Disciplinarum Scientia. Série: Naturais e Tecnológicas, Santa Maria, v. 24, n. 3, p. 171-190, 2023.
Recebido em: 02.10.2023. Aprovado em: 07.12.2023.

ISSN 2176-462X

DOI: 10.37779/nt.v24i2.4701

REFORMULAÇÃO DE UM SISTEMA DE FINANCIAMENTO COLETIVO EM UM CONTRATO INTELIGENTE NA BLOCKCHAIN DA REDE ETHEREUM

REFORMULATION OF A CROWDFUNDING SYSTEM INTO A SMART CONTRACT IN THE ETHEREUM BLOCKCHAIN

William Lorenzoni Zottele[1], Fabrício Tonetto Londero[2], Sylvio Andre Garcia Vieira[3] e Ana Paula Canal[4]

RESUMO

A tecnologia *blockchain* e seus contratos inteligentes estão revolucionando os negócios *online* ao conectar usuários sem intermediários, aumentando a confiança. Os usuários podem armazenar dados em uma rede global imutável e criar os chamados *Smart Contracts*. O objetivo deste trabalho é criar um contrato inteligente de arrecadação de fundos coletivos usando a tecnologia *blockchain*. Neste, os colaboradores têm controle dos fundos arrecadados por meio de votações em pedidos de saques propostos pelo fundador do contrato. O contrato foi desenvolvido na linguagem de programação Solidity utilizando a plataforma Remix.

Palavras-chave: *Blockchain*, *Ethereum*, Remix, Rede P2P, Solidity.

ABSTRACT

Blockchain technology and its smart contracts are revolutionizing online business by connecting users without intermediaries, increasing trust between them. Users can store data on an immutable global network and create Smart Contracts. The aim of this article is to create a crowdfunding smart contract using blockchain technology. Contributors have control over funds raised through voting for withdrawals requests proposed by the contract´s founder. The contract was developed with Solidity programming language and Remix platform.

Keywords: *Blockchain, Ethereum, Remix, P2P, Solidity.*

1 Acadêmico do curso de Ciência da Computação - Universidade Franciscana - UFN. E-mail: williamzottele@gmail.com ORCID: https://orcid.org/0009-0008-0202-407X

2 Curso de Ciência da Computação - Universidade Franciscana - UFN. E-mail: fabricio.londero@ufn.edu.br. ORCID: https://orcid.org/0000-0002-4822-4981

3 Curso de Ciência da Computação - Universidade Franciscana - UFN. E-mail: sylvio@ufn.edu.br. ORCID: https://orcid.org/0000-0002-1484-4728

4 Curso de Ciência da Computação - Universidade Franciscana - UFN. E-mail: apc@ufn.edu.br. ORCID: https://orcid.org/0000-0001-6360-1688

INTRODUÇÃO

O presente projeto propõe a elaboração de um modelo de *Smart Contract* (Contrato Inteligente) que seja utilizado para fins de *crowdfunding*, conhecido como "vaquinhas online", para arrecadar possíveis recursos a uma ou mais carteiras. Nesse sentido, transformar um sistema de vaquinha em um contrato inteligente digital onde é possível a utilização de criptomoedas como forma de pagamento, torna-se ferramenta útil e facilitadora para quem possui criptomoedas e deseja permanecer anônimo em sua doação.

A ideia é auxiliar pessoas e ONGs a fazerem suas vaquinhas utilizando contratos inteligentes, os quais permitem o desempenho, o monitoramento e a execução dos termos contratuais sem envolvimento de terceiros.

Com o passar dos anos a *blockchain* tem se designado uma tecnologia de banco de dados distribuído que registra todas as transações que aconteceram em uma rede P2P (*peer-to-peer*). A rede da *Ethereum* torna-se uma opção para pessoas utilizarem serviços de desenvolvimento para criação de aplicações em uma plataforma descentralizada. Essa rede também pode ser utilizada como forma de serviços bancários sem a necessidade de providenciar todos os detalhes sobre sua vida pessoal, podendo transferir ativos de pessoa para pessoa. É um modelo de computação distribuída que soluciona o problema em relação à confiança de um sistema centralizado. Desta forma, em uma rede *blockchain*, vários nós trabalham entre si para proteger e manter um conjunto de registros de transações compartilhadas, sem depender de apenas uma parte confiável (KHAN *et al*., 2021).

A *Ethereum* possui um ecossistema maduro, uma grande base de usuários, e é amplamente adotada. O que é um fator crucial para projetos que buscam visibilidade e integração com outros projetos.

Apesar de terem outras redes *blockchains* com baixas taxas de transações compatíveis com a *Ethereum Virtual Machine (*EVM), a *Ethereum* é a que oferece o maior nível de interoperabilidade, o que a torna uma boa opção para o desenvolvimento deste projeto. Um contrato desenvolvido na rede *Ethereum* também pode ser executado em outras redes *blockchains* desde que sejam compatíveis com a EVM.

Neste sentido, criar um modelo de contrato que seja capaz de ser auto-executado reduzindo os custos de transação, formalizando negociações entre as partes, é o diferencial no campo competitivo dos negócios, pois os Contratos Inteligentes trazem economia tanto de recursos, quanto de tempo (KHAN *et al*., 2021).

Conforme a tecnologia avança, os sistemas estão evoluindo cada vez mais e com isso nasce uma nova forma de fazer e cumprir contratos, sem precisar de uma terceira parte validadora. Isso pode aumentar a confiança entre as partes envolvidas, pois proporciona que as partes envolvidas em um contrato realmente cumpram o que foi designado para cada uma.

O objetivo deste trabalho é criar um *Smart Contract* que possibilite a arrecadação de fundos de criptomoedas na rede *Ethereum* utilizando a linguagem de programação Solidity.

REFERENCIAL TEÓRICO

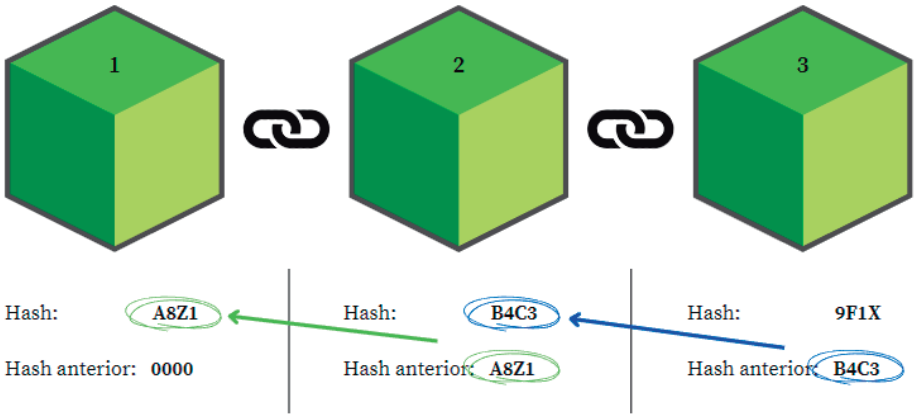
O aporte teórico do trabalho envolve *Blockchain*, especificamente da rede *Ethereum* e como criar um contrato inteligente usando a linguagem de programação Solidity.

Blockchain é um livro-razão (documento contábil) compartilhado e imutável que facilita o processo para registrar transações e rastrear ativos em uma rede empresarial. Um ativo pode ser tangível (uma casa, um carro, dinheiro, terras, …) ou intangível (propriedade intelectual, patentes, direitos autorais e criação de marcas). Praticamente qualquer item de valor pode ser rastreado e negociado em uma rede de *blockchain*, o que reduz os riscos e os custos para todos os envolvidos (GUPTA, 2019).

Segundo Orcutt (2018) "O principal motivo para usar a blockchain é permitir que as pessoas, em particular aquelas que não confiam umas nas outras, compartilhem dados valiosos de maneira segura e inviolável" (ORCUTT, 2018, p. 1).

Todos os conteúdos dos blocos da *blockchain* são escritos e gravados em um livro-razão digital chamado *ledger* e depois de gravados não podem ser apagados. Logo, se qualquer conteúdo do bloco for alterado, a função *hash* também será alterada (YANO *et al.*, 2018).

Figura 1 - Exemplo de *Blockchain*.



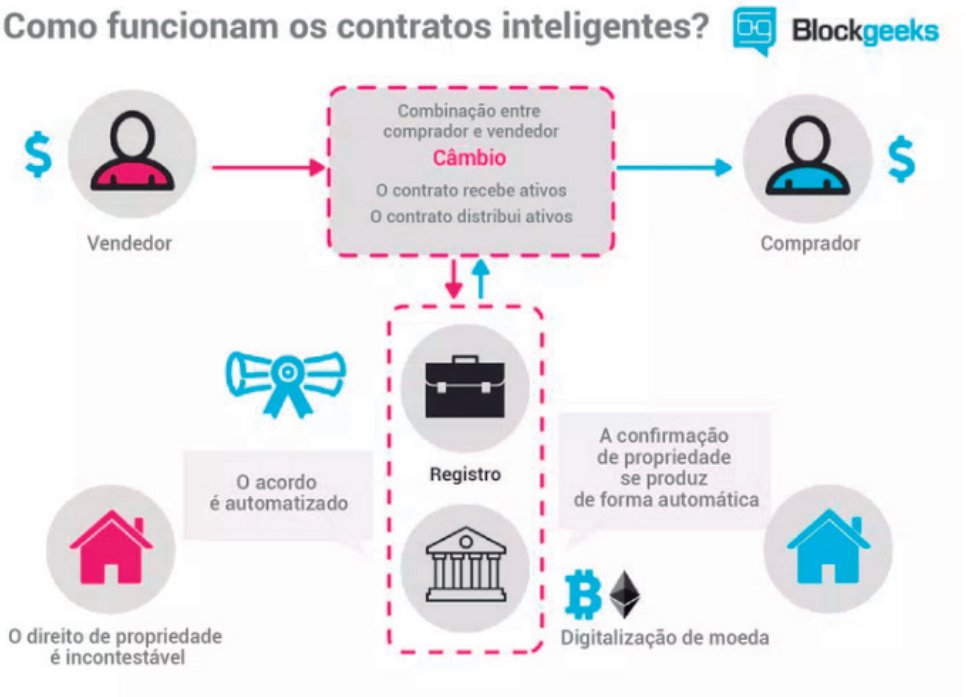
Fonte: Elaborado pelo autor.

Visto que cada bloco da *Blockchain* contém a *hash* do bloco anterior, não é possível modificar nenhum bloco sem mudar completamente a corrente. Por este motivo esta corrente de blocos funciona como uma *ledger* digital completamente imutável. Como no exemplo da Figura 1, todos os blocos possuem uma *hash* e essa *hash* é incluída no próximo bloco. Se alguma informação de um bloco for removida ou alterada a *hash* do bloco irá mudar por completo e o próximo bloco da rede não reconhecerá o *hash* do bloco anterior e assim sucessivamente se propagando pela rede inteira e a invalidando.

Nesse contexto, contrato inteligente ou *Smart Contract* é um acordo ou um conjunto de regras que governa uma transação de negócios, é armazenado na *blockchain* e é executado automaticamente como parte de uma transação (GUPTA, 2019).

O termo *Smart Contract* remete a uma variedade de coisas diferentes. Na década de 1990, o criptógrafo Nick Szabo cunhou o termo e o definiu como "um conjunto de promessas, especificadas em formato digital, incluindo protocolos dentro dos quais as partes cumprem as outras promessas" (SZABO, 1996, p. 28). De acordo com Cardoso (2018), contratos inteligentes podem funcionar como contas "multi-assinaturas", de modo que os fundos são gastos apenas quando uma porcentagem exigida de pessoas concordam" (CARDOSO, 2018, p. 13).

Figura 2 - Exemplo de funcionamento de um Contrato Inteligente.



Fonte: (CARDOSO, 2018, p. 13).

Como pode ser observado na Figura 2, os termos e os ativos de um contrato são codificados e armazenados em um bloco dentro da *Blockchain*. Este contrato passa a ser distribuído e copiado na rede entre os nós da *Blockchain*. Após o cumprimento dos termos do contrato, o contrato é executado e é verificada a transferência de compromissos automaticamente (CARDOSO, 2018).

Ethereum é um software executado em uma rede de computadores que garante que dados e programas, os contratos inteligentes sejam replicados e processados em todos os computadores da rede, sem um coordenador central. O propósito é criar um computador mundial descentralizado e autossustentável, resistente à censura e imparável (LEWIS, 2016).

A *Ethereum* é uma rede de acesso livre a dinheiro digital e serviços consistentes para todos, sem importar sua origem ou local. É uma tecnologia construída pela comunidade da criptomoeda Ether (ETH) e milhares de aplicativos que podem ser usados (ETHEREUM, 2021).

Solidity é uma linguagem de programação de alto nível, orientada a objetos com forte influência das linguagens Python, JavaScript e C++. É executada pela *Ethereum Virtual Machine* (EVM) e é destinada à implementação de contratos inteligentes na rede *Ethereum* (ETHEREUM, 2021). Solidity é tipada, suporta herança, bibliotecas e tipos complexos definidos pelo usuário. Com Solidity é possível criar contratos para votação, vaquinhas, leilões às cegas, e carteiras multi-assinadas (ETHEREUM, 2021). A Solidity surgiu em 2014, como uma proposta de Gavin Wood, sendo desenvolvida por uma equipe da Ethereum liderada por Christian Reitwiessner (SIQUEIRA; DUQUE, 2020).

A criação do contrato inteligente para *crowdfunding* foi feita no ambiente de trabalho integrado Remix (responsável por fazer a interação com a *blockchain* da rede *Ethereum*). Remix é uma IDE (*Integrated Development Environment*) que suporta a linguagem de programação Solidity.

Plataformas de *crowdfunding* são *websites* que possibilitam a interação entre fundadores e o público. As promessas financeiras podem ser feitas e coletadas na plataforma. Geralmente, os arrecadadores de fundos cobram uma taxa pelo uso da plataforma de *crowdfunding* se a campanha de arrecadação de fundos for bem-sucedida. Como retorno, é esperado que as plataformas providenciem segurança e praticidade no uso dela.

A maioria das plataformas operam com o sistema de "tudo ou nada". Isso significa que se a campanha conseguir alcançar o objetivo, o organizador ganha o dinheiro arrecadado. E se a campanha não atingir o objetivo, todos os colaboradores receberão o seu dinheiro de volta.

TRABALHOS CORRELATOS

Foram pesquisados trabalhos relacionados e escolhidos trabalhos relevantes à temática de contratos inteligentes e à tecnologia da *blockchain*. O trabalho realizado por Yano *et al.* (2018) apresentou um contrato inteligente para rastrear a localização da cadeira produtiva da carne de bois em fazendas por meio das transações que acontecem no próprio contrato. A simulação foi feita no ambiente de desenvolvimento Remix.

O artigo apresentado por Kushwaha *et al.* (2022) aborda vulnerabilidades de contratos inteligentes com seus métodos de prevenção, detecção e ferramentas de análises sobre problemas. O trabalho de Ashari *et al.* (2020) faz uma análise de como implementar a tecnologia da *blockchain* e contratos inteligentes no processo de *crowdfunding.*

É possível destacar aspectos que apontam a semelhança dos trabalhos. Em Yano *et al.* (2018), o ambiente de desenvolvimento Remix para a criação de seu contrato inteligente, trazendo conhecimento sobre os passos de desenvolvimento na plataforma do contrato inteligente. O trabalho de Kushwaha *et al.* (2022), o conhecimento sobre as vulnerabilidades de contratos inteligentes e seus

métodos de prevenção, foi importante para que no desenvolvimento do contrato inteligente não ocorressem erros que poderiam deixar o contrato vulnerável. Por fim, o trabalho de Ashari *et al.* (2020), faz uma análise de como implementar a tecnologia da *blockchain* em um esquema de *crowdfunding* utilizando contratos inteligentes, o objetivo final deste projeto.

METODOLOGIA

Inicialmente o trabalho foi desenvolvido por meio de estudos bibliográficos sobre os conteúdos de *Blockchain*: Khan *et al.* (2021), Gupta (2019), Orcutt (2018). *Smart Contracts*: Khan *et al.* (2021), Szabo (1996), Cardoso (2018). *Ethereum*: Lewis (2016), Kushwaha *et al.* (2022) e a linguagem de programação Solidity: Ethereum (2021).

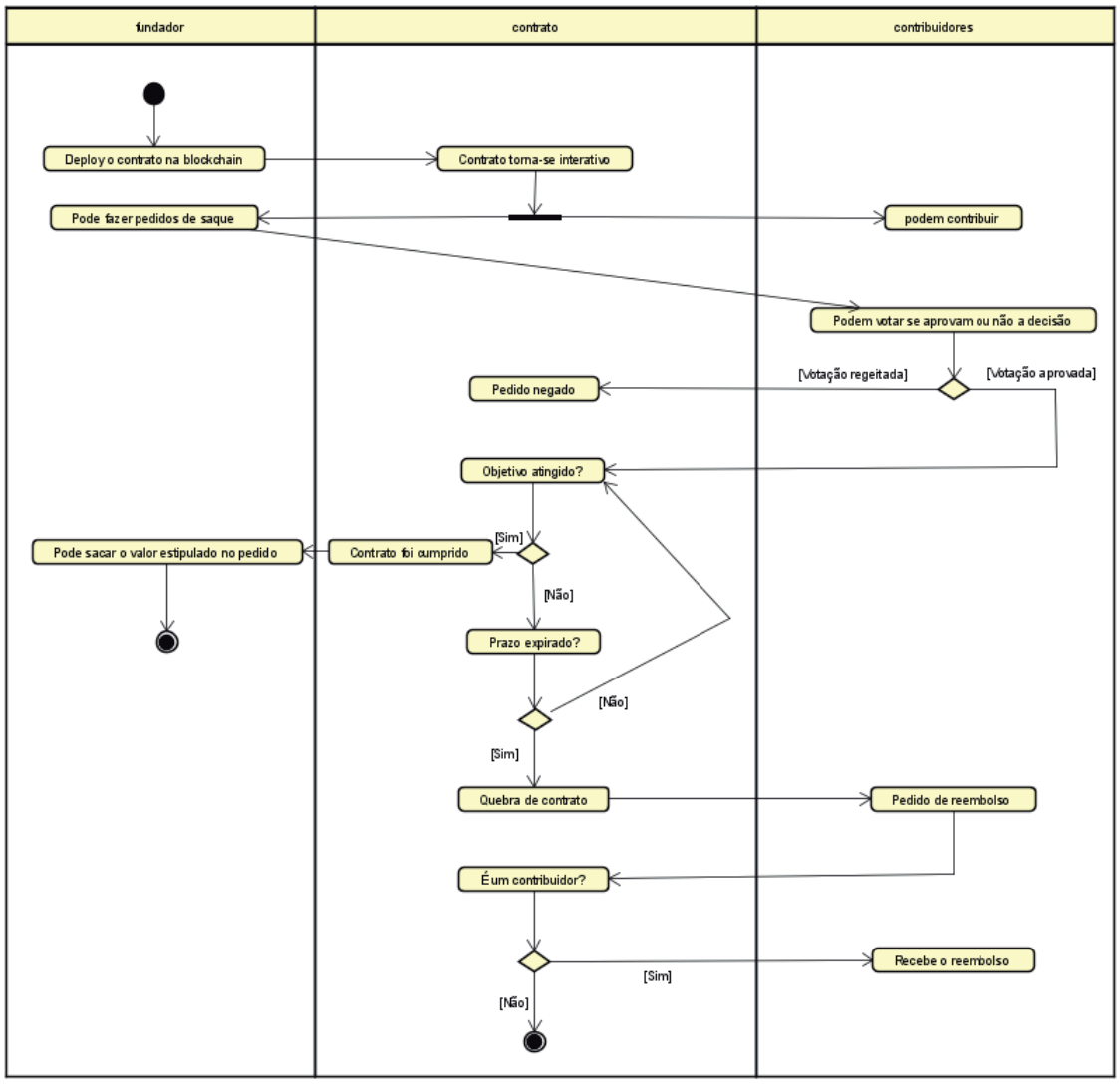
A metodologia ágil utilizada para o desenvolvimento do projeto foi a metodologia FDD (*Feature-Driven Development*). Segundo Pressman (2010), existem cinco atividades metodológicas que definem a abordagem FDD: Desenvolver um modelo geral, Construir uma lista de funcionalidades, Planejar por funcionalidades, Projetar por funcionalidades, Desenvolver por funcionalidade (PRESSMAN, 2010).

PROPOSTA DA CRIAÇÃO DO CONTRATO INTELIGENTE

Na era da Internet, qualquer pessoa pode doar dinheiro com a finalidade de ajudar projetos. Porém, as plataformas de *crowdfunding* ainda apresentam muitas falhas, como no momento de identificar a veracidade do projeto de *crowdfunding* ou da pessoa por trás dele. Qualquer pessoa pode criar uma conta, em uma plataforma de *crowdfunding*, postar um projeto, fazer o marketing, arrecadar uma quantia e simplesmente abandonar o projeto e fugir com o dinheiro arrecadado.

Algumas plataformas de *crowdfunding* atuais não conseguem prevenir uma campanha de arrecadação com más intenções. Existem vários casos de fraudes, nos quais pessoas de má índole se aproveitam de notícias trágicas para fundar projetos falsos de *crowdfunding* mesmo sem ter nenhum tipo de ligação com as vítimas de tal acontecimento.

Figura 3 - Diagrama de Atividades.



Fonte: Elaborado pelo autor.

Este problema pode ser solucionado pela construção de um contrato inteligente na rede *Ethereum*, que possibilita o controle de como o gerente da campanha gasta o dinheiro que foi arrecadado por meio de votação entre os financiadores da campanha. A proposta deste trabalho é construir um contrato inteligente que possa prevenir que pessoas sejam enganadas pelo fundador do projeto, optando por criar um contrato inteligente com mais transparência e segurança para que os contribuidores tenham confiança no projeto. Neste sentido, foi proposto:

- O fundador irá iniciar uma campanha de *crowdfunding* com um objetivo específico, a quantidade de dinheiro necessária e o prazo limite para a arrecadação.
- Os contribuidores irão financiar o projeto por meio de doações em ETH.
- O fundador, para poder usar o dinheiro arrecadado, terá de criar um pedido de saque. Este pedido poderá ser feito a qualquer momento, independente se a campanha alcançou ou não o seu objetivo.

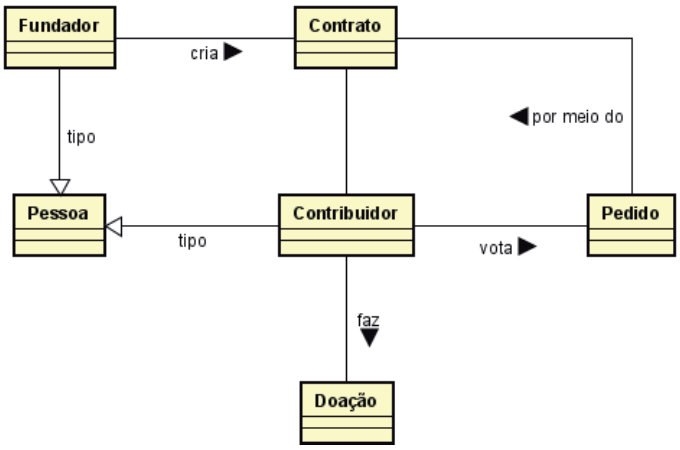
- Depois que o pedido for criado, os contribuidores poderão começar a votar se querem ou não aprovar o pedido de saque.
- Se mais de 50% dos contribuidores apoiarem a decisão, o fundador terá permissão para gastar o valor especificado no pedido de saque.
- Cada contribuinte terá um percentual diferente de votação, conforme a quantidade de ETH doados (quanto maior a quantia em ETH, maior o impacto percentual).
- Se o objetivo da arrecadação não for alcançado até o prazo limite, os contribuidores poderão fazer um pedido de reembolso.

Desta forma, o contrato está baseado no fluxo de processos ilustrado no Diagrama de Atividades da Figura 3. Nas próximas seções, a análise, projeto e desenvolvimento do contrato são descritos, conforme as etapas da metodologia FDD.

DESENVOLVER UM MODELO GERAL

O primeiro processo da metodologia FDD é o desenvolvimento de um modelo geral, possibilitando uma visão geral das entidades do domínio da aplicação (Figura 4). No diagrama, é possível observar as entidades conceituais do domínio e as suas relações.

Figura 4 - Diagrama de Domínio.



Fonte: Elaborado pelo autor.

Os próximos passos da metodologia são o desenvolvimento de uma lista de funcionalidades do sistema e o planejamento por funcionalidade.

CONSTRUIR UMA LISTA DE FUNCIONALIDADES E PLANEJAR

No segundo e terceiro processos da metodologia FDD, devem ser listados todos os requisitos funcionais e não funcionais do sistema e ordenar as funcionalidades conforme a prioridade em que serão desenvolvidas. O Quadro 1 contém a lista dos Requisitos Funcionais (os destacados em verde são ações dos contribuidores e os destacados em azul, do fundador) e o Quadro 2, a lista dos Requisitos Não Funcionais.

Quadro 1 - Requisitos Funcionais.

Requisitos Funcionais	Complexidade	Relevância
RF01 – Receber doações	Média	Essencial
RF02 – Fazer pedidos de saques	Média	Essencial
RF03 – Votar em pedidos de saque	Baixa	Essencial
RF04 – Consultar o saldo existente	Baixa	Essencial
RF05 – Consultar quantos contribuidores existem	Baixa	Essencial
RF06 – Consultar se existe pedido de saque pendente	Baixa	Essencial
RF07 – Consultar o prazo final do contrato	Baixa	Essencial
RF08 – Sacar a quantia informada no pedido de saque, caso o pedido seja aprovado pelos contribuintes.	Média	Essencial
RF09 – Fazer um pedido de reembolso caso ocorra quebra de contrato	Média	Essencial
RF10 – Consultar o saldo total do contrato	Baixa	Essencial
RF11 – Consultar se o contrato atingiu o objetivo final	Baixa	Essencial

Fonte: Elaborado pelo autor

Quadro 2 - Requisitos não Funcionais.

RNF01 – Linguagem de Programação: O contrato será desenvolvido em Solidity.
RNF02 – Ambiente de desenvolvimento: O ambiente de desenvolvimento que será utilizado é o Remix.
RNF03 – Blockchain: O deploy do contrato será feito na rede da Ethereum.

Fonte: Elaborado pelo autor

PROJETAR E DESENVOLVER POR FUNCIONALIDADES

As etapas Projetar e Desenvolver por Funcionalidades são iterativas e são descritas nesta seção. No ambiente Remix, com a linguagem de programação Solidity, o contrato foi implementado na rede *Ethereum*. O desenvolvimento do contrato inicia com a versão do compilador e o nome dado para o contrato, como mostrado na Figura 5. Nas linhas seguintes ocorrem as declarações de atributos e métodos do contrato.

Figura 5 - Declaração de atributos do contrato.

```
1   pragma solidity ^0.8.18;
2
3   contract Crowdfunding {
4
5       address public owner;
6       uint public totalFunds;
7       uint public numDonors;
8       uint public deadline;
9       uint public fundingGoal;
10      bool public goalReached = false;
11      uint public numRequests;
12      mapping(address => uint) public donations;
13      mapping(address => uint) public donationPercentages;
14      mapping (uint => WithdrawalRequest) public WithdrawalRequests;
15      event DonationReceived(address donor, uint amount);
16
```

Fonte: Elaborado pelo autor

Na linha 18 (Figura 6) ocorre a criação de um *modifier* chamado *onlyOwner*, esse *modifier* é usado para modificar o comportamento de um método onde se pretende tornar a função executável apenas pelo *owner* do contrato.

Figura 6 - Criação do *modifier onlyOwner*.

```
17    // Modificador para restringir o acesso apenas ao dono do contrato
18    modifier onlyOwner() {
19        require(msg.sender == owner, "Only the contract owner can perform this action.");
20        _;
21    }
```

Fonte: Elaborado pelo autor

O método *constructor* na linha 24, da Figura 7 é especialmente usado para inicializar os atributos do contrato, este é o método responsável pelo *deploy* do contrato.

Figura 7 - Iniciando os estados dos atributos usando o método *constructor*.

```
23    // Construtor que define o endereço do dono do contrato e a meta de arrecadação e prazo limite
24    constructor(uint _fundingGoal, uint _deadline) {
25        owner = msg.sender;
26        fundingGoal = _fundingGoal;
27        deadline = block.timestamp + _deadline;
28    }
```

Fonte: Elaborado pelo autor

As doações do contrato ocorrem por meio do método *donate* (Figura 8) que começa na linha 31, esse método é de tipo *payable* que significa que o método recebe Ether como parâmetro. As checagens são feitas por meio dos métodos *require,* para checar as entradas e as condições antes da execução, por exemplo, se a condição for falsa o método *require* interrompe a execução imediatamente.

Figura 8 - Método responsável pelas transferências de Ethers para o contrato.

```
30      // Função que permite a doação de ether para o contrato, desde que o prazo não tenh
31      function donate() public payable {
32          require(block.timestamp < deadline, "The deadline for donations has passed.");
33          require(!goalReached, "The funding goal has already been reached.");
34          require(msg.value > 0, "Donation amount must be greater than zero.");
35          donations[msg.sender] = msg.value;
36          totalFunds += msg.value;
37          numDonors++;
38          emit DonationReceived(msg.sender, msg.value);
39
40          if (totalFunds >= fundingGoal) {
41              goalReached = true;
42          }
43      }
```

Fonte: Elaborado pelo autor.

As aprovações de saques são feitas pelo método *approveWithdrawalRequest* (Figura 9). Neste, estão presentes os cálculos dos pesos de votação do usuário, conforme a quantia doada e o total de fundos que o contrato possui. O método é executado e o cálculo é feito após o usuário clicar no botão *approveWithdrawalRequest* somando o peso da votação no atributo *numOfApprovals* e alterando o estado de aprovação do usuário para *true*.

Figura 9 - Método responsável pelo número de aprovação de um pedido de saque e pelo cálculo do peso de votação de um usuário.

```
92      // Função para um donor poder aprovar um pedido de saque
93      function approveWithdrawalRequest(uint requestIndex) public {
94          require(goalReached, "The funding goal has not been reached yet.");
95          require(donations[msg.sender] > 0, "You must be a donor to approve a withdrawal request.");
96
97          WithdrawalRequest storage request = WithdrawalRequests[requestIndex];
98          require(!request.executed, "The withdrawal request has already been executed.");
99          require(!request.approvals[msg.sender], "You have already approved this withdrawal request.");
100
101         uint donationPercentage = (donations[msg.sender] * 100) / totalFunds;
102         donationPercentages[msg.sender] = donationPercentage;
103
104         uint votingWeight = (donationPercentages[msg.sender]);
105
106         request.numOfApprovals += votingWeight;
107         request.approvals[msg.sender] = true;
108     }
109
```

Fonte: Elaborado pelo autor.

Após um pedido de saque ser aprovado o *owner* do contrato, por meio do botão *transferWithdrawalRequest,* dispara o método *transferWithdrawalRequest* (Figura 10) método responsável por checar se o pedido foi aprovado por mais de 50% dos *donors* do contrato e que se verdadeiro possibilita a requisição, ou seja, *request* é alterado para *true* e é feita a transferência dos fundos especificados no endereço do pedido de saque.

Figura 10 - Método para checar aprovação de saque e fazer transferência ao *owner*.

```
110     // Função que transfere os fundos descritos no pedido de saque especificado
111     function transferWithdrawalRequest(uint requestIndex) public onlyOwner{
112         require(goalReached, "The funding goal has not been reached yet.");
113 
114         WithdrawalRequest storage request = WithdrawalRequests[requestIndex];
115         require(!request.executed, "The withdrawal request has already been executed.");
116 
117         if (request.numOfApprovals > 50) {
118 
119             request.executed = true;
120             payable(owner).transfer(request.amount);
121         }
122     }
123 }
```

Fonte: Elaborado pelo autor.

Em caso de não cumprimento dos termos do contrato e o prazo para a doação expirar, há o método *refund* (Figura 11), responsável pelo reembolso dos usuários que doaram para o contrato. Esse pode ser chamado pelo usuário em caso de o prazo para a arrecadação de fundos ter expirado, assim o doador consegue seu dinheiro de volta.

Figura 11 - Método responsável pelo reembolso dos usuários em caso de não cumprimento dos termos do contrato.

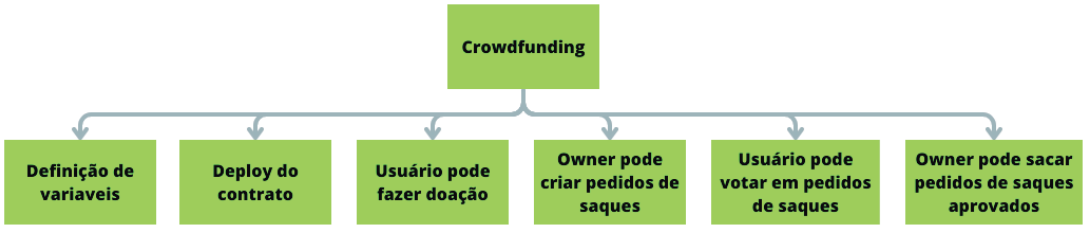
```
76      // Função para que os donors recebam o reembolso caso o contrato não for cumprido
77      function refund() public {
78          require(block.timestamp >= deadline, "The deadline for donations has not passed yet.");
79          require(!goalReached, "The funding goal has already been reached.");
80 
81          uint donation = donations[msg.sender];
82          require(donation > 0, "You have not made any donations to this campaign.");
83 
84          donations[msg.sender] = 0;
85          totalFunds -= donation;
86          numDonors--;
87          emit DonationReceived(msg.sender, donation);
88 
89          payable(msg.sender).transfer(donation);
90      }
```

Fonte: Elaborado pelo autor.

SIMULAÇÃO DE FINANCIAMENTO COLETIVO COM *SMART CONTRACT*

A simulação foi realizada no ambiente de desenvolvimento Remix, ferramenta responsável por integrar com a *blockchain* da rede *Ethereum*. Também, todas as simulações e testes do *Smart Contract* foram feitos no Remix. O *Smart Contract Crowdfunding* foi dividido em seis atividades, contemplando sua estrutura analítica (Figura 12).

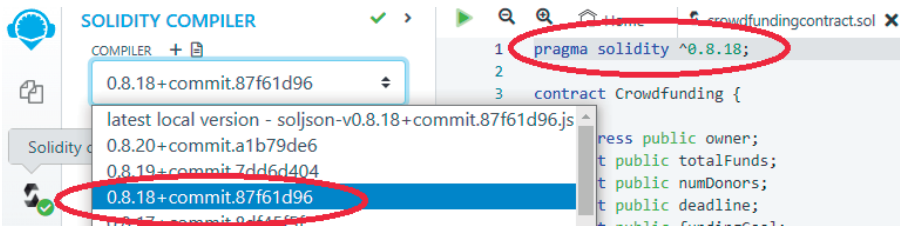
Figura 12 - Estrutura analítica do contrato *Crowdfunding*.



Fonte: Elaborado pelo autor.

Inicialmente, é selecionado o compilador do contrato para que possa checar se não existe algum erro que possa impedir a execução do contrato (Figura 13).

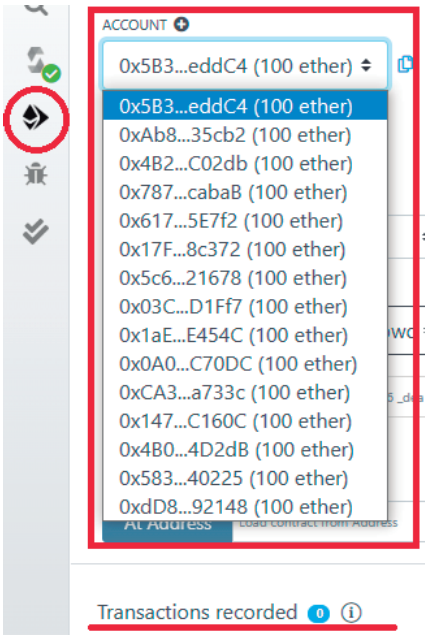
Figura 13 - Ambiente de desenvolvimento Remix, selecionando o compilador.



Fonte: Elaborado pelo autor.

Assim que o contrato for compilado, a simulação pode ser iniciada clicando-se na aba "*Deploy and run transactions*" e selecionada uma das 15 contas disponibilizadas pela plataforma para que seja feito o *deploy* do contrato. Nesta simulação são utilizadas as 4 primeiras contas, como na Figura 14. O primeiro endereço é o endereço responsável pelo *deploy* do contrato na rede, portanto o endereço ``0x5B3...eddC4'' será o *Owner* do contrato e os demais endereços serão os doadores do contrato.

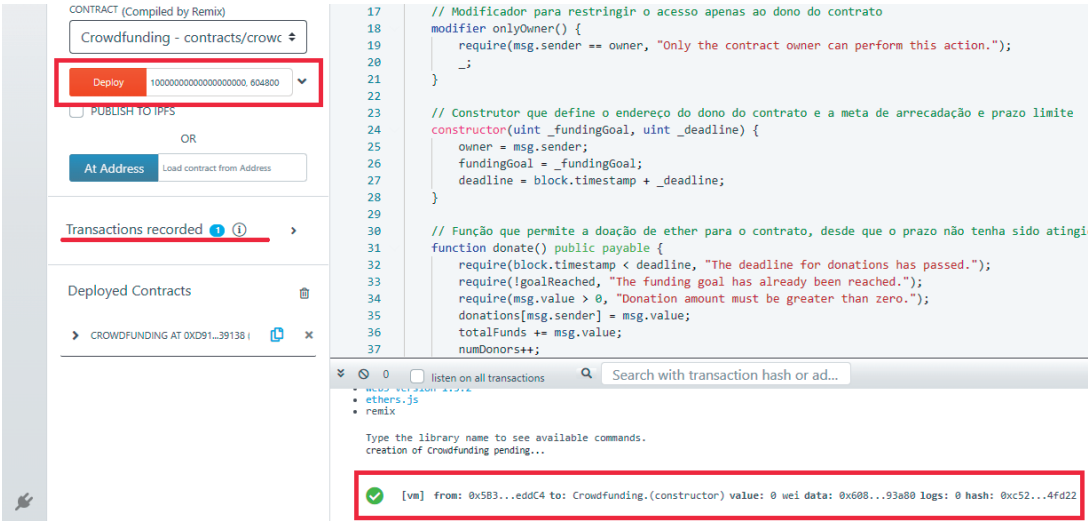
Figura 14 - Ambiente de desenvolvimento Remix, selecionando contas.



Fonte: Elaborado pelo autor.

A primeira interação do contrato é feita pelo endereço ``0x5B3...eddC4", no botão de *Deploy,* chamando o *constructor* do contrato e tornando o endereço *owner* do mesmo. Os atributos passados são o objetivo a ser atingido em Wei (menor denominação da moeda Ether, 1 Ether equivale a 1e18 Wei) e o prazo do contrato em segundos. O sucesso da transação é verificado pelo sinal de checagem verde na parte inferior da imagem da Figura 15 e o número de transações do contrato é acrescentado. Percebe-se que o saldo da conta ``0x5B3...eddC4" tem uma leve alteração, isto acontece em todas as transações do contrato em consequência das taxas de transações da rede Ethereum que servem para evitar *spams* na rede.

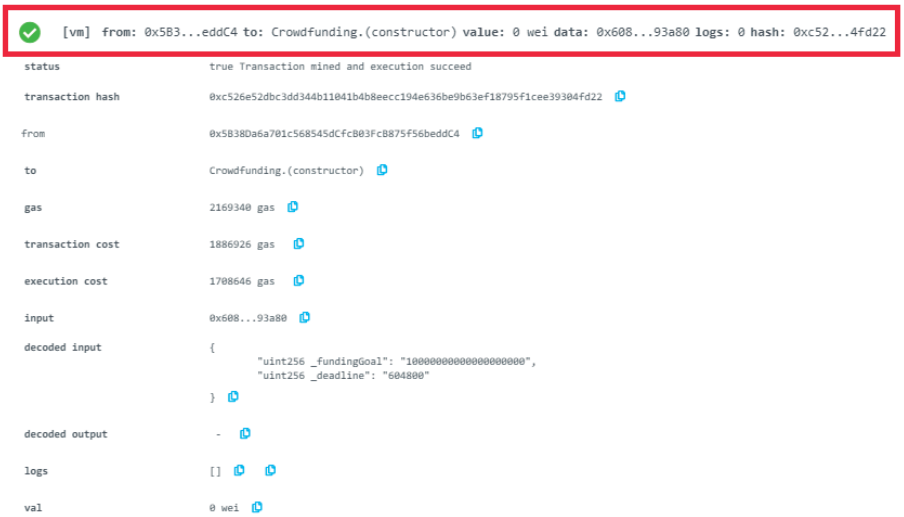
Figura 15 - *Deploy* do contrato.



Fonte: Elaborado pelo autor.

É possível obter todas as informações relacionadas à transação clicando na transação desejada. Como por exemplo na Figura 16, status, *hash* da transação, endereço remetente, operação executada e informações sobre custos da transação.

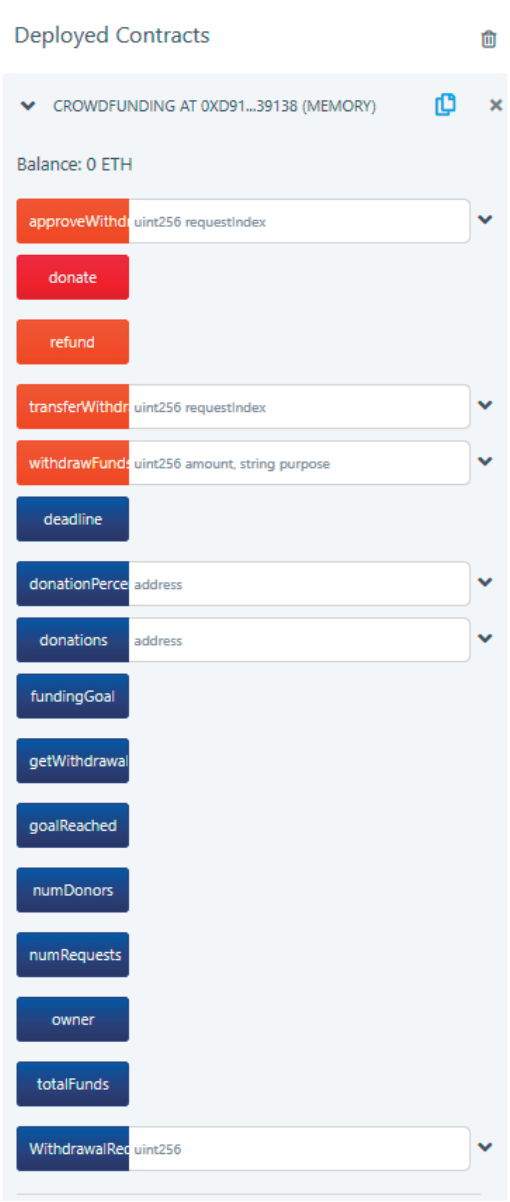
Figura 16 - Detalhes do *deploy*.



Fonte: Elaborado pelo autor.

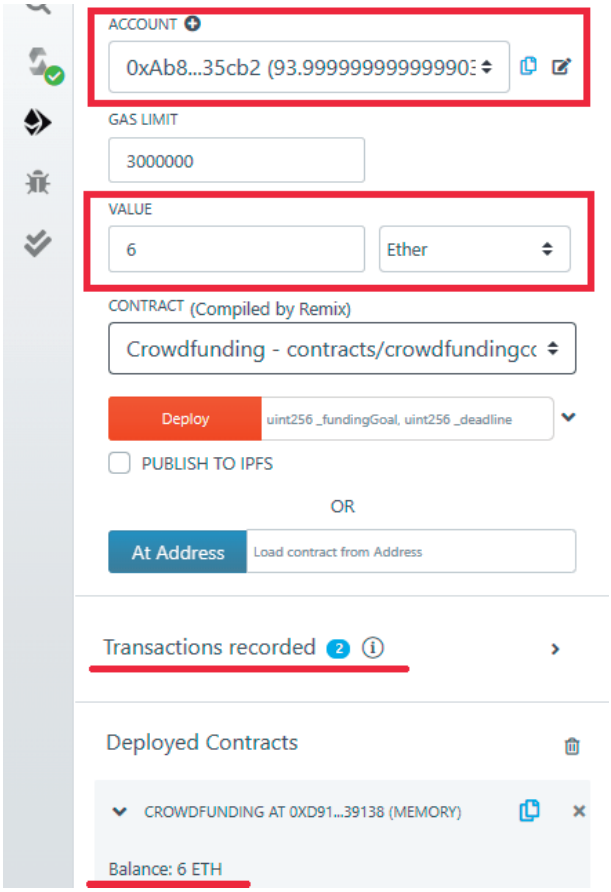
A Figura 17 contém os atributos do contrato em azul, os métodos em laranja e o método *donate* em vermelho

Figura 17 - *Calls* e Funções.



Fonte: Elaborado pelo autor.

Figura 18 - Fazendo uma doação.



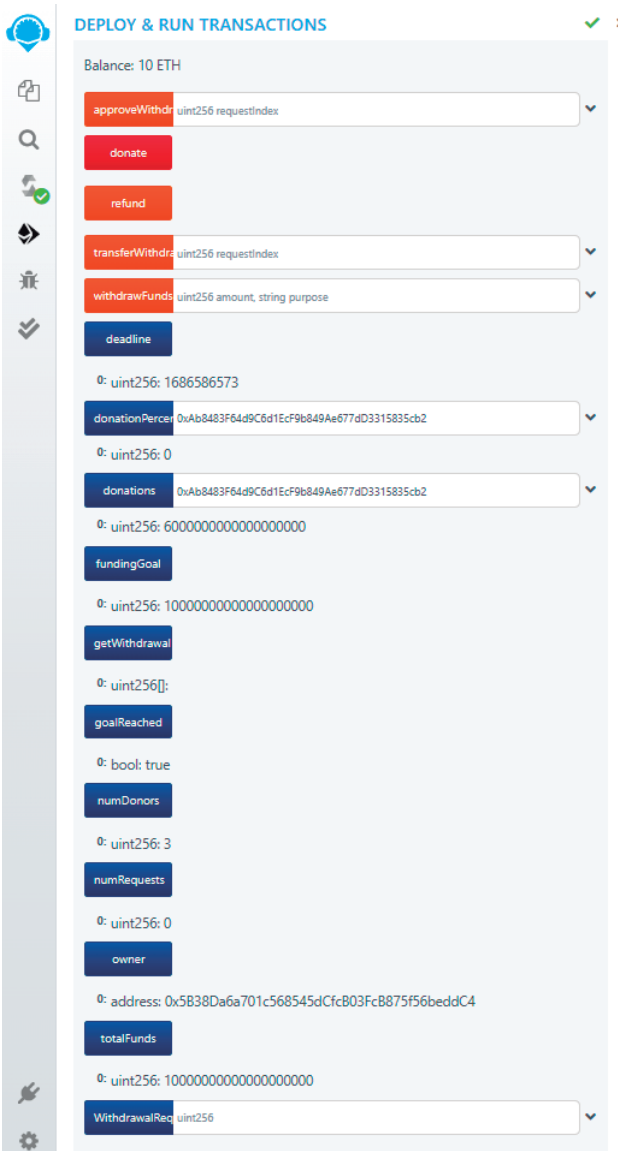
Fonte: Elaborado pelo autor.

A variedade nas cores identifica o tipo da variável. Atributos constantes não criam transações, ou seja, não há mudanças em seu estado, apenas retorna o valor armazenado no contrato e não será aplicada taxas de gás (taxa paga na rede em troca do uso do poder computacional da plataforma) em sua execução e por isto recebem a cor azul. Os métodos que têm seu estado alterado durante sua execução e que não recebem Ether são métodos *non-payable* e recebem a cor laranja. Já o método *donate* é um método *payable* que recebe Ether em sua execução e por isto sua cor é vermelha.

As outras duas doações são feitas pelos endereços ``0x4B2...C02db'' de seis Ether (Figura 18) e ``0x787...cabaB'' de quatro Ether, alcançando a meta do contrato: dez Ether.

Após finalizar as doações, o conteúdo das variáveis pode ser conferido para saber se a meta foi atingida, quantos doadores o contrato possui, qual o endereço do *owner*, quanto um doador específico doou, como na Figura 19.

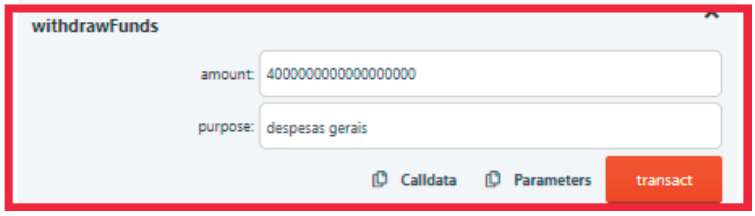
Figura 19 - Checando o conteúdo das variáveis.



Fonte: Elaborado pelo autor.

Com o objetivo do contrato cumprido, o *owner* pode fazer pedidos de saque, se esses são aprovados, podem ser sacados diretamente para o seu endereço. O pedido de saque deve conter o valor em Wei que será retirado do montante do contrato e o propósito do saque. Apenas o *owner* conseguirá realizar um pedido de saque e o seu endereço deve estar selecionado, como na Figura 20.

Figura 20 - Fazendo um pedido de saque.



Fonte: Elaborado pelo autor.

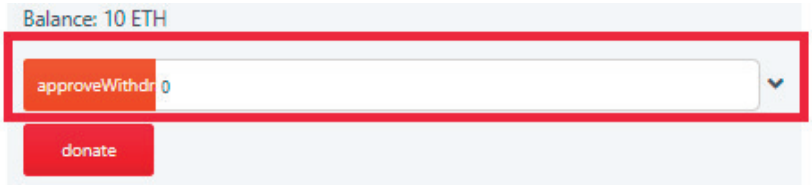
Após um pedido de saque ser criado, é possível consultar quantos pedidos de saques (*numRequests*) pelo botão *getWithdrawalRequests*. Também, é possível consultar um pedido de saque específico por meio do botão *WithdrawalRequests*, que disponibiliza todos os pedidos de saques criados no contrato. Para cada pedido, tem-se o valor de saque, o seu propósito, se o saque já foi executado ou não e a porcentagem do número de aprovações do pedido. O cálculo da porcentagem é baseado no valor que cada endereço doou, podendo assim uma pessoa ter peso de votação maior do que outras (Figura 21).

Após um pedido de saque ser consultado, endereços que doaram para o contrato podem decidir se apoiam ou não um determinado pedido de saque. Para aprovar um pedido de saque, o usuário deve informar o índice do pedido do saque no botão *approveWithdrawalRequest*. O contrato faz todas as checagens se o endereço é válido para aprovar um pedido de saque. Também, é feito o cálculo do peso de voto que o endereço possui e computado o percentual do voto em relação ao valor total do contrato (Figura 22).

Figura 21 - Consultando o pedido de saque.

Fonte: Elaborado pelo autor

Figura 22 - Aprovar um pedido de saque.

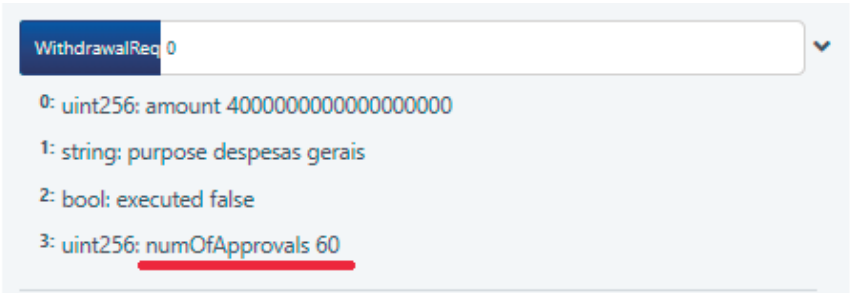


Fonte: Elaborado pelo autor

É possível consultar o pedido de saque depois do voto, a fim de checar se o número de aprovações alcançou mais de 50%. Na Figura 23 há um exemplo com aprovação de 60%.

Depois que um pedido de saque receber mais de 50% de aprovação entre os doadores do contrato, o *owner* do contrato pode fazer a requisição do saque. Isso é realizado, inserindo o índice do pedido aprovado no método *transferWithdrawalRequest*.

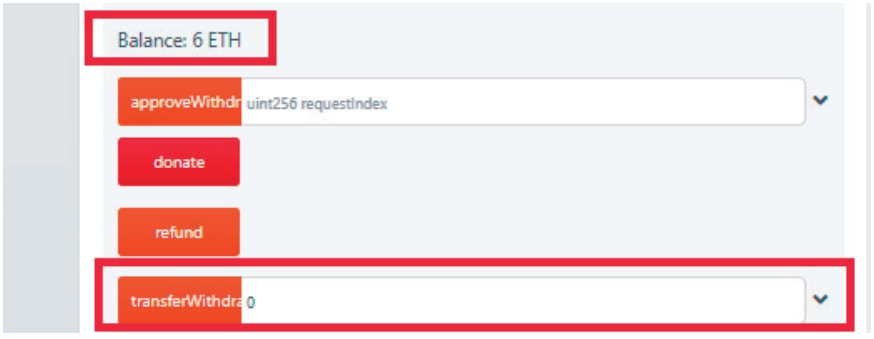
Figura 23 - Consultando o pedido de saque depois do voto.



Fonte: Elaborado pelo autor.

O contrato faz a checagem e se todos os requisitos forem aceitos, então a quantia especificada no pedido é transferida para o endereço do *owner*. Na Figura 24, nota-se que o saldo do contrato é debitado de 10 ETH para 6 ETH e que o saldo do endereço do *owner* é aumentado em 4 ETH.

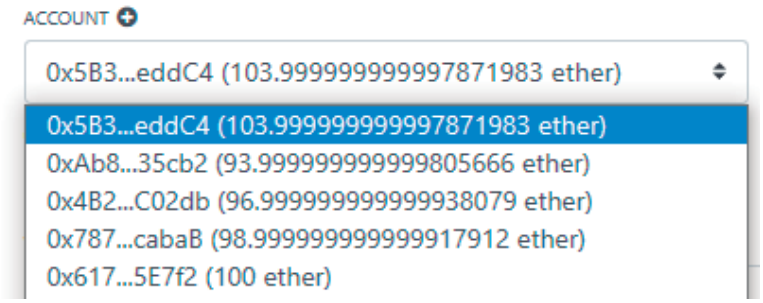
Figura 24 - Realiza o saque do pedido aprovado.



Fonte: Elaborado pelo autor.

Por fim podem ser verificados os novos saldos das contas após um saque ter sido realizado (Figura 25).

Figura 25 - Visualizar as contas após um saque.



Fonte: Elaborado pelo autor.

CONCLUSÃO

O presente trabalho apresentou o desenvolvimento de um contrato inteligente utilizando a tecnologia da *blockchain* criando um sistema de financiamento coletivo capaz de ser executado automaticamente, assim que os termos do contrato forem cumpridos ou quebrados, resultando em um sistema confiável e atrativo para este público-alvo. Além disso, foi possível garantir que os fundos arrecadados fossem seguros e que todas as transações fossem rastreadas ao longo do projeto.

Com o desenvolvimento deste trabalho, fica evidente que a utilização de contratos inteligentes baseados em *blockchain* podem transformar o modo que as transações financeiras são realizadas, fornecendo segurança, transparência e eficiência para os participantes de um sistema de financiamento coletivo. Além disso, abre portas para futuras aplicações e pesquisas no campo das tecnologias descentralizadas, impulsionando avanços e inovações nessa área que se encontra em constante evolução.

Como trabalhos futuros é interessante o desenvolvimento de uma interface intuitiva para o usuário para que o sistema seja mais agradável e proporcione maior facilidade de uso, favorecendo a usabilidade.

REFERÊNCIAS

LEWIS, A. **A gentle introduction to ethereum.** s.l.: 2016. Disponível em https://bitsonblocks.net/2016/10/02/gentleintroduction-ethereum. Acesso em: mar. 2023.

CARDOSO, B. **Contratos inteligentes:** descubra o que são e como funcionam. s.l.: 2018. Disponível em https://brunonc.jusbrasil.com.br/artigos/569694569/contratosinteligentes-descubra-o-que-sao-e--como-funcionam. Acesso em: mar. 2023.

ETHEREUM. **Solidity**. s.l.: 2021. Disponível em https://docs.soliditylang.org/en/latest/index.html. Acesso em: mar. 2023.

ETHEREUM. **What is Ethereum?** s.l.: 2021. Disponível em https://ethereum.org/en/what-is-ethereum. Acesso em: mar. 2023.

ASHARI, F. *et al.* Smart Contract and Blockchain for Crowdfunding Platform. **International Journal of Advanced Trends in Computer Science and Engineering**, v. 9, p. 3036-3041, 2020.

YANO, I. H. *et al.* Modelo de rastreamento bovino via smart contracts com tecnologia blockchain. **Comunicado Técnico**, 130, Embrapa, pp. 2- 21, 2018

GUPTA, M. **Blockchain for Dummies**. IBM Limited Editon: John Wiley & Sons, 2019.

ORCUTT, M. **How secure is blockchain really?** s.l.: 2018. Disponível em https://www.technologyreview.com/2018/04/25/143246/howsecure-is-blockchain-really. Acesso em: mai. 2023.

SZABO, N. Smart contracts: building blocks for digital markets. **Extropy: The Journal of Transhumanist Thought**, v.16, p. 28, 1996.

SIQUEIRA, P. C. D.; DUQUE, J. W. G.. Get together: estudo e implementação de uma aplicação blockchain usando smart contracts. **Revista H-Tec Humanidades e Tecnologias**, Edição Especial EIC 2019, p. 203-219, 2020.

PRESSMAN, R. S. **Engenharia de Software**: Uma Abordagem Profissional. 7 ed. s.l.: McGraw Hill, 2010.

KHAN, S. N, *et al.* Blockchain smart contracts: Applications, challenges, and future trends. **Oeer-to--Peer Netw. Appl**, v.14, pp. 2901-2925, 2021.

KUSHWAHA, S. S., *et al.* Systematic Review of Security Vulnerabilities in Ethereum Blockchain Smart Contract. **IEEE Access**, v.10, p. 6605-6621, 2022.