Reformulação de um sistema de financiamento coletivo em um contrato inteligente na *Blockchain* da rede *Ethereum*

William Lorenzoni Zottele, Ana Paula Canal
Curso de Ciência da Computação
UFN - Universidade Franciscana
Santa Maria - RS
w.zottele@ufn.edu.br, apc@ufn.edu.br

***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 pequenos programas chamados *Smart Contracts*. O objetivo deste trabalho é criar um contrato inteligente de arrecadação de fundos coletivos usando a tecnologia *blockchain*. Os colaboradores tem 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* : Smart Contract; Contrato Inteligente; crowdfunding; Solidity; Remix;**

I. Introdução

O presente projeto propõe a elaboração de um modelo de contrato inteligente que seja utilizado para fins de *crowdfunding*, conhecido como "vaquinhas online", na hora de arrecadar possíveis recursos para 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 tornando uma 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 que permite que os desenvolvedores criem qualquer programa ou aplicação em uma plataforma descentralizada. Podem também utilizar essa rede 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 por inteiro 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 de forma distribuída sem depender de apenas uma parte confiável [1].

A. Justificativa

Nos tempos atuais, pensar e descobrir soluções para os problemas é fundamental. 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 [1].

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 aumenta ainda mais a confiança entre as partes envolvidas, pois não é necessário ter uma terceira pessoa que faça com que ambas as partes envolvidas em um contrato realmente cumpram o que foi designado para cada um.

B. Objetivo Geral

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

II. Referencial Teórico

Nessa seção serão discutidas definições e conceitos da *Blockchain*, especificamente da rede *Ethereum* e como criar um contrato inteligente usando a linguagem de programação Solidity.

A. Blockchain

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 [2].

A *blockchain* constitui-se por estruturas de dados. Assim como dados são logicamente colocados juntos e armazenados. Outras estruturas de dados podem ser armazenadas em linhas, colunas, tabelas, formando as bases de dados, podem ser arquivos de texto, valores separados por vírgula, imagens, listas, e assim por diante [3].

Segundo Mike Orcutt "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" [4, 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 [5].

A Figura 1 mostra um exemplo de *Blockchain*.

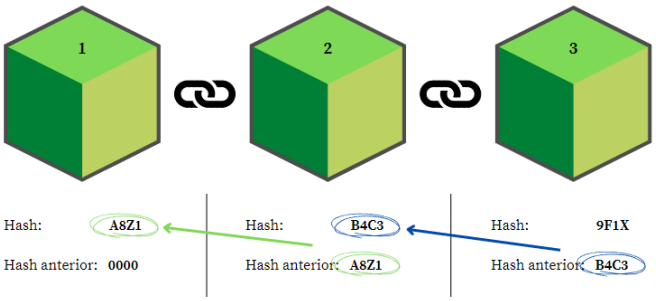

Figura 1. Exemplo de *Blockchain*.

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.

A função *hash* é uma função que utiliza uma fórmula matemática que permite o mapeamento de dados de tamanho variável para pequenos dados de tamanho fixo. Permitindo a identificação de qualquer alterações em um bloco de dados volumoso, avaliando apenas a *hash* gerada [5]. A Figura 2 mostra um exemplo do uso da função SHA365.

Na Figura 2, foi utilizada a fórmula SHA-256 de criptografia [6], por exemplo no texto "Contrato Inteligente1" usando a função *hash* SHA-256, obtemos "0c8680..." como resultado. Já no segundo exemplo, "Contrato Inteligente2", obtemos "e9b2c5...". Nota-se que apenas 1 caracter é trocado entre os 2 exemplos e mesmo assim obtemos uma *hash* completamente diferente.

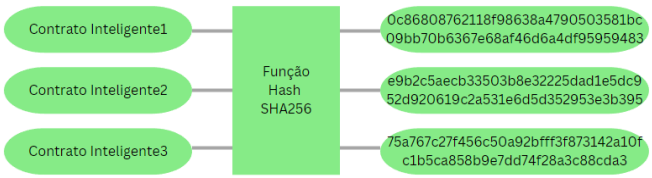

Figura 2. Exemplo de Criptografia.

B. *Contrato Inteligente*

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 [2].

O termo *Smart Contract* tem sido usado ao longo dos anos para descrever uma grande 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" [7, p.28].

De acordo com Bruno 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" [8, p.13].

A Figura 3 ilustra um exemplo de funcionamento de um contrato inteligente.

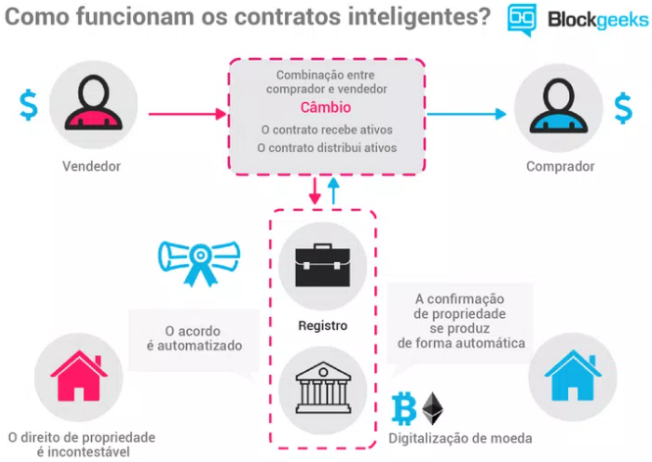

Figura 3. Exemplo de funcionamento de um Contrato Inteligente. [8]

Como observado na Figura 3, 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* [8].

Após o cumprimento dos termos do contrato, o contrato é executado conforme os termos nele contido e verifica a transferência de compromissos automaticamente [8].

C. Ethereum

Ethereum é um software executado em uma rede de computadores que garante que dados e pequenos programas de computador chamados contratos inteligentes sejam replicados e processados em todos os computadores da rede, sem um coordenador central. A visão é criar um computador mundial descentralizado e autossustentável, resistente à censura e imparável [9].

O *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 [10].

D. Solidity

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 foi designada para implementar contratos inteligentes na rede *Ethereum* [11].

Solidity é tipada, suporta herança, bibliotecas e tipos complexos definidos pelo usuário entre outras características. Com Solidity é possível criar contratos para votação, vaquinhas, leilões às cegas, e carteiras multi-assinadas [11].

A Solidity surgiu em 2014 como uma proposta de Gavin Wood, sendo desenvolvida por uma equipe da Ethereum liderada por Christian Reitwiessner [12].

Este trabalho foi desenvolvido no ambiente de trabalho integrado Remix, que é a ferramenta 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. Foi usada para a programação, testes e *deploy* do contrato inteligente.

E. Plataformas de crowdfunding

Plataformas de *crowdfunding* são *websites* que possibilitam a iteração entre fundadores e o público. As promessas financeiras podem ser feitas e coletadas por meio de uma 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. Em retorno, é esperado que as plataformas providenciem segurança e praticidade no uso da mesma.

A maioria das plataformas no geral 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.

F. Trabalhos correlatos

Foram pesquisados trabalhos relacionados à temática de contratos inteligentes e a tecnologia da *blockchain*.

O trabalho realizado por Yano [5] apresentou um contrato inteligente com o intuito de rastrear a localização da cadeira produtiva da carne de bois em fazendas de cria, recria ou engorda por meio das transações que acontecem no próprio contrato. A simulação foi feita no ambiente de desenvolvimento Remix, é a mesma ferramenta utilizada na simulação do contrato inteligente deste projeto.

O artigo apresentado por Kushwaha [13] fala sobre 24 vulnerabilidades de contratos inteligentes com seus métodos de prevenção, detecção, e ferramentas de análises sobre três problemas raízes e 17 subproblemas. Dando exemplos de ataques e ferramentas de detecção com seus sugeridos métodos de prevenção. Também classifica as vulnerabilidades de contratos inteligentes e suas causas.

O trabalho de Ashari [14] faz uma análise de como implementar a tecnologia da *blockchain* e seus contratos inteligentes nos esquemas dominantes do processo de *crowdfunding*. O resultado do estudo indica que contratos inteligentes da *blockchain* podem ser aplicados nos esquemas de *crowdfunding*.

Com base nos trabalhos relacionados citados, é possível destacar alguns fundamentos que apontam a semelhança dos trabalhos. Começando pelo trabalho realizado por Yano [5], que utilizou o ambiente de desenvolvimento Remix para a criação de seu contrato inteligente trazendo conhecimento sobre os passos de desenvolvimento na plataforma que foi utilizada na projeção do contrato inteligente deste trabalho. No trabalho de Kushwaha [13], foi acrescentado o conhecimento sobre as vulnerabilidades de contratos inteligentes e seus métodos de prevenção, que foi importante para que no desenvolvimento do contrato inteligente não ocorressem erros que poderiam deixar o contrato vulnerável. Por fim, o trabalho de Ashari [14], faz uma análise de como implementar a tecnologia da *blockchain* em um esquema de *crowdfunding* utilizando contratos inteligentes, que é o objetivo final deste projeto.

III. Metodologia

Inicialmente o trabalho foi desenvolvido com bases nos estudos bibliográficos sobre os conteúdos de *Blockchain*: Khan [1], Gupta [2], Lewis [3], Orcutt [4]. *Smart Contracts*: Khan [1], Szabo [7], Cardoso [8]. *Ethereum*: Lewis [9], Kushwaha [13] e a linguagem de programação Solidity [11].

A metodologia ágil utilizada para o desenvolvimento do projeto é a metodologia FDD (*Feature-Driven Development*).

Segundo Pressman (2010), existem cinco atividades metodológicas que definem a abordagem FDD, denominadas como processos no FDD (Figura 4), sendo elas: Desenvolver um modelo geral, Construir uma lista de funcionalidades, Planejar por funcionalidades, Projetar por funcionalidades, Desenvolver por funcionalidade [15].

IV. Proposta da criação do contrato inteligente

Na era da Internet, qualquer pessoa pode doar dinheiro com a finalidade de ajudar projetos. Porém, nem tudo é tão

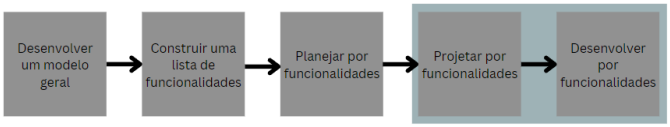

Figura 4. Adaptação do Processo FDD de Pressman (2010).

bom quanto parece. As plataformas de *crowdfunding* ainda apresentam muitas falhas ou brechas na hora de identificar a veracidade do projeto de *crowdfunding* ou da pessoa por trás do mesmo. 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 onde, pessoas de má índole se aproveitam de notícias trágicas que passam na TV, internet ou até mesmo rádio para fundar projetos falsos de *crowdfunding* mesmo sem ter nenhum tipo de ligação com as vítimas de tal acontecimento.

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 da criação 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 mais confiança no projeto.

A. Planejamento do Contrato

Nesta proposta de contrato inteligente é planejado o seguinte fluxo de processo:

• 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 terá que criar um pedido de saque para poder usar o dinheiro arrecadado. 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.

A Figura 5 mostra o Diagrama de Atividades, conforme o processo especificado.

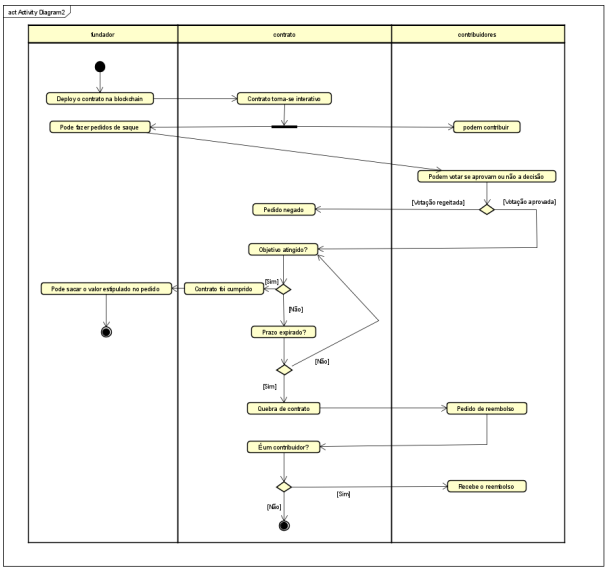

Figura 5. Diagrama de Atividades.

Nas próximas seções, a análise e projeto do contrato são descritas conforme as etapas do FDD.

B. Desenvolver um Modelo Geral

A primeira parte da metodologia FDD é o desenvolvimento de um modelo geral. Este modelo é uma introdução da metodologia que incorpora o projeto.

A Figura 6 ilustra o Diagrama de Domínio, nele estão as classes do domínio do problema.

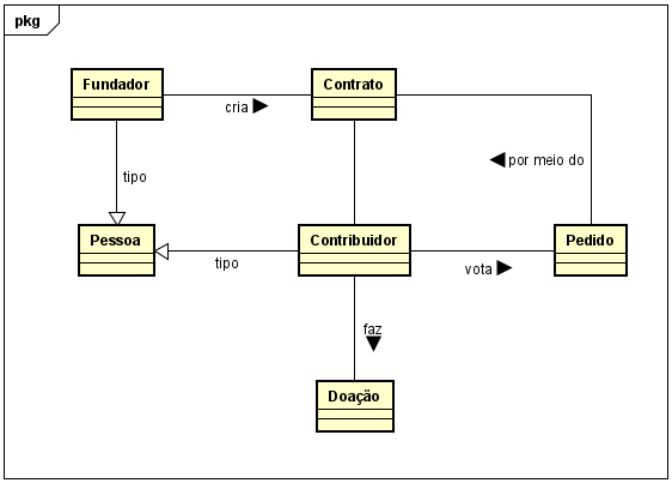

Figura 6. Diagrama de Domínio.

No diagrama, é possível observar as entidades conceituais do domínio e as relações que possuem entre eles. Dando uma visão mais ampla e em conjunto dos componentes presentes no sistema.

Depois do diagrama de domínio, o próximo passo da metodologia é o desenvolvimento de uma lista de funcionalidades do sistema.

C. Construir uma lista de funcionalidades

No segundo processo da metodologia FDD, devem ser listados todos os requisitos funcionais e não funcionais do sistema e ordenar as funcionalidades por prioridade em que serão desenvolvidas. Assim como listado na Figura 7 e na Figura 8.

RF 01 – O contrato deverá possibilitar receber doações de contribuidores.	**Complexidade:** Média	**Relevância:** Essencial
RF 02 – O contrato deverá possibilitar o fundador de fazer pedidos de saques.	**Complexidade:** Média	**Relevância:** Essencial
RF 03 – O contrato deverá possibilitar que os contribuidores possam votar em pedidos de saques.	**Complexidade:** Baixa	**Relevância:** Essencial
RF 04 – O contrato deverá permitir que o fundador consulte o saldo existente.	**Complexidade:** Baixa	**Relevância:** Essencial
RF 05 – O contrato deverá possibilitar que o fundador consulte quantos contribuidores existem.	**Complexidade:** Baixa	**Relevância:** Essencial
RF 06 – O contrato deverá possibilitar que o fundador consulte se existe um pedido de saque pendente.	**Complexidade:** Baixa	**Relevância:** Essencial
RF 07 – O contrato deverá possibilitar que o fundador consulte o prazo final do contrato.	**Complexidade:** Baixa	**Relevância:** Essencial
RF 08 – O contrato deverá possibilitar que o fundador saque a quantia informada no pedido de saque, caso o pedido seja aprovado pelos contribuidores.	**Complexidade:** Média	**Relevância:** Essencial
RF 09 – O contrato deverá possibilitar os contribuidores de fazer um pedido de reembolso caso ocorra quebra de contrato.	**Complexidade:** Média	**Relevância:** Essencial
RF 10 – O contrato deverá possibilitar os contribuidores de consultar o saldo total do contrato.	**Complexidade:** Baixa	**Relevância:** Essencial
RF 11 – O contrato deverá possibilitar os contribuidores de consultar se o contrato atingiu o objetivo final.	**Complexidade:** Baixa	**Relevância:** Essencial

Figura 7. Requisitos 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.

Figura 8. Requisitos não funcionais.

D. Planejar por Funcionalidades

O objetivo desta etapa do processo é produzir o plano de desenvolvimento do contrato, listando as funcionalidades que serão implementadas na sequencia de desenvolvimento. De acordo com a lista de funcionalidades apresentadas no segundo processo da metodologia FDD, o terceiro processo desta metodologia é feito o planejamento da mesma. A Figura 9 ilustra o planejamento das funcionalidades, mostrando a sua complexidade, relevância e a estimativa de tempo (em dias) para o desenvolvimento de cada uma delas.

Sequência de Desenvolvimento	Funcionalidade	Complexidade	Relevância	Tempo (Dias)
1	RF01	Média	Essencial	10
2	RF02	Média	Essencial	5
3	RF03	Baixa	Essencial	3
4	RF04	Baixa	Essencial	3
5	RF05	Baixa	Essencial	3
6	RF06	Baixa	Essencial	3
7	RF07	Baixa	Essencial	3
8	RF08	Média	Essencial	5
9	RF09	Média	Essencial	5
10	RF10	Baixa	Essencial	3
11	RF11	Baixa	Essencial	3

Figura 9. Planejamento das Funcionalidades.

E. Projetar e Desenvolver por Funcionalidades

A quarta e quinta etapa que fazem parte desta metodologia, projetar e desenvolver por funcionalidades, são iterativas e são descritas no nesta subseção.

O contrato foi desenvolvido no ambiente de desenvolvimento Remix, e a linguagem de programação utilizada foi a linguagem Solidity para a programação de contratos inteligentes na rede *Ethereum*.

O desenvolvimento do contrato inicia com a versão do compilador e na terceira linha o nome que foi dado para o contrato, como mostrado na Figura 10. Nas linhas seguintes ocorrem as declarações 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
```

Figura 10. Declaração de atributos do contrato.

Na linha 18 (Figura 11) ocorre a criação de um *modifier* chamado *onlyOwner*, esse *modifier* é usado para modificar o comportamento de um método onde se presente torna a função executável apenas pelo *owner* do contrato.

```
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      }
```

Figura 11. Criação do *modifier onlyOwner*.

O método *constructor* na linha 24 da Figura 12 é especialmente usado para inicializar os estados dos atributos

do contrato, este é o método responsável pelo *deploy* do contrato.

```
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    }
```

Figura 12. Iniciando os estados dos atributos usando o método *constructor*.

As doações do contrato ocorrem por meio do método *donate* (Figura 13) que começa na linha 31, esse método é de tipo *payable* que significa que ela é um método que recebe Ether como parâmetro. As checagens são feitas por meio dos métodos *require* que são métodos responsáveis por checar as entradas e as condições antes da execução, por exemplo, se a condição for falsa o método *require* interrompe a execução imediatamente.

```
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    }
```

Figura 13. Método responsável pelas transferências de Ethers para o contrato.

As aprovações de saques são feitas pelo método *approveWithdrawalRequest* (Figura 14) na Linha 93, nela estão presentes os cálculos dos pesos de votação do usuário levando em consideração a quantia doada e o total de fundos que o contrato possui. O método é executado e o calculo é feito após o usuário clicar no botão *approveWithdrawalRequest* somando o peso da votação no atributo *numOfApprovals* e setando o estado de aprovação do usuário para *true*.

```
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
```

Figura 14. Método responsável pelo numero de aprovação de um pedido de saque e pelo calculo do peso de votação de um usuário.

Após um pedido de saque ser aprovado o *owner* do contrato pode interagir com o método *transferWithdrawalRequest* (Linha 111) por meio do botão *transferWithdrawalRequest*, método responsável por checar se o pedido foi aprovado por mais de 50% dos *donors* do contrato e que se verdadeiro seta o *request* como *true* e faz a transferência dos fundos especificados no endereço do pedido de saque. A Figura 15 mostra o método *transferWithdrawalRequest*.

```
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   }
```

Figura 15. Método responsável por checar se um pedido de saque foi aprovado e por fazer a transferência para o *owner*.

Em caso de não cumprimento dos termos do contrato e o prazo para doação expirar, temos o método *refound* (Linha 77), responsável pelo reembolso dos usuários que doaram para o contrato. Esse pode ser chamado pelo usuário em caso do prazo para a arrecadação de fundos ter expirado, assim o doador consegue seu dinheiro de volta (Figura 16).

```
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    }
```

Figura 16. Método responsável pelo reembolso dos usuários em caso de não cumprimento dos termos do contrato.

F. Simulação de um financiamento coletivo utilizando um Smart Contract

A simulação foi realizada pelo ambiente de desenvolvimento Remix, que é a ferramenta responsável por fazer a interação com a *blockchain* da rede *Ethereum*, nela foram feitas todas as simulações e testes do *Smart Contract*.

No exemplo da Figura 17, o *Smart Contract Crowdfunding* foi dividido em seis atividades para demonstrar a sua estrutura analítica.

Figura 17. Estrutura analítica do contrato *Crowdfunding*.

Inicialmente, é selecionado o compilador do contrato para que possa checar se não existe algum erro que possa impedir a execução do contrato (Figura 18).

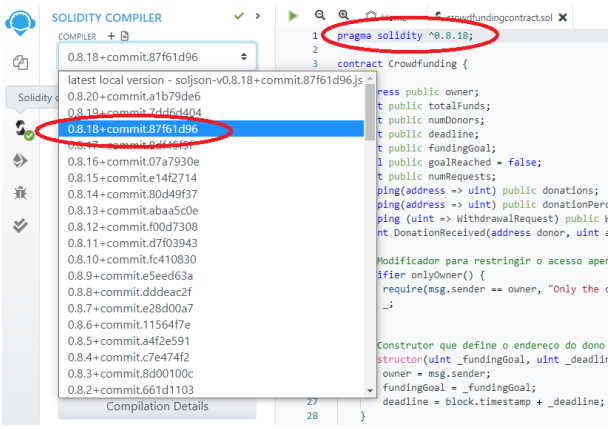

Figura 18. Ambiente de desenvolvimento Remix, selecionando o compilador.

Assim que o contrato for compilado, a simulação pode ser iniciada clicando-se na aba *"Deploy and run transactions"* e após isto devemos selecionar 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 19. O primeiro endereço será 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. Como ainda não há nenhuma transação no contrato o contador de transações está zerado.

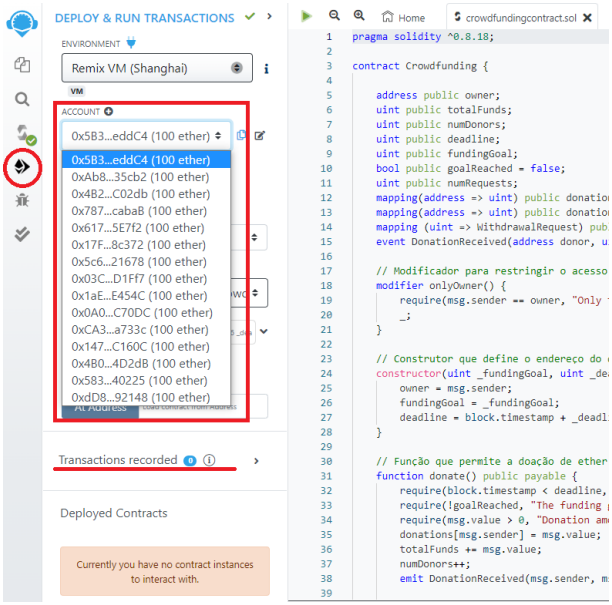

Figura 19. Ambiente de desenvolvimento Remix, selecionando contas.

A primeira iteração do contrato é feita pelo endereço "0x5B3...eddC4" apertando no botão de Deploy que faz a chamada do *constructor* do contrato tornando o endereço *owner* do mesmo. Os atributos passados são o objetivo a ser atingido em Wei e o prazo do contrato em segundos. Nota-se que o sucesso da transação é verificado pelo sinal de checagem verde na parte inferior da imagem da Figura 20 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 de *loops* infinitos acidentais ou hostis.

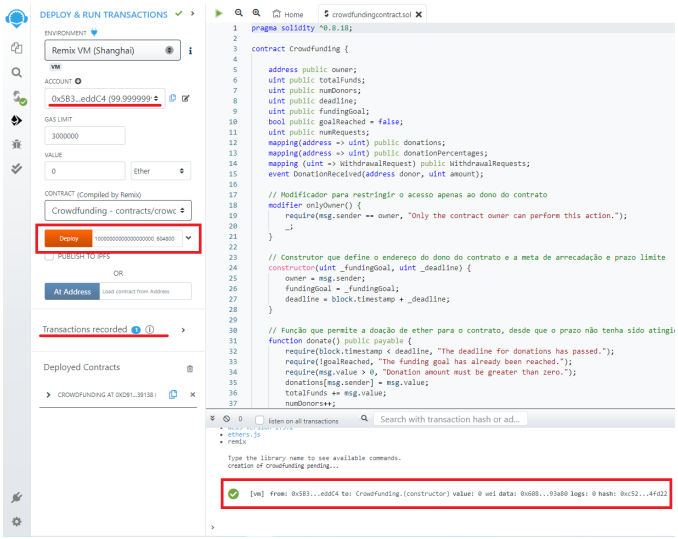

Figura 20. Deploy do contrato.

É possível observar os detalhes de uma transação clicando-se na transação desejada, podendo observar todas as informações relacionadas a transação. Como por exemplo na Figura 21, status, *hash* da transação, endereço remetente, operação executada e informações sobre custos da transação.

Figura 21. Detalhes do deploy.

Após a chamada do *constructor* (*Deploy* do contrato), é possível fazer a consulta dos atributos e executar os métodos presentes no contrato. A Figura 22 mostra os atributos em azul, os métodos em laranja e o método *donate* em vermelho. Essa variedade nas cores ocorre para identificar o tipo da variável, atributos constantes não criam transações então clicando nelas não causará 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 uma método *payable* que recebe Ether em sua execução e por isto sua cor é vermelha.

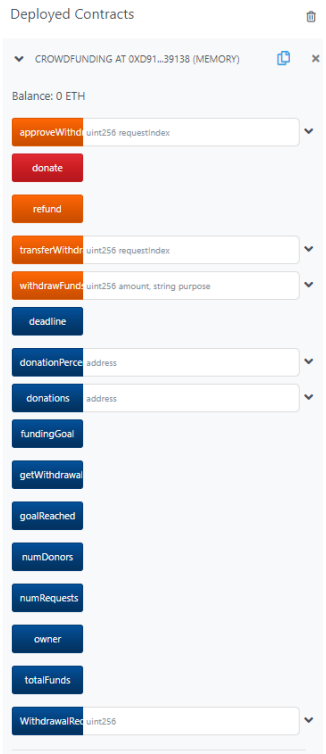

Figura 22. Calls e Funções.

Com o contrato acessível, é possível fazer as doações e transações dos fundos do contrato. Nesta parte foram feitas as 3 operações de doações. Primeiro foi selecionado o endereço do primeiro doador, "0xAb8...35cb2" e a quantia doada é de 6 Ether. Percebe-se que o saldo do contrato é aumentado para 6 ETH (Figura 23) e que o número do contador de transações subiu de 1 para 2 e a confirmação da execução é mostrada na tela.

As outras 2 doações são feitas pelos endereços "0x4B2...C02db" de 3 Ether e "0x787...cabaB" de 1 Ether. Fazendo com que a soma das doações resulte em 10 Ether,

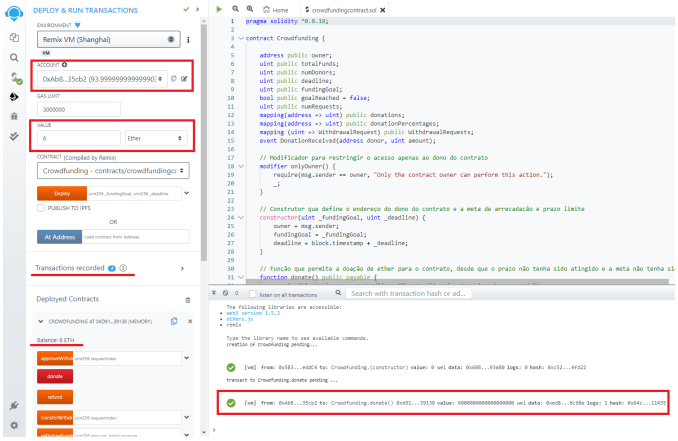

Figura 23. Fazendo uma doação.

cumprindo a meta do contrato. Após finalizar as doações podemos checar o conteúdo das variáveis para saber se a meta foi atingida, quantos doadores o contrato possui, qual o endereço do *owner*, quanto um doador específico doou, entre outras opções... como na Figura 24.

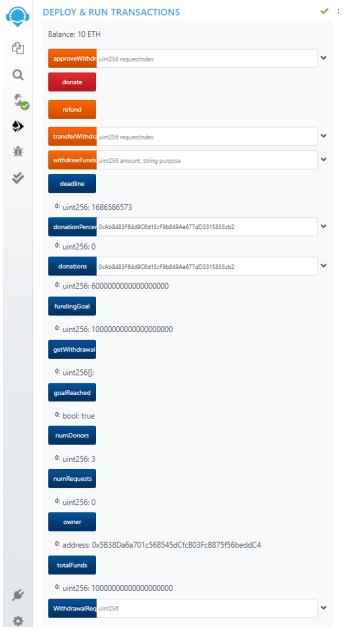

Figura 24. Checando o conteúdo das variáveis.

Com o objetivo do contrato cumprido, o *owner* pode fazer pedidos de saque que se aprovados podem ser sacados diretamente para o seu endereço. O pedido de saque deve conter o valor em Wei que deseja ser retirado do montante do contrato e o propósito do saque. Apenas o *owner* conseguirá realizar um pedido de saque e o endereço do *owner* precisa estar selecionado, como na Figura 25.

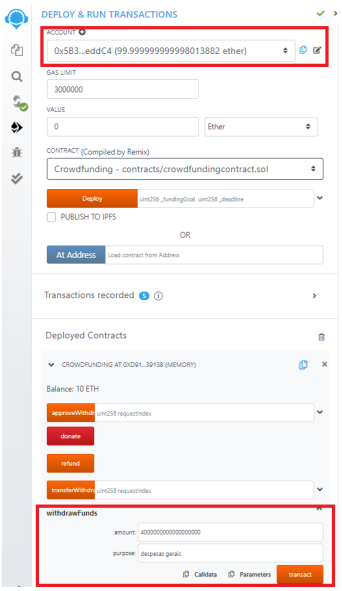

Figura 25. Fazendo um pedido de saque.

Após um pedido de saque ser criado, é possível consultar quantos pedidos de saques (*numRequests*) e quais são os índices desses pedidos pelo botão *getWithdrawalRequests*. Também é possível consultar um pedido de saque específico colocando o número do índice do pedido e clicando no botão *WithdrawalRequests*, que faz a chamada do método *WithdrawalRequests* onde armazena todos os pedidos de saques criados no contrato. Neste método podemos observar o valor de saque especificado no pedido, o propósito do saque, se o saque já foi executado ou não, e a porcentagem do número de aprovações do pedido. Lembrando que o cálculo da porcentagem é baseado no valor de que cada endereço doou, podendo assim uma pessoa ter um peso de votação maior do que outras (Figura 26).

Figura 26. Consultando pedido de saque.

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 colocar o índice do pedido do saque no botão *approveWithdrawalRequest* e pressioná-lo. O contrato faz todas as checagens se o endereço é valido para aprovar um pedido de saque, faz também o cálculo do peso de voto que o endereço possui e computa o voto em uma porcentagem do valor total do contrato (Figura 27).

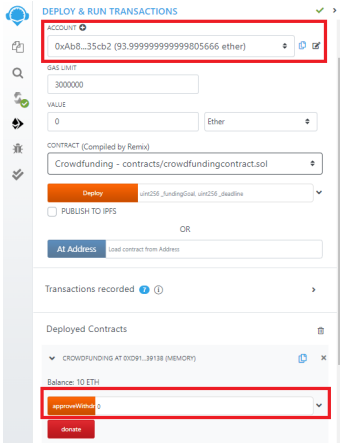

Figura 27. Aprovar um pedido de saque.

É possível consultar o pedido de saque pra checar se o pedido de saque de índice 0 para checar se o número de aprovações alcançou mais de 50% de aprovação. Na Figura 28 há um exemplo com aprovação de 60%.

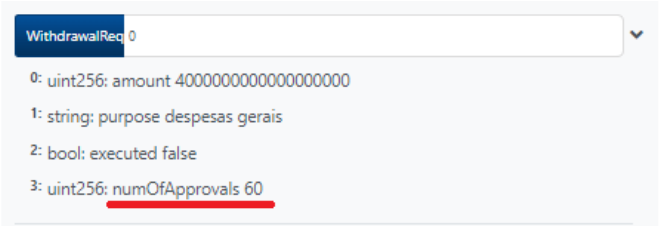

Figura 28. Consultando pedido de saque depois do voto.

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 deste pedido. A requisição do saque deve ser feita pelo *owner* do contrato que deve colocar o índice do pedido aprovado no método *transferWithdrawalRequest*. O contrato faz a checagem e se todos os requisitos forem aceitos a quantia especificada no pedido é transferida para o endereço do *owner*. 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, como na Figura 29.

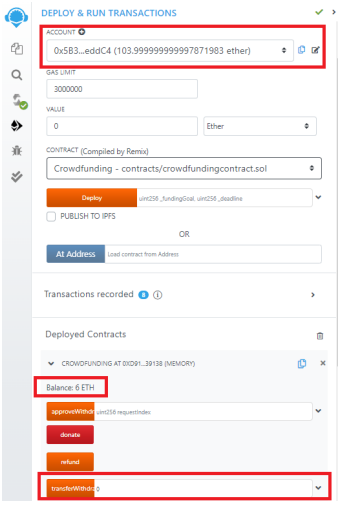

Figura 29. Realiza o saque do pedido aprovado.

Por fim podem ser verificados os novos saldos das contas após um saque ter sido realizado (Figura 30).

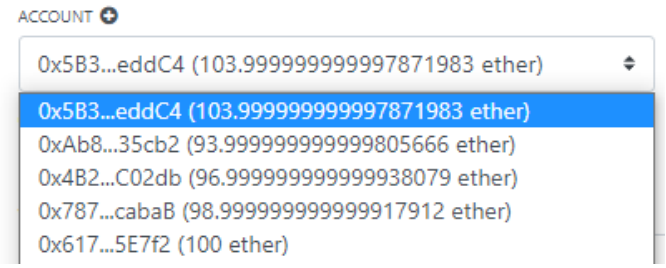

Figura 30. Visualiza as contas após um saque.

V. Conclusão e Trabalhos Futuros

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 mais intuitiva para o usuário para que o sistema seja mais agradável e tenha mais facilidade de ser utilizado, tornando-o um sistema de maior usabilidade.

Referências

[1] S.N. Khan et al. “Blockchain smart contracts: Applications, challenges, and future trends.” Em: *Peer-to-Peer Netw. Appl.* 14 (2021), pp. 2901–2925.

[2] M. Gupta. *Blockchain for Dummies*. IBM, 2019.

[3] A. Lewis. “A gentle introduction to blockchain technology”. Em: (2015). Disponível em https://bitsonblocks.net/2015/09/09/gentle-introduction-blockchain-technology.

[4] M. Orcutt. “How secure is blockchain really?” Em: (2018). Disponível em https://www.technologyreview.com/2018/04/25/143246/how-secure-is-blockchain-really.

[5] I. H. Yano et al. “Modelo de rastreamento bovino via smart contracts com tecnologia blockchain”. Em: *COMUNICADO TÉCNICO 130 Embrapa* (2018), pp. 2–21.

[6] Tim van Werkhoven Wouter Penard. “On the secure hash algorithm family”. Em: *Cryptography in context* (2008), pp. 1–18.

[7] Nick SZABO. “Smart contracts: building blocks for digital markets.” Em: *EXTROPY: The Journal of Transhumanist Thought.* 16 (1996), p. 28.

[8] B. Cardoso. “Contratos inteligentes: descubra o que são e como funcionam”. Em: (2018). Disponível em https://brunonc.jusbrasil.com.br/artigos/569694569/contratos-inteligentes-descubra-o-que-sao-e-como-funcionam.

[9] A. Lewis. “A gentle introduction to ethereum”. Em: (2016). Disponível em https://bitsonblocks.net/2016/10/02/gentle-introduction-ethereum.

[10] Ethereum. “What is Ethereum?” Em: (2021). Disponível em https://ethereum.org/en/what-is-ethereum.

[11] Ethereum. “Solidity”. Em: (2021). Disponível em https://docs.soliditylang.org/en/latest/index.html.

[12] P. C. D. Siqueira e J. W. G. Duque. “Get together: estudo e implementação de uma aplicação blockchain usando smart contracts”. Em: *Revista H-Tec Humanidades e Tecnologias - Edição Especial EIC 2019* (2020), pp. 203–219.

[13] S. S. Kushwaha et al. “Systematic Review of Security Vulnerabilities in Ethereum Blockchain Smart Contract”. Em: *IEEE Access* 10 (2022), pp. 6605–6621.

[14] F. Ashari. ;et al. “Smart Contract and Blockchain for Crowdfunding Platform.” Em: *International Journal of Advanced Trends in Computer Science and Engineering* 9 (2020), pp. 3036–3041.

[15] R. S. Pressman. *Engenharia de Software: Uma Abordagem Profissional*. 7 ed., McGraw Hill, 2010.