

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Marco Túlio dos Reis Pereira

**Desenvolvimento de aplicações de controle e
rastreamento utilizando Blockchain**

Uberlândia, Brasil

2021

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Marco Túlio dos Reis Pereira

**Desenvolvimento de aplicações de controle e
rastreamento utilizando Blockchain**

Trabalho de conclusão de curso apresentado
à Faculdade de Computação da Universidade
Federal de Uberlândia, Minas Gerais, como
requisito exigido parcial à obtenção do grau
de Bacharel em Sistemas de Informação.

Orientador: Ivan da Silva Sendin

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2021

Marco Túlio dos Reis Pereira

Desenvolvimento de aplicações de controle e rastreamento utilizando Blockchain

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 24 de Maio de 2021:

Ivan da Silva Sendin
Orientador

Renan Gonçalves Cattelan

William Chaves de Souza Carvalho

Uberlândia, Brasil
2021

Dedico este trabalho em especial aos meus pais que me apoiaram durante o decorrer do curso, e também ao meu orientador Ivan Sendin que me auxiliou para que fosse possível este desenvolvimento.

Agradecimentos

Agradeço a todo o corpo docente da Faculdade de Computação da Universidade Federal de Uberlândia que contribuiu para minha formação acadêmica e profissional. Em especial, agradeço ao meu orientador que me deu auxílio e contribuiu para o desenvolvimento deste trabalho. Deixo minha gratidão também aos meus pais e namorada que me apoiaram durante o decorrer do curso e que ajudaram para que eu conseguisse chegar até o fim.

“Suba o primeiro degrau com fé. Não é necessário que você veja toda a escada. Apenas dê o primeiro passo.”

Martin Luther King Jr

Resumo

Este trabalho tem como objetivo estudar e entender os conceitos da Blockchain, em seguida desenvolver uma aplicação de exemplo. Para desenvolver essa aplicação foi necessário compreender o Hyperledger, ferramenta utilizada para criação de uma rede Blockchain privada, sendo necessário saber como conseguir integrar uma aplicação desenvolvida com o Hyperledger, com a finalidade de demonstrar essa possibilidade de implementação, bem como a contribuição dessa tecnologia.

Palavras-chave: Blockchain, Hyperledger, Consenso, Livro-Razão, Cadeia de blocos.

Lista de ilustrações

Figura 1 – Representação de relacionamento entre os blocos da Blockchain. Como cada bloco possui o <i>hash</i> do bloco anterior as informações dos blocos não podem ser alteradas sem serem percebidas pelos participantes da rede.	16
Figura 2 – Fluxo de comunicação da aplicação	26
Figura 3 – Apresentação do comando executado no terminal para subir os nós. . .	27
Figura 4 – Representação do líder nó 0 em uma rede com seis nós de forma circular.	28
Figura 5 – Representação do líder nó 1 em uma rede com seis nós de forma circular.	28
Figura 6 – Representação do ultimo nó sendo líder e o próximo líder.	29
Figura 7 – Representação do envio da mensagem do nó líder aos nós secundários. .	29
Figura 8 – Representação da transmissão de mensagem aceita pelos nós secundários.	30
Figura 9 – Representação da transmissão de mensagem de <i>commit</i> por todos os nós da rede.	30
Figura 10 – Representação da transmissão da mensagem invalida pelo nó líder para os nós secundários.	31
Figura 11 – Representação da resposta dos nós secundários invalidando o novo bloco.	31
Figura 12 – Requisição para criar diamante valido.	32
Figura 13 – Resposta com a URL para consultar status da transação.	33
Figura 14 – Consulta na API do Hyperledger	33
Figura 15 – Retorno de consulta de diamantes.	34
Figura 16 – Exemplo de requisição para transferir diamante.	35
Figura 17 – Consulta do status da transferência realizada.	35
Figura 18 – Retorno dos diamantes do usuário que realizou a transferência.	36
Figura 19 – Retorno de diamantes do usuário que recebeu a transferência.	36
Figura 20 – Requisição de transferência realizado pelo usuário não detentor do diamante.	37
Figura 21 – Retorno de requisição invalida.	37
Figura 22 – Requisição de transferência de um diamante não existente.	38
Figura 23 – Retorno de requisição invalida.	38
Figura 24 – Retorno de consulta dos diamantes do produtor utilizado no teste. . . .	39
Figura 25 – Apresentação do cadastro do quarto diamante.	40
Figura 26 – Retorno da consulta com o status da transferência.	40

Lista de abreviaturas e siglas

APT	<i>Advanced Packaging Tool</i>
API	<i>Application Programming Interface</i>
XML	<i>eXtensible Markup Language</i>
GPG	<i>GNU Privacy Guard</i>
HTTP	<i>Hypertext Transfer Protocol</i>
PBFT	<i>Practical Byzantine Fault Tolerance</i>
POM	<i>Project Object Model</i>
PoS	<i>Proof of Stake</i>
PoW	<i>Proof of Work</i>
URL	<i>Uniform Resource Locator</i>
WFDB	<i>World Federation of Diamond Bourses</i>

Sumário

1	INTRODUÇÃO	11
2	FUNDAMENTAÇÃO TEÓRICA	13
2.1	Trabalhos Correlatos	13
2.2	Blockchain	13
2.2.1	Livro Razão	14
2.2.2	Blockchain como uma estrutura de dados	15
2.2.3	Algoritmos de consenso	17
2.2.3.1	Proof of Work	18
2.2.3.2	Proof of Stake	19
2.3	Hyperledger	19
2.3.1	Instalação do Hyperledger Sawtooth e dependências do projeto	20
2.3.1.1	Docker	20
2.3.1.2	Maven	21
2.3.2	Instalação do Docker Linux Ubuntu 64 bits	21
2.3.3	Instalação do Docker Composer	22
2.3.4	Subindo o Hyperledger no contêiner Docker	22
3	APLICACAO	24
3.1	Motivação para o desenvolvimento da aplicação	24
3.1.1	Cenário de diamantes	24
3.2	Aplicação para controle de diamantes	25
3.2.1	Arquitetura do sistema	26
3.3	Mecanismo de consenso utilizado na aplicação	27
3.3.1	Características do modelo de consenso	27
3.3.2	Definição do nó líder	28
3.3.3	Funcionamento do consenso para um bloco aprovado	29
3.3.4	Funcionamento do consenso para um bloco não aprovado	30
3.4	Testes realizados	32
3.4.1	Cadastro de diamante aceito	32
3.4.2	Transferência de diamante aceita	34
3.4.3	Transferência de diamante já transferido	37
3.4.4	Transferência de diamante não existente	38
3.4.5	Limite de produção atingido	39
4	CONCLUSÃO	41

REFERÊNCIAS	42
APÊNDICES	44
APÊNDICE A – ALGORITMO PARA CRIAR BLOCO	45
APÊNDICE B – ALGORITMO PARA COMUNICAR COM HY- PERLEDGER	49
APÊNDICE C – ALGORITMO DE CONSENSO	54

1 Introdução

Atualmente a Internet tem sua grande importância na sociedade, isso foi possível graças ao avanço tecnológico e a necessidade que os usuários têm de utilizar a Internet para fazer suas atividades. Antes o que demandava muito tempo para ser realizado, hoje pode ser feito de uma maneira mais rápida, como exemplo de ações que foram facilitadas pela Internet podem ser citadas: transferências, compras e pesquisas.

Com todos esses avanços surgiu o Bitcoin: moeda digital que é formada por diversas tecnologias e que torna possível as transações de valores diretamente entre os usuários participantes de uma rede ([ANTONPOULOS, 2014](#)).

Utilizando o Bitcoin é possível realizar a transferência de valores entre os seus participantes, uma das características importantes do Bitcoin é ser um sistema descentralizado, retirando a necessidade da existência de um servidor central e com o poder unicamente dele de processar as transações ([ANTONPOULOS, 2014](#)). Sendo assim, não é necessária uma instituição bancária para manter o controle dos processos que são realizados no Bitcoin, caso que ocorre nas moedas fiduciárias ou lastreadas. No Bitcoin, cada usuário possui uma chave criptográfica que é combinada com o algoritmo executado na camada de aplicação e que consegue comprovar que aquele usuário é de fato o detentor do valor que está sendo transferido através do sistema.

Essa maneira de validação é permitida pela tecnologia chamada Blockchain que está implementada no Bitcoin. A Blockchain também segue a ideia de ser um sistema distribuído e descentralizado e, na medida em que as transações ocorrem, são gerados blocos que vão sendo adicionados à estrutura de dados.

Os benefícios de utilizar um sistema de validação descentralizado e mais confiável possibilitaram o surgimento de estudos atrelados à criação de novas aplicações baseadas nessa plataforma. Um exemplo é o estudo realizado por ([KARNOUPAKIS; HARGRAVE, 2020](#)), em que é citado o uso de Blockchain em uma instituição de caridade, tendo como diferencial um processo de doação transparente, onde o valor doado seria repassado diretamente para o indivíduo beneficiado, sem a passagem por terceiros, sendo permitido aos doadores acompanharem suas transações realizadas e terem a certeza no destino do valor doado. Analisando esses fatores apresentados pelo autor surge a identificação da necessidade de estudo dos sistemas já existentes e a possibilidade de aprimoramento e criação de novos sistemas utilizando a Blockchain.

Baseando-se em todo esse cenário descrito, esse trabalho tem como objetivo principal realizar um estudo e entender o funcionamento e as principais características da Blockchain. Após esse entendimento, será escolhida uma ferramenta que possibilite o de-

envolvimento de uma aplicação engajada nos conceitos da Blockchain, demonstrado como essa tecnologia pode contribuir sendo utilizada em um sistema de exemplo.

2 Fundamentação teórica

2.1 Trabalhos Correlatos

Nesta seção, são apresentados alguns dos trabalhos que estão relacionados com o estudo de aplicações que utilizam Blockchain e que podem ser relevantes para desenvolver a aplicação deste trabalho.

Em ([CHUEN, 2015](#)) é apresentado um estudo sobre sistemas de votação eletrônica, nesse estudo são realizadas algumas críticas a esses programas, entre os problemas encontrados pelo autor está a centralização do código fonte e banco de dados da aplicação de votos, sendo assim apenas uma indivíduo é detentor dessas informações, por esse motivo esses sistemas não conseguem ter uma confiança tanto dos eleitores, como de quem organiza as eleições. Dentro desse estudo também são identificados outros fatores que prejudicam esse tipo de aplicação, são eles: questões de segurança e falta de confiabilidade. Considerando isso, o autor apresenta um método para resolver esse problema identificado, desenvolvendo um sistema de votação baseado na Blockchain. Dessa forma, seria criado um método seguro, completo, gratuito e tendo uma rede distribuída.

Outra pesquisa de grande importância é a realizada por ([UK GOVERNMENT DIGITAL SERVICE, 2015](#)). Nesse estudo, são apresentadas algumas das vantagens da adoção de sistemas baseados em Blockchain pelo governo, como por exemplo: transparência nas transações do governo, redução de custos, segurança contra ataques, crescimento econômico, redução das fraudes em transações realizadas e eficiência das aplicações oferecidas, chegando à conclusão que essa tecnologia é realmente de grande utilidade para o governo e seria uma possível forma de funcionamento futuro.

A plataforma ([EVERLEDGER, 2021](#)) fornece soluções baseadas em Blockchain para rastreabilidade de produtos diversos, como por exemplo: pedras preciosas, bebidas e obras de arte. Um dos principais objetivos dessa solução é estabelecer o controle do histórico de um determinado produto, para garantir aos usuários uma maior confiança de seus clientes na compra desses itens. Através dessa aplicação é possível ter uma maior transparência e segurança na comercialização dos diferentes produtos possibilitados.

2.2 Blockchain

A Blockchain pode ser definida como uma estrutura de blocos encadeados que forma uma coleção de registros, onde é possível adicionar novos blocos ao final dessa estrutura, sendo o controle desses blocos realizado pelo protocolo de consenso, determinando

o que pode ser ou não adicionado ao conjunto (RAJ, 2019).

Além dessa breve descrição, podemos citar os principais pontos fortes dessa tecnologia e que acabam contribuindo para o crescimento e aceitação dela. Um desses pontos é o funcionamento de forma distribuída e descentralizada, isso ocorre pois a validação dos blocos que são inseridos na Blockchain não é feita diretamente em um ponto central e único da aplicação, pois não se tem a existência desse ponto central, mas se tem a validação feita de maneira descentralizada.

Pode ser citado como um exemplo de funcionamento de Blockchain, o modelo utilizado no Bitcoin que é uma rede *peer-to-peer*, em que os nós participantes da rede verificam e propagam os blocos. Com essa propagação, os nós adicionam o novo bloco ao final de sua estrutura, fazendo com que os participantes da rede compartilhem da mesma cadeia de blocos entre eles, sendo que quando uma informação é adicionada à cadeia, essa informação será imutável (HARGRAVE, 2020).

Esse meio de validação da Blockchain é o que garante uma aceitação de todos os membros da rede, e essa característica do compartilhamento de dados no modelo *peer-to-peer* permite que a Blockchain possa ser inserida em diversas aplicações (RAJ, 2019).

Apesar dessa tecnologia apresentar esse benefício é importante também entender como isso é possível, sendo esse o objetivo da próxima seção que apresenta a forma de como é possibilitada ter essa confiança na cadeia de blocos.

2.2.1 Livro Razão

Seguindo essa necessidade de ter a confiança na cadeia de blocos, surge o que é conhecido como livro razão, algo já utilizado e conhecido na contabilidade e que consiste no registro de forma cronológica de todo tipo de transação realizada para controle de valores, sendo esse registro realizado de forma física pelo contador, em que ele não consegue alterar as informações já inseridas no livro, mas se tem a possibilidade de escrever novos dados.

Em soluções como Bitcoin em que se tem a presença de um valor envolvido é sempre necessário garantir a segurança, afirmando que não exista nenhum tipo de inconsistência nos valores apresentados dentro desse tipo de aplicação. Uma das maneiras de garantir essa integridade é tendo um controle do fluxo de movimentações realizadas, onde todo o valor que for repassado para um determinado usuário deve realmente ser removido da conta do usuário que está repassando o valor.

Uma diferença do que acontece na Blockchain para o que ocorre na contabilidade é que na contabilidade geralmente o livro razão é centralizado, sendo gerenciado por um banco ou contador que controla a entrada e saída de dinheiro, ao contrário disso, na Blockchain é utilizado um livro razão de maneira descentralizada, isso significa que não se tem as operações sendo realizadas em um único lugar, mas essas operações são

compartilhadas com diversos nós presentes na aplicação. Nessa abordagem, cada transação é gravada como bloco de dados que são encadeados na estrutura, pode-se imaginar vários blocos em fila e interligados em sequência ([HARGRAVE, 2020](#)).

Dessa forma descrita obtém-se o livro razão, onde toda transação tem sua origem e destino mapeados, nada do que é manipulado na estrutura fica sem ser registrado, portanto consegue-se manter o histórico de todo evento realizado na cadeia de blocos.

Apesar do que foi descrito podem surgir dúvidas sobre quais os reais benefícios de fazer isso utilizando essa tecnologia e não fazer da maneira convencional que seria realizar o controle através de um banco ou contabilidade. É nessa parte em que se tem o diferencial, sendo esse o fato de não poder remover um bloco após inseri-lo na estrutura, consequentemente um funcionário do banco ou contabilidade não pode ser subornado a alterar as informações já existentes na aplicação ([HARGRAVE, 2020](#)).

Pode surgir o questionamento de como tudo isso consegue funcionar sem ter alguém no controle. É nessa parte onde entra um dos pontos chaves para inserir na Blockchain que é o consenso. Através do protocolo onde se tem regras pré-estabelecidas, os nós que participam da rede podem validar as informações que são transacionadas, então após os nós chegarem a um consenso, o novo dado pode ser persistido na estrutura.

Essa maneira de realizar a validação com um consenso descentralizado, onde não se tem autoridades responsáveis pelo banco de dados, gera alguns benefícios, entre eles podem ser citados: economia de custos, maior velocidade e confiança nas transações ([BASHIR, 2018](#)).

Como descrito o livro razão utilizado pela Blockchain tende a suprir alguns problemas existentes em algumas áreas e é algo interessante para buscar ser utilizado pois consegue entregar confiança, segurança e transparência em sistemas.

2.2.2 Blockchain como uma estrutura de dados

Relacionando uma Blockchain como sendo uma estrutura de dados é necessário antes definir as estruturas de dados existentes, bem como entender seu funcionamento e algumas das funcionalidades ou manipulações possíveis.

Existem diversas formas de se armazenar dados, umas das maneiras é guardar esses dados em uma lista ou matriz, podendo-se fazer operações necessárias através da iteração nessas estruturas. Apesar dessas operações serem de simples implementação, deve-se pensar na possibilidade de uma grande quantidade de dados. Porque nessas situações pode ser necessário realizar muitas iterações para encontrar uma determinada informação presente no conjunto de dados, caso a estrutura de dados selecionada para fazer esse processamento não seja a melhor, pode se ter um tempo alto para percorrer a estrutura até encontrar a informação buscada ([MORIN, 2014](#)).

Dessa maneira, podemos definir diferentes tipos de estruturas para diferentes aplicações ou até para uma mesma aplicação. Existem diferentes tipos de estruturas que podem ser citadas, algumas das mais conhecidas são: Lista, Fila, Árvore Binária, Tabela Hash e Grafo.

De acordo com a estrutura selecionada, podemos ter diferentes funcionalidades que podem-se diferenciar de acordo com a abordagem que for selecionada. Um exemplo disso é a Lista encadeada não ordenada em que se consegue inserir um elemento sem restrição de posição no conjunto de dados, diferindo-se disso temos o exemplo da Pilha em que não é possível inserir um elemento em qualquer posição desejada, porque existe uma regra que todo elemento em uma pilha deve ser inserido após o último elemento, seguindo o conceito já conhecido onde o primeiro elemento a entrar em uma pilha é o último a sair. Dessa forma, podemos ter diferentes funcionalidades e regras definidas de acordo com o objetivo final. Todavia existem funcionalidades entre as estruturas que se assemelham as presentes na Blockchain, como por exemplo buscar uma determinada informação, listar os blocos presentes na cadeia e inserir um novo bloco.

Após essas definições é necessário entender a qual estrutura a Blockchain se assemelha em seu funcionamento, tendo como uma representação a Figura 1.

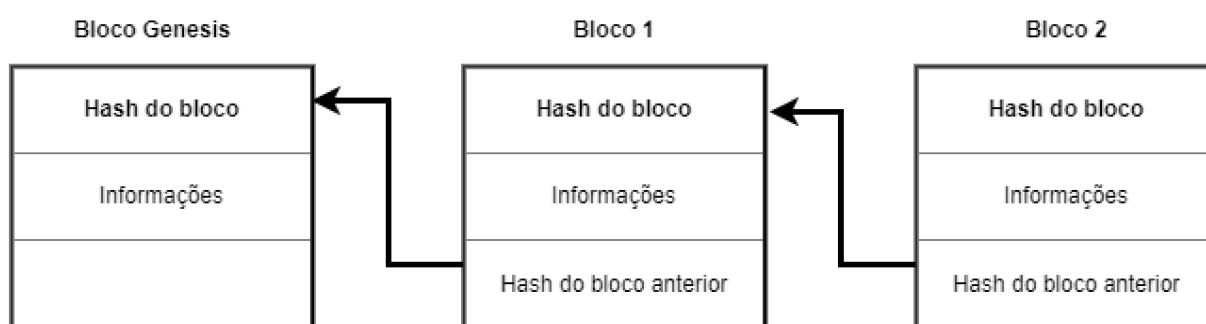


Figura 1 – Representação de relacionamento entre os blocos da Blockchain. Como cada bloco possui o *hash* do bloco anterior as informações dos blocos não podem ser alteradas sem serem percebidas pelos participantes da rede.

A estrutura da Blockchain é uma lista ordenada com a referência de um bloco atual para o seu bloco pai. Também pode ser vista como uma pilha vertical, onde os blocos são inseridos em camadas e tendo como base o primeiro bloco inserido na estrutura. É utilizado também a altura ou distância do primeiro bloco para o bloco presente no topo da estrutura ou último bloco inserido para se ter referência na estrutura existente. Cada bloco presente na Blockchain pode ser identificado por um *hash* presente em suas informações, gerado através de uma função *hash* criptográfica, como também existe uma mesma informação *hash* que aponta para o vizinho anteriormente inserido ou seu antecessor (ANTONPOULOS, 2014).

A diferença das outras estruturas citadas anteriormente é que essa tecnologia não

possui algumas das funções presentes em outras estruturas de dados, mas também temos algumas funcionalidades que são semelhantes. A seguir, podemos citar as manipulações que são possíveis na Blockchain e que se assemelham a uma estrutura de dados:

Listar Blocos: existe disponível a possibilidade de listar os blocos existentes na lista. Como já descrito anteriormente é possível percorrer a estrutura pela relação que os dados mantêm apontando para o vizinho anterior;

Listar Informações de um Bloco: após selecionado um bloco também é possível ter acesso as informações presentes no bloco;

Buscar um Bloco: possibilidade de buscar um bloco desejado. Sendo possível percorrer a estrutura e encontrar um bloco para validação ou para se confirmar a hierarquia desse tipo de conjunto de dados;

Inserir Bloco: apesar de não ser possível editar um bloco ou removê-lo, é possível inserir um novo bloco no final da estrutura. Fazendo assim com que seja possível adicionar novos registros a estrutura.

Como citado existem algumas manipulações encontradas na maioria das estrutura de dados mais conhecidas e que não se aplicam a Blockchain, são elas:

Remoção de Blocos: nessa estrutura não é possível remover um determinado bloco, todos os blocos que são inseridos vão continuar presentes durante o crescimento da estrutura de dados. Pode ser algo visto como uma desvantagem, mas é algo comentado anteriormente em que podemos ter uma abordagem diferente de acordo com o que é desejado, sendo o que torna essa tecnologia forte. Aumentando sua utilidade quando é necessário uma interface em que a estrutura não pode ser modificada;

Edição de Bloco: como não é possível remover um bloco, também não existe a possibilidade de modificar um bloco existente, todos os dados inseridos na cadeia de blocos serão imutáveis durante todo o processo de manipulações, sendo isso o que atende a necessidade do software desenvolvido neste trabalho. Mantendo sempre a integridade dos dados presentes na estrutura e contribuindo para os casos de implementações que precisam de manter um histórico de suas transações.

2.2.3 Algoritmos de consenso

Como apresentado, a Blockchain é uma tecnologia que funciona de maneira descentralizada, portanto não necessitando de uma entidade central para realizar o controle das transações que são feitas na aplicação, esse funcionamento é possível através do consenso, pois toda nova informação salva na Blockchain só pode ser persistida após um

consenso ser alcançado entre os nós participantes da rede, sendo que esse acordo entre os pares da rede pode ser obtido pelos diversos algoritmos de consenso existentes ([BASHIR; PRUSTY, 2019](#)).

Para que seja de melhor entendimento esse consenso, pode-se citar alguns exemplos de aplicações conhecidas e que são desenvolvidas para se chegar a um consenso, são essas, sistema de votação em que diversos usuários vão votar e definir algo através do consenso entre eles, como também os sistemas de votação realizados de maneira informal, como por exemplo os usuários de redes sociais que podem votar positivo em uma publicação. Porém, esse modelo de votação por parte humana é uma forma não confiável e que não consegue lidar com diversas transações, todavia a Blockchain consegue fornecer através do seu consenso uma maneira automatizada que possibilita fazer a validação das transações presentes na rede ([YUAN, 2019](#)).

A seguir serão descritos alguns dos algoritmos de consenso existentes.

2.2.3.1 Proof of Work

Protocolo utilizado pelo Bitcoin em que os nós participantes da rede precisam passar pelo processo chamado de mineração, onde eles realizam um cálculo utilizando a função SHA256 para determinar o endereço do próximo bloco, considerando que os blocos mantêm um sequência e relação entre si, sendo que a medida que a complexidade para resolver esses problemas vai aumentando, também se tem o aumento do poder computacional necessário para resolver esses cálculos, tendo como um retorno para incentivar a participação desses nós o Bitcoin. Um dos problemas encontrados não é apenas o custo para manter um poder computacional alto, mas também o custo de manter tudo isso funcionando devido ao alto consumo de energia, sendo esse um dos motivos para a busca por outros consensos alternativos ([HARGRAVE, 2020](#)).

Apesar dos problemas citados o algoritmo de consenso PoW(*Proof of Work*) se apresenta seguro contra ataques, porque depois que um dos mineradores consegue resolver o problema matemático ele pode propor um novo bloco válido, caso os outros participantes detectem que o bloco inserido é inválido vão ser propostos blocos concorrentes, com isso se existir um fraudador tentando escrever blocos inválidos na cadeia, o único que vai continuar escrevendo nessa cadeia de blocos vai ser ele, mostrando que aquela ramificação criada por ele não é válida ([YUAN, 2019](#)).

O autor ([YUAN, 2019](#)) também apresenta um modelo de ataque chamado de *51 por cento* que seria os participantes da rede entrarem em consenso para fazer a ramificação de um bloco inválido parecer legítima, porém para fazer isso na rede do Bitcoin é necessário um poder computacional muito alto, portanto compensa mais para um minerador seguir as regras e receber sua recompensa do que tentar atacar a rede do Bitcoin.

2.2.3.2 Proof of Stake

O PoS (*Proof of Stake*) é uma alternativa ao algoritmo de consenso PoW, onde os mineradores participantes da rede conseguem validar proporcionalmente a quantidade de *tokens* que eles possuem, a vantagem dessa abordagem é que ela tem um custo computacional menor, porque ao invés dos nós ficarem minerando para receber a recompensa, ela vai ser distribuída proporcionalmente para todos. O problema desse modelo é que quem tem muito, sempre vai conseguir mais do que aqueles que possuem menos, sendo algo ruim porque pode ser uma desmotivação para participar da rede por parte daqueles que detêm pouco, mas por outro lado existe um incentivo para aqueles que possuem muitos *tokens* de manter a integridade na Blockchain e continuar contribuindo para que a mesma não seja fraudada ([HARGRAVE, 2020](#)).

2.3 Hyperledger

Aplicação disponibilizada com código fonte aberto sendo feita de forma colaborativa entre apoiadores da iniciativa, incluindo grandes empresas conhecidas da área de tecnologia, como Intel, IBM e Cisco. Além das grandes empresas citadas, existem mais de 250 empresas que são membros do projeto, tendo como a líder da iniciativa a Linux Foundation e com o objetivo de atender várias demandas dos setores que podem ser da tecnologia ou até mesmo fora do ramo de interesse tecnológico ([HYPERLEDGER, 2018d](#)).

O objetivo do Hyperledger é prover uma Blockchain de forma a atender o interesse de grupos privados, como já abordado anteriormente existe um problema na forma que é implementada a Blockchain voltada para o meio público. Como um exemplo bem prático podemos colocar a forma com que é feita no Bitcoin, onde temos uma recompensa de Bitcoin para os usuários que devem provar o seu esforço através do cálculo *hash*, ao contrário disso, quando temos uma aplicação voltada para o ambiente privado, pode não existir essa possibilidade de recompensar o usuário como é feito no Bitcoin.

Podem também aparecer alguns questionamentos, como por exemplo, como vai ser feita a aprovação para que um determinado bloco possa integrar a rede? Esse cenário é algo que surge para motivação da criação do Hyperledger, com a ideia de atender com soluções que a Blockchain convencional não pode atender, mantendo o mesmo conceito adotado da tecnologia.

O Hyperledger detém de uma quantidade grande de bibliotecas e aplicações baseadas em Blockchain, entre as principais estão:

Hyperledger Fabric: utilizado para o desenvolvimento de aplicações que utilizam de uma abordagem de grande escala, além de ser de desenvolvimento bem flexível e

permitindo atender a necessidade de diversos setores do mercado ([HYPERLEDGER, 2018a](#));

Hyperledger Iroha: desenvolvido em C++, possuindo os blocos salvos em arquivo e o estado armazenado em banco de dados PostgreSQL, onde essa metodologia permite uma implementação com análises mais complexas. Sendo possível desenvolver em Java, Python, Javascript e C++ utilizando o Iroha ([HYPERLEDGER, 2018c](#));

Hyperledger Indy: uma de suas principais características é a opção de poder funcionar com outras Blockchains, mas podendo ser utilizado de forma independente também ([HYPERLEDGER, 2018b](#));

Hyperledger Sawtooth: plataforma que tem como objetivo contribuir com o meio corporativo e atender interesses privados, mantendo a segurança e a maneira distribuída de realizar seus contratos e transações que ocorrem na rede interna da aplicação em que essa plataforma está implementada ([HYPERLEDGER, 2020](#)).

A escolha do Hyperledger Sawtooth foi feita baseando-se na modularidade e facilidade de desenvolver regras de negócios específicas para diferentes tipos de sistema, com a possibilidade de utilizar algumas das linguagens de programação mais usadas atualmente, como por exemplo, Java, JavaScript e Kotlin. Com isso é possível realizar um desenvolvimento de forma mais simples, sendo permitido ao programador construir uma aplicação com a linguagem que ele tem maior domínio.

2.3.1 Instalação do Hyperlegder Sawtooth e dependências do projeto

Antes de ser iniciado o desenvolvimento da aplicação é necessário ter instalado algumas dependências para que seja possível instanciar o Sawtooth e executar o projeto desenvolvido nesse trabalho. A seguir serão descritas algumas das ferramentas utilizadas nesse trabalho.

2.3.1.1 Docker

O Docker é uma ferramenta que permite ao desenvolvedor separar a sua aplicação em algo chamado contêiner e que permite isola-lá do ambiente implantado, dessa maneira os frameworks ou ferramentas utilizadas vão funcionar da forma esperada independente de ambiente ([DOCKER, 2019b](#)). Sendo o contêiner o responsável por empacotar o código e as dependências de uma aplicação através de uma imagem gerada e que se torna contêiner em tempo de execução, sendo possível executar essa ferramenta em aplicações baseadas em Linux ou Windows ([DOCKER, 2019a](#)).

2.3.1.2 Maven

Ferramenta utilizada para gerenciamento de dependência em projetos que utilizam a linguagem Java. Para que possam ser baixadas essas dependências do projeto, como por exemplo bibliotecas que serão utilizadas para realizar uma determinada função dentro da aplicação, deve ser adicionada essa informação dentro de um arquivo XML(*eXtensible Markup Language*) utilizado pelo Maven para baixar as bibliotecas chamado de POM(*Project Object Model*), dentro desse arquivo existe uma forma correta de escrita onde as informações presentes no POM podem ser interpretadas, possibilitando ao Maven realizar o gerenciamento de todas as dependências do projeto e outras configurações (APACHE, 2020).

2.3.2 Instalação do Docker Linux Ubuntu 64 bits

É importante lembrar que para instalar o Docker é necessário a versão de 64 bits do Linux, podendo também ser instalado em sistemas operacionais com Windows. Comandos executados através do terminal:

Primeiro é necessário atualizar os pacotes:

```
sudo apt-get update
```

Após deve se adicionar a chave GPG(*GNU Privacy Guard*) para assinar os pacotes do Docker:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add-
```

Adicionar o repositório do Docker nas fontes do APT:

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

Atualizar os pacotes para ter acesso ao Docker que foi adicionado nos comandos anteriores:

```
sudo apt-get update
```

Instalar o Docker:

```
sudo apt-get install docker-ce
```

2.3.3 Instalação do Docker Composer

Comando para acessar a versão recente do Docker Composer, sendo a versão instalada nesta demonstração a de número 1.24.1 :

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.24.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

Aplicar as permissões de execução do programa:

```
sudo chmod +x /usr/local/bin/docker-compose
```

2.3.4 Subindo o Hyperledger no contêiner Docker

Antes de iniciar o ambiente é necessário baixar o arquivo de configurações `sawtooth-default.yaml`, esse arquivo é apenas para uma abordagem de desenvolvimento e visando um ambiente para teste de integração com a API (*Application Programming Interface*) do Sawtooth, com esse arquivo de configurações é possível subir um único nó para conseguir comunicar e publicar dados no Sawtooth. Além desse arquivo, existem mais dois outros arquivos, são eles, `sawtooth-default-pbft.yaml` e `sawtooth-default-poet.yaml`, tendo basicamente de diferença entre os dois arquivos o modelo de consenso.

Tendo o Docker Composer instalado e o arquivo de configuração baixado, o próximo passo é iniciar o Sawtooth para rodar no Docker, isso pode ser feito rodando o seguinte comando no diretório onde se encontra o arquivo `sawtooth-default.yaml`:

```
docker-compose -f sawtooth-default.yaml up
```

Em seguida é necessário verificar se a API está em execução consultando através do contêiner, para entrar no contêiner deve ser utilizado o comando a seguir:

```
docker exec -it sawtooth-shell-default bash
```

Em seguida é necessário verificar se a aplicação está disponível no contêiner fazendo a seguinte requisição:

```
curl http://rest-api:8008/blocks
```

Após confirmar que a API está sendo executada é necessário verificar se a máquina onde vai rodar a aplicação que comunicará com a API, tem acesso e consegue fazer requisições. Com isso basta fazer a seguinte requisição na porta em que foi configurada para a API rodar:

```
curl http://localhost:8008/blocks
```

Depois dos passos anteriormente realizados e confirmado que a API está funcionando e disponível para receber requisições, a próxima etapa é desenvolver e subir a aplicação que vai comunicar com o Hyperledger, onde os blocos vão ser de fato publicados.

3 Aplicação

3.1 Motivação para o desenvolvimento da aplicação

Visando o momento atual onde muitos setores do ramo de negócios privados ou até mesmo públicos necessitam de uma aplicação que consiga admitir a integridade das informações nela contida, garantindo que não exista nenhum tipo de fraude em seus sistemas, seja por motivos de fiscalização, conseguir mais confiança do cliente ou para garantir uma aplicação com dados não violáveis. Portanto, se tem a oportunidade de apresentar uma tecnologia para solucionar esse problema.

Abordando o assunto de confiança ou garantia nos dados presentes em uma aplicação, podem ser citados vários cenários de exemplo em que a Blockchain poderia ajudar a garantir o que está sendo solicitado, como por exemplo:

- Um cliente de uma determinada plataforma de produtos quer ter a garantia de que o seu comentário e avaliação não vai ser deletado ou alterado;
- Um usuário de um banco quer ter a certeza de que seu saldo não vai sofrer alterações indevidas;
- Garantia de transparência no controle da distribuição de vacinas da COVID-19;
- Um usuário de uma aplicação de votos quer a garantia de que seu voto realizado não vai ser adulterado.

Considerando essas necessidades surge o objetivo deste trabalho que é demonstrar o uso do Hyperledger para criação de um rede com Blockchain, apresentando a contribuição dessa tecnologia em um sistema de exemplo, tentando trazer as seguintes soluções: funcionamento de maneira descentralizada, um modelo de consenso confiável e uma estrutura imutável.

3.1.1 Cenário de diamantes

Com a abordagem deste trabalho foi selecionada uma aplicação de exemplo, tendo como objetivo o desenvolvimento de um protótipo de um sistema com os conceitos da Blockchain. Portanto, foi escolhida uma aplicação para controle no comércio e produção de diamantes, para demonstrar algo de um ambiente real e que pode receber essa implementação.

O comércio de diamantes é um dos mercados que pode receber boas contribuições da tecnologia Blockchain se aplicado da forma correta. Uma das incógnitas que podem surgir é o fato de não saber qual o histórico de um diamante que está sendo adquirido. Quando comprado um diamante pode ter grandes chances de ter sido fabricado através de mão de obra escrava.

Um exemplo de exploração escrava é a exploração de diamantes na Angola, onde são realizadas jornadas longas de trabalho tentando aumentar a produção de diamantes, consequentemente tendo um aumento na possibilidade de lucro, onde essa exploração de mão de obra é realizada através da comunidade local do país (MARQUES, 2011).

Esse histórico na venda de diamantes e a existência de algumas fraudes nas negociações, cria uma necessidade de controlar de uma forma mais transparente as transações feitas pelos vendedores e compradores. Com isso muito se foi investido para desenvolver algo para rastrear esses itens e garantir que o diamante negociado é autentico e não foi gerado de uma forma irregular.

Uma iniciativa foi feita para tentar conter isso, o projeto Kimberley que foi fundado em 2003, feito pela iniciativa do governo, apadrinhado pelas Nações Unidas, esse projeto tem como objetivo realizar a certificação dos diamantes, através de alguns mecanismos que conseguem realizar o controle com os produtos certificados, visando evitar o comércio ilegal (MARQUES, 2011).

O rastreamento da origem do diamante continua sendo o grande problema mesmo com os protocolos existentes, pois os consumidores querem ter a garantia que não estão financiando um trabalho escravo. Por isso, se tem o desafio dos comerciantes em garantir esse histórico confiável.

A própria WFDB (*World Federation of Diamond Bourses*) tem demonstrado o interesse e o reconhecimento de empresas que utilizam Blockchain em suas implementações e até mesmo apoiado os projetos existentes, apenas colocando como um critério que possa ser algo acessível para todos os usuários (BLOM, 2019).

Pode-se concluir que a utilização da Blockchain para solucionar o problema dos diamantes é algo real é que pode ser implementado nesse cenário como uma aplicação de exemplo para o foco deste trabalho.

3.2 Aplicação para controle de diamantes

Para garantir a integridade no mercado de diamantes foi desenvolvida uma aplicação simulando um ambiente onde os usuários possam ter o controle de suas pedras, esse sistema também vai servir para que uma determinada entidade possa gerenciar e monitorar se os vendedores ou exploradores estão seguindo as normas previstas, garantindo que

as seguintes regras de controle sejam respeitadas:

- O usuário pode realizar apenas a transferência de um item de sua posse legítima, garantindo que um diamante transferido não possa ter uma transferência duplicada ou a transferência de um item que tenha sido obtido de forma ilegal, como roubo ou extração predatória;
- O usuário não vai conseguir realizar a transferência de um diamante que não foi criado;
- Para o caso de produtores vai existir um limite máximo na produção de diamantes, portanto quando esse limite for atingido o próximo diamante cadastrado não será aceito. Esta limitação impede que os diamantes sejam obtidos de áreas de mineração clandestinas.

Tendo como foco apenas utilizar esse cenário para um exemplo real de aplicação utilizando Blockchain.

3.2.1 Arquitetura do sistema

A aplicação para controle de diamantes foi desenvolvida utilizando Java, Spring Framework, bibliotecas do Hypeledger para comunicar com a *API* e o Maven para gerenciar as dependências do projeto. O fluxo de funcionamento da aplicação se da seguinte forma demonstrada na Figura 2.

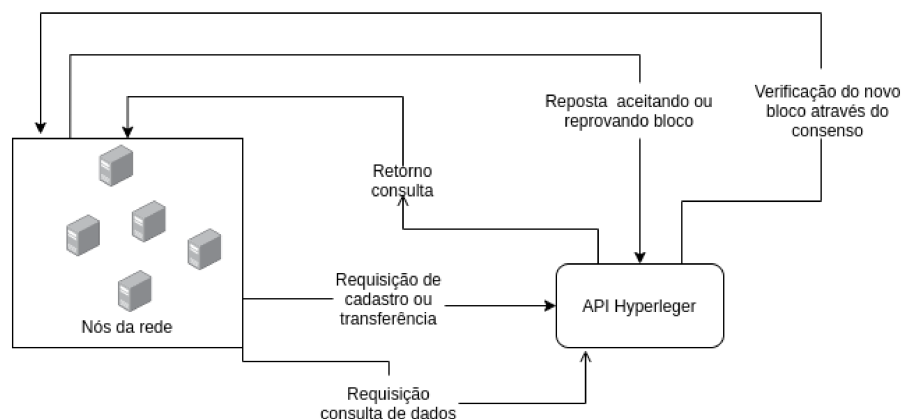


Figura 2 – Fluxo de comunicação da aplicação

No primeiro passo é disparada uma requisição do lado do cliente para o servidor da aplicação, podendo essa requisição ser a solicitação para visualizar ou gravar uma nova informação, em seguida a aplicação vai fazer a tratativa de toda a regra de negócio e encapsular essa requisição e fazer todo o processo necessário para que o Hyperledger possa entender e salvar esse bloco na Blockchain.

Para interagir com a aplicação o usuário do sistema deve realizar uma requisição HTTP(*Hypertext Transfer Protocol*), sendo disponível as seguintes funcionalidades:

Cadastrar diamante: permite ao usuário cadastrar um novo diamante;

Consultar diamantes: permite ao usuário verificar os diamantes que ele possui;

Transferir diamante: permite ao usuário realizar uma transferência para um outro usuário existente na rede.

3.3 Mecanismo de consenso utilizado na aplicação

O Hyperledger Sawtooth disponibiliza o mecanismo de consenso PBFT(*Practical Byzantine Fault Tolerance*) para casos onde é necessário utilizar mais de um nó para realizar as validações e até mesmo representar um ambiente real de consenso. Para o funcionamento desse meio de consenso é necessário, conforme descrito na documentação do Sawtooth ([BITWISE, 2020b](#)), que a rede tenha pelo menos quatro nós, caso a rede tenha menos que quatro nós ela não será iniciada e será apresentada falha.

Para subir mais de um nó utilizando o consenso PBFT, é necessário o arquivo `sawtooth-default-pbft.yaml` disponibilizado juntamente com a documentação do Hyperledger Sawtooth, nele pode ser configurado o número de nós que vão participar da rede, sendo necessário já ter sido configurado toda a instalação mencionada nas dependências do projeto para conseguir rodar a imagem no Docker. A seguir pode ser visto o comando para subir a imagem Docker com os nós configurados e a saída esperada do console:

```
marco@marco:~/Documentos$ docker-compose -f sawtooth-default-pbft.yaml up
Creating sawtooth-intkey-tp-python-default-3 ...
Creating sawtooth-intkey-tp-python-default-0 ...
Creating sawtooth-rest-api-default-4 ...
Creating sawtooth-settings-tp-default-0 ... done
Creating sawtooth-intkey-tp-python-default-2 ...
Creating sawtooth-pbft-engine-default-0 ...
Creating sawtooth-rest-api-default-3 ...
Creating sawtooth-xo-tp-python-default-2 ...
Creating sawtooth-validator-default-0 ...
Creating sawtooth-validator-default-2 ...
Creating sawtooth-pbft-engine-default-2 ...
Creating sawtooth-xo-tp-python-default-4 ...
Creating sawtooth-validator-default-4 ...
```

Figura 3 – Apresentação do comando executado no terminal para subir os nós.

3.3.1 Características do modelo de consenso

Conforme descrito na documentação ([BITWISE, 2020a](#)) o Sawtooth PBFT é um mecanismo de consenso que se baseia em votação e que tem as seguintes características:

- A segurança e a vitalidade da rede são garantidas mesmo tendo nós mal intencionados ou com algum problema, considerando que exista a quantidade mínima de nós na rede;

- O algoritmo utiliza um nó líder responsável por produzir blocos candidatos e os outros nós votam para aprovar ou não o bloco, sendo que o líder é alterado de maneira circular utilizando o algoritmo *round-robin*;
- Uso de comunicação constante para estabelecer o consenso e garantir um nó líder saudável;

3.3.2 Definição do nó líder

A escolha do nó líder na rede se dá de maneira sequencial percorrendo a lista circular com todos os nós da rede, obtendo-se o líder utilizando a fórmula:

$$p = v \pmod{n},$$

onde p é a posição do próximo nó líder na lista sequencial, v é a visualização que está ocorrendo no momento do cálculo, basicamente a cada trocar de líder é somado mais um para a visualização, n é quantidade de nós existentes na rede.

Na Figura 4 pode ser visto o estado inicial na primeira visualização que seria a de número 0 e o líder da rede seria o nó de número 0:

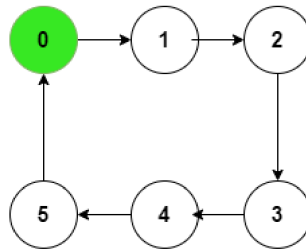


Figura 4 – Representação do líder nó 0 em uma rede com seis nós de forma circular.

Para definir o próximo nó vai ser utilizado a fórmula disponibilizada anteriormente, dessa maneira a próxima visualização seria a 1 e a rede possui 6 nós, então deve ser feito o cálculo utilizando $1 \pmod{6}$, tendo como resultado 1 que é a próxima posição da lista circular, como apresentado na Figura 5.

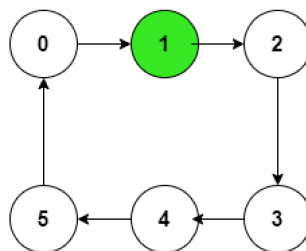


Figura 5 – Representação do líder nó 1 em uma rede com seis nós de forma circular.

Algo importante de se esclarecer é que quando chegar no último nó como está sendo utilizado uma lista circular vai ser retornado para o primeiro nó presente na lista. Como

apresentado a seguir em que temos o nó 5 como líder, no caso a próxima visualização vai ser a de número 6, nesse caso será utilizado o seguinte cálculo passando a visualização e a quantidade de nós presente na rede, da seguinte forma $6 \bmod 6$, tendo como resultado o nó de número 0, sendo o primeiro membro na lista e sendo o próximo na ordem, de acordo com a Figura 6.

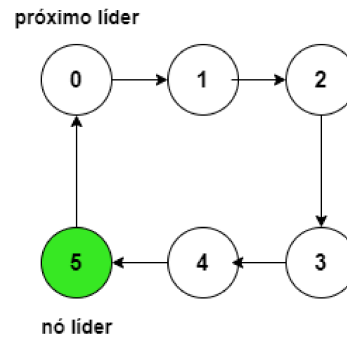


Figura 6 – Representação do ultimo nó sendo líder e o próximo líder.

Portanto cada um dos nós vai ser o líder seguindo a ordem da lista circular, sempre retornando ao primeiro nó no final da lista. Essa troca de nós pode ocorrer em um determinado intervalo que pode ser configurado.

3.3.3 Funcionamento do consenso para um bloco aprovado

A seguir será descrito o funcionamento do consenso na rede com cinco nós que é a quantidade utilizada para testes configurada anteriormente.

Primeiro o nó líder vai estar preparado para publicar um novo bloco e os outros nós secundários vão aguardar a chegada da mensagem. Com isso, o nó primário vai propor a criação de um novo bloco, depois de ser realizada todas as validações o bloco é transmitido pela rede.

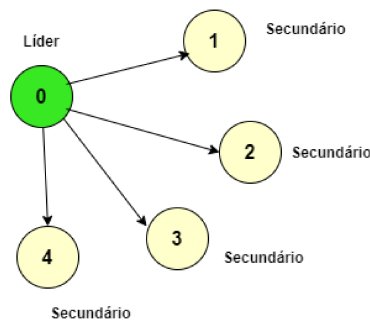


Figura 7 – Representação do envio da mensagem do nó líder aos nós secundários.

Depois disso, os nós secundários vão realizar a validação da mensagem, considerando que todos os nós realizaram a validação e que a mensagem foi aceita, cada um

dos participantes, exceto o líder, vai salvar aquela mensagem e enviar para os outros nós secundários, inclusive o nó líder vai receber a mensagem nessa fase de preparo.

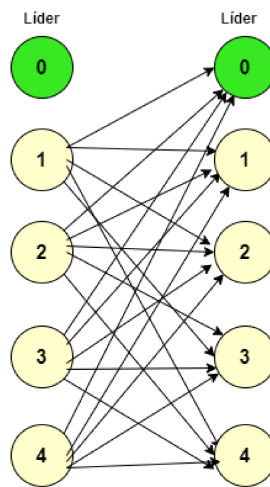


Figura 8 – Representação da transmissão de mensagem aceita pelos nós secundários.

Na próxima fase todos os nós, inclusive o líder, realizam o envio de mensagem para *commit*. Nesse caso, é esperado em todos os participantes que eles tenham uma mensagem presente em seu log. A Figura 9 demonstra o envio da mensagem de *commit*.

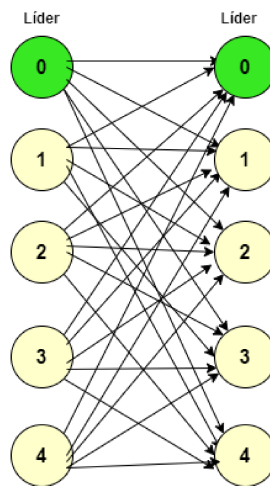


Figura 9 – Representação da transmissão de mensagem de *commit* por todos os nós da rede.

Recebida a mensagem cada um dos nós vai confirmar o bloco com seu validador e após o bloco ser salvo os outros nós aguardam novamente um novo bloco ser criado.

3.3.4 Funcionamento do consenso para um bloco não aprovado

A seguir será descrito o funcionamento do consenso para o caso de um bloco não aprovado por conter alguma inconsistência, podendo o nó que gerou esse bloco ser um

fraudador ou ter realizado algum procedimento errado no momento de gerá-lo. Primeiro o nó líder vai enviar o bloco para que os nós secundários possam realizar a validação.

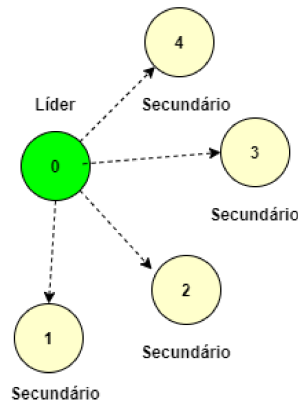


Figura 10 – Representação da transmissão da mensagem invalida pelo nó líder para os nós secundários.

Depois da mensagem ser enviada para os nós secundários vai ocorrer o processo de validação, para verificar se aquele bloco pode ser salvo na cadeia existente. Para esse caso vamos assumir que o líder enviou um novo bloco que vai ser recusado pelos outros nós da rede.

Após o bloco ser enviado, os nós secundários vão realizar a validação. Nesse caso, será verificado que aquele bloco não pode ser integrado à rede, sendo assim será replicada uma mensagem para os outros nós recusando o bloco, segundo apresentado na Figura 11.

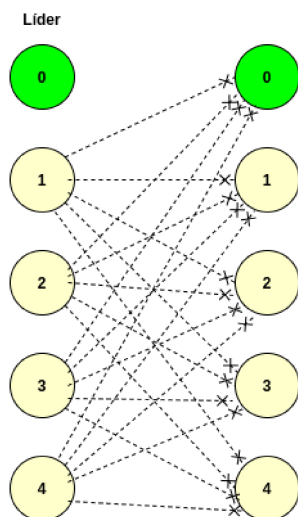


Figura 11 – Representação da resposta dos nós secundários invalidando o novo bloco.

Depois do consenso entre os nós membros da rede o novo bloco não será persistido, conforme a Figura 11 demonstrada anteriormente os nós secundários enviam uma mensagem para os outros nós da rede. Nessa mensagem, existe a reprovação do bloco e o

comando para selecionar um novo líder, como demonstrado na Seção 3.3.2, o próximo nó vai ser o líder.

3.4 Testes realizados

Nessa seção serão descritos os teste realizados para as funcionalidades de cadastro, transferência e consulta de diamantes, buscando garantir o desenvolvimento realizado na aplicação. O envio das requisições foi realizado utilizando o Postman, programa destinado para o teste de APIs (*Application Programming Interface*) visando simplificar o desenvolvimento desse tipo de sistema (POSTMAN, 2021).

3.4.1 Cadastro de diamante aceito

No primeiro cenário de teste será validado o cadastro de um diamante realizado com sucesso, sendo criado por um usuário com permissão na rede, para criar o novo diamante devem ser passados os seguintes parâmetros conforme apresentado na Figura 12:

privateKey: chave privada do usuário, utilizada para assinar as transações e confirmar a autenticidade de quem envia a requisição, sendo feita a validação pelos outros membros da rede através da chave pública do usuário;

description: uma descrição sobre o diamante que está sendo cadastrado.

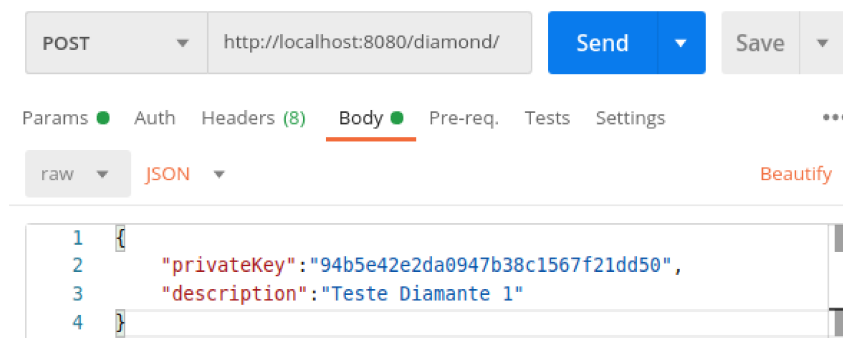


Figura 12 – Requisição para criar diamante valido.

Após o envio da requisição vai ser retornada uma URL (*Uniform Resource Locator*) originada da API do Hyperledger, como apresentado na Figura 13, essa URL de retorno permite consultar o status da transação enviada. Podendo os status apresentados nos testes utilizados serem os seguintes:

Committed: significa que a transação foi aceita através do consenso, persistindo o novo bloco com as informações passadas;

Invalid: significa que a transação não foi aceita através do consenso, dessa maneira não será persistido o novo bloco.

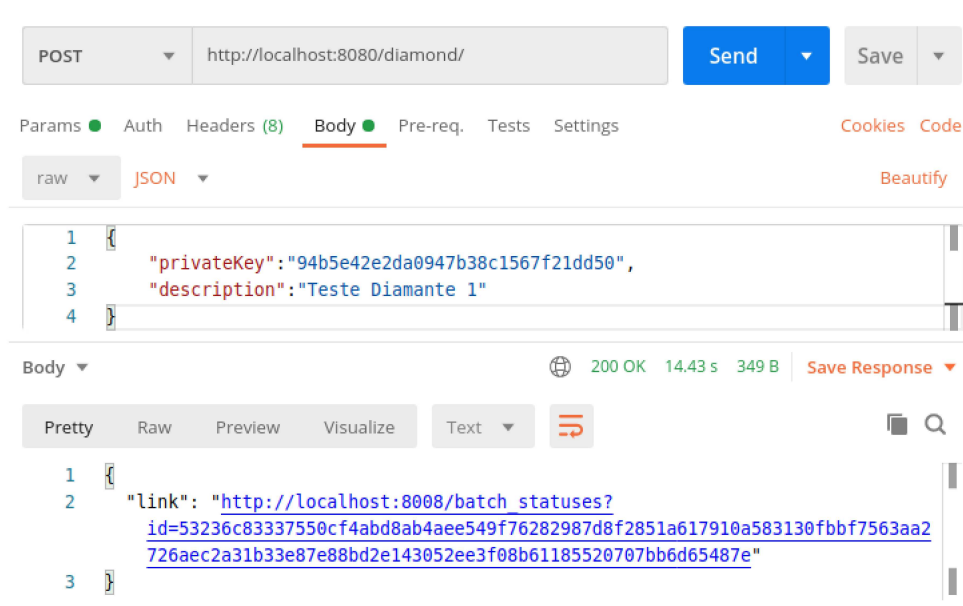


Figura 13 – Resposta com a URL para consultar status da transação.

Utilizando a URL é necessário consultar o status da transação para verificar se a mesma foi aceita, realizando a consulta é possível verificar que o status da transação é *committed*, como demonstrado na Figura 14.

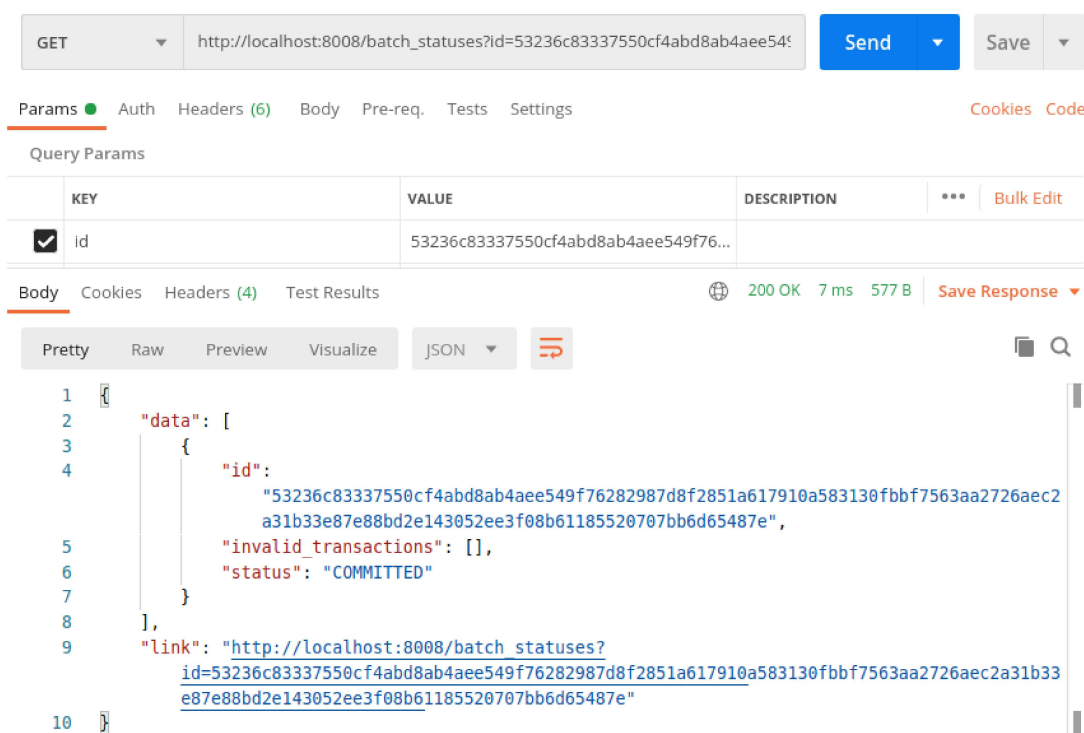


Figura 14 – Consulta na API do Hyperledger

Depois de ser confirmado que a transação foi aceita, precisa ser validado se o novo diamante está sendo apresentado no momento em que o usuário consultar os seus diamantes, informando a chave privada dele no corpo da requisição. Na Figura 15 é possível verificar o retorno da requisição, contendo as seguintes informações:

address: endereço do bloco;

publicKeyOwner: chave pública do usuário proprietário do diamante;

description: descrição feita ao realizar o cadastro do diamante;

diamondsPreviousAddress: item preenchido em casos onde o diamante foi recebido por transferência, sendo preenchido com o endereço da transferência anterior, utilizado para manter um histórico das transferências realizadas e no algoritmo de consenso implementado.

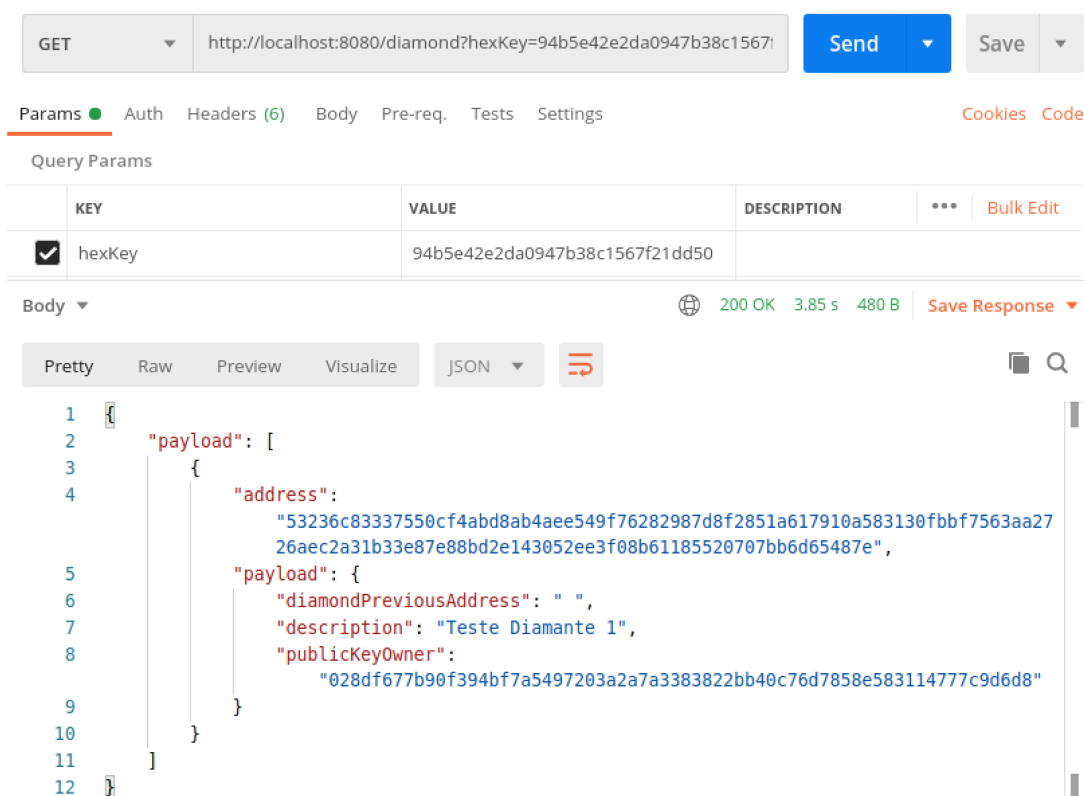


Figura 15 – Retorno de consulta de diamantes.

3.4.2 Transferência de diamante aceita

Nesse segundo teste será realizada a transferência do diamante para um outro usuário da aplicação, sendo necessário ao usuário que realiza a transferência informar sua

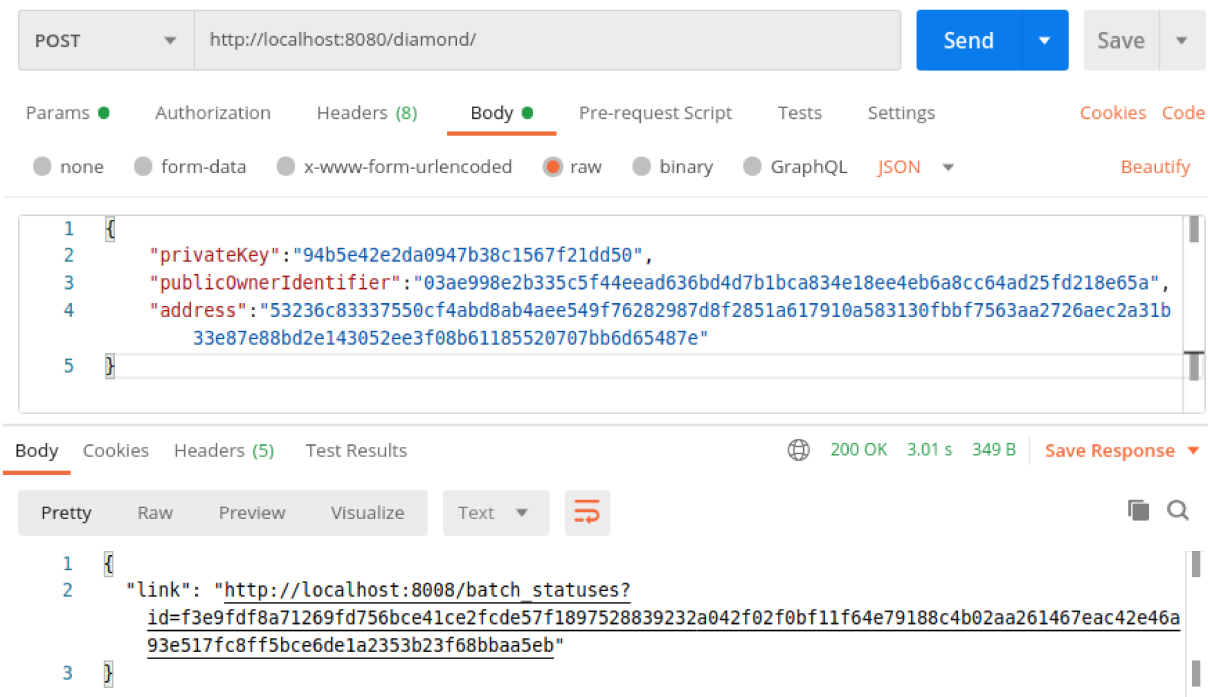


Figura 16 – Exemplo de requisição para transferir diamante.

chave privada, o endereço do diamante criado e o hexadecimal da chave pública do usuário que vai receber o diamante. Sendo apresentado um exemplo de requisição na Figura 16.

Depois de enviada a requisição de transferência vai ser retornada a URL para consultar o status na API do Hyperledger, realizando a consulta vai ser apresentado o status de *committed* na resposta, como evidenciado na Figura 17.

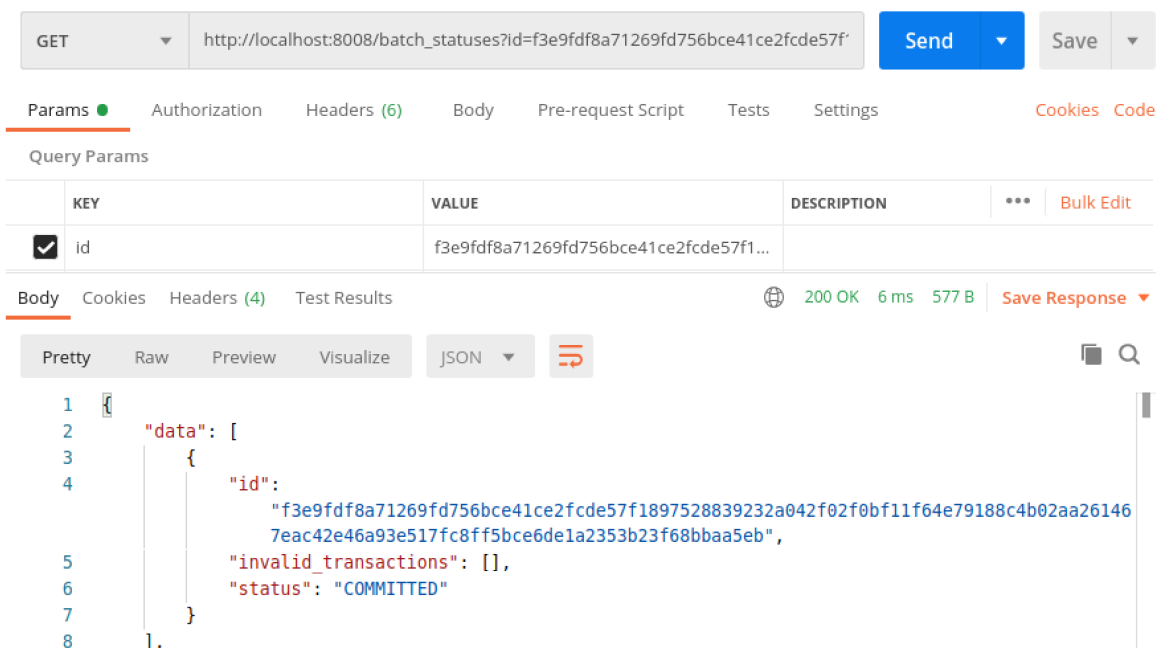


Figura 17 – Consulta do status da transferência realizada.

Comprovado que o diamante foi transferido, será necessário verificar se quando o primeiro usuário consultar seus diamantes, não será retornado o diamante transferido. Na Figura 18 pode ser evidenciado que nenhuma informação foi retornada.

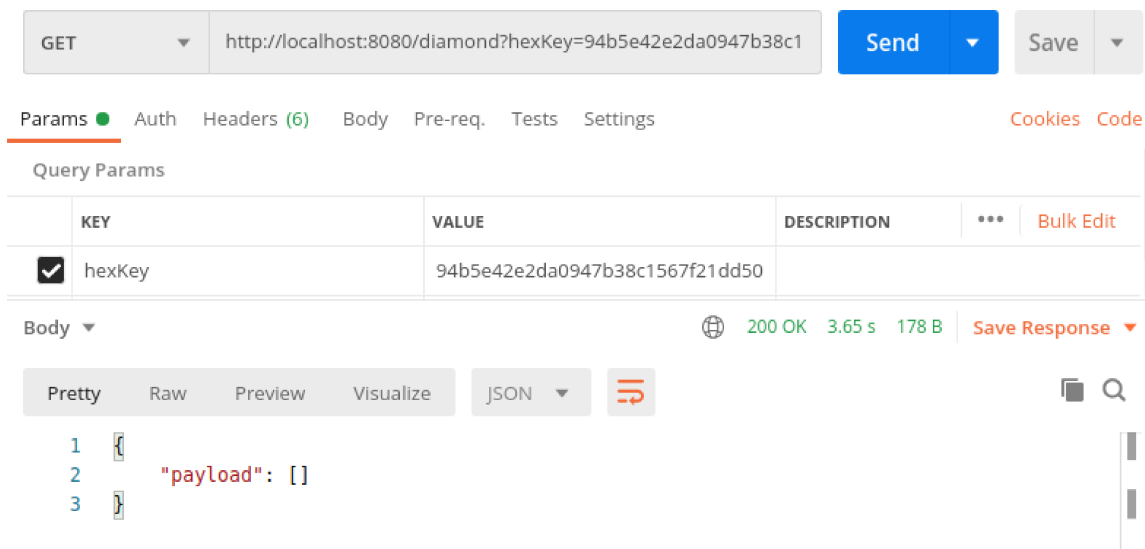


Figura 18 – Retorno dos diamantes do usuário que realizou a transferência.

Para concluir o teste, o próximo passo necessário é verificar se o diamante vai aparecer no momento em que o segundo usuário realizar requisição de consulta. Conforme apresentado na Figura 19, pode ser comprovado que o diamante está sendo retornado após a consulta do usuário que recebeu a transferência.

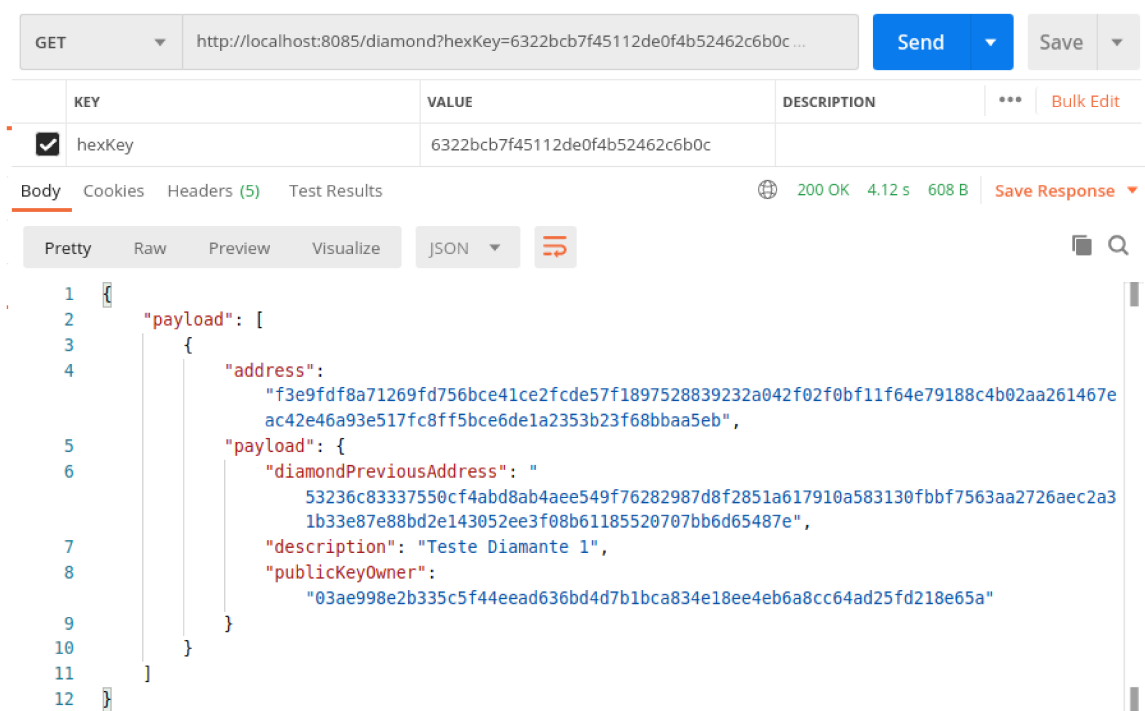


Figura 19 – Retorno de diamantes do usuário que recebeu a transferência.

3.4.3 Transferência de diamante já transferido

Para garantir a integridade da aplicação, nesse segundo cenário de teste será validado que o diamante enviado, não vai poder ser transferido novamente pelo usuário que não detém a posse do diamante. Na Figura 20 pode ser visto a requisição enviada pelo usuário, em conjunto também consta o retorno com a URL para consultar o status da transferência.

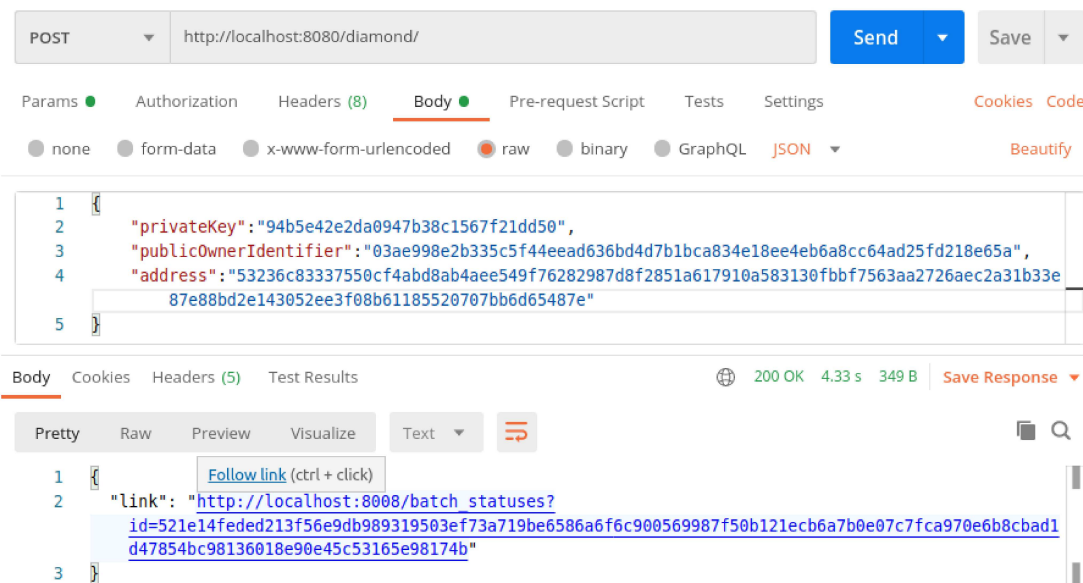


Figura 20 – Requisição de transferência realizado pelo usuário não detentor do diamante.

Consultando o status da transação pode ser verificado que os status é *invalid* apresentado na Figura 21, portanto a transferência não foi aceita pelo consenso, sendo esse o resultado esperado e garantindo a não violação por um usuário fraudador.

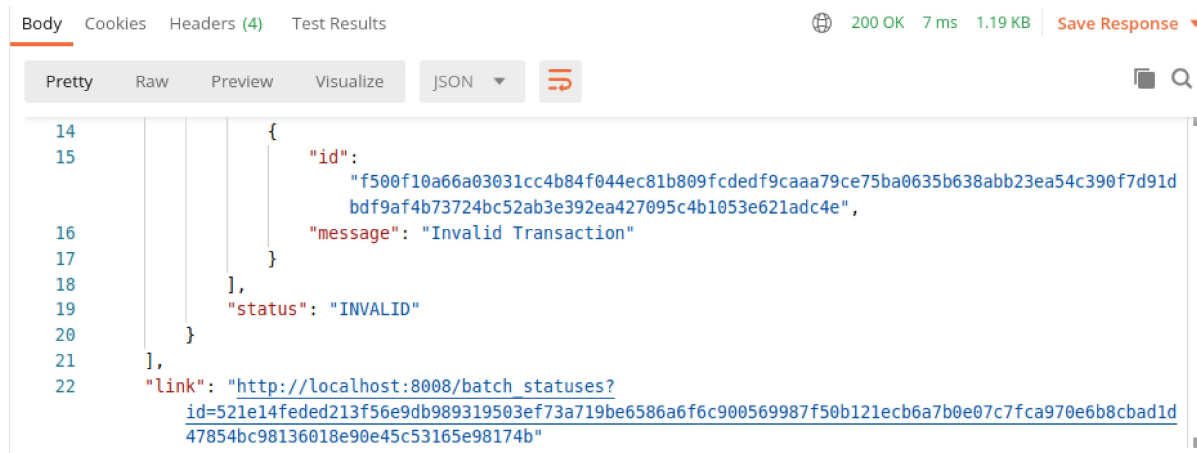


Figura 21 – Retorno de requisição invalida.

3.4.4 Transferência de diamante não existente

Nesse cenário de teste vai ser validado a tentativa de envio de um diamante não existente. O usuário realiza esse envio passando um endereço de transferência não existente, na Figura 22 pode ser observado o exemplo da requisição e o retorno para consultar o status da transferência.

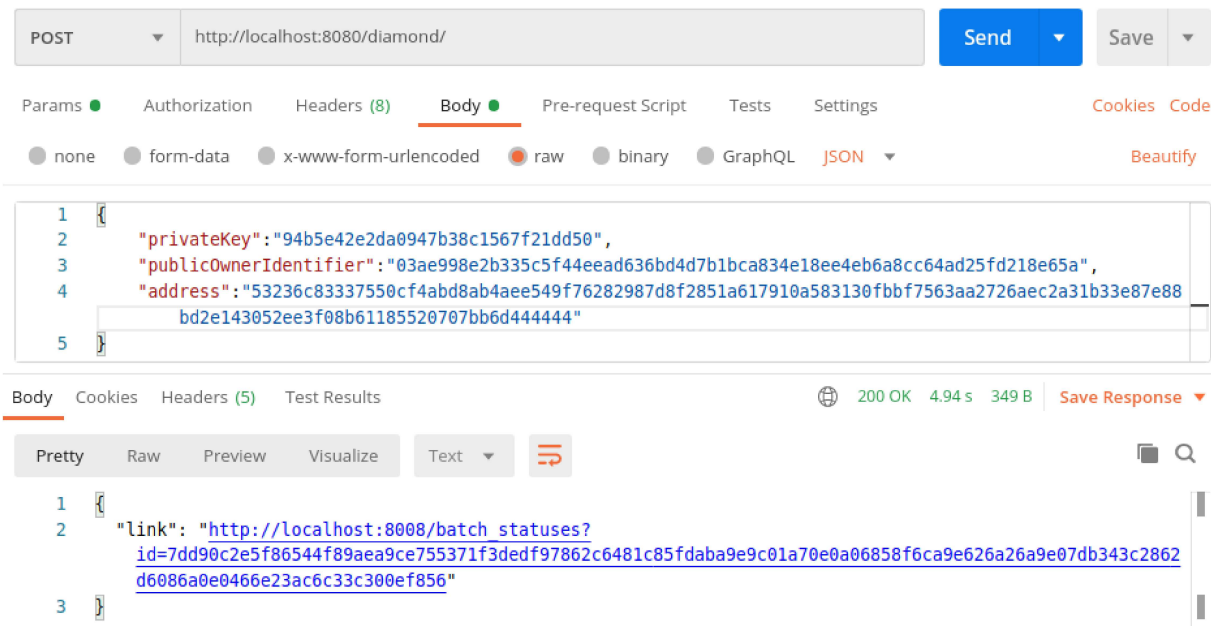


Figura 22 – Requisição de transferência de um diamante não existente.

Consultando o status da transferência, pode ser constatado que o resultado é o esperado, estando o status dela como *invalid*, congruente com o apresentado na Figura 23. Sendo assim também é garantido que os usuários não vão conseguir realizar o envio de um diamante não existente para outro usuário.



Figura 23 – Retorno de requisição invalida.

3.4.5 Limite de produção atingido

Nessa ultima seção será apresentado um cenário onde o produtor atingiu o número máximo possível de diamantes cadastrados, onde a quantidade máxima utilizada para esse teste são de três diamantes, podendo esse valor ser alterado de acordo com a necessidade proposta. Previamente foram cadastrados os diamantes, na Figura 24 pode ser visualizado o retorno dos três diamantes já cadastrados para o produtor utilizado.

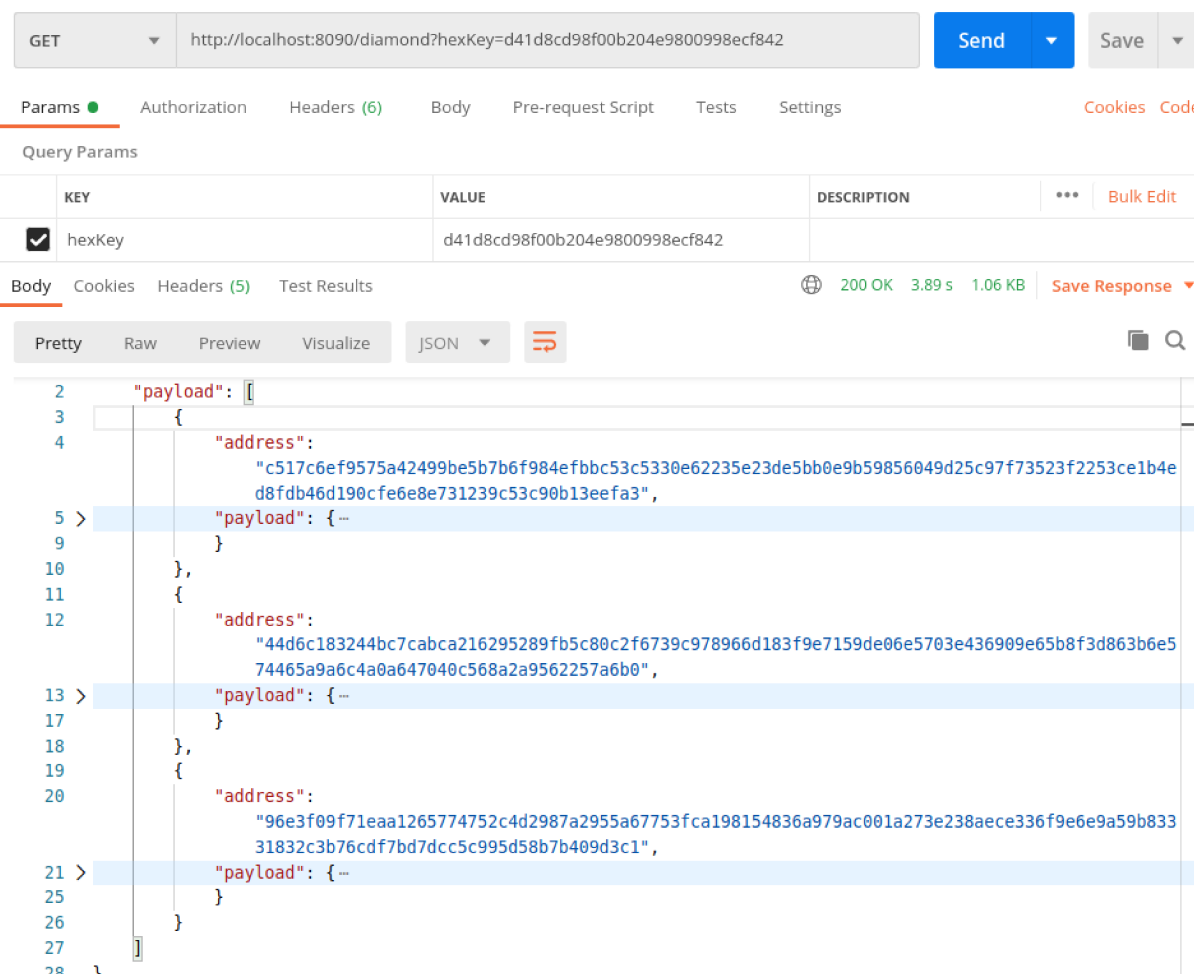


Figura 24 – Retorno de consulta dos diamantes do produtor utilizado no teste.

Depois de ter cadastrado os três diamantes vai ser realizado o envio do quarto diamante, com a intenção de verificar se o produtor vai conseguir realizar o cadastro. Na Figura 25 é apresentado o exemplo de envio do quarto diamante, contendo o retorno para consultar o status da transferência.

Realizado o envio o próximo passo é verificar se o cadastro não foi aceito, sendo esse o resultado esperado, considerando que o produtor possui o número máximo de pedras permitidas. O retorno com o status do cadastro pode ser evidenciado na Figura 26, podendo ser constatado o status *invalid* que é o esperado para esse caso, respeitando o limite de cadastro de diamantes permitido.

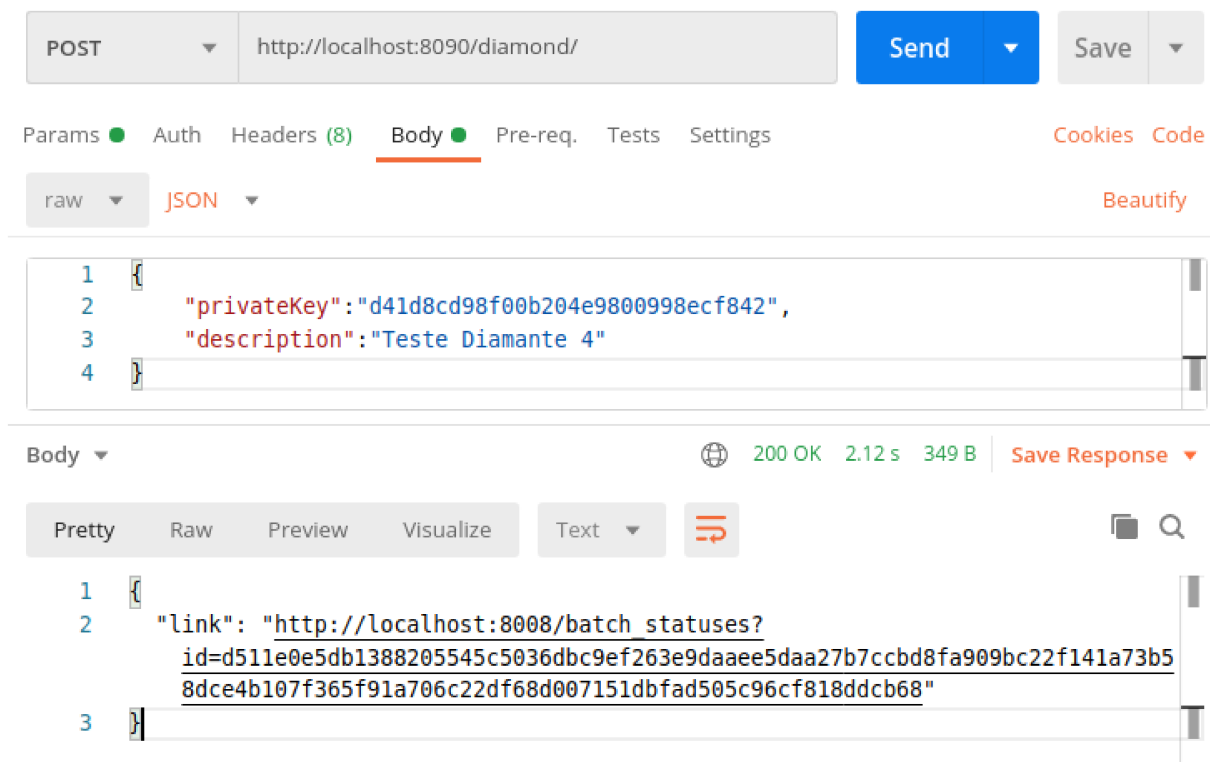


Figura 25 – Apresentação do cadastro do quarto diamante.

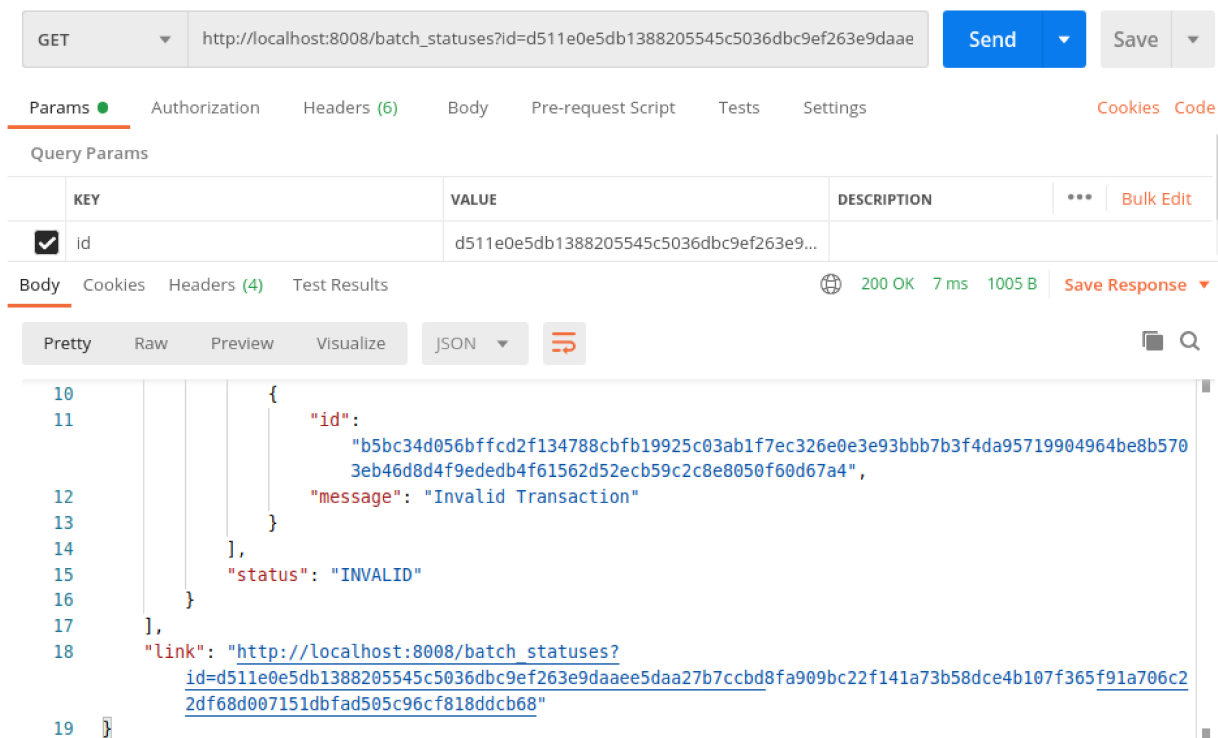


Figura 26 – Retorno da consulta com o status da transferência.

4 Conclusão

Com o desenvolvimento da aplicação realizada neste trabalho foi possível demonstrar a viabilidade de implementação de um sistema que precisa ser descentralizado e ao mesmo tempo prover confiança nas transações. O uso da ferramenta Hyperledger tornou a implementação de uma solução baseada em Blockchain uma tarefa relativamente fácil: as necessidades da aplicação escolhida foram atendidas pela ferramenta de forma bem transparente. Assim podemos afirmar que o desenvolvimento de aplicações mais complexas, porém com os mesmos tipos de requisitos, não deve encontrar muitas dificuldades. Apesar do Hyperledger facilitar essa implementação, é importante lembrar que no presente momento do desenvolvimento deste trabalho, essa é uma ferramenta não muito utilizada, sendo que não é possível encontrar uma quantidade considerável de informações além da documentação da aplicação, como por exemplo: demonstrações ou aplicações práticas realizando o uso dessa ferramenta. Portanto, para utilizar o Hyperledger é necessário ter como principal apoio a documentação oficial dessa solução.

Ainda, pode ser afirmado que é possível desenvolver uma rede descentralizada, para assim adicionar mais confiança dos clientes que participam da rede e fazem o uso da aplicação, tudo isso possibilitado através do mecanismo de consenso, onde não existe a manipulação de uma entidade e minimizando a possibilidade de fraude nos dados presentes no sistema. Com o mecanismo de consenso também se tem uma maior transparência nas transações realizadas, com a validação realizada pelos nós participantes da rede. Além de garantir que os blocos que não podem ser aprovados não serão persistidos, não se tendo a aprovação de informações publicadas por nós fraudadores. E sem a possibilidade de excluir ou editar um bloco, garantindo que todas as informações presentes na aplicação são autênticas e não sofreram algum tipo de alteração, então em qualquer aplicação que a Blockchain estiver implementada, será garantido que a informação presente na cadeia de blocos é a mesma desde o seu ciclo de vida inicial.

Portanto, conclui-se que a Blockchain, pode ser adicionada e agregar uma camada de confiança nas aplicações existentes e que necessitam de seu uso.

Referências

ANTONPOULOS, A. *Mastering Bitcoin*. 1. ed. Gravenstein Highway North: O'Reilly Media, 2014. Citado 2 vezes nas páginas 11 e 16.

APACHE. *Apache Maven Project*. 2020. Disponível em: <<https://maven.apache.org/>>, último acesso em 2020-11-09. Citado na página 21.

BASHIR, I. *Mastering Blockchain Second Edition*. Birmingham, United Kingdom: Packt Publishing Ltd, 2018. Citado na página 15.

BASHIR, I.; PRUSTY, N. *Advanced Blockchain Development*. Birmingham: Packt Publishing Ltd, 2019. Citado na página 18.

BITWISE. *Introduction Sawtooth PBFT*. 2020. Disponível em: <<https://sawtooth.hyperledger.org/docs/pbft/releases/latest/introduction-to-sawtooth-pbft.html>>, último acesso em 2020-11-11. Citado na página 27.

BITWISE. *Using PBFT Consensus*. 2020. Disponível em: <<https://sawtooth.hyperledger.org/docs/pbft/releases/latest/using-pbft-consensus.html>>, último acesso em 2020-11-11. Citado na página 27.

BLOM, E. *How Traceability, Provenance and Blockchain Technology Are Becoming the New Mantras in Rough Diamond Trading*. 2019. Disponível em: <<https://www.wfdb.com/media-news-press/presidents-blogs/952-how-traceability-provenance-and-blockchain-technology-are-becoming-the-new-mantras-in-rough-diamond-trading>>, último acesso em 2020-03-25. Citado na página 25.

CHUEN, D. L. K. *Handbook of Digital Currency*. 1. ed. Singapore Management University: Nikki Levy, 2015. Citado na página 13.

DOCKER. *O que é um contêiner?* 2019. Disponível em: <<https://www.docker.com/resources/what-container>>, último acesso em 2020-11-02. Citado na página 20.

DOCKER. *Why Docker?* 2019. Disponível em: <<https://www.docker.com/why-docker>>, último acesso em 2019-11-20. Citado na página 20.

EVERLEDGER. *Tech for Good Blockchain Solutions*. 2021. Disponível em: <<https://www.everledger.io/>>, último acesso em 2021-05-25. Citado na página 13.

HARGRAVE, S. J. *What Is Blockchain?* United States of America: O'Reilly Media, 2020. Citado 4 vezes nas páginas 14, 15, 18 e 19.

HYPERLEDGER. *Hyperledger Fabric*. 2018. Disponível em: <<https://www.hyperledger.org/projects/fabric>>, último acesso em 2019-11-20. Citado na página 20.

HYPERLEDGER. *Hyperledger Indy*. 2018. Disponível em: <<https://www.hyperledger.org/projects/hyperledger-indy>>, último acesso em 2019-11-20. Citado na página 20.

HYPERLEDGER. *Hyperledger Iroha*. 2018. Disponível em: <<https://www.hyperledger.org/projects/iroha>>, último acesso em 2019-11-20. Citado na página 20.

HYPERLEDGER. *O que é a Hyperledger?* 2018. Disponível em: <<https://pt-br.hyperledger.org/>>, último acesso em 2019-11-20. Citado na página 19.

HYPERLEDGER. *Hyperledger Sawtooth*. 2020. Disponível em: <<https://www.hyperledger.org/use/sawtooth>>, último acesso em 2020-11-09. Citado na página 20.

KARNOUPAKIS, E.; HARGRAVE, S. J. *Blockchain Success Stories*. [S.l.]: O'Reilly Media, 2020. Citado na página 11.

MARQUES, R. *Diamantes de Sangue - Corrupção e Tortura em Angola*. Lisboa, Portugal: Tinta-da-china, 2011. Citado na página 25.

MORIN, P. *Open Data Structures (in pseudocode)*. 1. ed. [S.l.], 2014. Citado na página 15.

POSTMAN. *API Platform: API Tools Solutions*. 2021. Disponível em: <<https://www.postman.com/api-platform/>>, último acesso em 2021-03-05. Citado na página 32.

RAJ, K. *Foundations of Blockchain*. Birmingham, United Kingdom: Packt Publishing, 2019. Citado na página 14.

UK GOVERNMENT DIGITAL SERVICE. *Distributed Ledger Technology: beyond block chain*. UK Government: Open Government Licence, 2015. Disponível em: <https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/492972/gs-16-1-distributed-ledger-technology.pdf>, último acesso em 2020-11-11. Citado na página 13.

YUAN, M. J. *Building Blockchain Apps*. [S.l.]: Packt Publishing Ltd, 2019. Citado na página 18.

Apêndices

APÊNDICE A – Algoritmo para criar bloco

```

1  @Component
   public class SawtoothUtils {
3
   private static final String FAMILY_NAME = "diamond";
5   private static final String FAMILY_VERSION = "1.0";

7   public byte[] sendMessageToSawtooth(String description, String
       diamondsPreviousAddress, String hexKey, String publicKeyOwner)
       throws NoSuchAlgorithmException, UnsupportedEncodingException {
       Secp256k1PrivateKey secp256k1PrivateKey = Secp256k1PrivateKey.
           fromHex(hexKey);
9       Secp256k1Context context = new Secp256k1Context();

11      Signer signer = new Signer(context, secp256k1PrivateKey);

13      ByteArrayOutputStream messageEncoded = null;

15      if(publicKeyOwner != null) {
           messageEncoded = encodeMessage(description,
               diamondsPreviousAddress, publicKeyOwner);
17      } else {
           messageEncoded = encodeMessage(description,
               diamondsPreviousAddress, signer.getPublicKey().hex());
19      }

21      TransactionHeader transactionheader = createTransactionHeader(
           messageEncoded, signer);
       Transaction transaction = createTransaction(transactionheader,
           messageEncoded, signer);
23      BatchHeader batchHeader = createBatchHeader(transaction, signer);
       Batch batch = createBatch(batchHeader, transaction, signer);

25      return batchToByteBatchList(batch);

27  }

29

   public ByteArrayOutputStream encodeMessage(String description, String
       diamondsPreviousAddress, String publicKeyOwner) {
31      ByteArrayOutputStream payload = new ByteArrayOutputStream();
       try {
33          new CborEncoder(payload).encode(new CborBuilder()
               .addMap()
35              .put("description", description)

```

```

        .put("diamondsPreviousAddress", diamondsPreviousAddress
            == null ? "" : diamondsPreviousAddress)
37        .put("publicKeyOwner", publicKeyOwner)
        .end()
39        .build());
    } catch (CborException e) {
41        e.printStackTrace();
    }
43    return payload;
}

45    public String getAddress(String familyName) throws
        UnsupportedEncodingException {
47        return Utils.hash512(familyName.getBytes("UTF-8")).substring(0, 6);
    }

49    public String hash(ByteArrayOutputStream payload) throws
        NoSuchAlgorithmException {
51        MessageDigest messageDigest = MessageDigest.getInstance("SHA-512");
        messageDigest.reset();
53        messageDigest.update(payload.toByteArray());
        return BaseEncoding.base16().toLowerCase().encode(messageDigest.
            digest());
55    }

57    public TransactionHeader createTransactionHeader(ByteArrayOutputStream
        messageEncoded, Signer signer) throws NoSuchAlgorithmException,
        UnsupportedEncodingException {
        TransactionHeader header = TransactionHeader.newBuilder()
59            .setSignerPublicKey(signer.getPublicKey().hex())
            .setFamilyName(FAMILY_NAME)
61            .setFamilyVersion(FAMILY_VERSION)
            .addInputs(getAddress(FAMILY_NAME))
63            .addOutputs(getAddress(FAMILY_NAME))
            .setPayloadSha512(hash(messageEncoded))
65            .setBatcherPublicKey(signer.getPublicKey().hex())
            .setNonce(UUID.randomUUID().toString())
67            .build();

69        return header;
    }

71    public Transaction createTransaction(TransactionHeader
        transactionHeader, ByteArrayOutputStream messageEncoded, Signer
        signer) throws NoSuchAlgorithmException {
73        String signature = signer.sign(transactionHeader.toByteArray());

```

```

75     Transaction transaction = Transaction.newBuilder()
        .setHeader(transactionHeader.toString())
77     .setPayload(ByteString.copyFrom(messageEncoded.toByteArray()
        ()))
        .setHeaderSignature(signature)
79     .build();

81     return transaction;
83 }

public BatchHeader createBatchHeader(Transaction transaction, Signer
signer) {
85     List<Transaction> transactions = new ArrayList();
        transactions.add(transaction);

87     BatchHeader batchHeader = BatchHeader.newBuilder()
        .setSignerPublicKey(signer.getPublicKey().hex())
89     .addAllTransactionIds(
        transactions
91         .stream()
93         .map(Transaction::getHeaderSignature)
        .collect(Collectors.toList())
95     )
        .build();

97     return batchHeader;
99 }

public Batch createBatch(BatchHeader batchHeader, Transaction
transaction, Signer signer) {
101     List<Transaction> transactions = new ArrayList();
        transactions.add(transaction);
103

105     String batchSignature = signer.sign(batchHeader.toByteArray());

107     Batch batch = Batch.newBuilder()
        .setHeader(batchHeader.toString())
109     .addAllTransactions(transactions)
        .setHeaderSignature(batchSignature)
111     .build();

113     return batch;
115 }

public byte[] batchToByteBatchList(Batch batch) {
117     byte[] batchListBytes = BatchList.newBuilder()
        .addBatches(batch)

```

```
119         .build()
120         .toArray();
121
122     return batchListBytes;
123 }
}
```

APÊNDICE B – Algoritmo para comunicar com Hyperledger

```

1 public class SawtoothGateway {
2
3     @Autowired
4     SawtoothUtils sawtoothUtils;
5
6     public ResponseEntity postDiamond(String description, String
7         diamondsPreviousAddress, String hexKey, String publicKeyOwner)
8         throws NoSuchAlgorithmException, UnsupportedEncodingException,
9         CborException {
10         String urlPost = ConfigurationConstants.SAWTOOTH_POST_BATCH_URL;
11         byte[] sawtoothRequest = null;
12
13         if (diamondsPreviousAddress != null) {
14             RestTemplate restTemplate = new RestTemplate();
15             ResponseEntity<HyperledgerReturn> responseHyperledger =
16                 restTemplate.getForEntity(ConfigurationConstants.
17                     SAWTOOTH_GET_BLOCKS, HyperledgerReturn.class, String.class);
18             String descriptionDiamond = getDescriptionAndLocation(
19                 diamondsPreviousAddress, responseHyperledger);
20             sawtoothRequest = sawtoothUtils.sendMessageToSawtooth(
21                 descriptionDiamond, diamondsPreviousAddress, hexKey,
22                 publicKeyOwner);
23         } else {
24             sawtoothRequest = sawtoothUtils.sendMessageToSawtooth(
25                 description, "", hexKey, publicKeyOwner);
26         }
27
28         ResponseEntity<String> response = null;
29
30         RestTemplate restTemplate = new RestTemplate();
31         try {
32             response = restTemplate.postForEntity(urlPost, sawtoothRequest,
33                 String.class);
34         } catch (Exception e) {
35             e.printStackTrace();
36         }
37         return response;
38     }
39
40     public ArrayList<HashMap<String, Object>> getDiamonds(String hexKey) {

```

```

32     ArrayList<HashMap<String , Object>> diamonds = new ArrayList();
33     String urlGet = ConfigurationConstants.SAWTOOTH_GET_BLOCKS;
34     ResponseEntity<HyperledgerReturn> response = null;

36     RestTemplate restTemplate = new RestTemplate();

38     try {
39         response = restTemplate.getForEntity(urlGet , HyperledgerReturn.
40             class , String.class);
41         diamonds = filterDiamondsByUserAndFamily(response , getPublicKey
42             (hexKey) , ConfigurationConstants.DIAMOND_FAMILY);
43         filterAndRemoveDiamondsTransferred(response , diamonds ,
44             ConfigurationConstants.DIAMOND_FAMILY);

46     } catch (Exception e) {
47         e.printStackTrace();
48     }
49     return diamonds;
50 }

52 public ArrayList<HashMap<String , Object>> filterDiamondsByUserAndFamily
53 (ResponseEntity<HyperledgerReturn> response , String publicKey ,
54 String family) throws CborException {
55     ArrayList<HashMap<String , Object>> diamonds = new ArrayList();

56     for (DataHyperledger dataHyperledger : response.getBody().
57         getDataHyperledger()) {
58         for (Batch batch : dataHyperledger.getBatchList()) {
59             for (Transaction transaction : batch.getTransaction()) {
60                 if (transaction.getHeader().getFamilyName().equals(
61                     family)) {
62                     String payload = transaction.getPayload();
63                     byte[] decodedBytes = Base64.getDecoder().decode(
64                         payload);
65                     co.nstant.in.cbor.model.Map decodedMap = (co.nstant
66                         .in.cbor.model.Map) CborDecoder.decode(
67                             decodedBytes).get(0);
68                     String[] payloadArray = decodedMap.getValues().
69                         toString().replaceAll("\\[", "").replaceAll("\\]",
70                             "").split(",");
71                     String description = payloadArray[0];
72                     String publicKeyPayload = payloadArray[1];
73                     String diamondPreviousAddress = payloadArray[2];
74                     publicKeyPayload = publicKeyPayload.replaceAll(" ",
75                         "");

76                     if (publicKeyPayload.equals(publicKey)) {

```

```

66         HashMap<String , Object> diamondHashMap = new
            HashMap<>();
        HashMap<String , String> payloadMap = new
            HashMap();
        payloadMap.put("description", description);
68         payloadMap.put("publicKeyOwner",
            publicKeyPayload);
        payloadMap.put("diamondPreviousAddress",
            diamondPreviousAddress);
70
        diamondHashMap.put("address", batch.
            getHeaderSignature());
72         diamondHashMap.put("payload", payloadMap);
        diamonds.add(diamondHashMap);
74     }
76     }
78     }
80     return diamonds;
82 }

public ArrayList<HashMap<String , Object>>
filterAndRemoveDiamondsTransferred(ResponseEntity<HyperledgerReturn>
response , ArrayList<HashMap<String , Object>> diamonds , String family
) throws CborException {
84     for (DataHyperledger dataHyperledger : response.getBody().
        getDataHyperledger()) {
        for (Batch batch : dataHyperledger.getBatchList()) {
86             for (Transaction transaction : batch.getTransaction()) {
                if (transaction.getHeader().getFamilyName().equals(
                    family)) {
88                     String payload = transaction.getPayload();
                    byte[] decodedBytes = Base64.getDecoder().decode(
                        payload);
90                     co.nstant.in.cbor.model.Map decodedMap = (co.nstant
                        .in.cbor.model.Map) CborDecoder.decode(
                            decodedBytes).get(0);

92                     for (int i = 0; i < diamonds.size(); i++) {
                        HashMap<String , Object> diamondHashMap =
                            diamonds.get(i);
94                         String address = diamondHashMap.get("address").
                            toString();

96                         if (decodedMap.toString().contains(address)) {

```

```

108         diamonds.remove(i);
109     }
110 }
111 }
112 }
113 }
114 }
115 }
116 }
117 }
118 }
119 }
120 }
121 }
122 }
123 }
124 }
125 }
126 }
127 }
128 }
129 }
130 }
131 }
132 }
133 }
134 }

    public String getDescriptionAndLocation(String lastAddress,
        ResponseEntity<HyperledgerReturn> response) throws CborException {
        String description = "";
        for (DataHyperledger dataHyperledger : response.getBody().
            getDataHyperledger()) {
            for (Batch batch : dataHyperledger.getBatchList()) {
                for (Transaction transaction : batch.getTransaction()) {
                    if (transaction.getHeader().getFamilyName().equals(
                        ConfigurationConstants.DIAMOND_FAMILY)) {
                        if (batch.getHeaderSignature().equals(lastAddress)) {
                            {
                                String payload = transaction.getPayload();
                                byte[] decodedBytes = Base64.getDecoder().
                                    decode(payload);
                                co.nstant.in.cbor.model.Map decodedMap = (co.
                                    nstant.in.cbor.model.Map) CborDecoder.decode
                                    (decodedBytes).get(0);
                                String[] payloadArray = decodedMap.getValues().
                                    toString().replaceAll("\\\\", "\\").split(",");
                                description = payloadArray[0];
                            }
                        }
                    }
                }
            }
        }
        return description;
    }

    public String getPublicKey(String hexKey) {
        Secp256k1PrivateKey secp256k1PrivateKey = Secp256k1PrivateKey.
            fromHex(hexKey);
        Secp256k1Context context = new Secp256k1Context();

        Signer signer = new Signer(context, secp256k1PrivateKey);

        return signer.getPublicKey().hex();
    }

```

136	} }
-----	-------------------------------------

APÊNDICE C – Algoritmo de Consenso

```

1  public class DiamondTransactionHandler implements TransactionHandler {
3
5      private String diamondNameSpace;
7
9      public DiamondTransactionHandler() {
10         try {
11             this.diamondNameSpace = Utils.hash512(
12                 this.transactionFamilyName().getBytes("UTF-8")).
13                 substring(0, 6);
14         } catch (UnsupportedEncodingException use) {
15             use.printStackTrace();
16             this.diamondNameSpace = "";
17         }
18     }
19
20     @Override
21     public String transactionFamilyName() {
22         return "diamond";
23     }
24
25     @Override
26     public String getVersion() {
27         return "1.0";
28     }
29
30     @Override
31     public Collection<String> getNameSpaces() {
32         ArrayList<String> namespaces = new ArrayList<>();
33         namespaces.add(this.diamondNameSpace);
34         return namespaces;
35     }
36
37     @Override
38     public void apply(TpProcessRequest tpProcessRequest, State state)
39         throws InvalidTransactionException, InternalError {
40         ArrayList<String> publicKeyList = loadPublicKeys();
41
42         try {
43             co.nstant.in.cbor.model.Map decodedMap = (co.nstant.in.cbor.
44                 model.Map) CborDecoder.decode(tpProcessRequest.getPayload().
45                 toByteArray()).get(0);
46             String[] payloadArray = decodedMap.getValues().toString()

```

```

        .replaceAll("\\\\[", "\\")
        .replaceAll("\\\\]", "\\")
        .split(",");
String lastAddress = payloadArray[2].replaceAll(" ", "");
String publicKey = payloadArray[1].replaceAll(" ", "");

if (lastAddress == null || lastAddress.isEmpty()) {
    RestTemplate restTemplate = new RestTemplate();
    ResponseEntity<HyperledgerReturn> response = restTemplate.
        getForEntity(ConfigurationConstants.SAWTOOTH_GET_BLOCKS,
            HyperledgerReturn.class, String.class);

    SawtoothGateway sawtoothGateway = new SawtoothGateway();
    ArrayList<HashMap<String, Object>> diamonds =
        sawtoothGateway.filterDiamondsByUserAndFamily(response,
            publicKey, ConfigurationConstants.DIAMOND_FAMILY);
    if ((!(userHavePermissionToCreateDiamond(publicKey,
        publicKeyList))) || diamonds.size() >=
        ConfigurationConstants.AMOUNT_MAXIMUM_OF_DIAMONDS)) {
        throw new InvalidTransactionException("Invalid
            Transaction");
    }
} else {
    ResponseEntity<HyperledgerReturn> response = null;
    RestTemplate restTemplate = new RestTemplate();
    response = restTemplate.getForEntity(ConfigurationConstants
        .SAWTOOTH_GET_BLOCKS, HyperledgerReturn.class, String.
        class);

    if (!(isValidAddressAndSignature(lastAddress,
        tpProcessRequest.getHeader().getSignerPublicKey(),
        response))) {
        throw new InvalidTransactionException("Invalid
            Transaction");
    }

    if (diamondHasTransferred(lastAddress, response)) {
        throw new InvalidTransactionException("Invalid
            Transaction");
    }
}
} catch (CborException e) {
    throw new InvalidTransactionException("Invalid Transaction");
}
}

```

```

73     public boolean userHavePermissionToCreateDiamond(String publicKey,
74         ArrayList<String> publicKeyList) {
75         for (String publicKeyUser : publicKeyList) {
76             if (publicKeyUser.equals(publicKey)) {
77                 return true;
78             }
79         }
80         return false;
81     }
82
83     public ArrayList<String> loadPublicKeys() {
84         ArrayList<String> publicKeyList = new ArrayList();
85         publicKeyList.add(ConfigurationConstants.PUBLIC_KEY_1);
86         publicKeyList.add(ConfigurationConstants.PUBLIC_KEY_2);
87         publicKeyList.add(ConfigurationConstants.PUBLIC_KEY_3);
88         publicKeyList.add(ConfigurationConstants.PUBLIC_KEY_4);
89         publicKeyList.add(ConfigurationConstants.PUBLIC_KEY_5);
90
91         return publicKeyList;
92     }
93
94     public boolean isValidAddressAndSignature(String lastAddress, String
95         key, ResponseEntity<HyperledgerReturn> response) throws
96         CborException {
97         for (DataHyperledger dataHyperledger : response.getBody().
98             getDataHyperledger()) {
99             for (Batch batch : dataHyperledger.getBatchList()) {
100                 for (Transaction transaction : batch.getTransaction()) {
101                     if (transaction.getHeader().getFamilyName().equals(
102                         ConfigurationConstants.DIAMOND_FAMILY)) {
103                         String payload = transaction.getPayload();
104                         byte[] decodedBytes = Base64.getDecoder().decode(
105                             payload);
106                         co.nstant.in.cbor.model.Map decodedMap = (co.nstant
107                             .in.cbor.model.Map) CborDecoder.decode(
108                             decodedBytes).get(0);
109                         String[] payloadArray = decodedMap.getValues().
110                             toString().split(",");
111                         String publicKey = payloadArray[1].replaceAll("\\\\",
112                             "").replaceAll(" ", "");
113
114                         if (batch.getHeaderSignature().equals(lastAddress)
115                             && publicKey.equals(key)) {
116                             return true;
117                         }
118                     }
119                 }
120             }
121         }
122     }

```

```

109     }
110   }
111   return false;
112 }
113
114 public boolean diamondHasTransferred(String diamondAddress,
115   ResponseEntity<HyperledgerReturn> response) throws CborException {
116   for (DataHyperledger dataHyperledger : response.getBody().
117     getDataHyperledger()) {
118     for (Batch batch : dataHyperledger.getBatchList()) {
119       for (Transaction transaction : batch.getTransaction()) {
120         if (transaction.getHeader().getFamilyName().equals(
121           ConfigurationConstants.DIAMOND_FAMILY)) {
122           String payload = transaction.getPayload();
123           byte[] decodedBytes = Base64.getDecoder().decode(
124             payload);
125           co.nstant.in.cbor.model.Map decodedMap = (co.nstant
126             .in.cbor.model.Map) CborDecoder.decode(
127               decodedBytes).get(0);
128           String[] payloadArray = decodedMap.getValues().
129             toString().split(",");
130           String lastAddress = payloadArray[2];
131           lastAddress = lastAddress.replaceAll(" ", "").
132             replaceAll("]", "");
133
134           if (lastAddress.equals(diamondAddress)) {
135             return true;
136           }
137         }
138       }
139     }
140   }
141   return false;
142 }
143 }

```