Essa auditoria pública na rede *ethereum* é denominada *Etherscan*. Na blockchain a auditoria pública das transações é livre para qualquer usuário acessar a qualquer momento, diferente de um sistema tradicional em que é contra a lei publicar todos os gastos de um usuário sem a permissão dele.

Fogang propõe um aplicativo de *crowdfunding* onde um usuário pode criar uma campanha de crowdfunding e buscar contribuições dos usuários do aplicativo. A campanha deve conter um título, uma breve descrição, prazo e a meta de valor a ser arrecadado. O aplicativo possui seções nas quais os usuários podem comprar os *tokens* integrados e usá-los para contribuir com a campanha de *crowdfunding* que desejam. Os usuários podem comprar a criptomoeda integrada utilizando moeda fiduciária e também podem converter essa criptomoeda em moeda fiduciária após o término do evento de arrecadação de fundos. O criador da campanha pode sacar os fundos contribuídos pelos usuários apenas se o objetivo da campanha for alcançado após o prazo (FOGANG, 2018).

Fogang desenvolveu a sua aplicação de crowdfunding com *smart contract*, utilizando a linguagem *Solidity*, dividindo em 4 contratos diferentes: o primeiro contrato permite de um token integrado na plataforma *Ethereum*. O segundo contrato permite que um usuário com um endereço de conta possa comprar o token integrado. O terceiro contrato é o contrato da campanha de *crowdfunding* em que o usuário deve criar uma campanha para uma determinada causa. O quarto e último contrato é um

contrato especial que facilita a criação e implantação do contrato de campanha crowdfunding na rede *blockchain* (FOGANG, 2018).

3 METODOLOGIA

Este capítulo aborda os requerimentos do funcionamento da aplicação e os diagramas de funcionamento do contrato.

3.1 REQUERIMENTOS

Os requerimentos do funcionamento desta aplicação são:

1. O administrador inicia uma campanha de *Crowdfunding* com uma meta e prazo especificados;

2. O público interessado contribui com o projeto enviando determinada quantia de ETH. Essa quantia deve ser igual ou superior à contribuição mínima definida pelo administrador;

3. O administrador deve criar uma solicitação de gasto quando desejar transferir os fundos. A solicitação deve conter descrição, endereço da carteira de quem receberá os fundos e o valor a ser enviado;

4. A partir do momento em que é criada a solicitação de gasto, os contribuintes podem votar nessa solicitação;

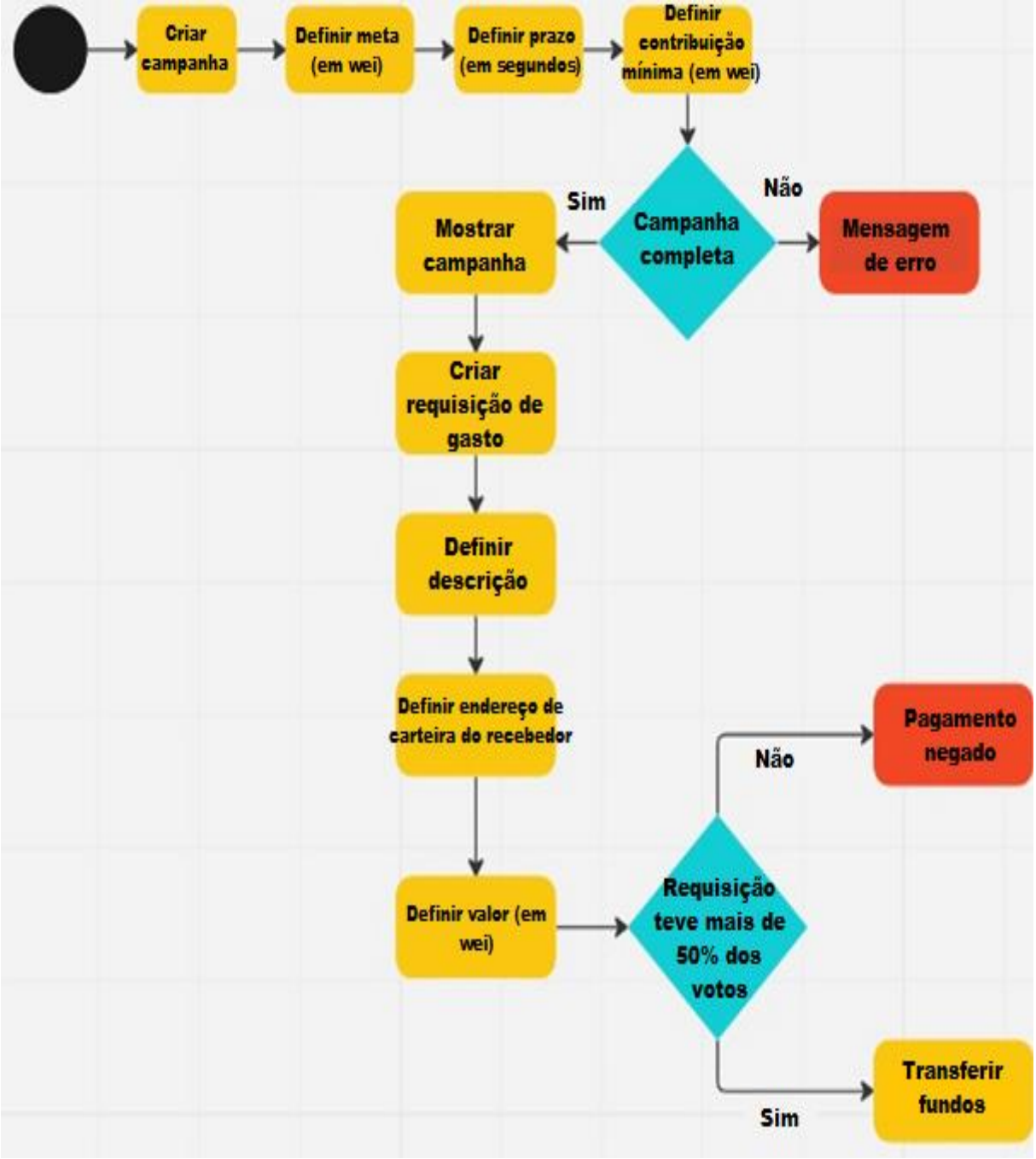
5. Se mais de 50% do total de contribuintes votarem positivamente, então o administrador deverá ter a permissão de transferir a quantia especificada na solicitação de gasto;

6. Caso a meta do valor do projeto não tenha sido alcançada até o prazo definido, os contribuintes podem requisitar reembolso.

3.2 DIAGRAMA DE FUNCIONAMENTO DO CONTRATO

A Figura 10 ilustra o diagrama da atividade desempenhada pelo administrador da campanha, desde o momento da criação da campanha até a transferência dos fundos arrecadados. Observa-se com o diagrama que a campanha somente é criada se o usuário definir uma meta (em *wei*), prazo (em segundos) e uma contribuição mínima (em *wei*). Caso as três condições sejam satisfeitas, a campanha é criada com sucesso. Outra observação é em relação a criação de requisição de gasto, cujo administrador deve definir uma descrição para a solicitação, o endereço de quem receberá o valor, e o valor que deseja transferir. A requisição de gasto somente é aceita se possuir mais de 50% dos votos dos investidores da campanha.

Figura 10: Diagrama da Atividade do Usuário Como Administrador da Campanha

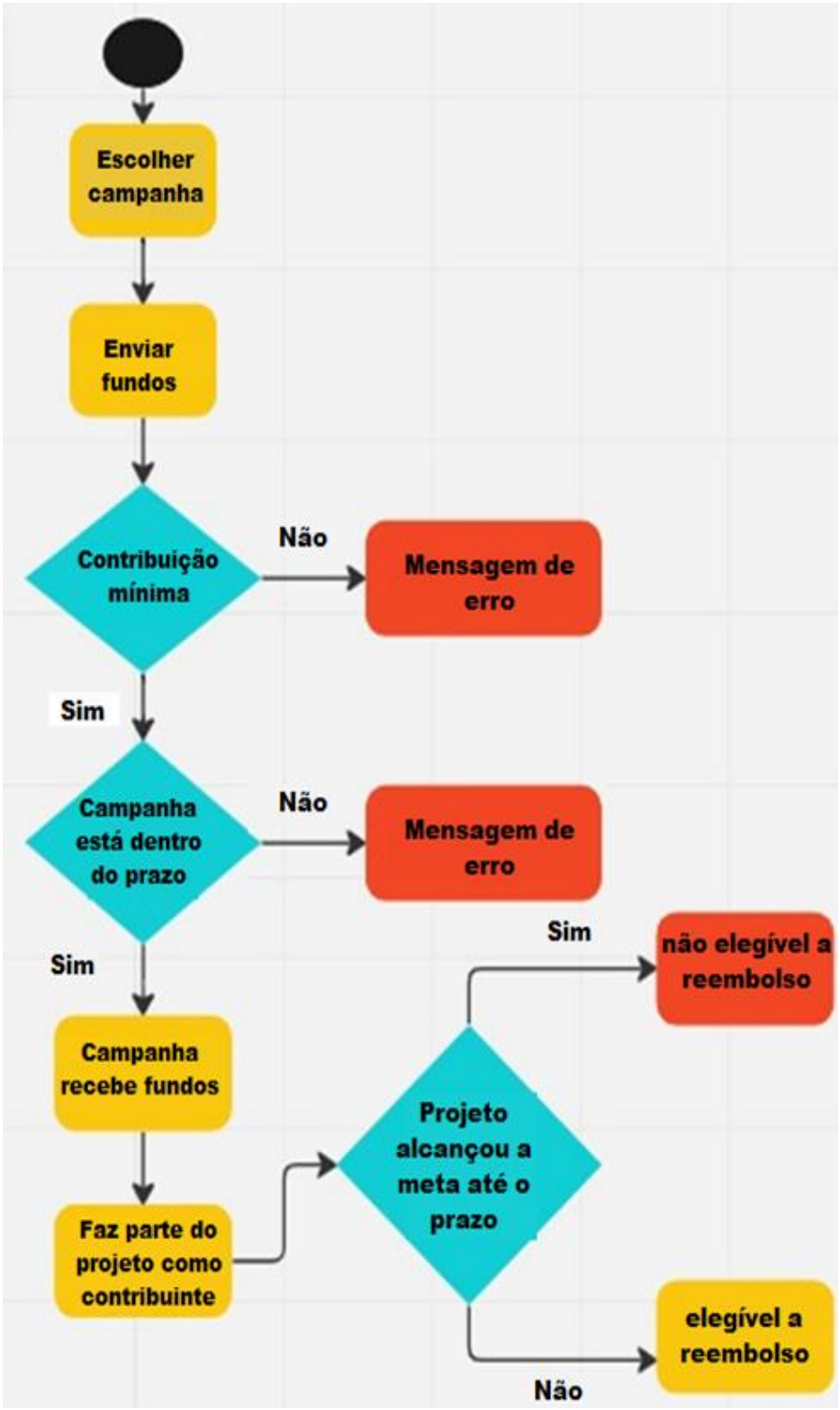


Fonte: Autor.

Já a Figura 11 ilustra o diagrama da atividade desempenhada pelos contribuintes da campanha, desde o momento da escolha da campanha até o momento de reembolso caso a campanha não atinja o valor de meta no prazo estipulado. Observa-se com o diagrama que o usuário somente fará parte do projeto se ele transferir um valor igual ou superior a contribuição mínima estipulada no

contrato. Outra observação é em relação a solicitação de reembolso, que somente será válida se a campanha não arrecadar o valor de meta no prazo estipulado.

Figura 11: Diagrama de Atividade do Usuário como Contribuinte da Campanha.



Fonte: Autor

4 DESENVOLVIMENTO DO *SMART CONTRACT*

Este capítulo aborda todas as etapas do desenvolvimento do *smart contract*, desde a declaração das variáveis de estado até a função que realiza o pagamento de uma requisição. Para fins deste trabalho, realiza-se o desenvolvimento do *smart contract* na IDE *Remix Ethereum*, que utiliza a rede *Ethereum* e linguagem de programação *Solidity*. De acordo com Alharby e Moorsel, (2017) a *Ethereum* é a plataforma mais popular para o desenvolvimento de *smart contracts*.

4.1 CONTRIBUINDO COM O CONTATO

Essa seção explica detalhadamente a parte do código que possibilita com que um contrato seja criado e que pessoas contribuam enviando fundos a esse contrato.

Primeiramente, declara-se as variáveis de estado do contrato, conforme o código-fonte ilustrado abaixo.

```
contract Crowdfunding {
mapping(address => uint256) public contribuintes;
address public admin;
uint256 public numContribuintes;
uint256 public minimoContribuicao;
uint256 public prazo; //timestamp
uint256 public meta;
uint256 public valorArrecadado;
```

Segue abaixo uma explicação breve sobre cada variável:

- contribuintes = retornam os endereços das contas que transferiram fundos ao projeto e os valores das quantias (em *wei*) que enviaram.
- *admin* = retorna o endereço da conta de quem criou o projeto;
- *numContribuintes* = retorna o número de pessoas que contribuíram com o projeto;
- *minimoContribuicao* = retorna qual é o valor mínimo (em *wei*), determinado pelo *admin,* que pode contribuir com o projeto;
- prazo = retorna o prazo do projeto (em *timestamp*);
- meta = retorna qual é a meta de arrecadação do projeto (em *wei*);
- valorArrecadado = retorna o valor total arrecadado até o momento (em *wei*).

Em seguida, tem-se o construtor que determina o mínimo de contribuição, e que definirá a meta e o prazo da campanha de acordo com o valor que o usuário incluir no momento do *deploy*.

```
constructor(uint256 _meta, uint256 _prazo) {
    meta = _meta;
    prazo = block.timestamp + _prazo;
    minimoContribuicao = 100 wei;
    admin = msg.sender;
}
```

A função “contribuir()” é chamada quando alguém deseja transferir dinheiro para o contrato. Essa função inicia verificando se o contrato está dentro do prazo e se o valor a ser transferido respeita o mínimo de contribuição, ambos estipulados pelo administrador.

```
function contribuir() public payable {
    require(block.timestamp < prazo, "Prazo expirou!");
    require(
        msg.value >= minimoContribuicao,
        "Contribuicao minima nao atendida!"
    );

    if (contribuintes[msg.sender] == 0) {
        numContribuintes++;
    }
    contribuintes[msg.sender] += msg.value;
    valorArrecadado += msg.value;
}
```

A condicional IF, que segue após as verificações, identifica o número de pessoas que contribuíram com o contrato de acordo com o endereço de suas carteiras. Um usuário pode transferir ETH diversas vezes para o mesmo contrato, mas a variável “*numContribuintes*” somente é incrementada 1 vez por endereço de carteira.

O valor enviado por um contribuinte é incluído na variável "contribuintes". A chave é o endereço e o valor é a quantidade total de *wei* enviado pelo endereço. E o valor atual recebido pelo contrato é adicionado à variável “valorArrecadado”.

Abaixo tem-se a parte do código que é responsável por fazer com que os valores transferidos a partir de contribuintes sejam enviados diretamente para o saldo do contrato.

```
receive() external payable {
    contribuir();
}
```

Finalmente, a função “obterSaldo()” retorna o saldo total do contrato.

```
function obterSaldo() public view returns (uint256) {
    return address(this).balance;
}
```

4.2 OBTENDO REEMBOLSO

Essa seção explica detalhadamente a parte do código que possibilita com que um contribuinte solicite reembolso. A campanha de *crowdfunding* possui uma meta e um prazo. Se a meta não é atingida dentro do prazo a campanha não pode ser executada com sucesso, cada usuário pode solicitar reembolso.

Para isso, é necessário criar uma função denominada "obterReembolso".

```
function obterReembolso() public {
    require(block.timestamp > prazo && valorArrecadado < meta);
    require(contribuintes[msg.sender] > 0);

    address payable recebedor = payable(msg.sender);
    uint256 valor = contribuintes[msg.sender];
    recebedor.transfer(valor);

    contribuintes[msg.sender] = 0;
}
```

O contribuinte pode requisitar reembolso se duas condições são satisfeitas: o prazo da campanha já passou e a meta não foi atingida. Deve-se verificar também se a pessoa que está solicitando o reembolso é realmente um contribuinte, ou seja, verifica-se se o endereço da conta já enviou dinheiro para o contrato.

Contudo, a função determina que, após as condições serem satisfeitas, o endereço da conta que solicita o reembolso deverá receber de volta em seu endereço o valor total da contribuição. Por fim, após o reembolso ser concluído, define-se o valor enviado por este contribuinte a zero na variável de mapeamento.

4.3 CRIANDO UMA SOLICITAÇÃO DE GASTOS

Essa seção explica detalhadamente a parte do código que possibilita com que o administrador crie uma solicitação de gastos. Um dos requerimentos da aplicação é que “o administrador deve criar uma solicitação de gasto para que possa gastar o dinheiro com o projeto” e “se mais de 50% do total de contribuintes votarem positivamente, então o administrador deverá ter a permissão de gastar a quantia especificada na solicitação de gasto” (seção 3.1).

Segundo Maia (2022), em uma aplicação onde o gerente do projeto deve criar uma solicitação de compra indicando o valor e o endereço da carteira digital do fornecedor, e somente consegue transferir os fundos caso haja a aprovação de 50%

dos financiadores, é mais difícil para aqueles que apenas pretendem se aproveitar do dinheiro.

A solicitação de gastos deve conter todas as informações necessárias para que os contribuintes estejam cientes do que votam. As informações da solicitação de gasto são armazenadas na variável de estrutura “Requisicao”.

```
struct Requisicao {
    string descricao;
    address payable recebedor;
    uint256 valor;
    bool concluido;
    uint256 numVotantes;
    mapping(address => bool) votantes;
}
```

Além das informações sobre a requisição (descrição, recebedor e valor a ser transferido), também tem as informações do status da requisição, número de votantes e o endereço da conta de cada votante. Quando a requisição tiver o mínimo de votos e o pagamento for realizado ao recebedor, o status da requisição é marcado como concluído, ou seja, o valor da variável concluído muda de *false* para *true*. Lembrando que o valor *default* de uma variável booleana é *false*.

Em uma campanha, o administrador pode criar mais de uma solicitação de gasto. Portanto, é necessária uma variável para armazenar todas as solicitações. A variável é do tipo *mapping*, em que as chaves são o índice da requisição e os valores da variável de estrutura “Requisicao”, conforme código-fonte abaixo. Como uma variável de mapeamento não incrementa o índice automaticamente (como um *array* faz), é necessário criar uma outra variável denominada “numRequisicoes” para armazenar o número de requisições.

```
mapping(uint256 => Requisicao) public requisicoes;

uint256 public numRequisicoes;
```

Depois de declarar as variáveis de estado, o próximo passo é declarar uma função denominada “criarRequisicao”, que será chamada pelo administrador quando ele quiser criar uma requisição de gasto. Como apenas o administrador pode criar as requisições, a função “criarRequisicao” só poderá ser chamada pelo administrador, portanto é necessário adicionar um modificador antes.

```
modifier apenasAdm() {
```

```
        require(
            msg.sender == admin,
            "Apenas o administrador pode chamar esta funcao!"
        );
        _;
    }

    function criarRequisicao(
        string memory _descricao,
        address payable _recebedor,
        uint256 _valor
    ) public apenasAdm {
        Requisicao storage novaRequisicao = requisicoes[numRequisicoes];
        numRequisicoes++;

        novaRequisicao.descricao = _descricao;
        novaRequisicao.recebedor = _recebedor;
        novaRequisicao.valor = _valor;
        novaRequisicao.concluido = false;
        novaRequisicao.numVotantes = 0;
    }
```

4.4 VOTAÇÃO EM SOLICITAÇÕES DE GASTOS

Essa seção explica detalhadamente a parte do código que possibilita aos contribuintes votar em solicitações de gastos. A função de votação deverá ser chamada pelos contribuintes para votar em uma solicitação de gasto. As requisições de gastos são salvas em uma variável de mapeamento, e cada requisição possui seu próprio índice. Assim, os contribuintes votarão em um número específico de requisição.

Para votar em uma requisição o usuário deve ser um contribuinte, portanto, para chamar a função de votação o valor associado ao endereço na variável de mapeamento do contribuinte deve ser maior que zero, caso contrário o contrato irá retornar "Você deve ser um contribuinte para votar!". Outra condição é que só pode um voto por contribuinte, caso contrário o contrato irá retornar “Você já votou!”. A Figura 21 mostra a implementação completa da função de votação das requisições.

```
    function votoRequisicao(uint256 _numRequisicoes) public {
        require(
            contribuintes[msg.sender] > 0,
            "Voce deve ser um contribuinte para votar!"
        );
        Requisicao storage essaRequisicao = requisicoes[_numRequisicoes];
```

```
        require(essaRequisicao.votantes[msg.sender] == false, "Voce ja
votou!");
        essaRequisicao.votantes[msg.sender] = true;
        essaRequisicao.numVotantes++;
    }
```

4.5 REALIZANDO O PAGAMENTO DE UMA REQUISIÇÃO

Essa seção explica detalhadamente a parte do código que possibilita com que o administrador realize o pagamento de uma requisição. A função responsável por realizar o pagamento de uma requisição será acionada apenas pelo administrador para transferir o dinheiro de uma requisição para um vendedor ou fornecedor. Essa função pode ser acionada apenas se o valor arrecadado for maior ou igual a meta e se mais que de 50% dos contribuintes tiverem votado na requisição em questão.

```
    function fazerPagamento(uint256 _numRequisicoes) public apenasAdm {
        require(valorArrecadado >= meta);
        Requisicao storage essaRequisicao = requisicoes[_numRequisicoes];
        require(
            essaRequisicao.concluido == false,
            "A requisicao foi concluida!"
        );
        require(essaRequisicao.numVotantes > numContribuintes / 2); //> 50%
votou na requisicao

        essaRequisicao.recebedor.transfer(essaRequisicao.valor);
        essaRequisicao.concluido = true;
    }
}
```

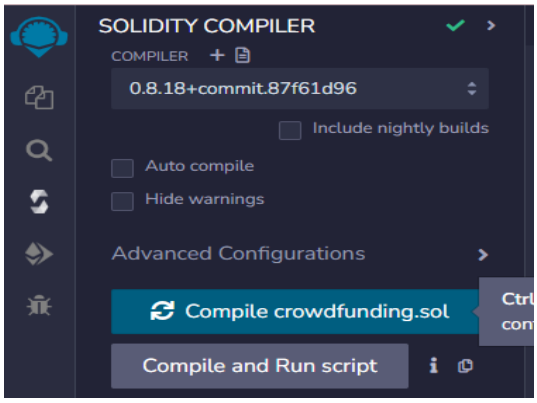
5 RESULTADOS

Este capítulo aborda todos os resultados do *smart contract* de acordo com cada etapa do desenvolvimento.

5.1 CONTRIBUINDO COM O PROJETO

Primeiramente, deve-se compilar o contrato para verificar se há possíveis erros no código, conforme figura 12.

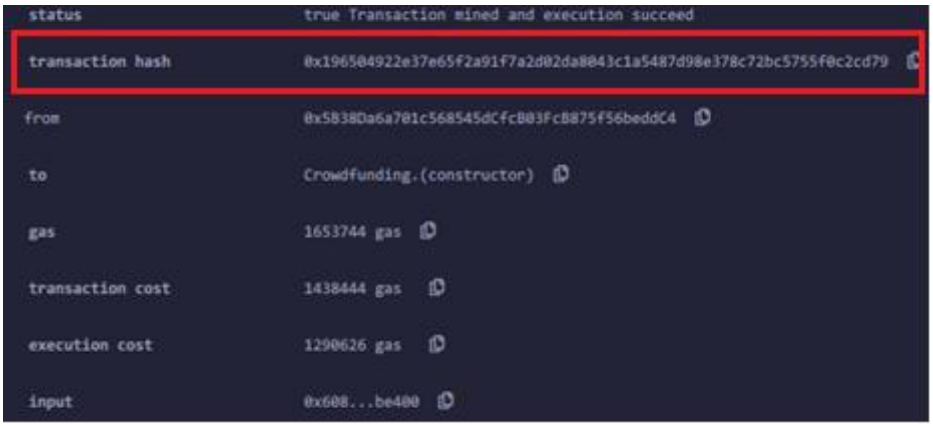
Figura 12: Compilação do Contrato.



Fonte: Autor.

Após a compilação, segue com o *deploy* e execução do contrato. Ao realizar o deploy aparece algumas informações básicas referente ao contrato, como ilustrado na figura 13. Observa-se que a "*transaction hash*" corresponde ao código *hash* da transação responsável pela criação do contrato (a seção 2.1.2 deste trabalho explica brevemente sobre o *hashing* de dados). A funções de *hash* são como impressões digitais para qualquer tipo de dado. (DRESCHER, 2018.)

Figura 13: Informações Básicas Referentes ao Contrato



Fonte: Autor.

No *deploy*, define-se pelo usuário a meta e o prazo do contrato (como explicado na seção 3.2.1). Portanto, se o usuário quiser que a meta seja 1 ETH e o prazo 10 dias, em *wei* e *timestamp* os valores ficam:

Cálculo da meta (ANTONOPOULOS; WOOD, 2019):

1 ETH=〖10〗^18 *wei* (Tabela 1)

Cálculo do prazo:

1 h= 60min= 3600 s

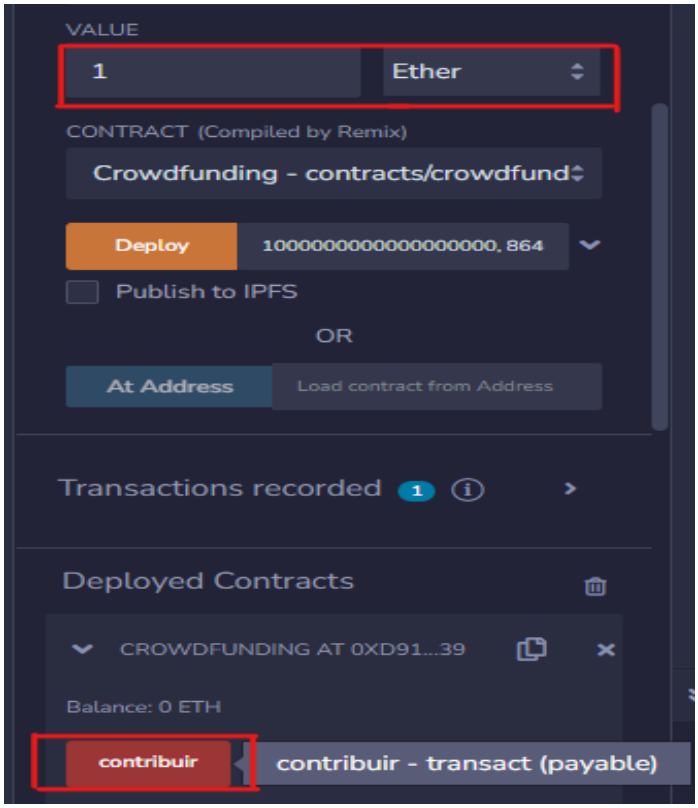
1 dia=24 h=24*3600s= 86400 s

10 dias= 10*86400=864000 s

O cálculo acima basicamente relaciona qual é o valor de 1 ETH expresso na unidade de medida *wei,* e quanto equivale 10 dias expresso em segundos, pois para fins da implementação as variáveis de meta e prazo devem ser expressas em *wei* e segundos, respectivamente.

Com o *deploy* feito, é possível contribuir com o contrato. Para simular a contribuição basta selecionar um dos endereços disponibilizados pela própria IDE, digitar o valor que gostaria de contribuir e, por fim, apertar o botão “contribuir”, como mostra na Figura 14.

Figura 14: Contribuindo com o Contrato.

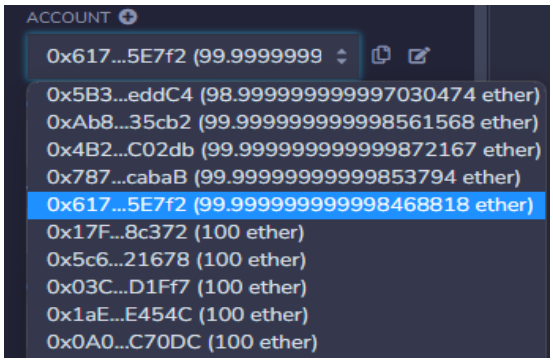


Fonte: Autor.

5.2 RECEBENDO REEMBOLSO

Para testar o funcionamento da função de reembolso é necessário realizar outro *deploy* com um prazo curto, por exemplo, 20 segundos. E dentro desses 20 segundos simular uma contribuição que seja menor que a meta do contrato. Portanto, mantendo a meta anterior de 1 ETH e simulando a contribuição de 100 *wei* da conta 0x167... (o número da conta é resumido em 5 dígitos seguidos de reticencias mais 5 dígitos, isso devido ao número da conta ser muito extenso), com um prazo de 20 segundos, tem-se que o saldo atual da conta do contribuinte é o mostrado na Figura 15.

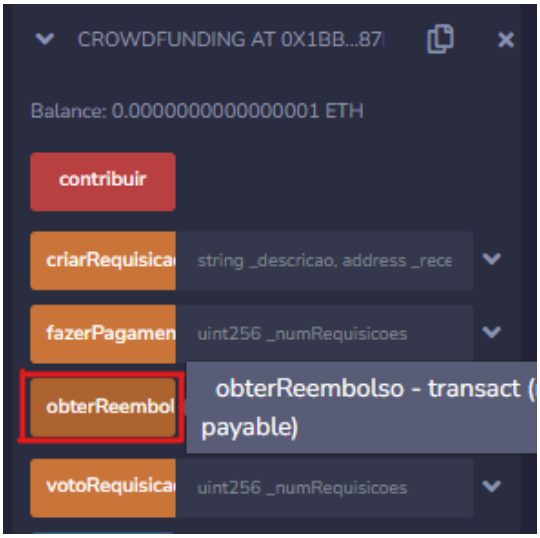
Figura 15: Saldo da Conta 0x167 Após a Contribuição.



Fonte: Autor.

Após os 20 segundos, a conta 0x617... pode solicitar o reembolso selecionando o botão “obterReembolso”, conforme ilustrado na Figura 16.

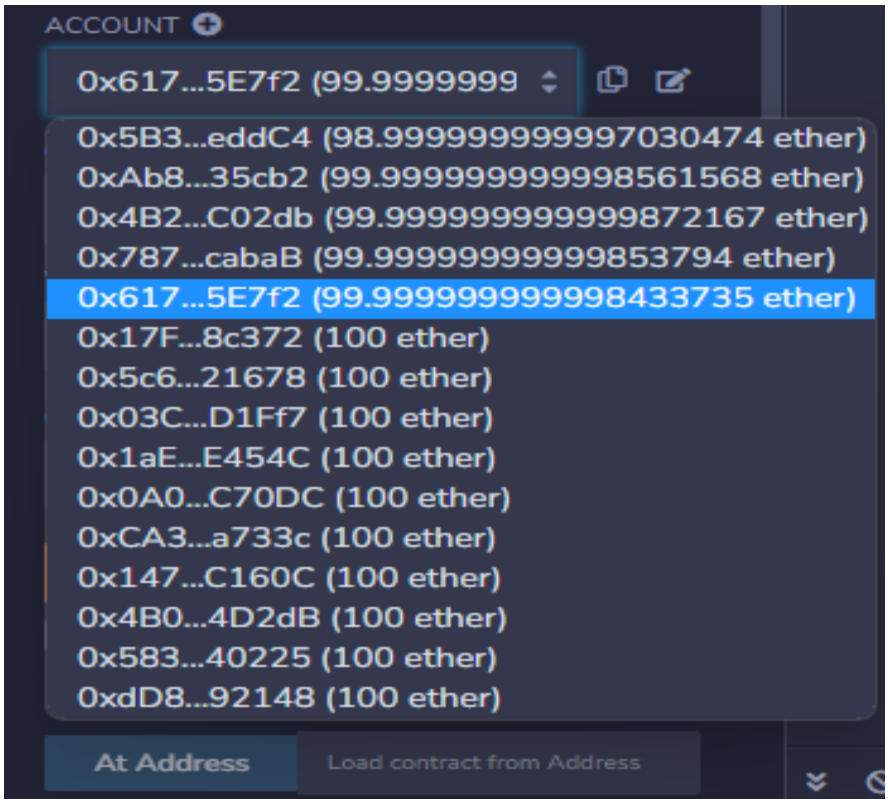
Figura 16: Botão obterReembolso.



Fonte: Autor.

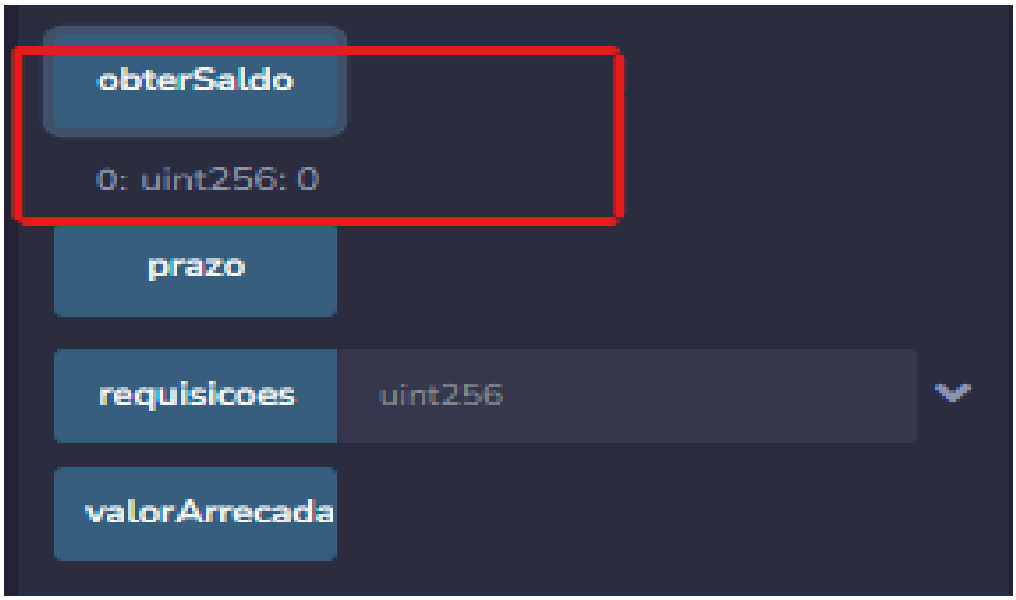
Após acionar o botão, é possível verificar o saldo da conta (Figura 17) e o saldo atual do contrato (Figura 18).

Figura 17: Saldo da Conta 0x617... Após o Reembolso.



Fonte: Autor.

Figura 18: Saldo do contrato após o reembolso.



Fonte: Autor.

De acordo com NAKAMOTO (2008) se o valor da transação de saída for menor do que o valor de entrada, a diferença se torna uma taxa de transação que é adicionada ao valor de estímulo do bloco, portanto, para calcular o valor recebido pelo contribuinte considerando a taxa de transação o cálculo é:

Valor recebido=Valor de entrada-taxa de transação

E para calcular a taxa de transação o cálculo é (PEASTER, 2020):

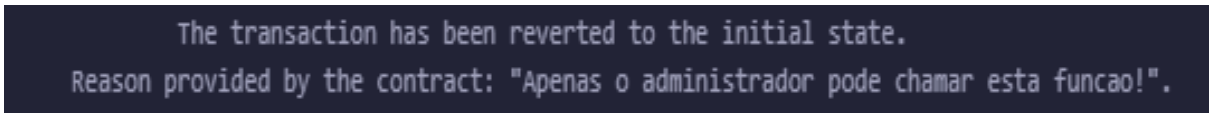
Taxa de transação=limite de *gas* [*wei*]×*gas* [*gwei*]×0,000000001 *eth*

5.3 CRIANDO UMA REQUISIÇÃO DE GASTOS

Para testar o funcionamento da função de solicitação de gastos é necessário realizar outro *deploy*. Neste caso, escolheu-se o endereço 0x617... para realizar o *deploy* e, portanto, ele é o administrador do contato. Assim sendo, somente a conta 0x617... poderá criar solicitação de gasto.

Ao simular outra conta criando uma solicitação de gasto, o contrato retorna a mensagem “Apenas o administrador pode chamar esta função”. Verifica-se isso na Figura 19.

Figura 19: Mensagem de Retorno Após um Usuário Comum Tentar Criar uma Requisição.



Fonte: Autor.

Ao simular a conta do administrador criando uma requisição de gastos, é possível visualizar a requisição ao indicar o índice da requisição na caixa ao lado de “requisicoes”, como mostra a Figura 20.

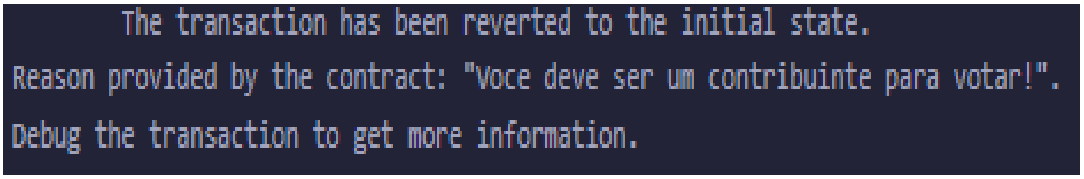
Figura 20: Visualização da requisição de índice 0.

Fonte: Autor.

5.4 VOTAÇÃO EM REQUISIÇÃO DE GASTO

Para testar o funcionamento desta função não será preciso realizar o deploy novamente, pois será reaproveitado o deploy do item anterior, cujo administrador do contrato é o endereço 0x617... . Um dos testes realizados mostra que o endereço 0x617... não pode votar pois não é um contribuinte, conforme ilustrado na Figura 21.

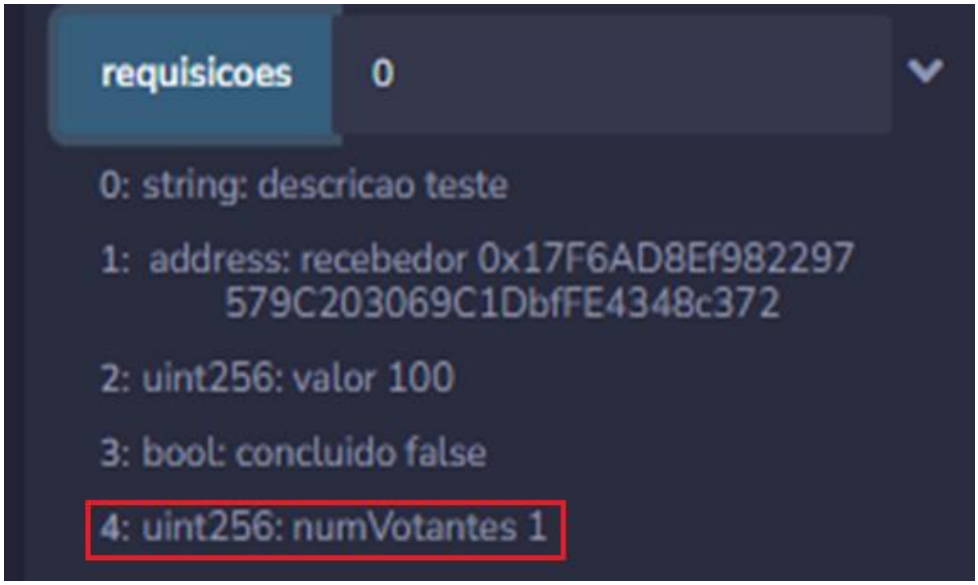
Figura 21: Mensagem de Retorno Após o Administrador Tentar Votar em uma Requisição.



Fonte: Autor.

Ao simular a conta do contribuinte votando na requisição de índice 0, é possível visualizar a requisição indicando que o número de votantes é 1, como mostra a Figura 22.

Figura 22: Atualização de votos na requisição de índice 0.



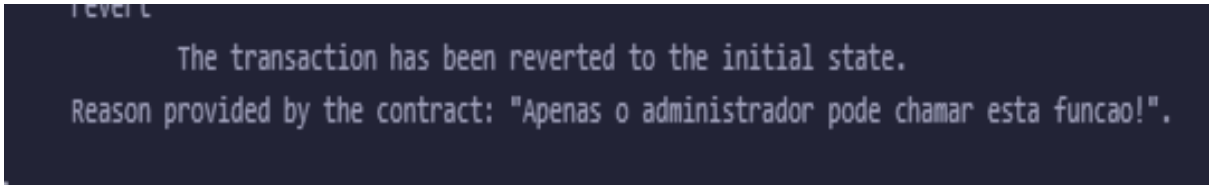
Fonte: Autor.

5.5 REALIZANDO O PAGAMENTO DE UMA REQUISIÇÃO

Para testar o funcionamento desta função não será preciso realizar o *deploy* novamente, pois será reaproveitado o *deploy* do item anterior, cujo administrador do contrato é o endereço 0x617. Um dos testes realizados mostra que um endereço

diferente de 0x617 não pode realizar o pagamento, pois é uma função permitida apenas ao administrador. A figura 23 ilustra a mensagem de retorno após um usuário comum tentar realizar o pagamento.

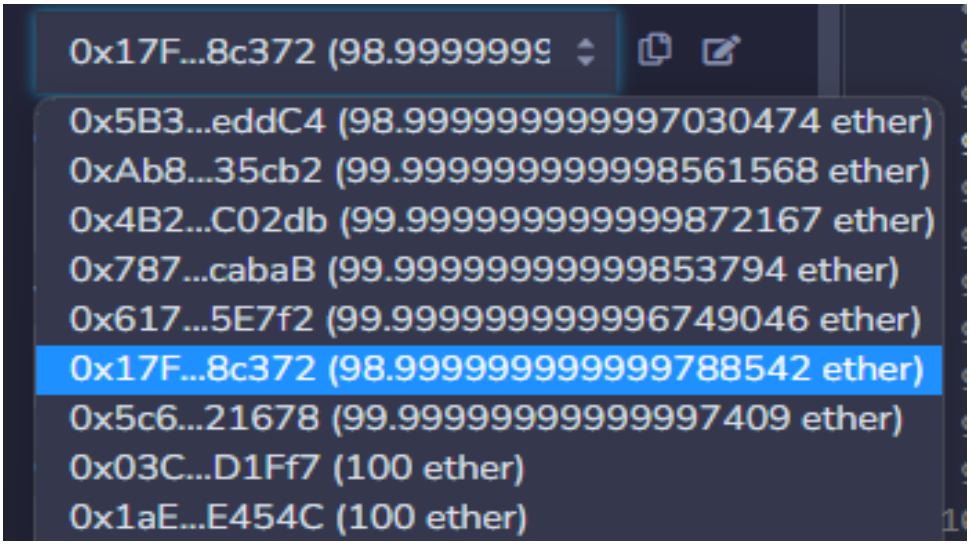
Figura 23: Mensagem de Retorno após um Usuário Comum Tentar Realizar o Pagamento.



Fonte: Autor.

Antes de testar a função sendo acionada pelo administrador, deve-se conferir se o pagamento realmente será realizado ao endereço do recebedor. Observa-se que o saldo atual do recebedor é 98.999999999999788542 *ether* (Figura 24) e que o valor a ser transferido de acordo com a requisição é de 100 *wei* (Figura 22).

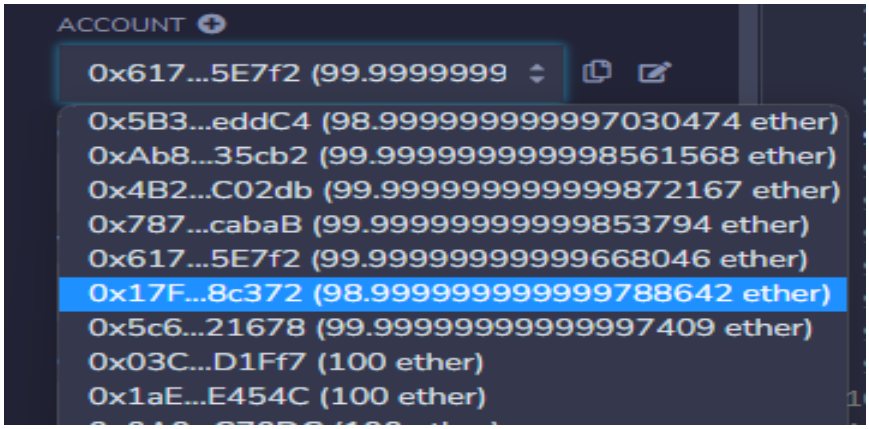
Figura 24: Saldo Atual do Recebedor.



Fonte: Autor.

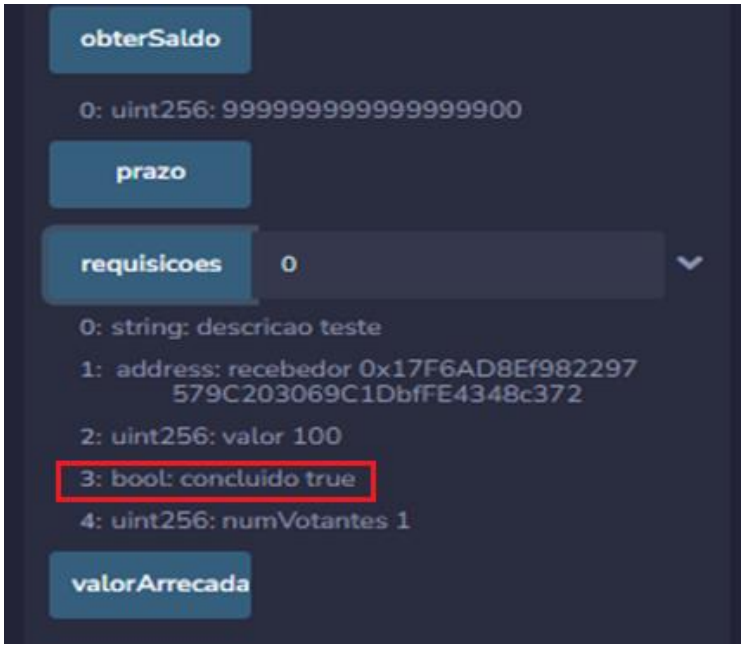
Ao simular a conta do administrador acionando a função do pagamento da requisição de índice 0, é possível visualizar que o endereço 0x17F... está com 100 *wei* a mais no seu saldo (Figura 25), que o saldo do contrato reduziu em 100 *wei* e que o status da requisição mudou para concluído = *true*, conforme Figura 26.

Figura 25: Novo saldo do endereço 0x17F... Após A Realização Do Pagamento.



Fonte: Autor.

Figura 26: Novo Saldo do Contrato e Status da Requisição de Índice 0.



Fonte: Autor.

A necessidade/obrigatoriedade de precisar criar requisições de gasto caracteriza este sistema de *crowdfunding* como um investimento do tipo ativo, de acordo com Schwienbacher e Larralde, (2010) os investimentos ativos permitem que os investidores participem do processo subsequente e afetam diretamente o resultado dos projetos do empreendedor.

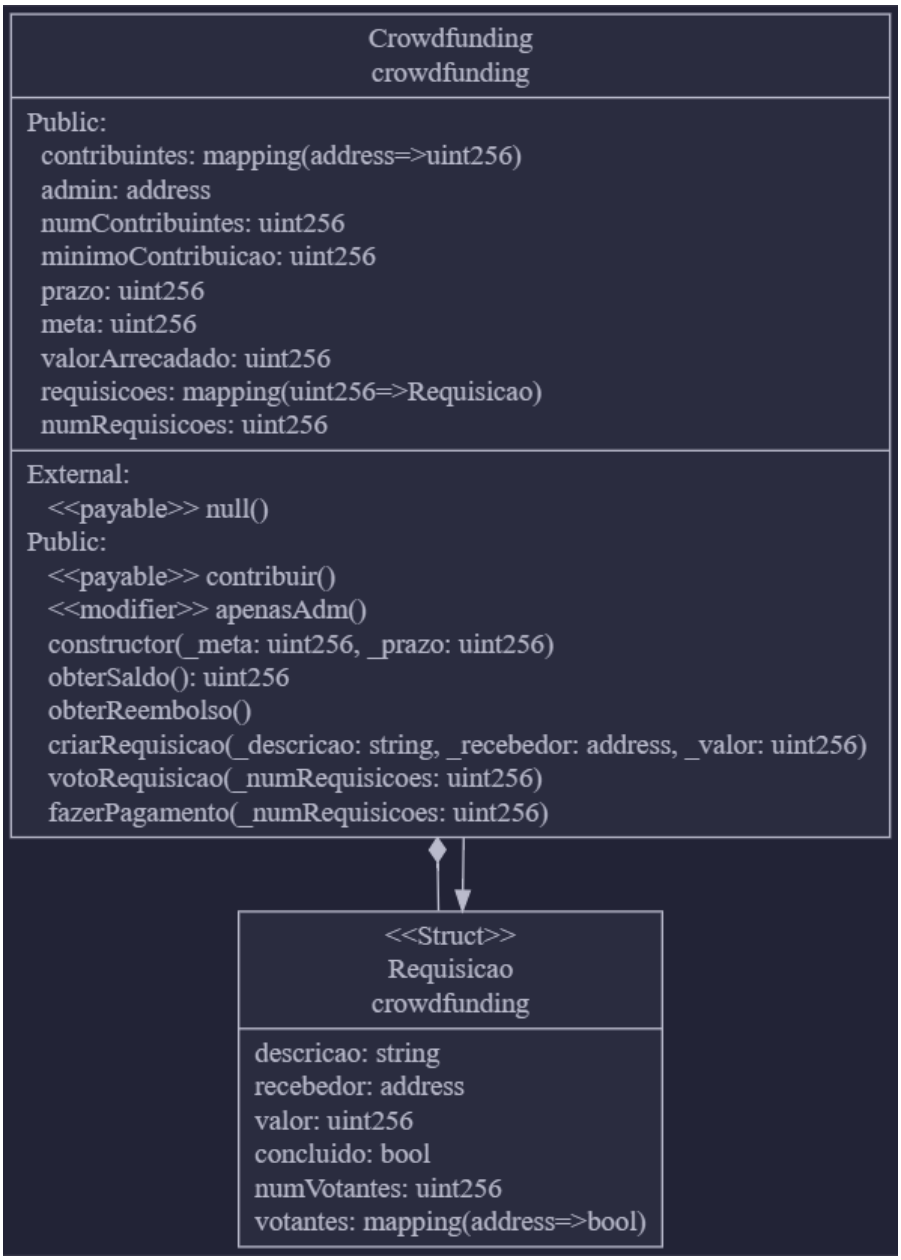
Observa-se também que, a ação realizada após a aprovação da requisição é semelhante a etapa 2 do exemplo de caso de uso de *smart contract* fornecido por

Zheng et al (2020), em que a transferência de fundos vai diretamente para um a carteira de um terceiro, que pode ser um fornecedor por exemplo.

5.6 DIAGRAMA DE CLASSE UML DO CONTRATO

A linguagem de programação Solidity utilizada no desenvolvimento do smart contract deste trabalho é uma linguagem orientada a objetos. Sendo assim, é possível obter um diagrama de classes. A partir do compilador IDE *Remix Ethereum*, foi possível gerar um diagrama de classes UML completo do contrato, conforme figura 27.

Figura 27: Diagrama de classe UML do contrato.



Fonte: Autor.

6 CONCLUSÕES

O presente trabalho cumpriu todos os seus objetivos, pois foi implementada uma solução de *crowdfunding* com a utilização de um *smart contract* em rede *blockchain*. O trabalho propõe e cumpre com o estudo bibliográfico referente assuntos pertinentes como *blockchain*, *smart contracts* e *crowdfunding* e, também, o estudo bibliográfico sobre trabalhos relacionados com o tema. Define-se os requisitos para que a aplicação execute a tarefa proposta e projeta-se diagramas para o funcionamento da aplicação. Após, valida-se a solução implementada em um ambiente de teste no *IDE Remix Ethereum*, onde constata-se a eficácia e desempenho da implementação.

As limitações deste trabalho englobam alguns itens como: a ausência de uma interface com o usuário (*front-end*); o contrato ficou apenas no ambiente de teste da *IDE Remix Ethereum*; na revisão de literatura não foi abordado sobre criptomoedas, o que são, como obtê-las e como ter uma carteira digital. Portanto, como trabalho futuro, espera-se o desenvolvimento de uma interface com o usuário e a implementação do contrato em um ambiente real e produtivo.

REFERÊNCIAS

ALHARBY, Maher; MOORSEL, Aad van. **Blockchain-based Smart Contracts: A Systematic Mapping Study.** Computer Science: Cornell University, 2017.

ANTONOPOULOS, Andreas M.; WOOD, Gavin. **Mastering Ethereum.** 1ª Edição, EUA: O'Reilly Media, 13 de novembro de 2018.

BAMAKAN, Seyed M. H.; MOTAVALI, Amirhossein; BONDARTI, Alireza B. **A survey of blockchain consensus algorithms performance evaluation criteria.** Expert Systems with Applications, v. 154, 2020.

BELLEFLAMME, Paul LAMBERT, Thomas; SCHWIENBACHER, Armin. **Individual Crowfunding Practices.** Venture Capital, 2013.

BELLEFLAMME, Paul; LAMBERT, Thomas; SCHWIENBACHER, Armin. **Crowfunding: Tapping the Right Crowd.** Eletronic Journal, 2012.

BUTERIN, Vitalik. **A next-generation smart contract and decentralized application platform**. White paper, 2014.

CERVANTES-ZACARÉS, Desamparados et al. **The relevance of crowdfunding in the entrepreneurial framework from a specialized media perspective.** Journal of Business Research, v. 158, 2023.

CHEN, Yuting; LIU, Bin. **Advertising and pricing decisions for signaling crowdfunding product's quality.** Computers & Industrial Engineering, v. 176, 2023.

DAI, Wei. **B-Money**, 1998. Disponível em: http://www.weidai.com/bmoney.txt.

DI PIERRO, Massimo. **What Is the Blockchain?** Computing in Science & Engineering, 2017.

DRESCHER, Daniel**. Blockchain básico: Uma introdução não técnica em 25 passos**, 1ª Edição, Editora Novatec, 18 de abril de 2018.

FAIRFIELD, Joshua. **Tokenized: The Law of Non-Fungible Tokens and Unique Digital Property.** Indiana Law Journal, v. 97, 2022.

FOGANG, Francis L. **Blockchain technology: decentralized crowdfunding application implementation on the ethereum platform**. CALIFORNIA STATE UNIVERSITY, 2018.

GOULART, Bruno B.; MARTINS, Paulo J. **Aplicação da Tecnologia Blockchain no Armazenamento de Documentos.** UNESC, 2022.

GRISHCHENKO, Ilya; MAFFEI, Matteo; SCHNEIDEWIND, Clara. **A semantic framework for the security analysis of ethereum smart contract.** Computer Science: Cornell University, v. 2, pp. 243–269, 2018.

JAGGI, Harish; JHA, Raj. **Rendezvous with Practical Solidity: Solidity for novice professionals, advanced programmers, architects, students and academicians**. Notion Press Media Pvt Ltd, 18 de dezembro de 2019.

KIRBY, Eleanor; WORNER, Shane. **Crowd-funding: An Infant Industry Growing Fast.** IOSCO, 2014.

MAIA, Delano J. H. **Uma Solução para Crowdfunding com Transparência em Investimentos Baseada em Blockchain.** UFCE, 2022.

NAKAMOTO, Satoshi. **Bitcoin: A Peer-to-Peer Electronic Cash System**. 2008.

NGUYEN, Cong T. et al. **Proof-of-Stake Consensus Mechanisms for Future Blockchain Networks: Fundamentals, Applications and Opportunities.** IEEEAcess, v. 7, 2019.

PEASTER, William M. **Ethereum as Explained**. DefiPrime, 2020.

QUISQUATERM Jean-Jacques; SERRET-AVILA, Xavier; MASSIAS, Henri. **Design of a secure timestamping service with minimal trust requirements**. 20th Symposium on Information Theory in the Benelux, Satoshi Nakamoto Institute, 1999.

RUBINTON, Brian J**. Crowdfunding: Disintermediated Investment Banking**. Advanced Finance Semiar, 2011.

SCHWIENBACHER, Armin; LARRALDE, Benjamin. **Crowdfunding of Small Entrepreneurial Ventures.** IN: LINGELBACH, David. **Handbook of Entrepreneurial Finance**. Editora da Universidade de Oxford, 28 de setembro 2010.

SONG, Woochul; SHI, Stone; XU, Victoria; GILL, Gursahib**. Advantages & Disadvantages of Blockchain Technology**. Blockchain Technology, 2016. Disponível em: https://blockchaintechnologycom.wordpress.com/2016/11/21/advantages-disadvantages/ . Acesso em: 11 de julho de 2023.

TAPSCOTT, Don; TAPSCOTT, Alex. **Blockchain Revolution: How the Technology Behind Bitcoin Is Changing Money, Business, and the World.** Londres: Editora Portfolio, 10 de maio de 2016.

WAN, Xiaole et al. **Blockchain technology empowers the crowdfunding decision-making of marine ranching.** Expert SYSTEMS With Application, v. 221, 2023.

WEISSTEIN, Eric W. "**Hash Function**." MathWorld - A Wolfram Web Resource. [s. d.]

WOOD, Gavin. **Ethereum: A Secure Decentralised generalised transaction ledger. EIP-150 revision.** Tech. Rep., p. 33, 2017.

ZHENG, Zibin, et al. **An overview on smart contracts: Challenges, advances and platforms.** Future Generation Computer Systems, v. 105, pp: 475 – 491, 2020.