PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ

PRÓ-REITORIA DE PESQUISA, PÓS-GRADUAÇÃO E INOVAÇÃO

COORDENAÇÃO DE INICIAÇÃO CIENTÍFICA

PIBIC

RELATÓRIO FINAL

CONTROLANDO A ENERGIA ELÉTRICA COM BLOCKCHAIN

ALTAIR OLIVO SANTIN

CURITIBA

02/09/2020

DANIEL MARQUES REFISKI
ALTAIR OLIVO SANTIN

CIENCIA DA COMPUTAÇÃO - PUCPR
MODALIDADE – ICV

CONTROLANDO A ENERGIA ELÉTRICA COM BLOCKCHAIN

Relatório Final apresentado ao Programa Institucional de Bolsas de Iniciação Científica, Pró-Reitoria de Pesquisa, Pós-Graduação e Inovação da Pontifícia Universidade Católica do Paraná, sob orientação do(a) **Prof(a). Altair Olivo Santin.**

LISTA DE FIGURAS

Figura 1: Representação do problema Trilema da Escalabilidade.....13
Figura 2: Geração de Fatura.....18
Figura 3: Realização do Pagamento.....17

LISTA DE TABELAS

Tabela 1: Tipos de visualização.....14
Tabela 2: SafeMath.....15
Tabela 3: InterfaceInvoice.....15
Tabela 4: Visibilidade.....16
Tabela 5: Invoice 16
Tabela 6: EnergyRegister.....17

SUMÁRIO

1. INTRODUÇÃO 9
2. OBJETIVOS 9
3. REVISÃO DE LITERATURA 10
4. MATERIAIS E MÉTODO 14
5. RESULTADOS 15
6. DISCUSSÃO 19
7. CONSIDERAÇÕES FINAIS 19
REFERÊNCIAS 20
ANEXO SMART CONTRACTS 1

RESUMO

Introdução: A blockchain é uma tecnologia em ascensão, contendo casos de estudo e aplicabilidade em diversas áreas da sociedade, com a ajuda dos smart contracts podemos criar relacionamentos complexos entre duas ou mais partes sem a necessidade de uma terceira parte verificável. **Objetivos:** o projeto visa desenvolver testes com contratos inteligentes que possam ser inseridas nos mais diversos cenários para o setor elétricos. Sendo um dos possíveis modelos o monitoramento do consumo, auditoria do consumo, smart grids e com isso poderemos verificar e validar sua aplicabilidade. **Materiais e Método:** Para realizar a pesquisa e testes foi usada a blockchain da Ethereum, IDE Remix e linguagem Solidity para escrever smart contracts com o intuito de buscar a melhor alternativa para o que estava sendo proposto na pesquisa. **Resultados:** Levando em consideração o custo do Ethereum atual, percebemos que nos valores atuais, as concessionarias teriam um custo de geração de noventa e sete centavos por fatura gerada**. Considerações Finais:** A blockchain possui real aplicabilidade no setor elétrico, entretanto, para definir se o custo real de uma transação é viável ou muito acima do aceitável, deveríamos levar em conta quanto uma concessionaria possui de despesa com geração de fatura, custo/colaborador, tempo de leitura, e muitas outras variáveis que esta pesquisa não conseguiu obter, pois estes dados não estavam disponíveis publicamente por se tratar de uma informação de competividade empresarial.

1. INTRODUÇÃO

A blockchain é uma tecnologia em ascensão, contendo casos de estudo e aplicabilidade em diversas áreas da sociedade. O termo blockchain foi cunhado pela primeira vez no paper “Bitcoin: A Peer-to-Peer Electronic Cash System” - Satoshi Nakamoto. Existem atualmente três gerações de blockchain. A primeira geração tem como propósito a transferência de um ativo (neste caso o bitcoin) sem a necessidade de uma terceira parte confiável.

A segunda geração é marcada pela utilização de smart contracts, logo seu objetivo é garantir que as ações brevemente acordadas por dois ou mais indivíduos seja honrada, caso contrário será executado uma determinada atividade a fim de resguardar os interesses da parte honesta.

Atualmente estamos na terceira geração. A qual é composta pelas organizações autônomas descentralizadas (DAOs), elas são baseadas em smart contracts com registro de transações e regras do programa mantidas na blockchain, dentre todas, a implementação de organização descentralizada autônoma mais famosa se chamava “The DAO”.

2. OBJETIVOS

Dessa forma o projeto visa desenvolver testes com contratos inteligentes que possam ser inseridas nos mais diversos cenários para o setor elétricos. Sendo um dos possíveis modelos o monitoramento do consumo, auditoria do consumo, smart grids e etc, com isso poderemos verificar e validar sua aplicabilidade. Vemos que a tecnologia pode garantir perfeitamente a integridade das informações via funções hash e criptografia assimétrica.

O projeto tem como objetivo garantir que ambas as partes interessadas no dado gerado pelos monitores possam ser armazenados para que seja evitada fraudes ou adulteração dessas informações.

3. REVISÃO DE LITERATURA

A acessibilidade de uma rede blockchain é amplamente discutida, não há evidências que prove qual dos modelos: Publico / Privado é o ideal. Pois, para responder essa questão, devemos antes analisar o escopo a qual ela está inserida, isso é, cada organização pode implementar da melhor maneira que venha a suprir todas as suas necessidades.

Não há uma regra que defina o padrão ideal de blockchain. Ambos os modelos possuem os seus defeitos e benefícios, se por um lado a modelo publica permite uma maior descentralização dos dados, o modelo privado permite resguardar informações confidenciais e importante para uma determinada organização.

Os Contratos inteligentes combinam protocolos com interfaces de usuário para formalizar e proteger relacionamentos através de redes de computadores (SZABO, 1997), em meados de 2015 a Ethereum se tornou o maior ecossistema distribuído de processamento de smart contracts do mundo, qualquer individuo com conhecimento na sua linguagem de programação Solidity[3] pode criar ou se aproveitar de modelos e especificações já desenvolvidas, com o propósito de resguardar um relacionamento descentralizado entre dois ou mais indivíduos. Isso é possível porque os smart contracts funcionam como a terceira parte confiável, e segue o princípio do manifesto cypherpunk - o código é lei, ou seja, uma vez que o smart contract foi programado e inserido na blockchain o mesmo não poderá mais ser excluído, entretanto, o mesmo pode ser versionado.

A linguagem Solidity suporta os mais diversos cenários, podemos inclusive criar um mecanismo de votação, totalmente sem nenhuma terceira parte confiável de modo a validar os blocos, um código em solidity se assemelha muito com javascript, mas com a desvantagem que ela é limitada a alguns métodos, pois esses códigos são copilados em bytecode e interpretados por uma EVM (Ethereum Virtual Machine).

No mercado de energia elétrica, o mesmo pode ser usado para gerenciar o consumo, emissão de fatura, e ser auditor em tempo real de todas as informações inseridas, em 2019, a Energy Web lançou a Energy Web Chain, a primeira

plataforma de blockchain empresarial de código aberto do mundo adaptado ao setor de energia. A Energy Web Chain é um fork da blockchain da Ethereum, sendo assim suporta smart contract concedendo a desenvolvedores a ferramenta necessária para que seja desenvolvida soluções em cima de sua estrutura, atualmente a Energy Web possui empresas parceiras como SHELL, SPGroup, Exelon e muitas outras concessionarias do ramo elétrico.

As empresas que desejarem criar sua blockchain de modo a se beneficiar dessa solução, primeiramente devem definir algumas perguntas, com o intuito de deixar claro o seu funcionamento, são elas:

Consenso: O consenso é a forma como uma blockchain valida seus blocos, a fim de não tolerar falhas ou manipulação de suas informações, é graças ao mesmo que uma blockchain não permite adulteração, pois uma vez alterada, o consenso irá falhar. Atualmente não existe um consenso certo ou errado, e sim, o consenso que melhor resolve um determinado problema, esse é o fator mais crucial na formação de uma blockchain, pois apenas com ela pode definir se ela será descentralizada, rápida ou segura.

A diversas categorias de consenso com suas vantagens e desvantagens, algumas dos principais consensos são:

POW (Proof of Work): É o primeiro e mais conhecido mecanismo de consenso e foi inventado pelo fundador do Bitcoin, Satoshi Nakamoto. No POW, um minerador que encontrar o hash primeiro terá permissão para adicionar um novo bloco da transação ao blockchain.

POS (Proof of Stack): É baseado na aposta de moeda dos participantes. Quanto mais moedas o usuário tiver, maior será a probabilidade de ele adicionar um novo bloco da transação ao blockchain, devido a isso as validações podem ocorrer mais rapidamente.

DPOS (Delegated Proof of Stake): É uma variação do POS. Com o DPOS, os detentores de moedas podem usar seu saldo para eleger uma lista de nós com permissão para adicionar novos blocos de transações ao blockchain. Os

detentores de moedas também podem votar na alteração do parâmetro de rede. O POS é o sistema quem elege randomicamente, enquanto o DPOS dá a todos os detentores de moedas mais influência e propriedade na rede.

PBFT (Practical Byzantine Fault Tolerance): Tolerância prática a falhas bizantinas (PBFT) foi apresentado por Miguel Castro e Barbara Liskov no Laboratório de Ciência da Computação do MIT em 1999. O PBFT é uma das soluções potenciais para o problema dos generais bizantinos. Com o PBFT, o objetivo é decidir se aceita ou não uma informação enviada ao blockchain. Cada parte ("geral") mantém um estado interno. Quando uma parte recebe uma mensagem, ela usa a mensagem com seu estado interno para executar um cálculo. Esse cálculo levará à decisão dessa parte sobre a mensagem. Em seguida, a parte compartilhará a decisão com todas as outras partes da rede. A decisão final é determinada com base no total de decisões de todas as partes. Um alto hashrate não é necessário neste processo porque PBFT depende do número de nós para confirmar a confiança. Assim que forem alcançadas respostas suficientes, a transação é verificada para ser uma transação válida.

DAG (Directed Acyclic Graph): É um grafo direcionado finito sem ciclos direcionados. É uma estrutura de dados bem conhecida na ciência da computação e frequentemente usada para resolver problemas como encontrar a melhor rota e processar os dados, com o DAG as transações podem ser executadas em diferentes cadeias simultaneamente.

Escalabilidade: Quando queremos construir ou definir qual blockchain usar, devemos também saber se nossa blockchain precisará ou não ser escalável, no caso de blockchains publicas, como Bitcoin e Ethereum o fator escalabilidade é crucial, pois quanto maior o número de usuários, maior também será a demanda por transações, e consequentemente superior será o congestionamento da rede, pois o tamanho do bloco não mudará automaticamente para suprir a nova demanda. Pensando nesse desafio Vitalik Buterin elaborou o Trilema da Escalabilidade, onde o

mesmo define que a escalabilidade de uma blockchain é desproporcional a três pilares fundamentais: Escalabilidade, Segurança e Descentralização.

Neste problema, quando uma rede é muito segura, ou seja, possui uma criptografia forte, ela irá demandar um poder computacional superior, fazendo com que demore mais para ser calculada, sendo assim, será muito mais lenta. Por outro lado, quando uma blockchain é altamente descentralizada ela deverá abdicar de uma rede com uma criptografia pesada, e deverá usar alguma forma de cálculo mais rápido, fazendo com que ela seja menos segura. Por fim se uma blockchain possui alta escalabilidade, ela devera abdicar de segurança e descentralização, fazendo com que a rede seja mais centralizada, sequencialmente será menos segura e descentralizada.

Figura 1: Representação do problema Trilema da Escalabilidade.

Fonte: O autor (2020).

Tipos: Definir qual será o tipo de uma blockchain é extremamente importante, pois com isso podemos definir se uma rede será pública, privada ou hibrida (Pública e Privada), com isso podemos definir como os indivíduos vão interagir com ela.

Tabela 1: Tipos de visualização

Tipo	Criar	Alterar	Ler	Deletar
Pública	Sim	Não	Sim	Não
Privada	Não	Não	Não	Não
Hibrida	Permissionário	Não	Permissionário	Não

Fonte: O autor (2020).

4. MATERIAIS E MÉTODO

Para realizar a pesquisa e testes foi usada duas blockchains, Ethereum e HyperLedger com o intuito de buscar a melhor alternativa para o que estava sendo proposto na pesquisa, sendo assim, optamos inicialmente pela Ethereum.

Uma blockchain é feita de peers ou como é usualmente chamado Nó. Com isso se faz necessário carregar um nó privado local, podendo ou não ser compartilhado na rede. A sua IDE é chamada de Remix e está sendo utilizada para o desenvolvimento dos smart contracts a qual farão partes dos testes de performance.

Para o desenvolvimento desta pesquisa o pesquisador optou por utilizar uma ferramenta chamada Ganache uma ferramenta útil para subir de forma performática uma testnet da rede Ethereum, poderia ser usado o próprio Geth do Ethereum para realizar esse procedimento, mas o pesquisador preferiu usar o Ganache com o intuito de auxiliar futuros pesquisadores a reproduzi-lo com maior assertividade e sem necessitar de muito conhecimento prévio para inicia.

A linguagem de programação utilizada foi a Solidity, pois essa é única linguagem suportada pela blockchain da Ethereum, ela se assemelha muito ao javascript com algumas diferenças sobre os tipos de dados suportados. Essa restrição de tipagem se dá a limitação de processamento da EVM (Ethereum Virtual Machine), pois é ela a responsável pelo processamento dos smart contracts. Após escrever os smart contracts todo o código é copilado pela EVM em Bytecode, no qual se encarregara de processar o smart contract sempre que um método for disparado, neste projeto a versão do compilador usada foi a **0.5.5+commit.47a71e8f.**

5. RESULTADOS

O smart contract foi contruido com o intuito de criar um fluxo de fatura automatizada com base em um smartgrid, onde a concessionaria poderia gerar a fatura com base no consumo do usuário, assim o cliente poderia paga-la sem a necessidade de envio de boletos, ou deslocamento de colaboradores para realizar a leitura.

Abaixo temos quatro smart contracts, com diferentes responsabilidades são eles:

SafeMath: Biblioteca para cálculos aritméticos, a fim de evitar um possível overflow no cálculo.

Tabela 2: SafeMath.

Método	Responsabilidade
mul	Multiplicação de números uint256
div	Divisão de números uint256
sub	Subtração de números uint256
add	Multiplicação de números uint256

Fonte: O autor (2020).

InterfaceInvoice: Este smart contract tem como funcionalidade fornecer uma interface de comunicação, com isso outros smart contracts poderão também se relacionar com nosso contrato o Invoice.

Tabela 3: InterfaceInvoice.

Método	Responsabilidade
subTotal	Fornecer interface para o método subTotal
actualConsumption	Fornecer interface para o método actualConsumption
expiration	Fornecer interface para o método expiration
feeRate	Fornecer interface para o método feeRate
paid	Fornecer interface para o método paid
daysLate	Fornecer interface para o método daysLate

Fonte: O autor (2020).

Para os próximos contratos, precisamos entender outro conceito importante na linguagem solidity ela é a visibilidade, ou seja, quão exporto um método pode ser em relação a outros smart contract ou aplicações externas

Tabela 4: Visibilidade.

Tipo	Descrição
Internal	Essas funções e variáveis de estado só podem ser acessadas internamente (ou seja, de dentro do contrato atual ou contratos derivados dele).
Private	Funções privadas e variáveis de estado são visíveis apenas para o contrato em que são definidas e não em contratos derivados.
Public	As funções públicas fazem parte da interface do contrato e podem ser chamadas internamente ou por meio de mensagens.
External	As funções externas fazem parte da interface do contrato, o que significa que podem ser chamadas a partir de outros contratos e por meio de transações.

Fonte: Documentação linguagem Solidity versão 0.5.5 (2020).

Invoice: A sua responsabilidade como o nome já descreve é definir as funções de leitura, monitoramento e pagamento da fatura do cliente.

Tabela 5: Invoice.

Método	Responsabilidade	Visibilidade
Constructor	Definir as informações básica para a criação do contrato.	Public
actualConsumption	Informar o consumo do cliente.	Public
expiration	Informar o vencimento da fatura em timestamp.	Public
feeRate	Multa por atraso.	Public
paid	Define se a fatura foi paga.	Public
daysLate	Informar os dias em Atraso.	Public

Fonte: O autor (2020).

EnergyRegister: Este é o smart contract que irá fazer toda a gestão dos pagamentos, clientes e faturas.

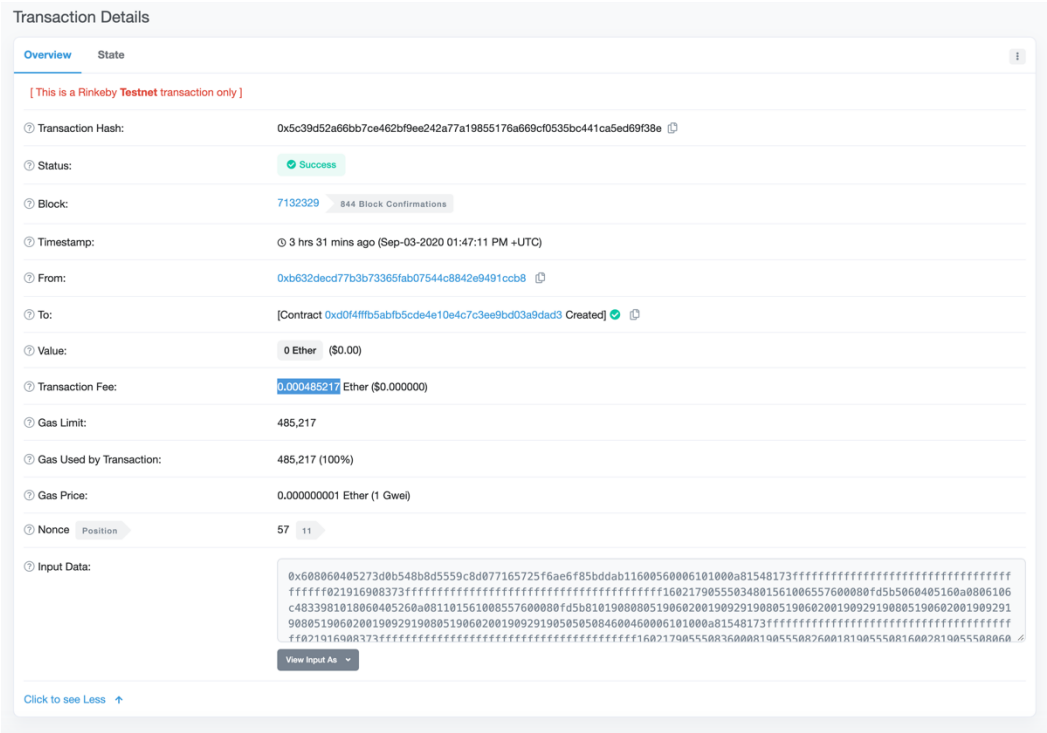
Tabela 6: EnergyRegister.

Método	Responsabilidade	Visibilidade
users	Buscar usuários e faturas.	Public
createInvoce	Criar uma fatura.	External
getInvoice	Buscar todos os dados de uma fatura em específica.	External

Fonte: O autor (2020).

Após todos os smart contracts criados o pesquisador criou algumas transações simulando interações reais com eles a fim de gerar alguns dados sobre custo, performance e aplicabilidade.

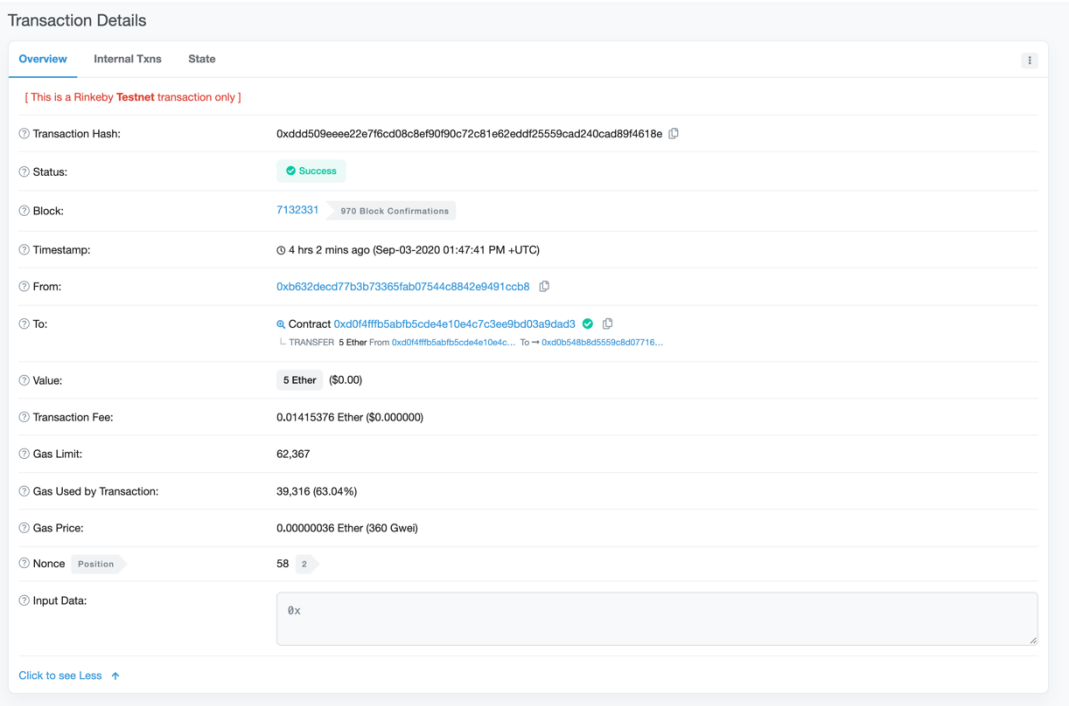
Figura 2: Geração De Fatura.



Fonte:https://rinkeby.etherscan.io/tx/0x5c39d52a66bb7ce462bf9ee242a77a19855176a669cf0535bc441ca5ed69f38e

A transação de pagamento só é possível após a concessionaria criar o invoice para o cliente, então será possível criar uma transação de pagamento. O valor pago por essa fatura foi de cinco ether (moeda da rede ethereum), mas poderia ser qualquer valor que a concessionaria definisse na hora da criação do invoice.

Figura 3: Realização do Pagamento.



Fonte:https://rinkeby.etherscan.io/tx/0xddd509eeee22e7f6cd08c8ef90f90c72c81e62eddf25559cad240cad89f4618e

Levando em consideração o custo do Ethereum atual, percebemos que nos valores atuais, as concessionarias teriam um custo de geração de 0,97 centavos por fatura gerada.

Para se chegar a esse cálculo, seguir a seguinte fórmula: **Fee da Transação** * **Valor Ethereum em Reais** = **Custo Real.**

Todos os códigos utilizados nessa pesquisa estão disponíveis no github do pesquisador em **https://github.com/Dkdaniz/EnergyContract.**

6. DISCUSSÃO

A pesquisa mostra o poder que os smart contract possuem, para realizar o gerenciamento descentralizado do processo e garantindo a integridade e faturamento desse tipo de serviço, permitindo uma confiança em toda a cadeia de pagamento das faturas elétricas, o mesmo caso pode ser aplicado também a geração autônoma de energia limpa, no qual o usuário e a concessionária poderiam chegar a um acordo, qual o valor deveria ser pago e assim a cada data acordada ser gerado uma invoice a concessionaria.

7. CONSIDERAÇÕES FINAIS

Como podemos observar nessa pesquisa que a blockchain possui real aplicabilidade no setor elétrico, entretanto, para definir se o custo real de uma transação é viável ou muito acima do aceitável, deveríamos levar em conta quanto uma concessionaria possui de despesa com geração de fatura, custo/colaborador, tempo de leitura, e muitas outras variáveis que esta pesquisa não conseguiu obter, pois, estes dados não estavam disponíveis publicamente por se tratar de uma informação de competividade empresarial.

REFERÊNCIAS

NAKAMOTO, Satoshi. Satoshi Nakamoto: Whitepaper Bitcoin, 2008. Disponível em: <https://bitcoin.org/bitcoin.pdf > Acesso em: 5 dez. 2019.

SZABO, Nick. Nick Szabo: Smart Contracts: Formalizing and Securing Relationships on Public Networks, 1997. Disponível em: <https://journals.uic.edu/ojs/index.php/fm/article/view/548/469> Acesso em: 5 dez. 2019.

HONIGMAN, Philippe. Philippe Honigman: What is a DAO?, 2019. Disponível em: <https://hackernoon.com/what-is-a-dao-c7e84aa1bd69> Acesso em: 5 dez. 2019.

DUPONT, Quinn. Quinn DuPont: Experiments in Algorithmic Governance: A history and ethnography of “The DAO,” a failed Decentralized Autonomous Organization, 2017. Disponível em: <https://web.archive.org/web/20170730133911/http://iqdupont.com/assets/documents/DUPONT-2017-Preprint-Algorithmic-Governance.pdf> Acesso em: 5 dez. 2019.

BEZERRA, R. R. M.. Rômulo Rodrigues de Morais Bezerra: Gerenciamento de uma rede de computadores em um ambiente corporativo utilizando o software zabbix. Disponível em: <https://semanaacademica.org.br/system/files/artigos/artigo_gerencia.pdf> Acesso em: 5 dez. 2019.

BARAN, Paul. Paul Baran: On Distributed Communications, 1964. Disponível em: <http://pages.cs.wisc.edu/~suman/courses/740/papers/baran.pdf> Acesso em: 5 dez. 2019.

Website Humana: O que você precisa saber sobre rede. Disponível em: <http://humana.social/o-que-voce-precisa-saber-sobre-redes/> Acesso em: 5 dez. 2019.

Website Nós Digitais. Disponível em: <http://wiki.nosdigitais.teia.org.br/Modelos_de_Rede> Acesso em: 5 dez. 2019.

GUEGAN, Dominique. Dominique Guegan: Public Blockchain versus Private Blockchain, 2008. Disponível em: <https://halshs.archives-ouvertes.fr/halshs-01524440/document> Acesso em: 5 dez. 2019.

Website logicsolutions: 5 Types of Blockchain Consensus Mechanisms Disponível em: < https://www.logicsolutions.com/5-types-blockchain-consensus-mechanisms/> Acesso em: 5 dez. 2019.

Website Solidity: Language documentation Disponível em: < https://solidity.readthedocs.io/en/v0.5.5/types.html/> Acesso em: 5 dez. 2019.

HUGHES, Eric. Eric Hughes: A Cypherpunk's Manifesto, 1993. Disponível em: <https://www.activism.net/cypherpunk/manifesto.html> Acesso em: 5 dez. 2019.

ETHEREUM. Ethereum Foundation: Solidity, 2019. Disponível em: <https://solidity.readthedocs.io/en/v0.5.12/> Acesso em: 5 dez. 2019.

ZAINUDDIN, Aziz. Aziz Zainuddin: Blockchain Public vs Private, 2019. Disponível em: <https://masterthecrypto.com/public-vs-private-blockchain-whats-the-difference/> Acesso em: 5 dez. 2019.

ENERDATA. Enerdata: Electricity domestic consumption, 2018. Disponível em: <https://yearbook.enerdata.net/electricity/electricity-domestic-consumption-data.html> Acesso em: 5 dez. 2019.

THOMAS, Stefan. Stefan Thomas: Crypto Conditions, 2018. Disponível em: <https://tools.ietf.org/html/draft-thomas-crypto-conditions-04> Acesso em: 5 dez. 2019.

Dong Ku David Im: The Blockchain Trilemma, 2018. Disponível em: <https://www.davidim.info/docs/blockchain_trilemma.pdf> Acesso em: 5 dez. 2019.

VISWANATHAN, Surya. Surya Viswanathan: The Scalability Trilemma in Blockchain, 2018. Disponível em: <https://medium.com/@aakash_13214/the-scalability-trilemma-in-blockchain-75fb57f646df> Ac esso em: 5 dez. 2019.

ANEXO SMART CONTRACTS

```
pragma solidity >=0.4.16 <0.7.0;

library SafeMath {
  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a * b;
    assert(a == 0 || c / a == b);
    return c;
  }

  function div(uint256 a, uint256 b) internal pure returns (uint256) {
    // assert(b > 0); // Solidity automatically throws when dividing by 0
    uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold
    return c;
  }

  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    assert(b <= a);
    return a - b;
  }

  function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    assert(c >= a);
    return c;
  }
}



contract InterfaceInvoice {
    function subTotal() external view returns (uint256);
    function actualConsumption() external view returns (uint256);
```

```
    function expiration() external view returns (uint256);
    function feeRate() external view returns (uint256);
    function paid() external view returns (bool);

    function daysLate() public view returns (uint256);
}

contract Invoice is InterfaceInvoice {
    using SafeMath for uint256;

    uint256 public subTotal;
    uint256 public actualConsumption;
    uint256 public expiration;
    uint256 public feeRate;
    address public user;
    address payable public energyCompany =
address(0xD0b548B8d5559c8D077165725f6Ae6F85bddab11);
    bool public paid;
    uint256 public totalizado;

    event paidInvoice(address indexed user, uint256 value);

    constructor (address _user, uint256 _total, uint256 _actualConsumption, uint256 _expiration, uint256
_feeRate) public {
        user = _user;
        subTotal = _total;
        actualConsumption = _actualConsumption;
        expiration = _expiration;
        feeRate = _feeRate;
        paid = false;
    }

    function daysLate() public view returns (uint256) {
        uint256 dayLate = 0;
        if(now > expiration){
```

```
            dayLate = now.sub(expiration).div(60).div(60).div(24);
        }
        return dayLate;
    }

    function lateFee() public view returns (uint256){
        uint256 fee = 0;
        if(now > expiration){
            uint256 dayLate = daysLate();
            fee = dayLate.mul(feeRate);
        }
        return fee;
    }

    function() payable external {
        require(msg.value > 0, 'value dont can be zero!');
        uint256 fee = lateFee();
        subTotal = subTotal.add(fee);

        energyCompany.transfer(msg.value);
        paid = true;
    }
}

contract EnergyRegister {

    mapping(address => address[]) public users;

    function createInvoce (address _user, uint256 _total, uint256 _actualConsumption, uint256 _expiration,
uint256 _feeRate) external returns(bool) {
        address newInvoice = address(new Invoice(_user, _total, _actualConsumption, _expiration,
_feeRate));
        address[] storage invoices = users[_user];
        invoices.push(newInvoice);
        users[_user] = invoices;
```

```
    }

    function getInvoice(address _invoice) external view returns ( uint256, uint256, uint256, uint256,
uint256, bool){
        InterfaceInvoice invoice = InterfaceInvoice(_invoice);
        uint256 total = invoice.subTotal();
        uint256 actualConsumption = invoice.actualConsumption();
        uint256 expiration = invoice.expiration();
        uint256 feeRate = invoice.feeRate();
        uint256 daysLate = invoice.daysLate();
        bool paid = invoice.paid();

        return (total,actualConsumption ,expiration, daysLate,feeRate, paid );
    }
}
```