ESTUDO DE APLICAÇÃO DE BLOCKCHAIN EM PROCESSOS INDUSTRIAIS

Marcos Vinicio da Silva Oliveira[1]

Vitor Natã Zanetta Santos[1]

Fabio Andrijauskas[2]

vitor_zanetta@hotmail.com; marcos.vinicio.silva@hotmail.com

[1]Alunos do Curso de Engenharia da Computação, Universidade São Francisco; Campus Itatiba

[2]Professor Orientador do Curso de Engenharia de Computação, Universidade São Francisco; Campus Itatiba.

Resumo. Com a repercussão em torno do Bitcoin, a arquitetura da criptomoeda chamada Blockchain também recebeu uma atenção promissora fora do setor financeiro. Com a finalidade de trazer segurança em suas transações, esse estudo mostra que essa tecnologia é útil para que o processo de fabricação de um produto tenha uma maior confiabilidade. Durante esse processo podem ocorrer diversas falhas devido ao corrompimento de informações, o Blockchain no ambiente industrial se torna uma possibilidade de principalmente de garantir confiabilidade e segurança nas informações durante todo o processo de fabricação. Esse artigo apresenta um sistema desenvolvido para a indústria utilizando a linguagem Javascript e para obtenção dos resultados foram simulados alguns processos. Em geral, o sistema apresentou desempenho em segurança e na confiabilidade dos dados, porém a sua utilização em ambientes de larga escala podem ocorrer problemas por necessidade do poder de processamento impactando no seu desempenho.

Palavras-chave: Blockchain, hash, criptografia, processos industriais

Abstract. With the repercussions around Bitcoin, a cryptocurrency architecture called Blockchain should also bring promising attention to the financial sector. With an improvement in bringing security to your transactions, this study shows that this technology is useful for a product's manufacturing process to have greater reliability. During this process, several failures can occur due to information corruption, the Blockchain in the industrial environment becomes a possibility to guarantee information security and security during the entire manufacturing process. This article presents a system developed for the industry using a Javascript language and to obtain the results some processes were simulated. In general, the system presents performance in security and data reliability, but its use in large-scale environments may occur due to the need for processing power, impacting its performance.

Key-words: Blockchain, hash, cryptography, process

1. Introdução

Incorporado por um ou mais indivíduos sobre o pseudônimo Satoshi Nakamoto, a criptomoeda Bitcoin e o conceito de Blockchain criou um grande engajamento em torno de sistemas de pagamento eletrônico usando um sistema ponto a ponto. Através do conceito de Blockchain, a criptomoeda Bitcoin fornece a infraestrutura que permite uma segurança direta em troca de valor entre participantes sem qualquer intermediário financeiro, a ideia era criar uma moeda virtual totalmente independente do sistema monetário (Nakamoto, 2008). O Blockchain é uma espécie de livro razão compartilhado em uma rede que permite armazenamento imutável de dados de transações verificados, portanto essa arquitetura representa confiabilidade em todas as transações (Al-Jaroodi, Mohamed, 2019). A tecnologia Blockchain é utilizada atualmente em áreas financeiras de criptomoedas, porém dado os benefícios de seu uso, como imutabilidade, transparência e rastreabilidade, novas aplicações estão sendo estudadas.

O Blockchain no ambiente industrial se torna uma possibilidade de principalmente de garantir confiabilidade e segurança nas informações durante todo o processo de fabricação, uma vez que o ambiente tradicional não tenha facilidade com que as informações sejam utilizadas mesmo ela sendo corrompidas devido à alteração de dados, na rede Blockchain quando um dado é modificado todo o processo é corrompido, sendo necessário a revisão de todas as etapas para ser válido novamente. Embora a tecnologia Blockchain possua confiabilidade, algumas redes onde essa tecnologia é aplicada podem haver recursos insuficientes para sua implementação.

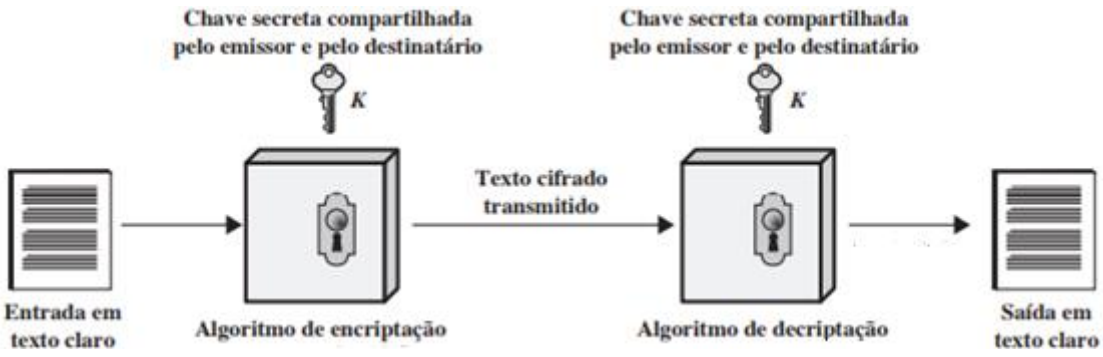
O objetivo do trabalho é planejar a aplicação da rede Blockchain para validação de processos de produção de produtos onde eles passam por etapas de modelagem, testes, embalagem e transporte, com isso cada etapa atribuir dados a peça para garantir que ela foi realizada e pode ser enviada a próxima etapa. Utilizando softwares e gráficos para simular a implementação da infraestrutura Blockchain, o estudo será estruturado da seguinte forma: Embasamento teórico para compreensão sobre a criptografia das informações e conceitos, simulação da aplicação Blockchain utilizando um software desenvolvimento para inserção e processamento dos dados para posteriormente ser realizado o armazenamento das informações no banco de dados, definições de métricas básicas para o funcionamento, utilização de software para avaliar o desempenho e resultados que serão apresentados as conclusões sobre o funcionamento da aplicação Blockchain observando seu processamento e a segurança das informações.

2. Referencial Teórico

A arquitetura de rede ponto-a-ponto (ou peer-to-peer) é um tipo de arquitetura diferente das redes populares, onde existe a necessidade de um cliente se conectar a um servidor para se comunicar com os demais. A principal característica é devido ao fato de ela não ser hierárquica, não necessitando de um servidor centralizado para que os clientes se comuniquem entre si, tornando-os cliente com o papel também de servidor fazendo que a rede seja descentralizada (Tanembaum, 2003). Uma rede onde não existe uma organização centralizada se torna vantajosa em ocasiões de falhas de processamento e armazenamento, visto que os demais nós da rede continuam a operar quando um ocorre problemas. Também apresenta um maior nível de escalabilidade, diferentemente de um modelo convencional cliente/servidor o servidor pode ficar sobrecarregado se muitos clientes trabalham simultaneamente, enquanto nos sistemas ponto a ponto à medida que o número de clientes aumenta, a rede tende a ter mais performance visto que eles também podem fazer o papel de servidor (Kamienski, 2005; Souto, 2005; et al. 2005).

A criptografia simétrica, ou encriptação de chave única, é o modelo mais comum e simples utilizado. A mensagem original é conhecida por texto claro e a mensagem codificada é conhecida como texto cifrado. O processo de conversão do texto claro em um texto cifrado se chama criptografia e o inverso ou ajuste chama-se decriptação (Stallings, 2015). O método de encriptação desse modelo ocorre através de uma chave secreta e um algoritmo de criptografia, somente com essa mesma chave a decriptação ocorre transformando esses dados criptografados. A segurança dessa criptografia pode variar de acordo com o tamanho e tipo do algoritmo da chave. A Figura 01 ilustra um processo onde ocorre a criptografia simétrica de uma informação passando pelos processos de encriptação e decriptação utilizando uma chave secreta compartilhada.

Figura 01: Criptografia Simétrica



Fonte: Adaptado de Stallings (2015)

No Blockchain é utilizada a função Hash para conversão de arquivos de tamanhos variados para uma palavra (chave) de tamanho único criptografada semelhante ao token onde se tem como característica a extrema dificuldade de decodificação da palavra gerada sem acesso ao arquivo original

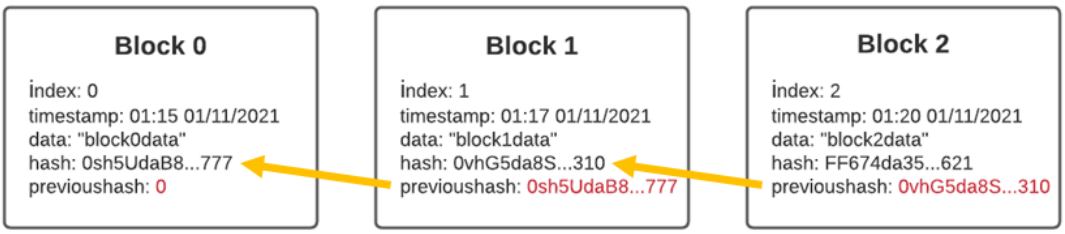
convertido. Uma outra importante característica é que dois arquivos distintos são extremamente difíceis a conversão para mesma palavra com o objetivo de evitar que dados sejam duplicados e enviados incorretamente pela função. Atualmente a função hash mais utilizada é a Secure Hash Algorithm (SHA) (Stallings, 2015).

Token é uma chave criptográfica que tem diversas aplicações com o objetivo de compartilhar dados de uma forma segura, existem alguns tipos de token como o “security token” (token de seguridade), normalmente esse tipo de token é utilizado em transações bancárias, também existe o “utility tokens”(tokens utilitários), que tem a funcionalidade de garantir o acesso à determinado serviço ou sistema (HOUBEN; SNYERS, 2018). Uma forma de verificar a sua autenticidade é realizando um comparativo entre o código armazenado e o código informado no momento do acesso pelo usuário.

A Árvore de Merkle é um tipo de estrutura de dados utilizada para verificação da integridade de grande quantidade de informações em sistemas distribuídos, ou seja, em sistemas descentralizados. Elas são árvores binárias e otimizam o uso das funções hash. Na ciência da computação, o termo "árvore" é utilizado para referir uma estrutura de dados ramificada, entretanto como nas árvores binárias essas árvores são exibidas de forma contrária, onde o nó "raiz" está estabelecido no topo de um diagrama e as "folhas" na parte de baixo sendo que cada um guarda uma informação e é interligado com até 2 nós chamados filhos (Antonopoulos, 2014). A árvore Merkle é construída de baixo para cima, onde cada nó armazena a sua função hash onde os nós superiores armazenam o hash da combinação dos nós inferiores repetindo esse processo até resultar na obtenção do nó raiz.

Em um sistema Blockchain o processo se divide em blocos que possui uma estrutura como: identificador do bloco, data e hora atual, dado que será transmitido, hash anterior e o hash atual. A criptografia baseada no identificador, data e hora atual e o dado transmitido é considerada a segurança pois dificulta que blocos diferentes façam a geração do mesmo hash. Em cenários onde a cadeia seja inutilizada devido a alguma alteração no dado transmitido, todos os blocos devem ser verificados e gerados novos hash. A figura 02 mostra esse modelo, uma estrutura Blockchain se refere a esse processo de verificação como mineração do bloco.

Figura 02: Estrutura do bloco



Fonte: Próprios autores (2021)

Trabalhos relacionados

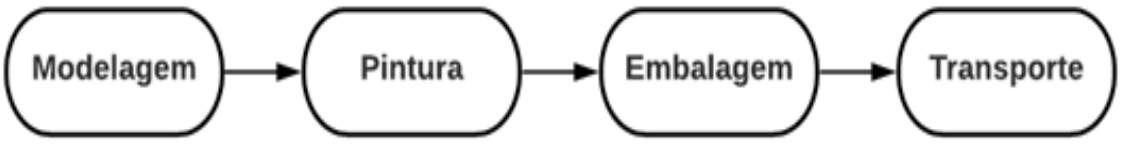
A avaliação de desempenho utilizando o programa Jmeter será baseado no artigo de Anderson e Gustavo (2020) tendo algumas diferenças como a não utilização do programa Docker como sistema de arquivos e do envio de arquivo JSON via POST para a rede Blockchain, enquanto que o desenvolvimento da rede será baseado na conjuntos dos tutoriais de CELKE (2019) para envio e recebimento de informações utilizando linguagem Javascript e o código de Savjee (2021) para criar a aplicação de processamento Blockchain em Javascript.

3. Metodologia

A implementação de uma aplicação Blockchain em um ambiente industrial de produção, deve ocorrer em ambientes que possuem processos com etapas e que necessitam de uma segurança em toda a sua operação. Para realização deste estudo foi definido inicialmente uma pesquisa sobre o tema e também os processos e a estruturação da aplicação Blockchain, após isso é definido ferramentas e linguagens para a implementação do projeto. Com a implementação realizada foram definidas as métricas e ferramentas para avaliação de desempenho da aplicação e na etapa final, analisadas as informações coletadas nos testes para obter os resultados.

Foi realizada uma aplicação Blockchain para verificação a fim de analisar o seu comportamento simulando um processo de produção e definir ferramentas de benchmarks para realizar a avaliação de desempenho durante a execução desse processo. Cada etapa do processo será definida em um bloco na rede que se conectará no bloco seguinte com outra etapa atribuída ao mesmo e assim sucessivamente até a finalização de todo o processo. Na finalização do processo a rede garantirá que nenhum dado seja alterado após a finalização. Caso seja necessário a alteração de uma informação em algumas das etapas, todas as etapas devem ser revisadas para conclusão do processo. A figura 03 ilustra em representação diagramática as etapas propostas na metodologia deste trabalho.

Figura 03: Etapas utilizadas para avaliação do Blockchain



Fonte: Próprios Autores (2021)

A aplicação foi desenvolvida em Javascript utilizando biblioteca NodeJS no software Visual Studio Code para simulação de rede Blockchain local, com uma quantidade de nós pré-definida. Ao ser

executada cria o bloco gênesis da cadeia e instancia os outros nós da Blockchain, que armazena uma cópia dos registros da cadeia de blocos. Após criados, os dados serão registrados no banco de dados Mongodb com a ferramenta Mongodb Compass para consulta de dados armazenados. Para entrada e alteração de dados será utilizado o link localhost:3000/ acessado por um navegador para cadastrar, consultar e alterar as informações.

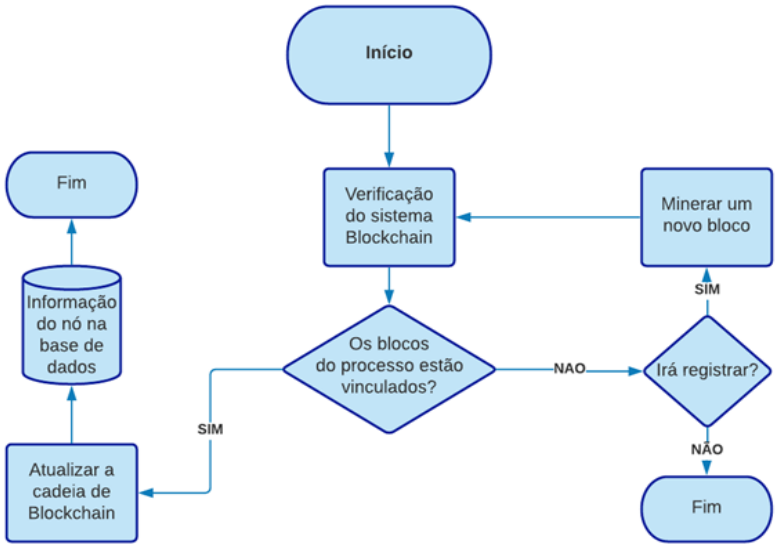
A Figura 04 ilustra o esquema da aplicação, onde será desenvolvido uma interface para entrar com os dados, a aplicação Blockchain e o banco de dados para registros levando em consideração de que o software desenvolvido simula os nós.

Figura 04: Representação esquemática da aplicação

Fonte: Próprios Autores (2021)

Para a inserção de dados, a aplicação desenvolvida irá realizar o processo ilustrado no fluxograma da Figura 05, onde os dados inseridos irão ser processados pelo sistema Blockchain verificando se os blocos estão vinculados. Se não o sistema irá minerar um novo bloco, se sim a cadeia Blockchain irá ser atualizada registrando os dados no banco de dados.

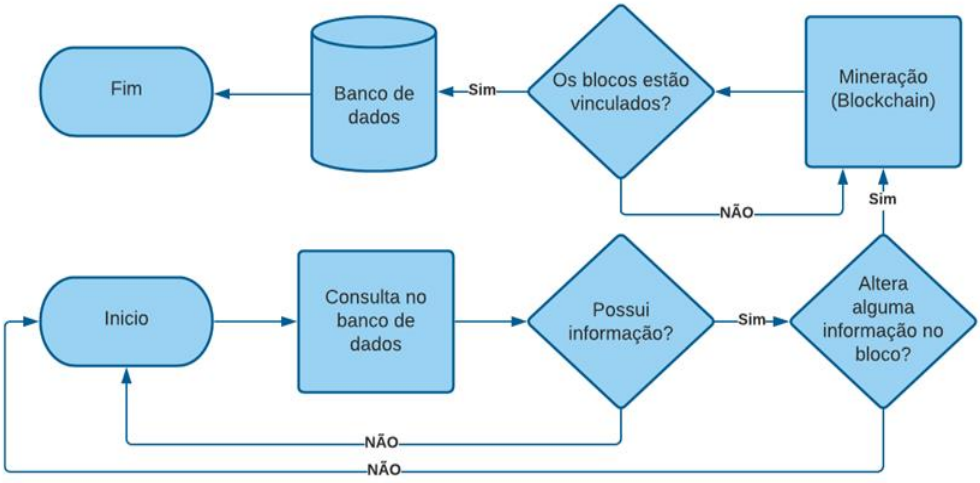
Figura 05: Fluxograma de inserção de dados



Fonte: Próprios Autores (2021)

Para a alteração de dados a aplicação desenvolvida irá realizar o processo ilustrado no fluxograma da Figura 06, que diferente do fluxo anterior o sistema primeiramente verificará no banco de dados a informação a ser alterada.

Figura 06: Fluxograma de alteração de dados

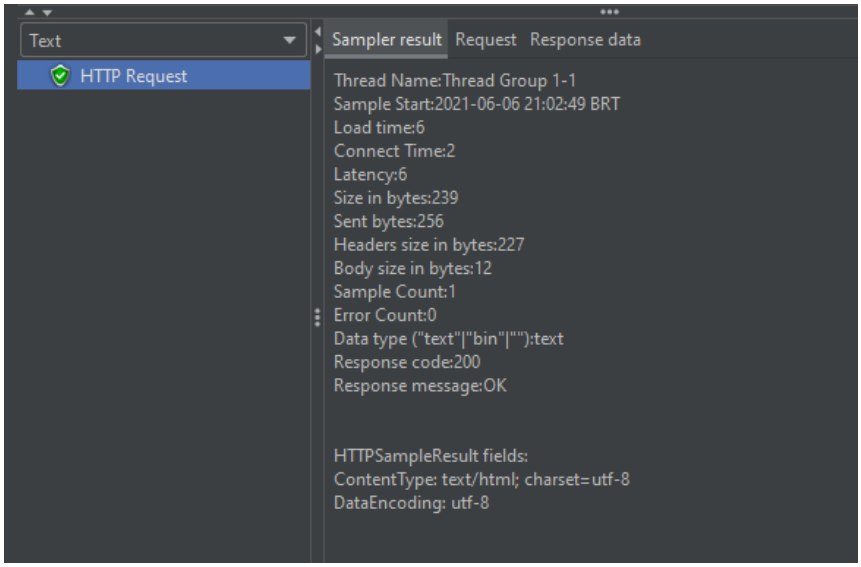


Fonte: Próprios Autores (2021)

Para avaliar o desempenho da rede Blockchain foram definidas algumas métricas, tais como o tempo de resposta para uma requisição, tempo de espera e processamento da resposta. Algumas situações foram realizadas para avaliar e testar o desempenho da rede utilizado um computador com processador intel i5-8265 U 1.80GHz com 8GB de memória RAM de sistema operacional Windows 10 apenas executando tarefas padrões e a aplicação Blockchain desenvolvida para um processo de 4 nós (bloco) para acesso simultâneo a 10, 20, 50 e 100 requisições. Para avaliar a segurança da informação é definido que o processo precisa manter todos os blocos encadeados (conectados) e que a geração de cada hash de cada bloco seja único em toda rede evitando duplicidade de informação.

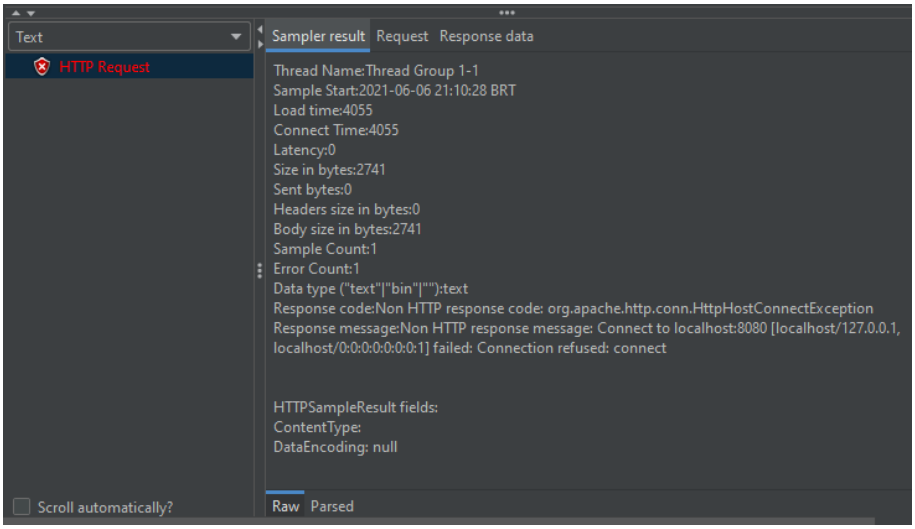
Na verificação do desempenho da aplicação implementada foi utilizada a ferramenta de monitoramento Jmeter. Essa ferramenta serve para testar o desempenho em recursos estáticos e dinâmicos. Ela pode ser usada para simular uma carga pesada em um servidor, grupo de servidores, rede ou objeto para testar sua força ou para analisar o desempenho geral sob diferentes tipos de carga realizando um POST (informações enviadas através de uma url e com os dados a serem processado como parâmetros) enviado para o servidor onde a aplicação Javascript realiza a captura dos dados e com isso realiza o processamento. Uma vez realizado o POST a ferramenta Jmeter recebe a informação que se a requisição foi aceita (Figura 07) ou recusada pelo servidor (Figura 08).

Figura 07 - Requisição aceita pelo servidor da aplicação



Fonte: Próprios Autores (2021)

Figura 08 - Requisição recusada pelo servidor da aplicação



Fonte: Próprios Autores (2021)

A fim de finalizar o estudo, foi realizada a etapa de análise de resultados que serão apresentadas no tópico seguinte referente as métricas obtidas na simulação realizada com as requisições da carga do teste, com o objetivo de obter conclusões sobre a implementação do Blockchain.

4. Resultados e Discussões

Apresentando o código desenvolvido para implementação, ele é dividido em 6 arquivos principais, considerando 3 arquivos da linguagem EJS junto com HTML para visualizar as informações (index.ejs, edita.ejs e show.ejs) e 3 arquivos de linguagem Javascript para tratamento das informações (server.js,mongo.js e main.js) para facilitar o entendimento e otimizar a inicialização da aplicação. Na página server é considerado o código fonte principal, uma vez que ele é responsável por inicializar a aplicação e obter informações das outras páginas na Figura 09 contendo as bibliotecas, configurações iniciais e constantes.Entre as linhas 14 à 19 é utilizado para definir o localhost para as páginas Index e show quando realizar uma requisição GET para chamar as funções conforme Figura 10.

Figura 09 - Bibliotecas, constantes e renderizações iniciais da página Server

```
1  const express = require('express') // Constraint para aplicações Get e Post
2  const bodyParser = require('body-parser') // Constraint para capturar html
3  const sha256 = require('crypto-js/sha256'); // Contraint criptografica SHA256
4  const {Blockchain,Block}=require('./Blockchain/main')//estruturar o bloco
5  const app = express()
6  var db = require('./dbmongo'); // Obter os parâmetros do banco de dados
7  const produto=db.Mongoose.model('produto', db.Produto, 'produto');//Esquema para
tratar  os     dados
8  // Objeto do arquivo main do Blockchain
9 const BLOCKCOIN = new Blockchain();
10 app.use(bodyParser.urlencoded({ extended: true})) // capturar o corpo html
11 app.set('view engine', 'ejs') // Configuração do ambiente para linguagem js
14 // renderização do arquivo index no localhost:3000/index
15 app.get('/index', (req, res) =>{
16 res.render('index.ejs') })
17 //renderização do arquivo show no localhost:3000/show
18 app.get('/show', function(req, res, next) {
19 res.render('show', { results: false });   });
```

Fonte: Próprios Autores (2021)

Após a renderização das páginas Index e Show é realizado nas linhas 23 à 43 (Figura 10) uma verificação se os dados estão corrompidos no banco de dados quando for feita uma requisição GET na página **localhost:3000/search** que é executada ao pressionar o botão procurar na página **show**.

Figura 10 - Consulta do banco de dados da página server

```
23 //Get na página localhost:3000/search é realizado a renderização da página Show
24 app.get('/search', function (req, res, next) {
25 //captura do dado presente na form da pagina show
26 var searchParams = req.query.query.toUpperCase().split(' ');
27 //realizando uma consulta no banco no campo tamanho
28 produto.find({ tamanho: { $all: searchParams } }, function (e, docs) {
29 docs.forEach(function(teste){
30 //comparando dado banco com dado minerado novamente com base na hora de criação
31 BLOCKCOIN.isempty();
32 BLOCKCOIN.addBlock(new Block(1, teste.datacriacao, {amount: teste.tamanho}));
33 BLOCKCOIN.addBlock(new Block(2, teste.datacriacao, {amount: teste.cor}));
34 BLOCKCOIN.addBlock(new Block(3, teste.datacriacao, {amount: teste.material}));
35 BLOCKCOIN.addBlock(new Block(4, teste.datacriacao, {amount: teste.veiculo}));
 comparando hash gerado acima com o hash salvo no banco,se igual não houve modific.
  se diferente houve modificação na cor , o dado no banco será corrompido.
37 if(teste.hash2==BLOCKCOIN.chain[2].hash){ console.log('dado correto'); }
38 else { console.log('dado cor divergente');
39 console.log(teste.hash2);console.log(BLOCKCOIN.chain[2].hash);
40 // console.log(JSON.stringify(BLOCKCOIN, null, 4));
41 produto.updateOne({"_id" : teste._id},
42 {$set: {"cor" : "Dado corrompido"}},
43 function( err, result ) { if (err) {res.send(err); } else { } }); }
```

Fonte: Próprios Autores (2021)

Após o comparativo dos blocos e alterações no banco caso necessário, é realizada uma consulta pelo tamanho do produto e renderizado na Página Show conforme mostra nas linhas 65 a 66(Figura 11) e finalizando a renderização o sistema preenche os campos da página Edita com os dados selecionados pelo usuário na Página Show conforme mostra nas linhas 67 a 71(Figura 11).

Figura 11 - Consulta e renderização dos dados da página server

```
64//consulta o tamanho no banco e renderiza os dados com o tamanho procurado
65 produto.find({ tamanho: { $all: searchParams } }, function (e, docs) {
66 res.render('show', { results: true, search: req.query.query, list: docs });});});
67 //monitora o get na /edita/_id e renderiza a procura na pagina edita
68 app.get('/edita/:_id', function (req, res, next) {
69 var id = req.params._id
70 produto.find({ _id : { $all: id } }, function (e, docs) {
71 res.render('edita', { search: id, list: docs }); }); });
```

Fonte: Próprios Autores (2021)

Após as renderizações,o código realiza a monitoração das requisições POST no **localhost:3000/cad-realizado** para salvar os dados no Banco MongoDB(Figura 12). Assim como é definido na etapa Moldagem entre as linhas 97 à 99 , os parâmetros para criação do bloco, é realizado semelhante para a criação dos blocos da pintura, embalagem e transporte.

Figura 12 - Código para definir os parâmetros do bloco da página server

```
80 // monitora o post da página /cad-realizado
81 app.post('/cad-realizado', (req, res) => {
82 // Função para obtenção da hora que será constituída no timestamp do bloco
83 let date_ob = new Date(); let date = ("0" + date_ob.getDate()).slice(-2);
84 let month=("0"+(date_ob.getMonth()+1)).slice(-2);let year=date_ob.getFullYear();
85 let hours = date_ob.getHours(); let minutes = date_ob.getMinutes();
86 let seconds = date_ob.getSeconds(); let mili = date_ob.getMilliseconds();
87 var hora = year + "-" + month + "-" + date + " " + hours + ":" + minutes + ":" +
88 seconds + ":" + mili; var i = 0;
89 BLOCKCOIN.isempty(); // Limpa os dados do vetor
90 BLOCKCOIN.addBlock(new Block(1, hora, {amount: req.body.nomemo}));//Adicionando as
                                variáveis do bloco
92 BLOCKCOIN.addBlock(new Block(2, hora, {amount: req.body.nomepi}));
93 BLOCKCOIN.addBlock(new Block(3, hora, {amount: req.body.nomeem}));
94 BLOCKCOIN.addBlock(new Block(4, hora, {amount: req.body.nometr}));
95 console.log(JSON.stringify(BLOCKCOIN, null, 4));
96 // captura de dados inseridos na pagina para envio no banco
97 const etapa1 = "Moldagem"; const tamanho = req.body.nomemo;
98 const phash1 = BLOCKCOIN.chain[i+1].previoushash; // hash do bloco anterior
99 const hash1 = BLOCKCOIN.chain[i+1].hash; // captura do hash
```

Fonte: Próprios Autores (2021)

Após a definição dos parâmetros dos blocos, é utilizado uma função presente na linha 123 para salvar os dados no banco de dados MongoDB, observado na Figura 13.

Figura 13 - Função para salvar os dados no banco da página server

```
118 // estrutura para inserção de dados no banco
119 const produ = new produto({datacriacao, etapa1, tamanho,
120 phash1, hash1, etapa2, cor, phash2,
121 hash2, etapa3, material, phash3, hash3, etapa4,veiculo,phash4,hash4 });
122 // tratativa de para salvar e direcionar para a página /show
123 try { produ.save(); res.redirect("/show"); } catch (err) {next(err);}})
```

Fonte: Próprios Autores (2021)

Semelhante a Função para salvar as informações , a Função para atualizar os dados possui a mesma estrutura para obter os dados se diferenciando no local que monitora requisição POST localizado em **localhost:3000/cad-alterado** e o método para atualizar os dados conforme (Figura 14) entre as linhas 158 a 163.

Figura 14 - Função para atualizar os dados da página server

```
158 produto.updateOne({"_id" : id},
159 {$set: {"cor" : cor, "material" : material,
160 "veiculo" : veiculo, "tamanho" : tamanho,
160 "phash1" : phash1 ,"phash2" : phash2 ,"phash3" : phash3 ,"phash4"
 phash4,"hash1" : hash1 ,"hash2" : hash2 ,
161 "hash3" : hash3 , "hash4" : hash4, "datacriacao" : hora}},
162 function( err, result ) { if (err) { res.send(err); } else {
 res.redirect("/show"); } }); })
```

Fonte: Próprios Autores (2021)

Com a finalização do código presente na página Server. O programa precisa estruturar os direcionamentos dos dados para os locais monitorados conforme página Index na Figura 15 no formulário que direciona as informações nas linhas 1 a 7.

Figura 15 - Formulário para salvar os dados da página Index

```
1 <h1>Formulário para Cadastrar Produto</h1>
2 <form action="/cad-realizado" method="POST">
3 <label>Moldagem</label> <input type="text" name="nomemo"><br><br>
4 <label>Pintura</label> <input type="text" name="nomepi"><br><br>
5 <label>Embalagem</label> <input type="text" name="nomeem"><br><br>
6 <label>Transporte</label>10 <input type="text" name="nometr"><br><br>
7 <button type="submit">Cadastrar</button> </form>
```

Fonte: Próprios Autores (2021)

Assim como na página Index, a página Show também realiza redirecionamento para locais monitorados observado nas linhas 1 a 6 para realizar a consulta e finalizando a página exibindo as informações consultadas conforme Figura 16.

Figura 16 - Página Show

```
1 <h1> Consulta de Produtos </h1>
2 <form action="/search" method="get"> <input type="text" name="query" />
4 <input type="submit" value="procurar" /><a href="/index">Novo Produto</a>
6 </form>
7 <% if(results){ if(list.length > 0){ %>
8 <b>Resultados da pesquisa por <%= search %>:</b> <ul>
10 <% list.forEach(function(item){ %>
11 <li> <a href="/edita/<%=item._id%>">Editar</a></li>
13 <li>Moldagem :<%= item.tamanho%> </li> <li>Pintura :<%= item.cor%> </li>
 15 <li>Embalagem :<%= item.material%> </li><li>Transporte:<%= item.veiculo%>
        </li><br><br> <% }); %> </ul> <% } else{ %>
20 <b>Nenhum resultado encontrado para <%= search %>!</b>
21 <% } } %> </center></body></html>
```

Fonte: Próprios Autores (2021)

Semelhante a página Index ,na próxima etapa do código realiza-se o redirecionamento para o local monitorado na página Edita diferenciando por exibir os dados consultados conforme mostra a Figura 17 entre as linhas 1 à 8.

Figura 17 - Página Edita

```
1 <b>Resultados da pesquisa por <%= search %>:</b>
2 <ul> <% list.forEach(function(item){ %>
3 <h1>Formulário para alterar Produto</h1>
4 <form action="/cad-alterado" method="POST">
5 <label>ID</label>
6 <input type="text" name= "nomeid" value="<%= search %>" readonly><br><br>
7 <label>Moldagem</label>
8 <input type="text" name="nomemo" value="<%= item.tamanho %>"><br><br>
```

Fonte: Próprios Autores (2021)

Finalizado o redirecionamento para os locais monitorados. O código abaixo realiza a conexão com o banco utilizando uma estrutura que possui o nome e o tipo do dado que deve ser inserido conforme as linhas 1 à 9 da página MongoDB (Figura 18).

Figura 18 - Conexão com o MongoDB

```
1 const mongoose = require('mongoose'); // Criando conexão no Mongo db
2 mongoose.connect('mongodb://localhost:27017/projetos');//Conecta ao mongo
3 // Esquema dos dados no banco
4 const produto = new mongoose.Schema({
5 etapa1: String, tamanho: String, phash1: String,hash1: String,
6 etapa2: String, cor: String, phash2: String, hash2: String,
7 etapa3: String,material: String, phash3: String, hash3: String,
8 etapa4: String,veiculo: String, phash4: String, hash4: String,
9 datacriacao: String, }, { collection: 'produto' } );
10// exportar a conexão do mongo e o esquema
11 module.exports = { Mongoose: mongoose, Produto: produto }
```

Fonte: Próprios Autores (2021)

Finalizando o código fonte, estrutura-se e importa para a Página Server a cadeia de blocos entre as linhas 1 à 30 da Página Blockchain conforme Figura 19.

Figura 19 - Estrutura de geração dos blocos da rede Blockchain

```
1 //constante para criptografia sha-256
2 const sha256 = require('crypto-js/sha256');
3 //atributos para um rede ,index,hora atual,dado,hash e hash anterior
4 class Block{ constructor(index, timestamp, data, previoushash = ''){
6 this.index = index; this.timestamp = timestamp;
7 this.data=data;this.previoushash=previoushash;this.hash=this.calculateHash();}
9 //calcular hash com base no index, hora, hash anterior e dado enviado
10 calculateHash(){ return sha256(this.index + this.previoushash + this.timestamp +
 12 JSON.stringify(this.data)).toString(); }}
13//classe geradora de cadeia de Blockchain
14 class Blockchain{
15 constructor(){ this.chain = [this.createGenesisBlock()]; }
18 //bloco genesis baseado na hora atual de execução
19 createGenesisBlock(){
20 //gerar dia da execução do programa
21 return new Block(0, '2021-01-01', "Bloco Genesis","0"); }
22 // Captura o ultimo bloco para adicionar o novo bloco
23 getLatestBlock(){ return this.chain[this.chain.length - 1]; }
24 // Criando um novo bloco
25 addBlock(newBlock){ newBlock.previoushash = this.getLatestBlock().hash;
26 newBlock.hash = newBlock.calculateHash(); this.chain.push(newBlock); }
27//limpar lista de blocos da rede Blockchain após os dados serem salvos
28 isempty(){ this.chain.length = 1;}}
29// Exportar as classes Blockchain e Block
30 module.exports.Blockchain = Blockchain; module.exports.Block = Block;
```

Fonte: Próprios Autores (2021)

Com a implementação do Código fonte juntamente com a instalação do Banco de Dados MongoDB, obtém uma rede Blockchain. Com isso pode-se obter dados sobre seu desempenho e segurança. Abaixo serão mostrados os testes de desempenho realizados com a utilização da ferramenta Jmeter para avaliar os picos de processamento (momento do maior uso do processador do computador durante o teste de requisição) relacionados à inclusão, consulta e alterações de informações utilizando cargas de 10, 20, 50 e 100 requisições simultâneas. Quando o teste é realizado, a ferramenta mostra algumas informações como tempo de teste, hora do início, latência, status, tentativas de conexão e outros. A Figura 20 mostra o exemplo de um teste com 10 requisições, essa mesma figura também é obtida nos testes de 10, 20, 50 e 100 requisições, porém os resultados foram diferentes.

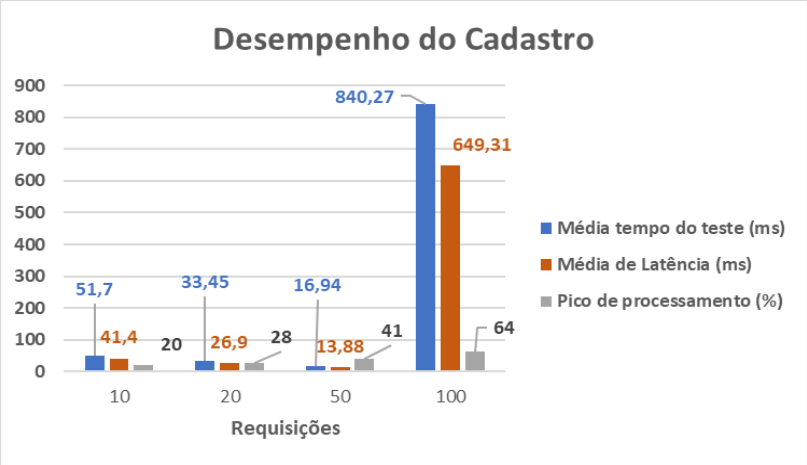
Figura 20 - Informações do Jmeter

Sample	Start time	thread name	label	Sample time (ms)	status	bytes	sent bytes	Latency	Connection
1	22:05:49.089	Thread Group 1-1	HTTP Request	19	Success	841	422	17	2
2	22:05:49.198	Thread Group 1-2	HTTP Request	31	Success	841	422	28	3
3	22:05:49.312	Thread Group 1-3	HTTP Request	38	Success	841	422	34	4
4	22:05:49.410	Thread Group 1-4	HTTP Request	38	Success	841	422	34	4
5	22:05:49.514	Thread Group 1-5	HTTP Request	27	Success	841	422	24	4
6	22:05:49.603	Thread Group 1-6	HTTP Request	13	Success	841	422	11	1
7	22:05:49.708	Thread Group 1-7	HTTP Request	32	Success	841	422	29	4
8	22:05:49.807	Thread Group 1-8	HTTP Request	30	Success	841	422	27	3
9	22:05:49.907	Thread Group 1-9	HTTP Request	35	Success	841	422	32	3
10	22:05:50.006	Thread Group 1-10	HTTP Request	29	Success	841	422	26	3

Próprios Autores (2021)

Com as informações obtidas, foi realizada uma média dos dados das colunas Sample Time (tempo do teste) e Latency (latência) e verificado o comportamento do processador do computador para geração do gráfico de desempenho de cadastro observado no Gráfico 01.

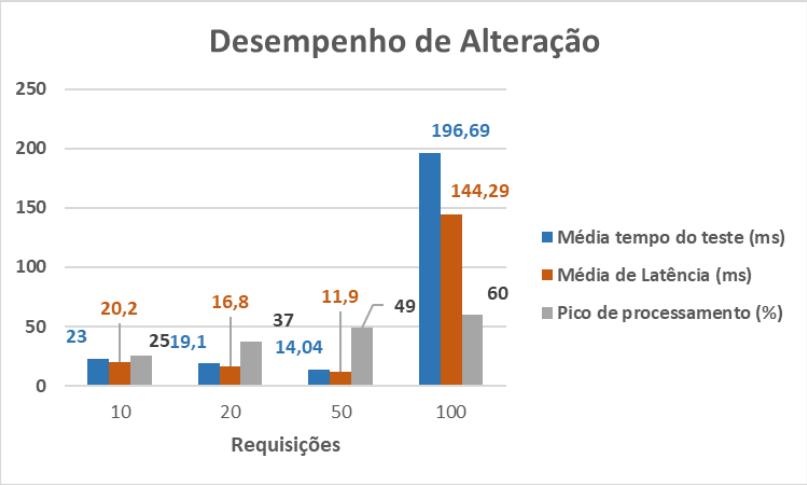
Gráfico 01 - Desempenho de cadastro



Fonte: próprios autores (2021)

Semelhante a geração do Gráfico 01, foram feitos testes de requisições para alterações dos produtos onde obteve as das informações das colunas **Sample Time (tempo do teste) e Latency (latência)** e verificado o comportamento do processador do computador para elaboração do Gráfico 02.

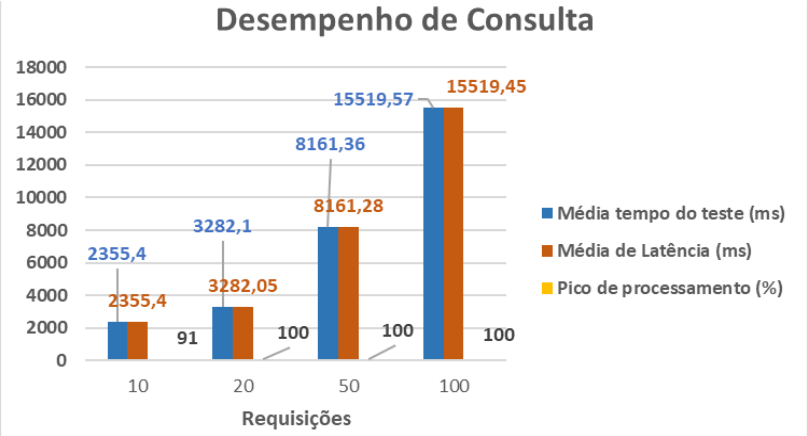
Gráfico 02 - Desempenho de alteração



Fonte: Próprios autores (2021)

Finalizando a elaboração do desempenho, foi realizado o Gráfico 03 baseado nas colunas **Sample Time (tempo do teste) e Latency (latência)** e verificado o comportamento do processador do computador nos testes de requisições feitas para consultar produtos.

Gráfico 03 - Desempenho de Consulta



Fonte: Próprios autores (2021)

Analisando esses resultados, conclui-se que o desempenho da consulta tem um maior tempo de teste e latência e esse tempo apenas aumenta conforme a quantidade de dados gerados no banco, com isso podem ocorrer erros e requisições poderão não serem processadas devido a picos de processamento (momento do maior uso do processador do computador durante o teste de requisição) gerado pelo computador, pois nela o programa faz a verificação se as informações não foram corrompidas e após exibe-as, diferentemente do cadastro que somente insere as informações e alteração que apaga e insere

as informações no banco de dados não precisando de processar a lógica de consulta realizada no código. Contudo essa avaliação de desempenho pode ser diferente conforme as configurações da maquina que será realizado o processamento, nesses testes foram utilizadas as métricas definidas na metodologia.

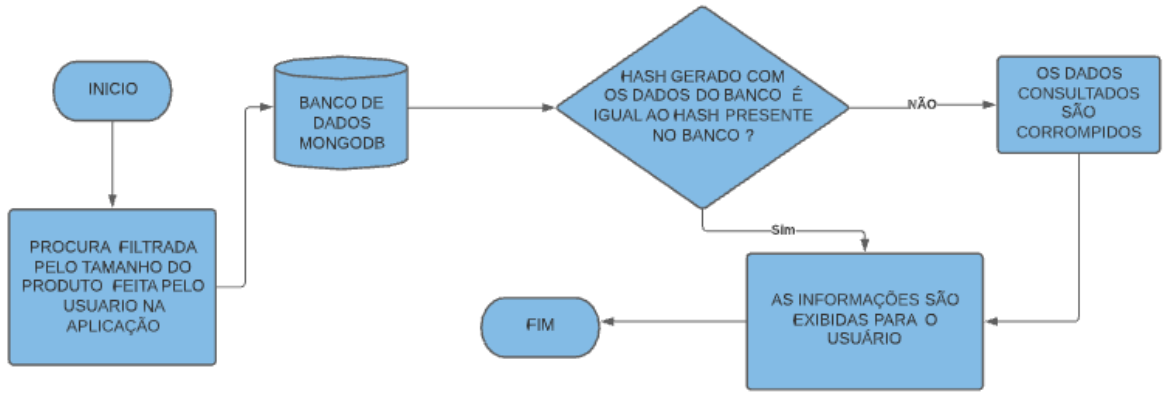
Utilizando a ferramenta Jmeter para realizar duas requisições, onde o intervalo de tempo entre elas se diferencia em milissegundos e o restante dos dados são iguais, observado no campo **timestamp** dos blocos e com isso o hash gerado é diferente do observado no campo **hash** dos blocos.O primeiro bloco gerado **marcado em vermelho** na Figura 21 é referente à primeira requisição feita e o segundo bloco gerado **marcado em azul** na Figura 21 é referente a segunda requisição feita.

Figura 21 - Geração de um bloco

Fonte: Próprios Autores (2021)

Sendo critério de segurança, a aplicação realiza uma comparação entre a criptografia do hash gerado presente no banco de dados MongoDB e a criptográfica de hash gerada no momento da consulta dos dados conforme o fluxograma da Figura 23. Caso tenha diferença entre os hash's ,os dados serão corrompidos antes de exibi-los, se for idêntico os hash's , os dados serão exibidos sem corrompê-los previamente.

Figura 23 - Comparação Hash



Fonte: Próprios Autores (2021)

5. Conclusão

Neste artigo, verificamos a possibilidade de implementação e verificação do desempenho e segurança da aplicação desenvolvida do Blockchain em Javascript simulando de forma abstrata processos que ocorrem na produção de um produto. Para este feito, foi realizada toda a parte de estruturação do estudo para aplicação dessa tecnologia e desenvolvimento do código fonte, concluindo que a aplicação do Blockchain pode ser executada em ambiente industrial e também em outros setores que existem uma grande quantidade de etapas em seu processo de forma a garantir a segurança e definir momentos de picos de processamento (momento do maior uso do processador do computador durante o teste de requisição) e consequentemente aumentar as possibilidades de falha em incluir, consultar ou alterar os dados causados pelo aumento de requisições simultâneas.

6. Referências Bibliográficas

NAKAMOTO, Satoshi; “Bitcoin: A Peer-to-Peer Electronic Cash System”. Acesso em 07 de abril de 2021, disponível em https://bitcoin.org/bitcoin.pdf

AL-JAROODI, Jameela; MOHAMED, Nader; “Blockchain in Industries: A Survey”. Acesso em 07 de abril de 2021, disponível em https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8662573

HOUBEN, R.; SNYERS, A. Cryptocurrencies and Blockchain: legal context and implications for financial crime, money laundering and tax evasion. 2018.

PENARD, W.; WERKHOVEN, T. v. On the Secure Hash Algorithm family. 2008. Disponível em: https://www.staff.science.uu.nl/~tel00101/liter/Books/CrypCont.pdf. Citado na página 18.

KAMIENSKI, Carlos; CALLADO, Arthur; SOUTO, Eduardo; SADOK, Djamel; ROCHA, João; DOMINGUES, Marco; “Colaboração na Internet e a Tecnologia Peer-to-Peer”.
Acesso em 03 de maio de 2021, disponível em https://www.researchgate.net/publication/255653983_Colaboracao_na_Internet_e_a_Tecnologia_Peer-to-Peer

OLIVEIRA, Walan;”Token Universitário por Meio de Blockchain” disponível em: https://ri.ufs.br/bitstream/riufs/12497/2/Walan_Marcel_Teles_Oliveira.pdf. Acesso em 03 de maio de 2021.

TANENBAUM, Andrew; “Redes de computadores”; 4ª Ed., Editora Campus (Elsevier)”. 2003.

ANTONOPOULOS, Andreas; “Mastering Bitcoin”; 1ª Ed., Editora O’Reilly. 2014

STALLINGS, William;”Criptografia e segurança de redes, princípios e práticas”; 6ª Ed., Editora Pearson. 2015

LINS, Fernando;”Simulação e Avaliação de Desempenho de uma Rede Blockchain Utilizando Containers Docker” disponivel em: https://www.e-publicacoes.uerj.br/index.php/cadinf/article/viewFile/46934/35686.Acesso em 10 de junho de 2021.

Selke(2019),"como-criar-o-formulário-com-nome-e-salvar-no-banco-de-dados" disponível em:https://celke.com.br/artigo/como-criar-o-formulario-com-node-e-salvar-no-banco-de-dados.Acessado em 10 de junho de 2021

Savjee(2021),”A simple Blockchain in Javascript”, disponivel em: https://github.com/Savjee/SavjeeCoin#readme: Acesso em 10 de junho de 2021