

Universidade Paulista - UNIP

Matheus Pereira Barbosa

***Smart Contracts* para transações fiscais**

Limeira

2019

Universidade Paulista - UNIP

Matheus Pereira Barbosa

***Smart Contracts* para transações fiscais**

Trabalho de conclusão de curso apresentado à banca examinadora da Faculdade UNIP, como requisito parcial à obtenção do Bacharelado em ciência da computação sob a orientação dos professores Me. Marcos Gialdi, Me. Antonio Mateus Locci.

Limeira

2019

Matheus Pereira Barbosa

***Smart Contracts* para transações fiscais**

Trabalho de conclusão de curso apresentado à banca examinadora da Faculdade UNIP, como requisito parcial à obtenção do Bacharelado em ciência da Computação sob a orientação do professor Me. Marcos Gialdi, Me. Antonio Mateus Locci.

Aprovada em de de 2019.

BANCA EXAMINADORA

DEDICATÓRIA

Dedico este trabalho aos meus amigos, familiares e professores que me apoiaram durante toda a trajetória do curso, tornando realidade os objetivos pessoais que foram traçados desde o início da graduação.

RESUMO

Este trabalho trata sobre a falta de informações e transparência em transações fiscais focando principalmente nas licitações da modalidade Pregão Eletrônico, que atualmente são as mais utilizadas devido ao fácil acesso via internet, onde empresas de todos portes e lugares do país podem participar e realizar lances. O *blockchain* foi apresentado como uma possível solução, já que esta tecnologia disponibiliza recursos que auxiliam a prevenção de fraudes através da criação de uma aplicação que simula o Pregão Eletrônico, na qual um contrato inteligente atua na autenticação, validação e monitoramento das transações, tornando o sistema mais seguro contra *bots* e possíveis fraudes.

Palavra-Chave: *Blockchain; Smart Contracts; Pregão Eletrônico; Transação.*

ABSTRACT

This paper deals with the lack of information and transparency of taxes, especially the licenses for use of the Electronic Auction, currently the most used due to the easy access to the Internet, where companies of all sizes and locations in the country can participate and perform bids. Blockchain has been presented as a possible solution, as this technology provides features that help prevent fraud by creating an application that simulates the Electronic Auction, which smart contract acts on authentication, validation and monitoring of tests, using the most secure system. against bots and possible scams.

Key Words: *Blockchain; Smart contracts; Electronic trading floor; Transaction.*

LISTA DE FIGURAS

Figura 01 – Bases de dados utilizadas na criação de <i>softwares</i> .	13
Figura 02 – Demonstração de chaves públicas.	15
Figura 03 - Representação de <i>hash</i> .	17
Figura 04 - Alteração do <i>hash</i> .	18
Figura 05 - Blocos vinculados.	19
Figura 06 - Exemplificação de <i>Smart Contract</i> .	20
Figura 07 - Exemplificação de código <i>Smart Contract</i> .	22
Figura 08 – Caso de Uso.	24
Figura 09 - Parte inicial do contrato.	25
Figura 10 - Parte inicial do contrato.	27
Figura 11 - Configuração do compilador.	28
Figura 12 - <i>Deploy</i> do contrato.	29
Figura 13 - Informações do órgão.	30
Figura 14 - Inserção de registro.	30
Figura 15 - Empresa vencedora.	31
Figura 16 - Teste de código.	32
Figura 17 - Compilador.	33
Figura 18 - Rotina de testes.	33
Figura 19 - Interação com o contrato.	34
Figura 20 - Demonstração da aplicação.	35
Figura 21 - Configuração da biblioteca <i>web3</i> .	36
Figura 22 - Informações do contrato.	37
Figura 24 - API encapsuladora de documentos.	37
Figura 25 - Interface da aplicação.	38
Figura 26 - Envio de proposta do Pregão Eletrônico.	39
Figura 27 - Elegendo vencedor.	39

Figura 28 - Vencedor do Pregão Eletrônico.	40
Figura 29 - Painel de exibição de transações.	41

SUMÁRIO

INTRODUÇÃO	13
1.1 Objetivo	14
1.2 Justificativa	14
1.3 Metodologia	14
2. <i>Blockchain</i>	14
2.1 Chave pública	16
2.2 <i>Hashing</i>	17
2.2.1 MD5	17
2.2.2 SHA-1	18
2.2.3 SHA-256	18
2.2.4 Keccak-256	19
2.2.5 Estrutura da <i>hash</i>	19
2.3 <i>Smart Contracts</i>	20
2.4 Pregão Eletrônico	23
2.4.1 Processos da licitação	24
3. Desenvolvimento da aplicação	24
3.1 IDE para desenvolvimento do código	26
3.1.1 Compilando o código	29
3.1.2 Processo de implantação	29
3.1.3 Simulando Pregão Eletrônico	30
3.2 Testes de desenvolvimento	32
CONCLUSÃO	42

INTRODUÇÃO

A insegurança política está presente na rotina dos Brasileiros, pois há dúvidas sobre o destino dos impostos pagos pela população. Segundo dados governamentais atualmente os maiores investimentos e gastos estão relacionados com o Ministério da Educação, Ministério da Defesa, República Federativa do Brasil e Ministério da Saúde, mas como é possível ter certeza que o destino final se concentra nessas áreas? (TRANSPARÊNCIA, 2018).

Ao debater o tema, percebe-se que há a necessidade de buscar uma solução sobre a falta de transparência referente às informações de como é aplicado o dinheiro público. Atualmente o governo brasileiro disponibiliza um portal onde é possível consultar através de painéis as áreas de investimento público. (TRANSPARÊNCIA, 2018).

O portal citado é o Portal da Transparência, nele pode se pesquisar sobre orçamentos aprovados, licitações, contratos, convênios, entre outras informações referentes a servidores públicos e despesas administrativas. Apesar de uma grande aplicação, será que ele exibe todas as informações? (TRANSPARÊNCIA, 2018).

A Plataforma disponibiliza dados em relação às licitações que ocorreram no ano de 2019, até o momento foram contratadas aproximadamente 82.600 licitações, com investimento de cerca de R\$ 23,77 bilhões de reais somando todas as modalidades, sendo elas Dispensa de Licitação, Pregão - Registro de Preço, Pregão e Inexigibilidade de Licitação. (TRANSPARÊNCIA, 2018).

A modalidade abordada será o Pregão Eletrônico, na qual busca igualdade no procedimento licitatório entre empresas. Essa categoria permitiu que o leque de empreendedores candidatos aumentasse e que empresas de pequeno porte e microempreendedores pudessem participar das licitações. (RAP, 2013).

Atualmente o valor empregado pelo Pregão é de aproximadamente 22,19% do total de licitações contratadas, obtendo o valor a cerca de R\$ 5,27 bilhões de reais distribuídos entre os ministérios de atuação. (TRANSPARÊNCIA, 2018).

Mesmo sendo uma categoria que busca transparência em seus processos há a ocorrência de fraudes na mesma, segundo Santos e Paixão (2017), devido o Pregão ser uma das modalidades mais utilizadas nos últimos tempos, houve um aumento significativo de fraudes.

Em contrapartida a ocorrência de fraudes há uma tecnologia que pode ser empregada para minimizar tal problemática, onde busca autenticar transações de uma forma transparente, funcional e segura, minimizando os gastos com sua aplicação e que tem como objetivo monitorar transações fiscais em tempo real, chamada *blockchain*. (BLOCKGEEKS, 2016).

O estudo do método a ser implementado busca inovar o processo de licitações eletrônicas através de uma aplicação que irá simular transações feitas através do Pregão Eletrônico, com o objetivo de exibir e assegurar que o dinheiro está sendo utilizado para um bem público. (COURSERA, 2019) O *blockchain* contém aplicações de *Smart Contracts*, ou seja, contratos inteligentes, estes contratos guardam informações e geram *logs* sobre a alteração dos dados que estão sendo manipulados. (BLOCKGEEKS, 2016).

1.1 Objetivo

O objetivo deste trabalho é desenvolver uma aplicação com o intuito de monitorar transações financeiras públicas através de *Smart Contracts* e exibir informações concretas para que os cidadãos possam acompanhar de maneira transparente a forma como o dinheiro público é aplicado.

1.2 Justificativa

Devido à falta de informação pública sobre os impostos, há a necessidade da criação de um projeto envolvendo os meios governamentais para que os dados sejam monitorados e autenticados, resultando na divulgação transparente à população brasileira.

1.3 Metodologia

A fim de alcançar o objetivo proposto, uma aplicação *blockchain* baseada na metodologia *Smart Contract* foi desenvolvida através das linguagens de programação *Solidity* e *React*, no qual foi desenvolvido um

software que simula as operações realizadas pelo Pregão Eletrônico, sendo capaz de monitorar transações em tempo real e o momento em que estão sendo registradas na rede de testes *Rinkeby*.

Os cursos introduziram o conhecimento de *Smart Contracts* e técnicas para desenvolvimento dos mesmos, permitindo a utilização desses conceitos para a realização de testes e criação de protótipos *Solidity* que serviram de base para o projeto final.

2. Blockchain

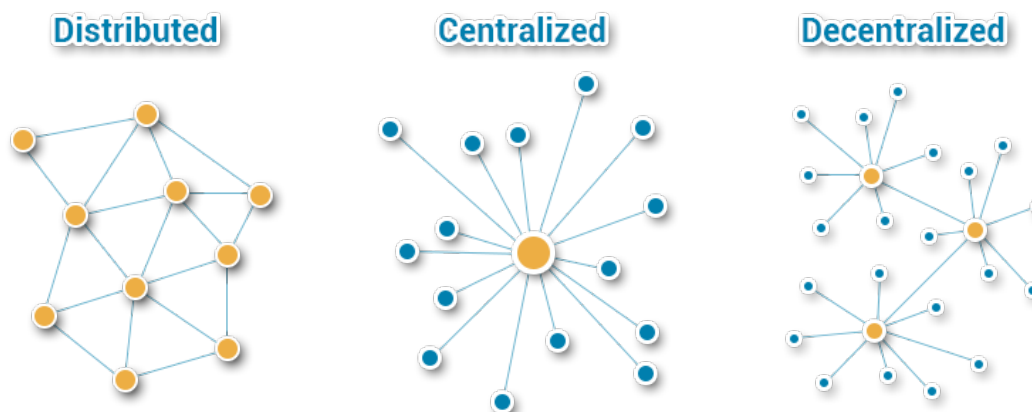
Esta tecnologia busca autenticar transações de uma forma transparente, sem necessariamente a presença de um sistema de intermédio, sendo totalmente funcional, segura e minimizando os custos envolvendo operações. (TAPSCOTT; TAPSCOTT, 2017). Tudo isso é possível graças ao *peer-to-peer*, ou seja, como é exibido em uma simples tradução, pessoa para pessoa ou ponto a ponto. (BLOCKGEEKS, 2016). Com base nessa metodologia foram criadas aplicações de sucesso, como o *Bitcoin* e *Ethereum*. (BLOCKGEEKS, 2016).

Segundo a Christidis e Devetsikiotis (2016), O *blockchain* pode ser entendido como registros que são alocados em blocos, contendo data e hora. Todos os blocos da cadeia de registro possuem uma identificação própria em forma de *Hash*, permitindo que seja possível acessar o bloco anterior e o bloco posterior. Isso impossibilita que haja alterações ou tentativas de fraude em uma cadeia de blocos que estão realizando transações financeiras e contratos inteligentes.

A criação de uma aplicação *blockchain* possui uma estrutura diferente dos sistemas comumente utilizados. Atualmente há inúmeras formas de criar *softwares*, como por exemplo, rede centralizada focada em apenas um nó, rede descentralizada que possui nós principais conectados entre si, e pôr fim a rede distribuída onde todos os nós estão conectados, abrangendo total

conhecimento de informações. A rede distribuída é responsável por aplicar na prática o *peer-to-peer*. (MODIAX, 2018).

Figura 01 – Bases de dados utilizadas na criação de *softwares*.



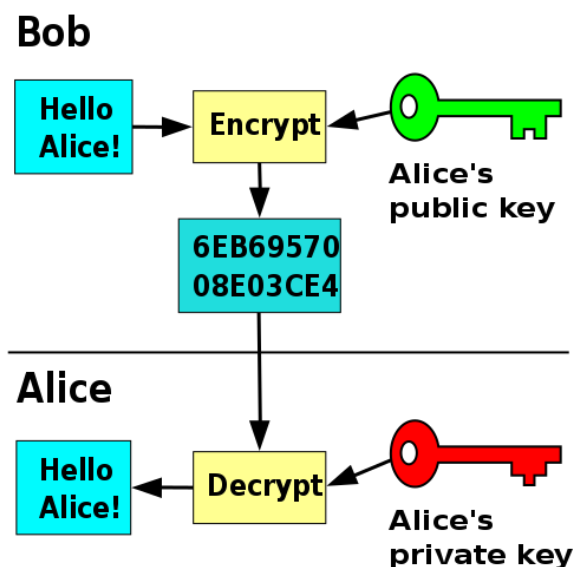
Fonte: Modiax – Qual a diferença entre as bases de dados centralizadas, descentralizadas e distribuídas?

2.1 Chave pública

Para garantir integridade em seus dados, aplica-se o conceito de chave pública também chamado de criptografia assimétrica. Este recurso se dá através de duas chaves, uma chave dita pública que é utilizada para criptografar os dados, e uma chave privada para descriptografar, sendo ambas matematicamente relacionadas. (GLOBSIGN, 2017).

Devido a chave pública ser utilizada apenas para a criptografia, não há problemas em ela ser exposta, como por exemplo os portais que exibem blocos do *Bitcoin* e *Ethereum*. Mesmo sendo matematicamente relacionadas, não é possível desvendar a chave privada por meio da pública, já que o único capaz de descriptografar a chave pública é a pessoa que possui a chave privada. Atualmente, as chaves privadas são armazenadas em *softwares* para autenticação, como certificados digitais. (SEARCHSECURITY, 2015).

Figura 02 – Demonstração de chaves públicas.



Fonte: Medium – *Blockchain (Public key and Private Key)*.

A figura 02 representa de forma ilustrativa como ocorre o processo de encriptação e decriptação de uma mensagem, na qual Bob com a sua chave pública encripta a palavra “Hello” e envia à Alice, transformando a mesma em um *hash*. Como visto anteriormente as chaves são matematicamente ligadas, sendo possível que Alice possa decriptar e descobrir a palavra que Bob enviou.

Além da chave pública ser uma ótima ferramenta para transacionar dados de forma segura, ela também contém inúmeros benefícios, tais como aplicar assinaturas digitais a documentos, assegurando assim que estes realmente foram autenticados pelo órgão expedidor, pois ele é o único proprietário da chave privada utilizada. Ao aplicar a assinatura, ocorre o processo de verificação, que inspeciona todo o documento e impede que mudanças sejam feitas após o documento estar assinado. (GLOBALSIGN, 2017). (SEARCHSECURITY, 2015).

2.2 Hashing

O objetivo final de um *hashing* é proteger a comunicação entre o envio de uma mensagem até o destino final impossibilitando possíveis alterações. O *hash* é gerado por meio de fórmulas, tais quais utilizam funções matemáticas

para transformar o conteúdo da mensagem em um valor. Atualmente há inúmeros métodos para transformar mensagens em *hash*, como por exemplo, MD5, SHA 1, SHA-256, Keccak-256. Para melhor esclarecimento abordaremos brevemente a seguir o funcionamento de cada uma das criptografias citada anteriormente. (BLOCKGEEKS, 2017)

2.2.1 MD5

A primeira a ser abordada é a MD5, metodologia criada pela empresa *RSA Data Security, Inc.*, cujo nome é *Message-Digest algorithm 5*. Essa criptografia é utilizada em um modelo unidirecional através de uma *hash* de 128 bits, uma de suas funcionalidades é comparar duas *hashes* para que uma aprovação de *login* seja feita. Entretanto esta tecnologia não é indicada para programas que necessitam de um alto nível de segurança. (LARA, 2008).

2.2.2 SHA-1

A segunda criptografia citada é a SHA-1, denominada como *Secure Hash Algorithm*, existem várias derivações de SHA, mas a que está sendo abordada é implementada em assinaturas digitais, na qual possui um padrão de implementação pela NIST (*National Institute of Standards and Technology*). (LARA, 2008). Tal criptografia contém uma metodologia de transformação a partir do momento de recebimento de uma mensagem, segundo Figueiredo (2008), “Transforma a mensagem em uma *hash* de 160 bits”.

2.2.3 SHA-256

A tecnologia SHA-256 é muito famosa por ser implementada na grande rede de moedas digitais *Bitcoin*, onde as transações são recebidas e realizadas por meio de um *hash* que possui o tamanho de 256 bits. (BLOCKGEEKS, 2017).

Figura 03 - Representação de *hash*.

INPUT	HASH
Hi	3639EFCD08ABB273B1619E82E78C29A7DF02C1051B1820E99FC395DCAA3326B8
Welcome to blockgeeks. Glad to have you here.	53A53FC9E2A03F9B6E66D84BA701574CD9CF5F01FB498C41731881BCDC68A7C8

Fonte: *Blockgeeks - Smart Contracts*.

Através da figura acima, pode ser observado que não importa o tamanho da mensagem, o *hash* sempre terá comprimento fixo. Devido às propriedades e funções que esta metodologia criptográfica possui, ela é capaz de se auto proteger, uma dessas funções é representada pelo “Efeito Avalanche”, no qual mesmo fazendo pequenas alterações na sua entrada, o *hash* é renovado, como no exemplo a seguir. (BLOCKGEEKS, 2016).

Figura 04 - Alteração do *hash*.

INPUT	HASH
This is a test	C7BE1ED902FB8DD4D48997C6452F5D7E509FBCDBE2808B16BCF4EDCE4C07D14E
this is a test	2E99758548972A8E8822AD47FA1017FF72F06F3FF6A016851F45C398732BC50C

Fonte: *Blockgeeks - Smart Contracts*.

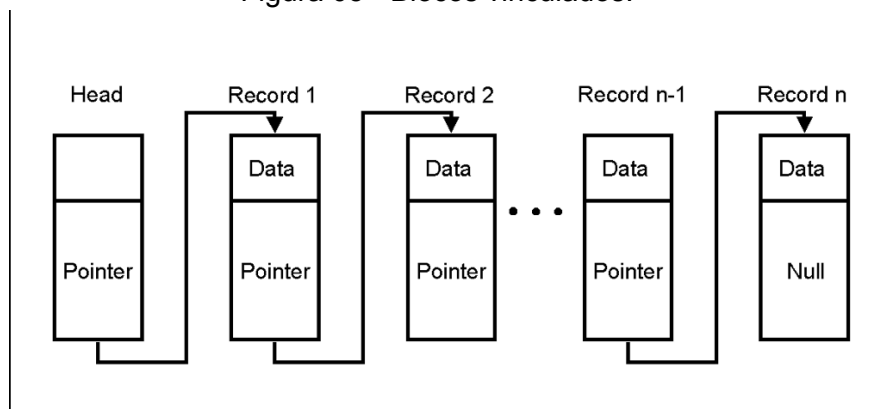
2.2.4 Keccak-256

A tecnologia Keccak trabalha com um *hash* de 256 bits, é utilizado por outra famosa rede de criptomoedas, a *Ethereum*. A Keccak foi criada por quatro desenvolvedores que são também responsáveis pelo desenvolvimento da função esponja. Ao instanciar uma função esponja o Keccak não apresenta limitações no tamanho da entrada e saída, podendo ser flexível à dimensão dos blocos e o espaço de armazenamento. (DINUR; DUNKELMAN; SHAMIR, 2012).

2.2.5 Estrutura da *hash*

Uma das estruturas mais importantes na estruturação de *hash*, são as listas vinculadas, elas formam uma cadeia de blocos responsáveis por armazenar o endereço entre os conjuntos subsequentes. Os ponteiros são responsáveis por guardar essas informações nos blocos da cadeia, tornando a rede *blockchain* imutável. (BLOCKGEEKS, 2017).

Figura 05 - Blocos vinculados.



Fonte: *Blockgeeks - What is a hashing?*

Ou seja, mesmo que essa rede sofra um ataque, e haja a tentativa de alteração de um de seus blocos, conseqüentemente todos os outros terão que ser alterados para que o ataque seja efetivo. São essas propriedades de *hash* que fazem uma rede *blockchain* ser tão segura, atestando sua imutabilidade. (BLOCKGEEKS, 2017).

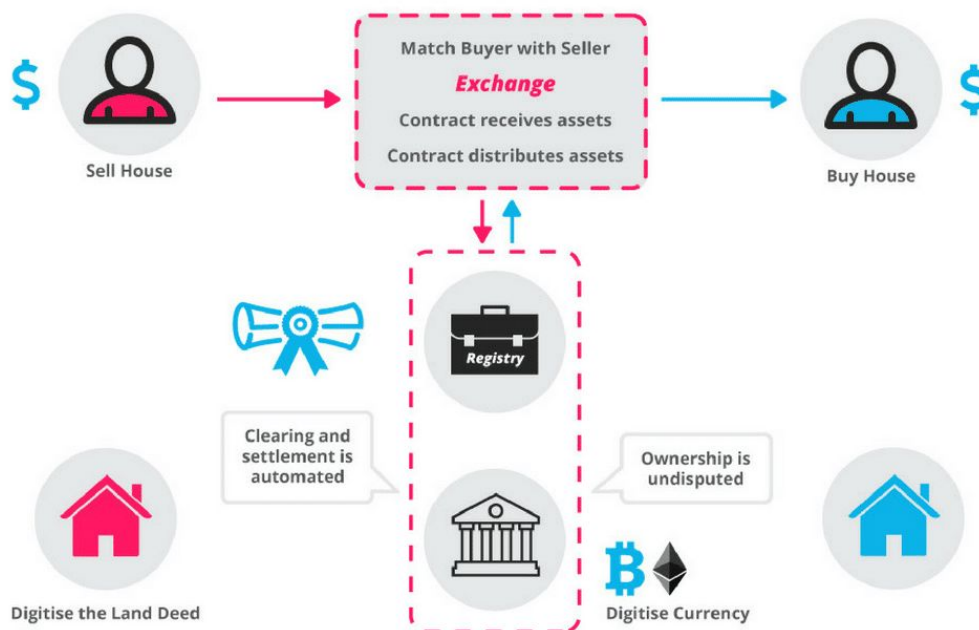
2.3 Smart Contracts

Os *Smart Contracts*, contratos inteligentes, buscam facilitar a negociação digital através de um ambiente descentralizado, onde não são necessários *softwares* intermediários para que as transações ocorram. (COURSERA, 2019). Muitos desses intermediários são amplamente utilizados no cotidiano, como aplicativos de bancos, venda online, logísticas de entrega, entre outros. Mas assim como *softwares* podem ser aprimorados e substituídos, ambientes físicos também estão sujeitos a sofrer mudanças e atualizações. (BLOCKGEEKS, 2016).

Um grande exemplo de meio físico discutido atualmente são os cartórios, nos quais são oferecidos serviços de autenticação de documentos, reconhecimento de firma, registro de imóveis, escrituras de venda e compra, entre outros. (CARDOSO; PINTO, 2018). Esse modelo futuramente pode ser facilmente substituído por uma rede blockchain aplicada por *Smart Contracts*, onde todos os registros serão incluídos na cadeia de dados, evitando fraudes, agilizando processos de autenticação e priorizando a acessibilidade dos usuários. (COURSERA, 2019).

A tecnologia *blockchain* atrai a atenção não apenas por ser segura e autêntica, mas também por seus benefícios. Atualmente transações bancárias realizadas por usuários são taxadas, tanto pessoas físicas quanto jurídicas, se submetem a pagar tributos pelo uso de máquinas de cartão. (COURSERA, 2019). No entanto os *Smart Contracts* contribuem a favor das transações realizadas através de uma rede *blockchain* por não possuírem custo, possibilitando ao usuário uma maior liberdade no número de transações. (BLOCKGEEKS, 2016).

Figura 06 - Exemplificação de *Smart Contract*.



Fonte: Jusbrasil - Contratos inteligentes: descubra o que são e como funcionam.

Pode ser observado na figura 06 uma definição simples de *Smart Contracts* utilizando o conceito venda de imóvel, na qual possuem dois atores principais, vendedor e comprador, que por meio dessa aplicação foram combinados para poderem trocar seus bens. Neste caso o funcionamento ocorre da seguinte forma, quando o vendedor receber a informação que o dinheiro da compra da casa foi enviado, sucessivamente ele poderá enviar toda a documentação necessária para provar que a casa não está mais em seu nome.

Em um meio físico essa transação seria realizada por meio de um cartório com auxílio de um tabelião para comprovar autenticidade da documentação desta venda. Através do *Smart Contract* toda a transação seria feita em um tempo muito mais curto, podendo ser paga em qualquer moeda digital e com a possibilidade de compradores e vendedores estarem no conforto de suas casas.

No presente momento, existem inúmeras formas de criar, manipular e desenvolver um *Smart Contract*, principalmente para construir uma aplicação baseada na estrutura da moeda digital *Ethereum*. (BLOCKGEEKS, 2016). Há duas linguagens de programação mais utilizadas para construir uma aplicação, a *Go Language* e a *Solidity*. (COURSERA, 2019). A *Solidity* é uma linguagem própria da *Ethereum*, que atuou na construção da própria moeda digital e também está sendo utilizada para criação deste trabalho. (COURSERA, 2019). Na figura 07 pode ser observado um exemplo de contrato inteligente:

Figura 07 - Exemplificação de código *Smart Contract*.

```

/* Allow another contract to spend some tokens in your behalf */
function approve(address _spender, uint256 _value)
    returns (bool success) {
    allowance[msg.sender][_spender] = _value;
    return true;
}

/* Approve and then communicate the approved contract in a single tx */
function approveAndCall(address _spender, uint256 _value, bytes _extraData)
    returns (bool success) {
    tokenRecipient spender = tokenRecipient(_spender);
    if (approve(_spender, _value)) {
        spender.receiveApproval(msg.sender, _value, this, _extraData);
    }
    return true;
}

/* A contract attempts to get the coins */
function transferFrom(address _from, address _to, uint256 _value) returns (bool success) {
    if (balanceOf[_from] < _value) throw; // Check if the sender has enough
    if (balanceOf[_to] + _value < balanceOf[_to]) throw; // Check for overflows
    if (_value > allowance[_from][msg.sender]) throw; // Check allowance
    balanceOf[_from] -= _value; // Subtract from the sender
    balanceOf[_to] += _value; // Add the same to the recipient
    allowance[_from][msg.sender] -= _value;
    Transfer(_from, _to, _value);
    return true;
}

/* This unnamed function is called whenever someone tries to send ether to it */
function () {
    throw; // Prevents accidental sending of ether
}

```

Fonte: Blockgeeks - *Smart Contracts*.

2.4 Pregão Eletrônico

O Pregão é caracterizado por utilizar recursos tecnológicos para aplicação de licitações, podendo ser utilizado a distância por seus participantes e órgãos licitatórios por meio da internet. Os participantes do Pregão Eletrônico são empresas que fornecerão determinado serviço ou material para o órgão correspondente à sua área de atuação. (FÓRUM, 2015).

Esse tipo de licitação proporciona a seus participantes e cidadãos uma experiência de transparência ao decorrer de todo o processo. Tal modalidade permitiu que as licitações fossem realizadas com maior rapidez, além do aumento no número de empresas interessadas a disputar e apresentar suas propostas. O processo como um todo atua também dificultando a formação de cartéis e minimizando a ocorrência de possíveis fraudes. (LICITAÇÃO, 2015).

2.4.1 Processos da licitação

Atualmente para que as empresas participem das licitações em forma de pregão precisam se inscrever no site do órgão licitante, após este primeiro passo, os fornecedores recebem uma senha para permitir o processo de certificação da empresa. Ao certificar a empresa os fornecedores estarão preparados à participar de licitações que ocorrerão no órgão cadastrado renovando seu cadastro uma vez ao ano. (IFTO, 2014). (LICITAÇÕES, 2018).

Para que uma empresa possa concorrer com outros fornecedores ela deve seguir os padrões apresentados no edital do órgão licitante. As propostas ocorrem como em uma sala de bate papo ou chat, mas sem que haja reconhecimento de nenhum dos participantes por parte dos outros, com o objetivo de não ocorrer nenhum tipo de fraude no processo. (LICITAÇÃO, 2015).

Esse tipo de licitação é bastante semelhante aos leilões reversos, pois os leilões tradicionais iniciam-se do menor valor e terminam no maior, já o processo de Pregão recebe propostas com a intenção dos participantes disputarem pelo menor valor até que sobre apenas o último lance. Todo esse processo tem início em uma data especificada pelo órgão expedidor, podendo se estender por até semanas. Ao escolher a proposta de menor valor, o órgão analisa se a empresa que apresentou o seu lance está habilitada e certificada pelo edital. (LICITAÇÃO, 2015).

3. Desenvolvimento da aplicação

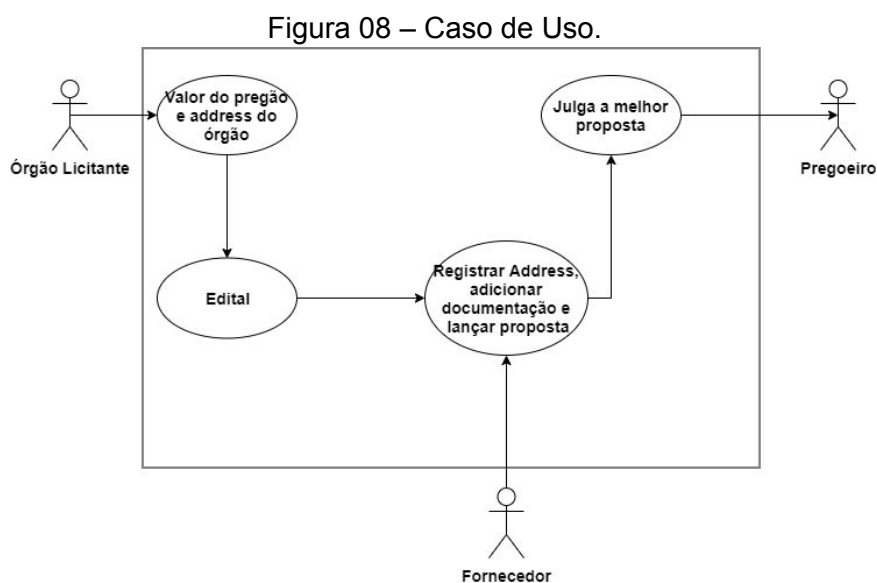
Para ter uma ampla visualização do projeto, houve a necessidade de aplicar engenharia de *software* a metodologia *use case*, adotada para facilitar todos os processos de criação e aplicação, apoiando o desenvolvedor e o conduzindo em todas as etapas. Com ela foi possível separar as fases e seus respectivos atores, primeiramente realizou-se a localização dos atores. (SOUZA, 2016).

O principal acionista no processo de implementação das licitações é o órgão licitante, devido a isso o primeiro ator foi definido como “Órgão

Licitante”. O papel deste autor é incluir o valor do pregão, além de seu endereço em formato de *hash* através de sua carteira digital e publicar o edital com as normas que deverão ser seguidas pelos fornecedores.

O próximo autor é o “Fornecedor”, responsável por se registrar no pregão por meio do seu endereço, também localizado na sua carteira digital e registrar o valor de sua proposta, desta maneira, a empresa fornecedora poderá apresentar inúmeras propostas do início ao final da licitação.

Ao final do pregão é escolhido o fornecedor ganhador, neste momento o autor “Pregoeiro” entra em ação e elege a empresa ganhadora com a proposta de menor valor. O pregoeiro que é encontrado nessa aplicação foi automatizado por lógica de programação, a realização deste *use case* pode ser concluída graças ao *software* draw.io, onde são encontradas várias ferramentas para projetar *softwares* e outras soluções.



Fonte: Desenvolvida pelo autor.

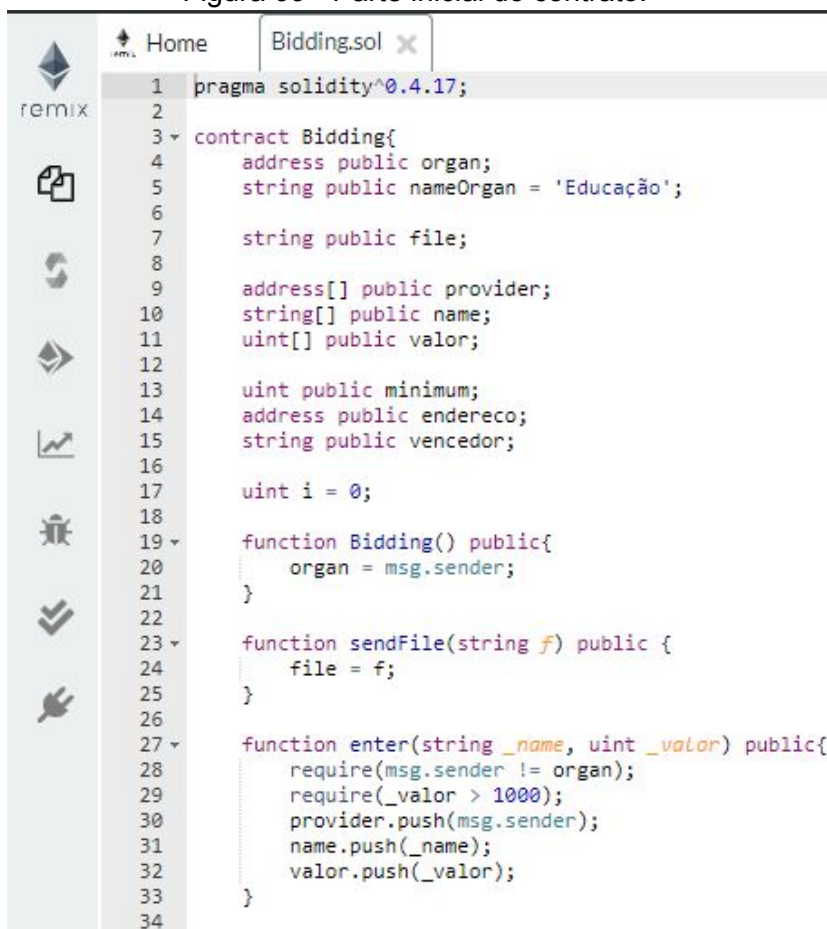
3.1 IDE para desenvolvimento do código

Como já citado anteriormente, o código foi escrito em *Solidity*, linguagem desenvolvida para a criação do *Ethereum* e *Smart Contracts*, devido a isso houve a necessidade de simular as interação do usuário com o contrato inteligente. O *software Remix IDE* proporciona uma representação do código

em forma aplicação ao ser compilado, o mesmo é capaz de depurar, testar e implantar contratos, desenvolvido em *javascript* pode ser utilizado no navegador e localmente.

Para poder implantar um código no *Remix IDE* é obrigatório incluir o nome do contrato, a aplicação desenvolvida neste trabalho recebeu a denominação de “*Bidding.sol*”, na qual possui a extensão “.sol”, ou seja, escrito em linguagem *Solidity*, ao definir o nome o contrato pode ser iniciado.

Figura 09 - Parte inicial do contrato.



```
1 pragma solidity^0.4.17;
2
3 contract Bidding{
4     address public organ;
5     string public nameOrgan = 'Educação';
6
7     string public file;
8
9     address[] public provider;
10    string[] public name;
11    uint[] public valor;
12
13    uint public minimum;
14    address public endereco;
15    string public vencedor;
16
17    uint i = 0;
18
19    function Bidding() public{
20        organ = msg.sender;
21    }
22
23    function sendFile(string f) public {
24        file = f;
25    }
26
27    function enter(string _name, uint _valor) public{
28        require(msg.sender != organ);
29        require(_valor > 1000);
30        provider.push(msg.sender);
31        name.push(_name);
32        valor.push(_valor);
33    }
34
```

Fonte: Desenvolvida pelo autor.

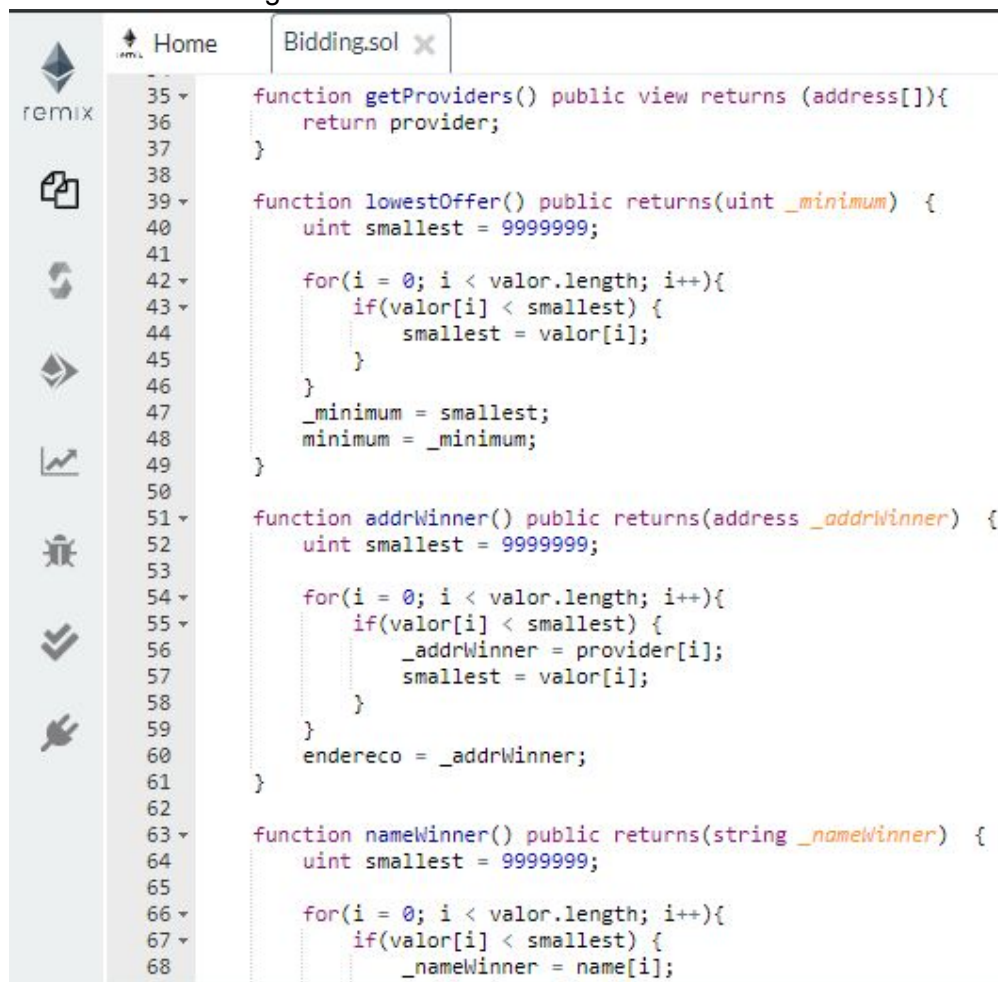
Primeiramente deve ser mencionado a versão que esse contrato será escrito, neste caso a versão 0.4.17 foi utilizada, mesmo não sendo a versão mais atualizada no momento, se comportou muito bem aos testes e está completamente funcional.

Analisando a figura 09 é possível identificar as variáveis existentes, a variável “*organ*” do tipo *address* é responsável por guardar o endereço do órgão licitante assim como foi explicado no *use case*. Já variável “*nameOrgan*” do tipo *string* guarda o nome do órgão e a variável “*file*” do tipo *string* irá receber a documentação que será transformada em *hash*, uma semelhança entre elas é a definição de público, ou seja, o resultado pode ser observado para consulta.

As variáveis que apresentam os colchetes são definidas como *array*, devido ao fato de poder incluir inúmeros participantes em um pregão eletrônico, foram denominadas da seguinte forma: *provider*, *name* e *value*. A variável “*provider*” do tipo *address*, inclui o endereço do participante, a variável “*name*” do tipo *string*, inclui o nome fantasia da empresa e a variável “*value*” do tipo *uint* recebe o valor da proposta.

Logo após a adição dessas variáveis, é necessário a inserção de algumas outras funções para o contrato funcionar, a variável “*minimum*” é uma delas e age com o tipo *uint* capturando o menor valor assim que a função é acionada. Em seguida a variável “endereço” com o tipo “*address*” receberá da função “*addrWinner*” o endereço vencedor. Já função “*nameWinner*” enviará o nome fantasia vencedor para a variável “*name*” do tipo *string*, enquanto a variável “*i*” é utilizada apenas para contador do ciclo de repetição das funções.

Figura 10 - Parte inicial do contrato.



```

35  function getProviders() public view returns (address[]){
36      return provider;
37  }
38
39  function lowestOffer() public returns(uint _minimum) {
40      uint smallest = 9999999;
41
42      for(i = 0; i < valor.length; i++){
43          if(valor[i] < smallest) {
44              smallest = valor[i];
45          }
46      }
47      _minimum = smallest;
48      minimum = _minimum;
49  }
50
51  function addrWinner() public returns(address _addrWinner) {
52      uint smallest = 9999999;
53
54      for(i = 0; i < valor.length; i++){
55          if(valor[i] < smallest) {
56              _addrWinner = provider[i];
57              smallest = valor[i];
58          }
59      }
60      endereco = _addrWinner;
61  }
62
63  function nameWinner() public returns(string _nameWinner) {
64      uint smallest = 9999999;
65
66      for(i = 0; i < valor.length; i++){
67          if(valor[i] < smallest) {
68              _nameWinner = name[i];

```

Fonte: Desenvolvida pelo autor.

Um contrato inteligente é repleto de funções as quais são responsáveis por executar os seus processos, a primeira função do *Bidding* se assegura de que só órgão poderá gerar este contrato. A próxima função é denominada de *enter*, é a função encarregada de registrar os fornecedores, neste caso recebendo um nome simbólico e a proposta mínima acima de mil reais fictícios, esta função não pode receber lances do autor do órgão.

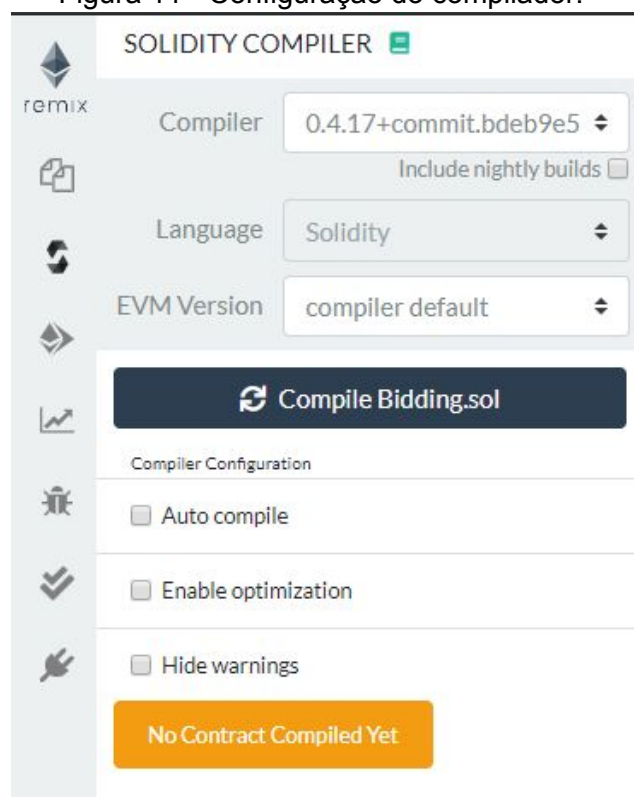
A função *“sendFile”* é utilizada para encapsular a documentação da empresa, ja *“getProviders”* tem a capacidade de exibir o número de participantes do contrato em tempo real. As funções *“lowestOffer”*, *“addrWinner”* e *“nameWinner”*, possuem basicamente a mesma lógica devido ao fato de utilizarem laço de repetição para revelar o ganhador. No quesito de menor oferta, a função *“lowestOffer”* revela a menor proposta, *“addrWinner”*

exibe o endereço vencedor e “*nameWinner*” exibe o nome fictício vencedor, todas elas só poderão ser acessadas pelo órgão licitante.

3.1.1 Compilando o código

Primeiramente deve-se ser escolhida a versão do compilador, pois a linguagem e as outras configurações já estão preparadas para o uso; após escolhida a versão, basta clicar para compilar.

Figura 11 - Configuração do compilador.



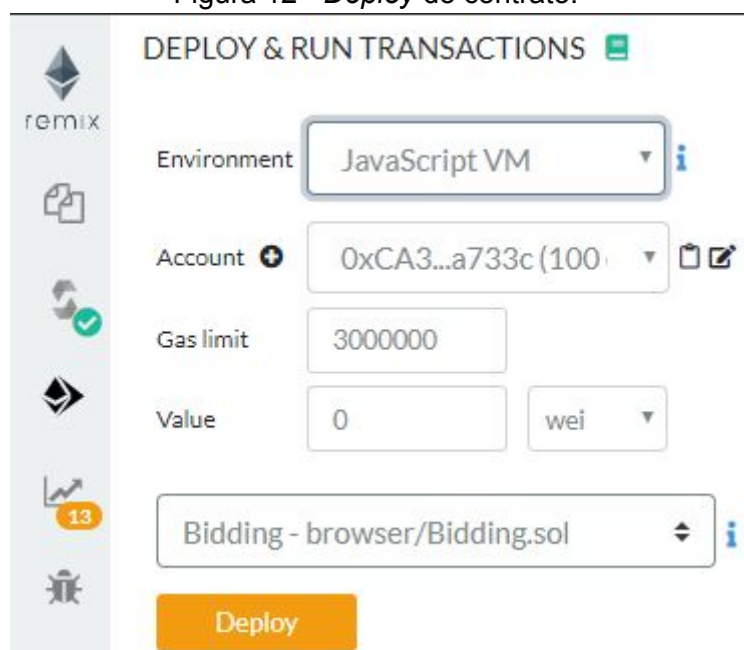
Fonte: Desenvolvida pelo autor.

3.1.2 Processo de implantação

O *deploy* do código deve ser iniciado pelo órgão licitante, nesta seção há duas listas dinâmicas: a primeira com o nome de *Environment*, define o modelo de execução, podendo ser executada em três formatos, *JavaScript VM*, por meio de uma API com Web3 e localmente através do Web3. Já a segunda lista dinâmica contém os endereços, é necessário escolher um deles para ser o órgão licitante.

Ao escolher o endereço do órgão licitante, todos os outros poderão ser usados para simular as empresas. Após a seleção do endereço, pode ser definido o *Gas limit* deste *Smart Contract*, e a opção “*Value*” é utilizada se houver a necessidade de pagar por uma transação. Ao configurar todas as opções acima, o *deploy* pode ser realizado.

Figura 12 - *Deploy* do contrato.

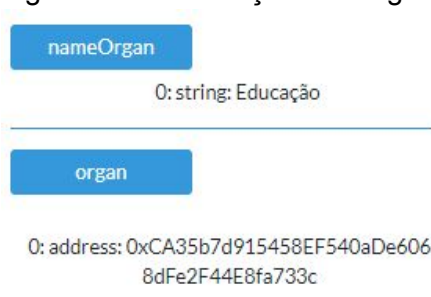


Fonte: Desenvolvida pelo autor.

3.1.3 Simulando Pregão Eletrônico

As variáveis públicas exibem o estado que foi definido a elas.

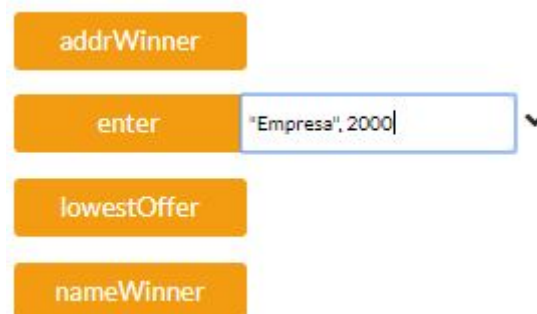
Figura 13 - Informações do órgão.



Fonte: Desenvolvida pelo autor.

Para simular a entrada de um fornecedor é essencial fazer a alteração do endereço, após modificadas as informações a empresa poderá utilizar a função enter.

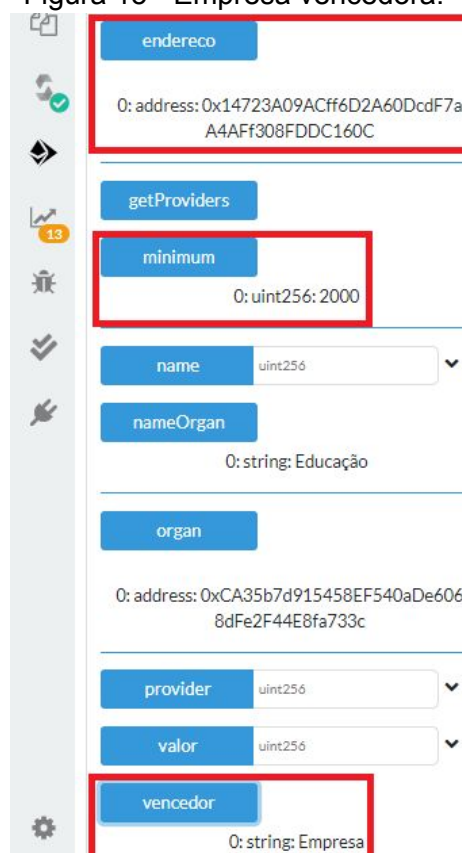
Figura 14 - Inserção de registro.



Fonte: Desenvolvida pelo autor.

Ao inserir o registro da empresa que irá participar, podemos acionar a funções e exibir o resultado.

Figura 15 - Empresa vencedora.



Fonte: Desenvolvida pelo autor.

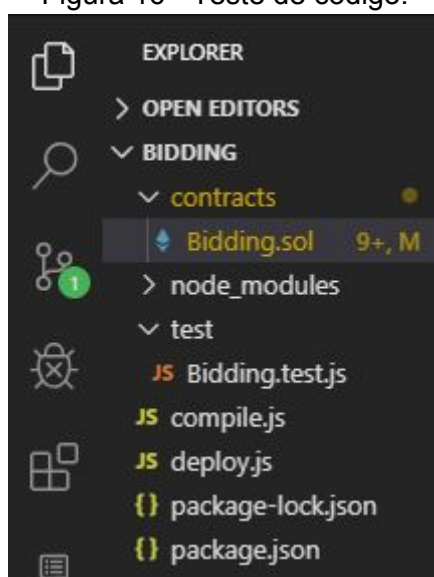
3.2 Testes de desenvolvimento

A metodologia de testes adotada no desenvolvimento da aplicação se distancia um pouco do uso de ferramentas tais como o *Remix IDE*, neste processo foi utilizada a execução de um editor de código que se mostrou mais que suficiente para realizar os testes antes da aplicação definitiva ao sistema final.

O editor *Visual Code* obteve um grande papel em todo o processo de desenvolvimento, o mesmo possibilitou que fossem instaladas várias dependências que atuaram a favor da conclusão do projeto, onde foi possível configurar o compilador do código e o ambiente para fazer implementação dos testes. Essas dependências foram instaladas através do *node* e *npm*.

O *npm* faz apenas o gerenciamento de pacotes do *Node.js*, tal ferramenta foi originada em 2009 com o conceito de *open source*, com o objetivo de auxiliar desenvolvedores instalar pacotes em suas aplicações e compartilhá-los. Já o *Node.js* apresenta um conceito inovador, onde busca processar serviços com muita eficiência e gerenciamento do consumo de memória, o que torna sua plataforma um local versátil de desenvolvimento sem que ocorram falhas e travamentos. (PEREIRA, 2014).

Figura 16 - Teste de código.



Fonte: Desenvolvida pelo autor.

A figura 16 é composta pelo mesmo código apresentado pelo trecho de testes realizados pelo *Remix IDE*. O que os diferencia é o formato da plataforma, pois a ferramenta *Remix* já está pronta, enquanto o *Visual Code* teve sua plataforma configurada do zero.

Ainda analisando a figura, pode-se observar a pasta *node_modules* concentrada logo abaixo do arquivo *Bidding.sol*, nela estão contidos todos os pacotes e dependências que a aplicação consome para se comunicar. Com destaque para algumas API's que fazem essa comunicação via internet, os pacotes citados foram instalados através do *npm*, com apenas o comando “*npm install*”.

Figura 17 - Compilador.



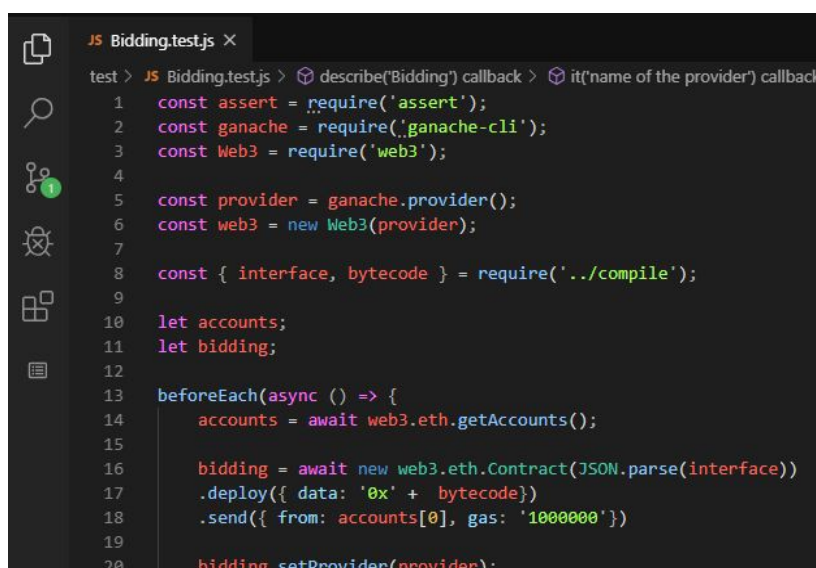
```
JS compile.js <
JS compile.js > ...
1  const path = require('path');
2  const fs = require('fs');
3  const solc = require('solc');
4
5  const biddingPath = path.resolve(__dirname, 'contracts', 'Bidding.sol');
6  const source = fs.readFileSync(biddingPath, 'utf8');
7
8  module.exports = solc.compile(source, 1).contracts[':Bidding'];
```

Fonte: Desenvolvida pelo autor.

O arquivo compilador tem o objetivo de definir o caminho do diretório no qual se encontra o código da aplicação e auxiliá-la a trabalhar em diferentes plataformas. A variável “*source*” faz a sincronização de leitura e define o modo como será lido.

A última linha definida por *module.exports* é responsável por exportar o conteúdo selecionado, apesar de um único contrato estar presente nesta demonstração podem ser adicionados mais itens de acordo com a necessidade do projeto.

Figura 18 - Rotina de testes.



```

test > JS Bidding.test.js > describe('Bidding') callback > it('name of the provider') callback
1  const assert = require('assert');
2  const ganache = require('ganache-cli');
3  const Web3 = require('web3');
4
5  const provider = ganache.provider();
6  const web3 = new Web3(provider);
7
8  const { interface, bytecode } = require('../compile');
9
10 let accounts;
11 let bidding;
12
13 beforeEach(async () => {
14   accounts = await web3.eth.getAccounts();
15
16   bidding = await new web3.eth.Contract(JSON.parse(interface))
17     .deploy({ data: '0x' + bytecode})
18     .send({ from: accounts[0], gas: '100000'})
19
20   bidding.setProvider(provider);

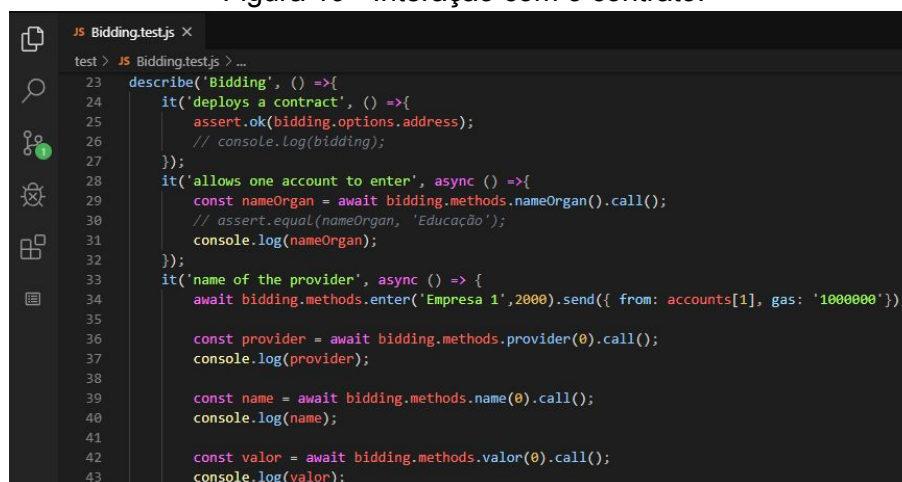
```

Fonte: Desenvolvida pelo autor.

A figura 18 representa a rotina de testes, logo no início faz a comunicação com algumas bibliotecas que disponibilizam ferramentas com capacidade de desenvolvimento em *Ethereum* de uma forma mais rápida, por exemplo a *web3*. Ela é capaz de exibir as contas que estão disponíveis para simular transações em uma rede de testes *Ethereum*, agindo diretamente com uma interface que utiliza o compilador criado na figura anterior.

Essa imagem também apresenta a forma de como o contrato será implantado, simulando a mesma operação do *Remix IDE*, com o *deploy* informando o endereço da conta e o *gas*, que se refere ao valor que poderá ser gasto pelo contrato.

Figura 19 - Interação com o contrato.



```

test > JS Bidding.test.js > ...
23 describe('Bidding', () =>{
24   it('deploys a contract', () =>{
25     assert.ok(bidding.options.address);
26     // console.log(bidding);
27   });
28   it('allows one account to enter', async () =>{
29     const nameOrgan = await bidding.methods.nameOrgan().call();
30     // assert.equal(nameOrgan, 'Educação');
31     console.log(nameOrgan);
32   });
33   it('name of the provider', async () => {
34     await bidding.methods.enter('Empresa 1',2000).send({ from: accounts[1], gas: '100000'});
35
36     const provider = await bidding.methods.provider(0).call();
37     console.log(provider);
38
39     const name = await bidding.methods.name(0).call();
40     console.log(name);
41
42     const valor = await bidding.methods.valor(0).call();
43     console.log(valor);

```

Fonte: Desenvolvida pelo autor.

Acima na figura 19, estão sendo representadas as interações com o *Smart Contract*, um exemplo é demonstrado pelo comando “*call()*”, exibindo o resultado das funções e variáveis públicas. Já o método “*send()*” faz a implementação de itens na rede *blockchain*, como simulado com a informação Empresa 1 e valor 2000. As inserções são acompanhadas do endereço e o *gas* da transação, na figura 20 pode-se observar uma demonstração este teste funcionando.

Figura 20 - Demonstração da aplicação.

```
✓ deploys a contract
Educação
✓ allows one account to enter
0xC69e2aC243d5232AADe8e967ad5009a1B35ea8ff
Empresa 1
2000
[ '0xC69e2aC243d5232AADe8e967ad5009a1B35ea8ff' ]
2000
0xC69e2aC243d5232AADe8e967ad5009a1B35ea8ff
Empresa 1
✓ name of the provider (471ms)

3 passing (903ms)
```

Fonte: Desenvolvida pelo autor.

Após a execução correta de todas as rotinas de testes que foram criadas para a preparar a aplicação, deu-se início ao processo de elaboração do *software*. O programa foi criado na *framework JavaScript React*, a qual possibilita a criação de interfaces de usuário de uma maneira simples e rápida, assim como plataformas *web* e desenvolvimento de aplicações nativas para *iOS* e *Android*. (EISENMAN, 2015).

A aplicação desenvolvida em *React* foi instalada através do gerenciador de pacotes *npm* para executar o comando desejado, *npm install create-react-app*. Este comando possibilita a criação de ambientes *React* de uma forma muito simples, ao mencionar outro comando no terminal o local de desenvolvimento é gerado, como por exemplo, *create-react-app bidding-react*.

O primeiro arquivo a ser configurado na aplicação foi a *web3*, biblioteca responsável pela interação com a rede *Ethereum*, tendo que ser instalada pelo gerenciador de pacotes npm e logo após importada na aplicação.

Figura 21 - Configuração da biblioteca web3.

```
1  import Web3 from 'web3';
2
3  const getProvider = async() =>{
4    |   await window.web3.currentProvider.enable();
5    | }
6
7  getProvider();
8
9  const web3 = new Web3(window.web3.currentProvider);
10
11 export default web3;
```

Fonte: Desenvolvida pelo autor.

Após configurar a biblioteca acima, foi criado um arquivo chamado *bidding.js*, neste arquivo contém o endereço do contrato que foi criado na rede blockchain e também todas as informações do mesmo, que são fornecidas por sua ABI com o papel de realizar interações por meio do *JavaScript*.

A biblioteca *web3* é importada neste arquivo para que interação seja realizada com a rede *Ethereum* de testes.

Figura 22 - Informações do contrato.

```
import web3 from './web3';

const address = '0x65dc079797Cdb838eC20DB6d818FD095474b9d5A';

const abi = [
  {
    "constant": true,
    "inputs": [
      {
        "name": "",
        "type": "uint256"
      }
    ],
    "name": "name",
    "outputs": [
      {
        "name": "",
        "type": "string"
      }
    ]
  }
]
```

Fonte: Desenvolvida pelo autor.

Figura 23 - Aplicação da biblioteca para interação.

```
export default new web3.eth.Contract(abi, address);
```

Fonte: Desenvolvida pelo autor.

Para que fosse possível adicionar documentos à rede blockchain com o intuito de impedir fraudes nos mesmos houve a necessidade de adicionar IPFS na aplicação. A API aplica uma assinatura digital aos documentos enviados pelas empresas transformando-os em *hash*, com o objetivo de remover possíveis duplicações na rede. (FREECODECAMP, 2018).

Figura 24 - API encapsuladora de documentos.

```
1 const IPFS = require('ipfs-api');
2 const ipfs = new IPFS({ host: 'ipfs.infura.io', port: 5001, protocol: 'https' });
3
4 export default ipfs;
```

Fonte: Desenvolvida pelo autor.

Após entender e exibir todos os ambientes criados para complementar a aplicação, restou somente o desenvolvimento no *front-end*, responsável por apresentar a interface do *software*, onde conceitos de HTML e CSS foram aplicados.

Figura 25 - Interface da aplicação.

Pregão Eletrônico

Orgão licitante: Educação Participantes: 1 Menor proposta: 5000

Insira a documentação

Escolher arquivo Nenhum arquivo selecionado

Proposta

Nome fantasia Valor da proposta

Entrar

Vencedor

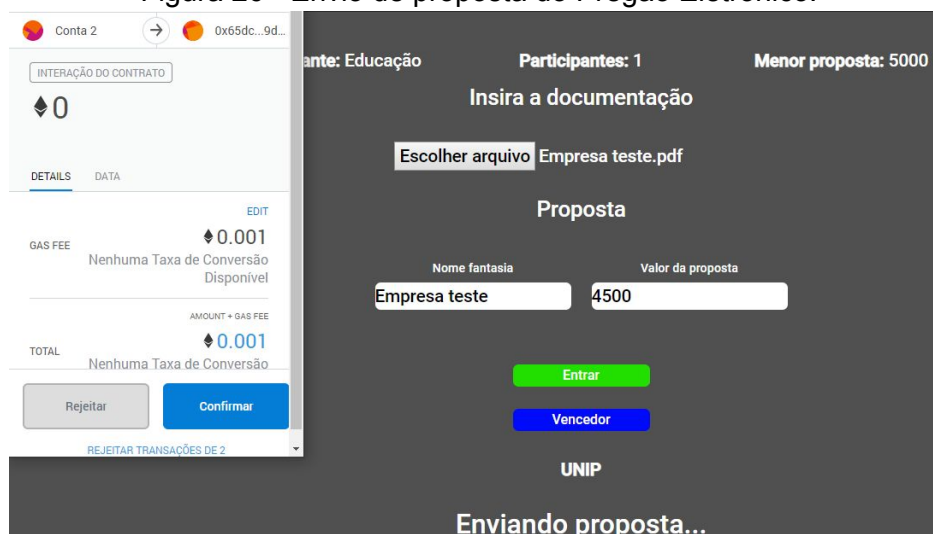
UNIP

Fonte: Desenvolvida pelo autor.

Na imagem 25 está representada uma simulação de Pregão Eletrônico disponibilizado pelo órgão licitante do Ministério da Educação. No momento representado pela aplicação se encontra apenas um participante com um lance fictício referente a R\$ 5.000,00 reais, pode-se observar que o atual vencedor é a faculdade UNIP.

Para exemplificar melhor a atuação do *software*, será realizada uma representação do processo para que seja possível entender como as informações são autenticadas.

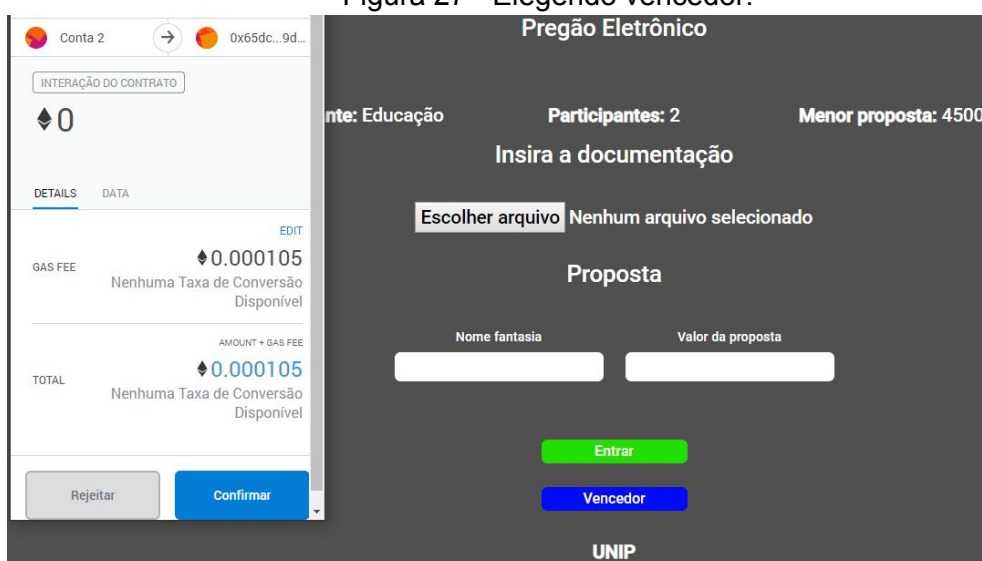
Figura 26 - Envio de proposta do Pregão Eletrônico.



Fonte: Desenvolvida pelo autor.

A proposta é enviada através de uma carteira digital chamada *MetaMask*, que fornece uma interface para seus usuários por meio do próprio navegador e assina transações realizadas em uma rede *blockchain* de testes. (ANTONOPOULOS, 2018). Esta etapa da aplicação é o primeiro momento do envio de proposta, no qual os campos foram preenchidos com as informações necessárias e todas as transações devem ser confirmadas para que sejam concluídas.

Figura 27 - Elegendo vencedor.



Fonte: Desenvolvida pelo autor.

Após a inserção dos dados na rede o número de participantes foi alterado para dois e o valor da proposta enviada pela empresa teste foi atualizado pela aplicação. No próximo estágio ocorre a eleição do vencedor com a menor proposta, apresentando o nome da empresa na parte inferior da página.

Figura 28 - Vencedor do Pregão Eletrônico.

Pregão Eletrônico

Orgão licitante: Educação Participantes: 2 Menor proposta: 4500

Insira a documentação

Escolher arquivo Nenhum arquivo selecionado

Proposta

Nome fantasia Valor da proposta

Entrar













Vencedor

Empresa teste

Fonte: Desenvolvida pelo autor.

Ao acessar a plataforma do *Etherscan* onde são simuladas as transações pelo *Rinkeby Test Net*, nota-se todas as transferências que foram realizadas no contrato. Há o registro das informações do horário de cada transação, *hash* dos participantes e os blocos que foram criados, de forma que tais dados possam ser acessados para consulta de forma transparente.

Figura 29 - Painel de exibição de transações.

Txn Hash	Block	Age	From	To	Value	[Txn Fee]
0xcd2666e8704716...	5351338	57 secs ago	0x7d3c6b901c7c5b...	 0x65dc079797cdb8...	0 Ether	0.000054944
0x08b85265169970...	5351332	2 mins ago	0x7d3c6b901c7c5b...	 0x65dc079797cdb8...	0 Ether	0.000048287
0xc0333eaf72f42294...	5351327	3 mins ago	0x7d3c6b901c7c5b...	 0x65dc079797cdb8...	0 Ether	0.000041039
0x3c1539ccc58781...	5351327	3 mins ago	0x7d3c6b901c7c5b...	 0x65dc079797cdb8...	0 Ether	0.000101445
0x82fb658e47fd1fc...	5339726	2 days 24 mins ago	0x01d71e393d6a01...	 0x65dc079797cdb8...	0 Ether	0.000059457
0x09475c09e7837b...	5339723	2 days 24 mins ago	0x01d71e393d6a01...	 0x65dc079797cdb8...	0 Ether	0.000009223
0xee5bd1e1f94d89b...	5339722	2 days 25 mins ago	0x01d71e393d6a01...	 0x65dc079797cdb8...	0 Ether	0.000041039
0x259d65031a852c...	5339721	2 days 25 mins ago	0x01d71e393d6a01...	 0x65dc079797cdb8...	0 Ether	0.000145889
0xa7e5501b4c0340...	5339707	2 days 28 mins ago	0x01d71e393d6a01...	 0x65dc079797cdb8...	0 Ether	0.000041039
0x84b8c88ea16312...	5339704	2 days 29 mins ago	0x01d71e393d6a01...	 0x65dc079797cdb8...	0 Ether	0.000041039
0x7b6dd73a565956...	5339701	2 days 30 mins ago	0x01d71e393d6a01...	 0x65dc079797cdb8...	0 Ether	0.000041039
0x224d3a74764646...	5339654	2 days 42 mins ago	0x01d71e393d6a01...	 0x65dc079797cdb8...	0 Ether	0.000088039

Fonte: *Etherscan Rinkeby Test Net* - Plataforma de testes *Ethereum*.

CONCLUSÃO

O trabalho aborda a falta de informações disponíveis sobre como o dinheiro público é aplicado, no qual foi exemplificado o processo de Pregão Eletrônico. Essa modalidade possibilita a ampliação do leque de empreendedores interessados, onde empresas de todos os portes e regiões do Brasil possam participar devida a facilidade de acesso via internet.

Neste projeto foi criada uma ferramenta que utiliza recursos para auxiliar na exibição dos processos e transações, a fim de evitar tentativas de fraude. A aplicação foi desenvolvida através de uma tecnologia chamada *blockchain* na qual contém a metodologia *Smart Contracts*, responsável por autenticar e gerar logs ao decorrer das implementações que são feitas na rede.

A aplicação obteve sucesso nos testes realizados, como esperado foram gerados logs de todas as transações na rede *blockchain*, envio de proposta, envio da documentação dos fornecedores. Os dados foram alocados em uma rede de testes denominada de *Rinkeby Testnet Explorer* que atua juntamente com a API IPFS de arquivos, eliminando duplicação de documentos na rede.

Dentre os procedimentos realizados é necessária a confirmação dos mesmos na carteira virtual *Metamask*, servindo como uma ferramenta de autenticação, dificultando então a aplicação de fraudes durante o processo. Esta ação submete o usuário a realizar uma série de confirmações durante o contrato, nos quais todas as transações e confirmações exibem a *hashing* do próximo bloco de criação.

Esta aplicação tem grande valor para a sociedade, pois apresenta dados que são gerados em tempo real, podendo despertar um maior interesse da população. Ela atua de forma didática instruindo de forma simples através destes registros a população a compreender como o dinheiro público é aplicado atualmente. Além de motivar a população a ter mais confiança nestas transições por meio de processos de autenticação.

A apresentação e implementação do *blockchain* como uma importante ferramenta com potencial para criação de aplicações, soluções, segurança e qualidade, que podem ser utilizadas não apenas no setor público mas também em setores privados.

Mostrou-se relevante a possibilidade de ampliação da aplicação através da implementação do conceito de I.A, Inteligência Artificial, através da qual espera-se ter mais precisão no ato de eleição dos fornecedores através da substituição do pregoeiro no processo. Além desta, a inclusão da tecnologia OCR, Reconhecimento Óptico de Caracteres, neste projeto poderá propiciar a verificação de falhas e informações divergentes na documentação cedida pelo fornecedor.

A disponibilização dos documentos que estão implementados na rede também mostra-se um alvo para expansão deste projeto, pois estes poderiam ser consultados para avaliações futuras, não mais sendo apenas alocados na rede por meio de uma *hash*.

Este trabalho foi muito importante para o amadurecimento pessoal, proporcionando a aquisição de experiência para enfrentar inúmeras situações

durante a criação de todo o projeto, preparando o autor para futuras melhorias sem que haja dificuldade neste processo. Conhecimentos adquiridos durante cursos, artigos, sites e vídeos foram essenciais para utilizar a tecnologia abordada. Estes recursos permitiram que os códigos que compõem a aplicação fossem criados favorecendo a elaboração do trabalho, assim como a graduação possibilitou que o autor usufrísse dos conhecimentos fornecidos pelos mestres.

REFERÊNCIAS BIBLIOGRÁFICAS

ANTONONOPOULOS, Andreas M.; PH.D., Gavin Wood. **Mastering Ethereum: Building Smart Contracts and DApps**. London: O'reilly, 2018.

Assymetric cryptography (public key cryptography). **SearchSecurity**, 2019. Disponível em: <<https://searchsecurity.techtarget.com/definition/asymmetric-cryptography>>. Acesso em: 26 de Agosto de 2019.

Blockchain Basics; Coursera: 2019.

CARDOSO, João; PINTO, Jefferson. **Blockchain e Smart Contracts: Um Estudo Sobre Soluções para Seguradoras**. Farolândia, Pág. 17, 2018.

CHRISTIDIS, Konstantinos; DEVETSIKIOTIS, Michael. **Blockchains and Smart Contracts for the Internet of Things**. Raleigh, Pág. 12, 2016.

CONTROLADORIA-GERAL DA UNIÃO. **Portal da Transparência**, 2019. Disponível em: <<http://www.portaltransparencia.gov.br/>>. Acesso em: 10 de Agosto de 2019.

DINUR, Itai; DUNKELMAN, Orr; SHAMIR, Adi. **New Attacks on Keccak-224 and Keccak-256**. Rehovot, Pág. 27, 2012.

EISENMAN, Bonnie. **Learning React Native: Building Native Mobile Apps with JavaScript**. Nova Iorque: O'reilly, 2015. 272 p.

FIGUEIREDO, Douglas Barbonaglia Sathler. **Implementação do Algoritmo Hash SHA-1 em Processador Digital de Sinais**. São José, Pág. 65, 2008.

HANDS ON: GET STARTED WITH INFURA AND THE IPFS ON ETHEREUM. **FreeCodeCamp**, 2018. Disponível em: <<https://www.freecodecamp.org/news/hands-on-get-started-with-infura-and-ipfs-on-ethereum-b63635142af0/>>. Acesso em: 15 de Setembro de 2019.

IFTO, Ead. **Prof Erna - Pregão Eletrônico**. 2014. (28m30s). Disponível em: <<https://youtu.be/IW2fkFqsAx8>>. Acesso em: 05 de Setembro 2019.

LARA, Pedro Carlos da Silva. **Implementação de uma API para Criptografia Assimétrica Baseada em Curvas Elípticas**. Petrópolis, Pág. 94, 2008.

LICITAÇÃO, Aprova. **Pregão Eletrônico: A Base Essencial para Entender Pregão Eletrônico | Marilete Aguilar**. 2015. (7m11s). Disponível em <<https://youtu.be/yp4wwjemcD8>>. Acesso em: 05 de Setembro 2019.

LICITAÇÕES, Magna. **#20 | Fase de Lances Pregão Eletrônica Comprasnet** |. 2018. (17m11s). Disponível em: <<https://youtu.be/PS2plygkAc>>. Acesso em: 05 de Setembro 2019.

PEREIRA, Caio. **Aplicações web real-time com Node.js**. São Paulo: Casa do Código, 2014. 216 p.

Qual a diferença entre as bases de dados centralizadas, descentralizadas e distribuídas? **Modiax**, 2018. Disponível em: <<https://www.modiax.com/descubra/educacao/bases-de-dados-descentralizadas-distribuidas/>>. Acesso em: 25 de Agosto de 2019.

REVISTA DE ADMINISTRAÇÃO PÚBLICA - **RAP: O pregão eletrônico e as contratações de serviços contínuos**. Rio de Janeiro: Revista de Administração Pública, 2013.

REVISTA FÓRUM DE CONTRATAÇÃO E GESTÃO PÚBLICA: **Licitação – Passado, presente e futuro**. Belo Horizonte: Editora Fórum, 2015.

Smart Contracts; Coursera: 2019.

Smart Contracts: The Blockchain Technology That Will Replace Lawyers. **BlockGeeks**, 2019. Disponível em: <<https://blockgeeks.com/guides/smart-contracts/>>. Acesso em: 05 de Setembro de 2019.

SOUZA, Alessandro. **Uma Linguagem Visual para Descrição de Use Cases**. Natal, Pág. 137, 2016.

TAPSCOTT, Don; TAPSCOTT, Alex. **Blockchain Revolution**. Toronto: Senai-sp, 2017. 392 p.

What Are Smart Contracts? **BlockGeeks**, 2016. Disponível em: <<https://blockgeeks.com/guides/smart-contracts/>>. Acesso em: 28 de Agosto de 2019.

What is a Public-key Cryptography? **GlobalSign**, 2017. Disponível em: <<https://www.globalsign.com/en/ssl-information-center/what-is-public-key-cryptography/>>. Acesso em: 26 de Agosto de 2019.

What is a Hashing? **BlockGeeks**, 2017. Disponível em: <<https://blockgeeks.com/guides/what-is-hashing/>>. Acesso em: 28 de Agosto de 2019.

What is Blockchain Technology? A Step-by-Step Guide For Beginners. **BlockGeeks**, 2016. Disponível em: <<https://blockgeeks.com/guides/what-is-blockchain-technology/>>. Acesso em: 15 de Agosto de 2019.