VINICIUS PEREIRA NUNES

INTEGRAÇÃO DE LEITOR NFC DE CELULAR COM BANCO DE DADOS DISTRIBUÍDO HYPERLEDGER IROHA POR COMUNICAÇÃO MQTT

Uberlândia, Minas Gerais

2024

VINICIUS PEREIRA NUNES

INTEGRAÇÃO DE LEITOR NFC DE CELULAR COM BANCO DE DADOS DISTRIBUÍDO HYPERLEDGER IROHA POR COMUNICAÇÃO MQTT

Monografia de Conclusão de Curso apresentada no curso de graduação em Engenharia Mecatrônica da Universidade Federal de Uberlândia, como parte dos requisitos para obtenção de título de **BACHAREL EM ENGENHARIA MECATRÔNICA**. Área de concentração: Engenharia Mecatrônica

Universidade Federal de Uberlândia

Faculdade de Engenharia Mecânica

Orientador: José Jean-Paul Zanlucchi de Souza Tavares

Uberlândia, Minas Gerais

2024

NUNES, Vinícius Pereira. **Integração de um sistema de RFID com banco de dados distribuído Hyperledger Iroha por comunicação MQTT**. 2024. Trabalho de Conclusão de Curso de Graduação em Engenharia Mecatrônica – Universidade Federal de Uberlândia, Uberlândia, 2024.

RESUMO

A tecnologia RFID é amplamente utilizada no ambiente empresarial, pois facilita a rastreabilidade da movimentação de itens e ativos em estoque e sua identificação, mas há o problema no compartilhamento das informações quando existem diversas organizações participantes no processo, o que poderia provocar divergência de dados entre eles. Com isso, esse trabalho tem o objetivo de propor uma forma de reduzir esse tipo de inconsistência com a implementação de um banco de dados distribuído através da blockchain Hyperledger Iroha, a qual permite que qualquer empresa, desde que obtenha as permissões necessárias, consiga acessar remotamente as informações de seus itens. Para isso, foi necessário modelar os processos através de uma conexão MQTT com um leitor de *tags* RFID do tipo NFC de celulares e o banco de dados distribuído *Iroha*. O resultado foi um aplicativo móvel que consulta informações da blockchain hospedada em servidor local baseado na leitura e gravação de *tags* NFC.

Palavras-chave: Blockchain. Banco de dados distribuído. Hyperledger Iroha. IoT. NFC. MQTT.

NUNES, Vinícius Pereira. **Integration of an RFID system with Hyper-ledger Iroha distributed database via MQTT communication**. 2024. Monograph of the Mechatronics Engineering Course Completion, Federal University of Uberlândia, Uberlândia, 2024.

ABSTRACT

RFID technology is widely used in the business environment, as it facilitates the traceability of the movement of items and assets in stock and their identification, but there is a problem in sharing information between the participating organizations of the process causing data divergence between them. Thus, this work aims to propose a way to minimize this inconsistency with the implementation of a distributed database through the Hyperledger Iroha blockchain, which allows any company, if they obtain the necessary permissions, to be able to remotely access the information of their items. To do this, it was necessary to model the processes through a MQTT connection with a cellphone NFC RFID tag reader and the Iroha distributed database. The result was a mobile application that consults the information from the blockchain hosted on localhost based on NFC tag reading and writing.

Keywords: Blockchain. Distributed database. Hyperledger Iroha. IoT. NFC. MQTT.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
BFT	Byzantine Fault Tolerance
CFT	Crash Fault Tolerance
DLT	Distributed Ledger Tecnology
ECDSA	Elliptic Curve Digital Signature Algorithm
HF	High Frequency
HL	Hyperledger Iroha
IoT	Internet of Things
JSON	JavaScript Object Notation
LF	Low Frequency
MQTT	Message Queuing Telemetry Transport
NDEF	NFC Data Exchange Format
NFC	Near Field Communication
P2P	Peer-to-Peer
PBFT	Practical Byzantine Fault Tolerance
PoET	Proof of Elapsed Time
RAM	Random Access Memory
RFID	Radio-Frequency Identification
SDK	Software Development Kit
TCP/IP	Transmission Control Protocol/Internet Protocol
UHF	Ultra-high Frequency
UI	User Interface

SUMÁRIO

1 INTRODUÇÃO

Identificação de rádio frequência ou RFID, é uma tecnologia baseada na emissão de ondas de rádio a uma determinada frequência para uma etiqueta (*tag)* a fim de rastrear informações do objeto *(DOMDOUZIS, 2007)*. Este tipo de tecnologia é amplamente implementado no mundo empresarial atual para substituir o ultrapassado sistema de código de barras (WANT, 2006). De acordo com Want, o sistema é composto por um leitor, uma antena e uma *tag.* No caso de uma *tag* passiva, através da indução eletromagnética o leitor emite um campo para a *tag* a fim de ler ou gravar informações; a antena emite e recebe as informações; e a etiqueta armazena os dados e o transmite tão logo é solicitado pela indução de uma antena.

O sistema RFID tem diversas aplicações em diversos campos, pode ser usado em hospitais para monitorar informações de pacientes, em sistemas de rastreamento de transporte de mercadorias e até mesmo em sistemas de pagamento através da tecnologia NFC (THANAPAL, 2017).

NFC é um sistema de transferência de dados RFID, também disponível em alguns *smartphones,* o qual opera na faixa de alta frequência e utiliza o formato de troca de dados NDEF (*NFC Data Exchange Format)* (Want, 2011).

Apesar de eficiente, o sistema *tag* e leitor RFID não possui a função de armazenamento de dados históricos, ou seja, não há a disponibilidade de todo o processo de produção na etiqueta, somente uma informação. Há o problema do custo dos leitores e *tags* RFID, a configuração do leitor RFID para cada parte do processo de uma cadeia de produção, e o fato da arquitetura Cliente-Servidor apresentar problemas de comunicação em ambientes altamente distribuídos. Isso pode ser corrigido com a implementação de um aplicativo de celular com leitor NFC embarcado que realize a comunicação com a *tag* via NFC e um banco de dados distribuído entre as partes envolvidas.

1.1 OBJETIVOS

Este trabalho possui como objetivo geral a criação de um banco de dados distribuído para registro de informações simulando uma cadeia de produção com base em dados de aplicativos de celulares integrados com leitores NFC. O banco de dados deve ser acessível remotamente para acompanhamento em tempo real dos registros realizados pelo leitor de RFID e leitores NFC de celulares. Os objetivos específicos deste projeto são:

1 Integrar um sistema de leitura RFID do tipo NFC embarcado em celular com um banco de dados distribuído.
2 Criar um sistema seguro e acessível em qualquer localização.
3 Utilizar o protocolo MQTT como mediador na comunicação entre os dispositivos.
4 Desenvolver um aplicativo móvel que realize a comunicação com o usuário e o banco de dados.

1.2 JUSTIFICATIVA

A cadeia de produção é uma rede de organizações composto de vários processos e atividades e vai desde a criação à venda do produto ao consumidor final (MENTZER et al, 2001). Cada companhia envolvida no processo possui sua própria logística, tecnologia e metodologia do processo, o que resulta na divergência de informações, imprecisão na rastreabilidade do produto, perda de suprimentos no decorrer dos processos, atrasos entre as etapas e prejuízo financeiro às companhias.

Com o avanço das tecnologias, há um crescimento na necessidade da digitalização de processos de uma cadeia de suprimentos devido a necessidade de uma conectividade inter e intraorganizacional. Com isso, as empresas vêm implementando ferramentas como internet das coisas (IoT), tecnologia *Blockchain*, inteligência artificial, RFID, entre outros (REJEB; KEOGH; TREIBLMAIER, 2019).

Os dispositivos IoT, juntamente com a tecnologia RFID, são utilizados em processos em cadeias de suprimentos, pois estes facilitam o processo de

rastreamento de remessas utilizando de tecnologias de monitoramento em tempo real, como o GPS. O uso dessas tecnologias é fundamental, pois garante à empresa competitividade de mercado. Ainda assim, há uma certa carência em conectar os diversos processos de produção principalmente quando eles se encontram em empresas distintas. Este trabalho atende estes requisitos, pois ele visa incluir as informações do processo em um banco de dados distribuído e acessível à todas as organizações atuantes.

Esse trabalho está organizado da seguinte forma, a saber, o capítulo 2 apresenta a fundamentação teórica do projeto, seguindo a metodologia utilizada e resultados obtidos no capítulo 3. A seguir é apresentada a conclusão e, por fim, a referência bibliográfica no capítulo 5.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 *REACT NATIVE*

2.1.1 *React*

React é uma biblioteca de código aberto, criada para desenvolvimento de interfaces de usuário (UI) em Javascript. Pode ser utilizada tanto para desenvolvimento Web como *mobile*. Sua grande vantagem é a possibilidade de criar pequenos componentes reutilizáveis, originando um sistema complexo (REACT, 2024).

2.1.2 Introdução ao *React Native*

React Native é um *framework* de código aberto criado pelo Facebook para desenvolvimento de aplicações iOS e Android utilizando *React*, sem que haja a necessidade de utilizar uma linguagem por plataforma (BODUCH, 2017).

Ao invés de utilizar um Modelo de Documento por Objetos (DOM), como é feito em HTML, o *React Native* utiliza chamadas assíncronas para cada sistema operacional (Android ou iOS), o qual, em seguida, aciona as API's nativas dos widgets, como no esquema da Figura 1 (BODUCH, 2017). Isso garante facilidade de desenvolvimento e otimização no funcionamento das aplicações *React Native*.

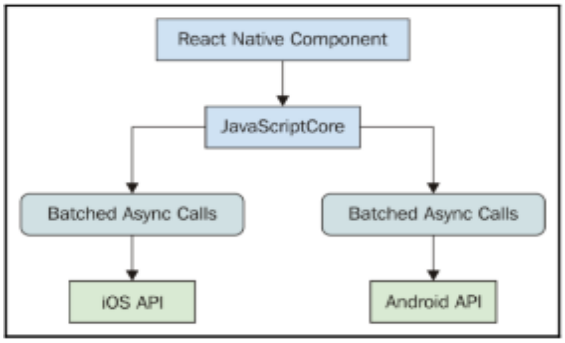

Figura 1 - Funcionamento do *React Native*

Fonte: BODUCH, Adam. React and React Native. Packt Publishing Ltd, 2017, p. 214.

2.2 TECNOLOGIA RFID

Segundo Hessel (2012), o RFID é uma tecnologia composta por um microchip acoplado a uma antena que identificam os objetos os quais são fixados a ele. Essa tecnologia foi desenvolvida com a proposta de substituir scanners ópticos, pois apesar de seus benefícios apresenta suas limitações. Embora ambas possuem semelhanças, o sistema RFID tem inúmeras vantagens em relação à outra tecnologia, como a ausência de necessidade da presença humana para manusear o leitor, alta usabilidade em diferentes tipos de superfícies e a tecnologia RFID dispensa o contato visual entre a *tag* e o leitor.

Os componentes do sistema RFID são as *tags*, antenas, leitores e *Middleware* (CHINELATTO, 2010). Na Figura 2 é representada a arquitetura básica do funcionamento dessa tecnologia.

Figura 2 - Esquema de funcionamento do sistema de RFID.

Nesse sistema, o leitor envia um sinal de radiofrequência através da antena, criando um campo eletromagnético em direção à etiqueta. Quando o sinal encontra uma *tag*, este é processado pelo microchip e transmitido de volta para o leitor, o qual transmite o sinal para um *Middleware* e armazena a informação para uma base de dados.

Há três modos de comunicação RFID: passivo, ativo e semi-passivo. A comunicação passiva é a mais utilizada, sendo ela composta por uma etiqueta que não necessita de alimentação integrada. Sua alimentação é obtida pelo próprio campo magnético. A comunicação ativa envolve uma *tag* com alimentação independente e, como consequência, esta consegue trabalhar em maiores distâncias entre leitor e etiqueta. Já na comunicação semi-passiva, a *tag* também possui alimentação interna, porém ainda necessita do campo eletromagnético do leitor para enviar suas informações (PAULA, 2021).

2.2.1 Etiquetas RFID (*Tags)*

A maioria das etiquetas RFID possuem somente antena e microchip. Segundo VOWELS (2010) o chip das etiquetas é composto por processador, memória e rádio transmissor. As etiquetas ativas e semipassivas também possuem uma bateria interna, portanto são mais caras. Essas são utilizadas em aplicações que exigem uma maior distância, como sistema de pedágios. As *tags* podem ser somente leitura ou leitura e gravação

As etiquetas passivas operam em diferentes faixas de frequência: baixa frequência (LF), alta frequência (HF) e ultra altas frequências (UHF) (VOWELS, 2010). A Tabela 1 apresenta um comparativo entre os tipos de *tags* em função da sua frequência. A faixa LF possui uma curta distância (até 10 cm) de leitura e pode ler poucas etiquetas simultaneamente. A HF é a mais utilizada em *tags* comerciais, pois possui uma distância de leitura razoável (até 30 cm), tem um baixo custo e pode ler várias etiquetas ao mesmo tempo. A UHF pode ler *tags* de 3 a 6 metros e é amplamente utilizada em veículos para realizar o pagamento automático de pedágios.

Tabela 1 - Comparativo entre faixas de frequência de *tags* RFID

Faixa de frequência	**Faixa de frequência** (Hz)	**Distância de leitura**	**Capacidade de leitura**
LF	125 a 134,2 kHz	Curta (até 0,1 m)	Poucas etiquetas simultaneamente
HF	13,56 MHz	Média (0,3 m)	Várias etiquetas simultaneamente
UHF	300 MHz a 3 GHz	De 3 a 6 metros	Muitas etiquetas simultaneamente

A frequência mais utilizada nas etiquetas é a HF (13,56 MHz) e é amplamente utilizada em cartões de transporte coletivo, pagamento por aproximação (NFC), entre outros.

As *tags* RFID também se classificam pela quantidade de armazenamento e pelo processamento (CHINELATO, 2010). Existem as *tags* de 1 bit que são responsáveis por armazenar somente a informação 0 ou 1. Podem ser utilizadas em sistemas anti-furto de lojas, onde a informação 0 é quando o produto foi pago pelo cliente e 1 quando não foi e deve disparar o leitor na entrada. Algumas situações exigem com que haja o maior número de informações, nesse caso utilizam-se *tags* de capacidade de 1 bit até 8 Kbytes. Sua tecnologia de armazenamento pode ser EEPROM (*Eletric Erasable and Programable Read-Only Memory*), RAM (*Random Access Memory*) ou SRAM (*Static Random Access Memory*).

A memória da etiqueta é dividida em 4 partes:

1. Memória reservada: parte da memória para reservar a *kill pasword*, responsável por desativar permanentemente a *tag*;
2. Memória EPC: essa parte da memória contém o código de produto eletrônico ou *Electronic Product Code* (EPC), deve possuir no mínimo 96 bits;
3. Memória TID: contém o número de identificação criado pelo fabricante;
4. Memória do usuário: nas *tags* que contém esse compartimento de memória, é onde são gravadas as informações enviadas pelo leitor.

2.2.2 Antenas e leitores

O leitor tem a função de se enviar sinais para a *tag* através da antena, as informações são processadas e enviadas a uma interface com o usuário, chamado de *Middleware*. Outro componente importante no sistema é o controlador do leitor, o qual é responsável por efetuar o disparo do leitor para realizar leituras ou gravações (CHINELATO, 2010).

As antenas podem ser mono-estáticas ou bi-estáticas. A primeira requer apenas uma antena para enviar e receber os sinais eletromagnéticos, enquanto a segunda possui uma antena para cada função.

O posicionamento das etiquetas pode provocar o efeito de sombreamento, o qual se refere a interferência de outras etiquetas quando estas se encontram empilhadas, evitando com que o sinal seja enviado para a etiqueta correta.

2.2.3 *Middleware RFID*

O *Middleware* é uma camada de software responsável por transformar as informações coletadas pelo leitor e transformá-las em dados para disponibilização às aplicações (CHINELATO, 2010). Em um ambiente industrial, onde são utilizados vários leitores simultâneos é necessária uma única interface que seja capaz de acessar as informações de todos. A Figura 3 apresenta as camadas de aplicação do *Middleware*. As camadas são:

1. Adaptador de leitor: essa camada interage com o próprio leitor de RFID, traduzindo os dados da *tag* para o formato que o *Middleware* possa trabalhar.
2. Gerenciador de evento: responsável por processar os dados obtidos através da camada anterior. Essa camada filtra e organiza os eventos evitando leituras desnecessárias.
3. Interface em nível de aplicação: camada que realiza a comunicação dos dados filtrados para aplicações de interação com o usuário final.

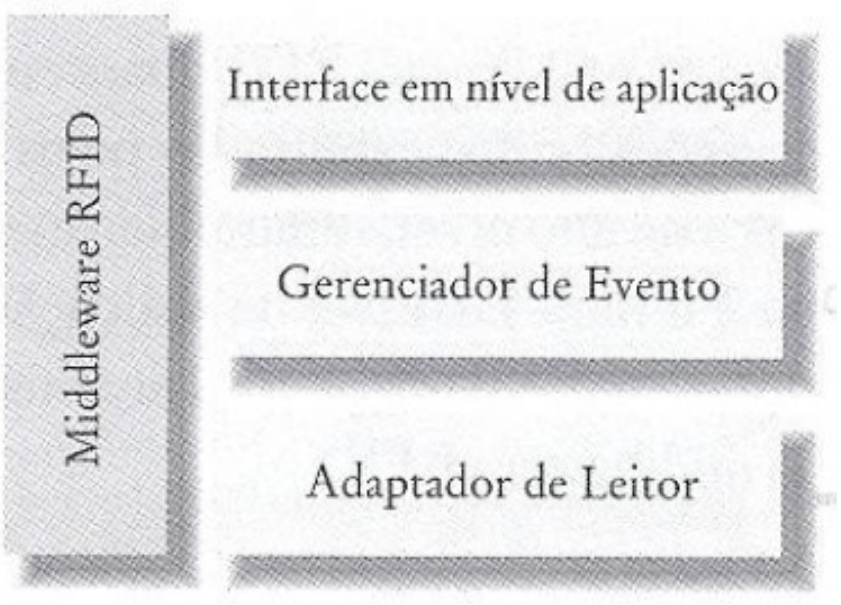

Figura 3 - Camadas de aplicação do *Middleware*

Fonte: (GLOVER, BHATT, 2007)

2.2.4 NFC (*Near Field Communication*)

NFC é uma comunicação de pequenas distâncias que foi desenvolvido pela primeira vez em 2002 pela Philips e Sony para comunicações sem contato, a qual contribuiu para diversas tecnologias conhecidas hoje em dia como a internet das coisas (IoT) (COSKUN, 2015).

A comunicação ocorre entre dois dispositivos compatíveis com a tecnologia onde um dispositivo inicia (iniciador) e o outro, conhecido como target apenas responde. Smartphones NFC utilizam sua própria bateria como fonte de energia, enquanto as *tags* utilizam a energia do campo magnético gerado pelo iniciador.

De acordo com ANDROID, 2024, existem 3 tipos de comunicação entre smartphones com NFC:

1. Modo leitura/gravação: método de comunicação entre um dispositivo e uma *tag*;
2. Modo P2P: permite a troca de informações entre dois dispositivos, o que possibilita a troca de dados diversos, como vídeos e fotos. Esse serviço é

chamado de Android Beam e foi descontinuado a partir do Android 14 em 2022. O iOS não oferece suporte a esse tipo de comunicação.

3. Modo de emulação de cartões: é o método mais utilizado na atualidade. Consiste em o dispositivo NFC simular um cartão para ser lido por um dispositivo externo como terminal de ponto de venda NFC.

Os dispositivos Android utilizam o método de troca de dados conhecido como NDEF (*NFC Exchange Data Format*), o qual encapsula as informações em um registro contendo cabeçalho, tipo de dados, ID, e a carga útil. Para realizar comunicação com a *tag*, esta deve ser previamente formatada em NDEF para que os dados sejam lidos/gravados da forma correta (ANDROID, 2024).

2.3 MQTT (*Message Queuing Telemetry Transport)*

2.3.1 Conceito

O MQTT foi desenvolvido em 1990 pela IBM com o objetivo de criar um método de transmissão de mensagens leves para locais com baixa velocidade de conexão. Somente quando o Facebook™ e a Apple™ utilizaram em aplicações de mensagens, o MQTT passou a ser amplamente reconhecido (GOODRICH, 2022).

Este método de troca de mensagens utiliza o protocolo TCP/IP, e divide a mensagem em pequenos pacotes. O MQTT é uma escolha adequada para comunicação para IoT pois utiliza números binários, o que torna mais fácil o processamento das informações pelos dispositivos. O esquema de funcionamento do MQTT é representado na Figura 4. Um usuário publica informação em um tópico e envia ao *Broker*, os usuários subscritos no mesmo tópico receberão a mensagem e podem atuar conforme a informação solicitada (abrir a porta na imagem). Após atuar ele envia a mensagem de sucesso ou falha ao primeiro o usuário, o qual recebe se estiver subscrito no mesmo tópico. O método necessita de poucos recursos pelo tamanho leve das mensagens, a comunicação é bi-direcional, suporta múltiplos dispositivos conectados simultaneamente, funciona em conexões limitadas e é seguro pois trabalha com TLS (*Transport Layer Security*) e métodos de autenticação (HIVEMQ, 2024).

O exemplo da Figura 4 exemplifica como funciona um sistema de automação residencial utilizando MQTT. O cliente MQTT, através de um aplicativo de *smartphone,* publica uma mensagem MQTT em um tópico “X” solicitando a abertura da porta, a qual é recebida pelo *Broker* e este é responsável por enviá-la a todos os clientes subscritos no tópico X. A casa possui um sistema de automação, o qual pode ser um simples Arduino, que está subscrito no tópico X e este recebe a mensagem. Se, de acordo com os algoritmos desenvolvidos no projeto de automação, a mensagem recebida for correspondente a abertura da porta, ela se abrirá e o sistema publica uma mensagem de sucesso ao mesmo tópico X. O *Broker* envia novamente a mensagem aos clientes subscritos no tópico, e o usuário finalmente tem a resposta de que a porta de sua casa foi aberta com sucesso através de seu aplicativo.

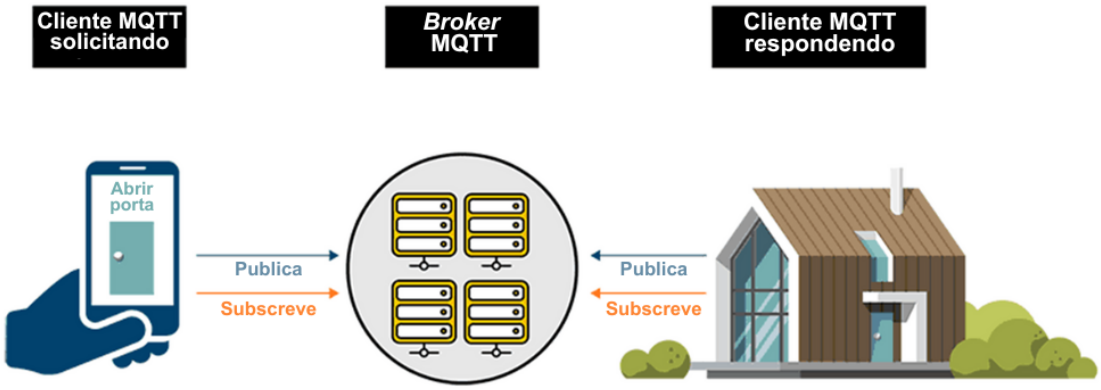

Figura 4 - HIVEMQ. Esquema de comunicação MQTT com IoT

Disponível em: https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/. Acesso em: 26 out. 2024.

2.3.2 Tópicos

Como mencionado anteriormente, as mensagens no MQTT são baseadas em subscrições e publicações de tópicos. Tópicos são definidos por uma *string* e funcionam como uma espécie de grupo onde as mensagens são subscritas e publicadas. Um tópico pode conter vários níveis de hierarquia como um endereço separado por “/”.

Um usuário deve informar ao *Broker* qual tópico ele deseja receber mensagens, então o *Broker* direciona todas as mensagens publicadas neste tópico ao usuário. Para a publicação funciona da mesma forma, o usuário informa a mensagem a qual deseja publicar e o tópico desejado e envia ao *Broker*, o qual direciona a todos os subscritos no mesmo tópico (HIVEMQ, 2024).

2.3.3 *Broker* e Cliente

A comunicação via MQTT necessita de dois componentes, o cliente e o *Broker*. O *Broker* atua como intermediador da comunicação, ele é responsável por receber, filtrar e enviar todas as mensagens da rede de comunicação. Quando a rede possui autenticação, este também é responsável pela validação das credenciais de acesso de cada cliente (HIVEMQ, 2024).

Um cliente é qualquer dispositivo, seja publicador ou subscritor, inserido na rede de comunicação. Qualquer dispositivo pode funcionar como um cliente, como um dispositivo mobile ou um microcontrolador Esp32 desde que estejam conectados ao *Broker*.

2.4 SISTEMA DISTRIBUÍDO

De acordo com (VAN STEEN, TANENBAUM, 2018), um sistema distribuído é um sistema composto por vários elementos computacionais independentes, os quais são chamados de nós, e estes trabalham com o objetivo de colaborar com um único sistema.

A principal característica do sistema distribuído é que ele funciona de forma descentralizada, pois ele não é controlado por um nó que envia comandos para os outros demais e sim são vários nós utilizando sua capacidade computacional para realizar coletivamente uma mesma tarefa. A vantagem desse tipo de sistema é que a falha de um nó não impacta negativamente no resultado, devido a sua capacidade de redistribuir a tarefa aos demais nós restantes.

Um sistema distribuído possui uma camada de software no topo do sistema operacional de cada nó, chamada *Middleware*. Esse software tem a função de gerenciar os recursos da rede, aproveitando as diferenças de hardware entre os nós participantes, sem que essas sejam notadas.

Na Figura 5, há um exemplo de 4 máquinas com seus sistemas operacionais, podendo ser distintos ou não, utilizando a mesma camada de sistema distribuído e colaborando para a execução de 3 aplicações, A. B e C.

Os computadores 1, 2, 3 e 4 estão conectados pelo mesmo *middleware* e possuem a mesma interface. Entretanto, este decide, através de algoritmos de otimização, que somente o computador 1 deverá executar a aplicação A, 2 e 3 deverão executar a aplicação B e o 4 deverá executar a aplicação C. Todas as aplicações possuem o mesmo objetivo final, que é garantir o funcionamento de um sistema em comum, mas cada computador opera de forma independente.

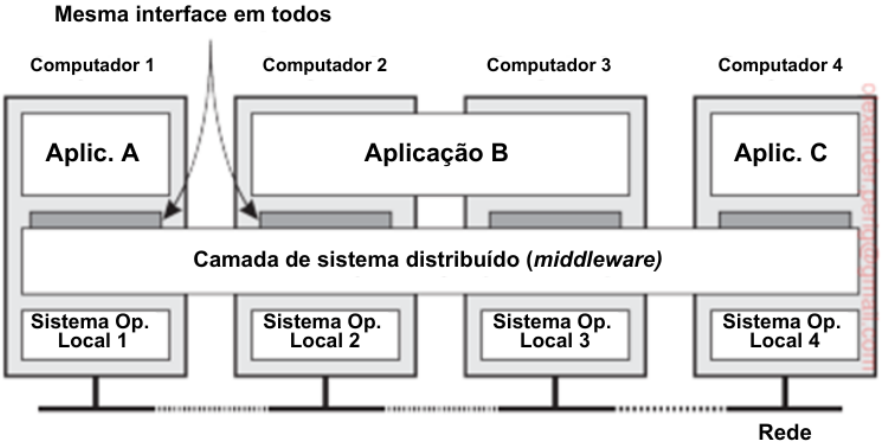

Figura 5 - vários computadores em um sistema distribuído utilizando a mesma camada de software

Fonte: Distributed Systems (VAN STEEN, TANENBAUM, 2018)

2.5 BLOCKCHAIN

Blockchain pode ser definido como uma corrente de blocos. É um banco de dados criado com a ideia de ser um sistema P2P, onde não há um controle central sobre a rede, o que permite que transações sejam enviadas entre *peers*

sem que haja a necessidade de uma entidade intermediária, como bancos (BASHIR, 2018). Cada informação é encapsulada em forma de blocos, os quais estão todos interligados ao bloco anterior da cadeia.

As transações são armazenadas no livro de transações, chamado de *ledger*. Este é criptografado e imutável e é compartilhado com todos os *peers* da rede, portanto todos devem possuir a mesma cópia do *ledger*, como no exemplo da Figura 6. O *ledger* não pode ser alterado, portanto os blocos somente podem ser adicionados à rede.

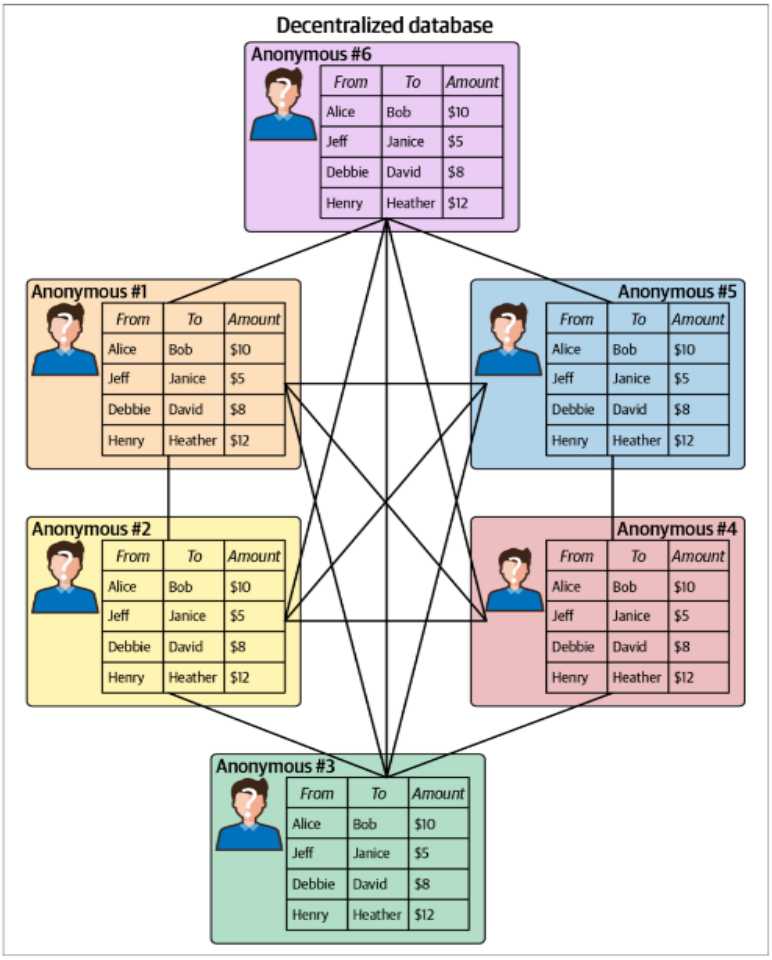

Figura 6 - Exemplo de uma blockchain com 6 peers

Fonte: Mastering Blockchain (LANTZ; CAWREY, 2020)

Na Figura 7 há um exemplo de como funciona uma transação em uma blockchain entre usuários “A” e “B”.

1. A deseja enviar dinheiro a B;

2. A transação é representada em forma de bloco, o qual é compartilhado com todos os participantes da *blockchain;*
3. Todos os participantes validam a transação e chegam em um consenso;
4. Após o consenso positivo, o bloco é adicionado ao final da cadeia;
5. O dinheiro é movido de A a B e o saldo de ambas as contas é atualizado.

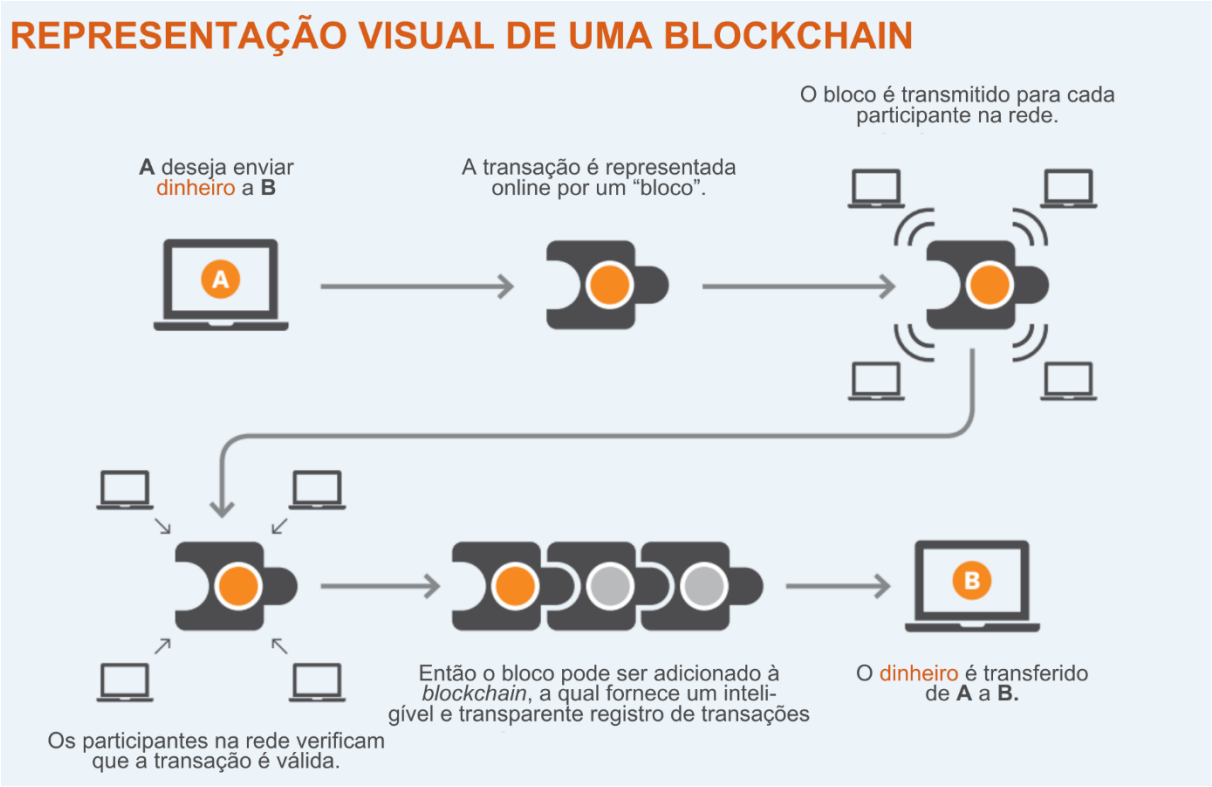

Figura 7 - Representação de uma transação de transferência monetária em uma blockchain

Fonte: MICHAEL, J.; COHN, A. L. A. N.; BUTCHER, Jared R. Blockchain technology. The Journal, v. 1, n. 7, p. 1-11, 2018.

Uma das principais características de uma *Blockchain* é sua atualização via consenso. Como dito no início deste capítulo, não há a necessidade de uma autoridade verificadora de transações para inserir uma transação na rede. Todas as transações devem passar por uma validação, a qual somente quando o consenso entre a maioria dos *peers* for alcançado o bloco será criado. Existem vários tipos de consenso, os quais serão discutidos no decorrer deste projeto.

A Figura 8 exibe um diagrama que demonstra o funcionamento de uma *Blockchain* em relação às camadas de rede. Esta se define como uma rede P2P, a qual opera sobre a camada da internet, como o protocolo TCP/IP. Nessa rede se realizam transações, as quais são encapsuladas em blocos, e estes são inseridos na *blockchain* através de um consenso. As transações alteram o estado atual da rede e podem ser criados *smart contracts*, que são aplicações para inserir transações de acordo com determinado gatilho. Todas essas informações são compartilhadas e armazenadas com os usuários/nós inseridos na rede.

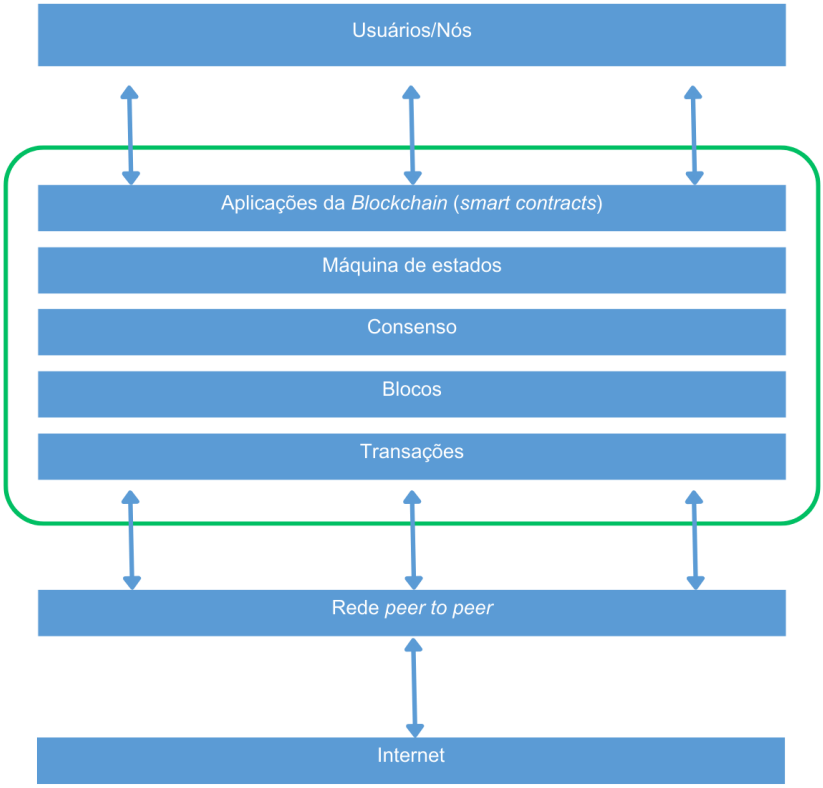

Figura 8 - Camadas de rede de uma blockchain

Fonte: Mastering Blockchain (BASHIR, 2018)

2.5.1 Elementos de uma *Blockchain*

De acordo com Bashir, 2018, há vários elementos genéricos em uma *blockchain*.

A Figura 9 exemplifica uma como os blocos de uma *blockchain* se comportam. Cada bloco contém duas partes principais, o *previous hash,* que é o *hash* do bloco anterior e o grupo de transações que compõe o bloco. *O* primeiro bloco gerado (bloco gênesis), não possui *hash* anterior pois é o primeiro bloco da cadeia. Entretanto, os demais sempre possuem a informação do *hash* anterior, criando uma rede inalterável e segura, pois qualquer alteração em um bloco alteraria o bloco seguinte.

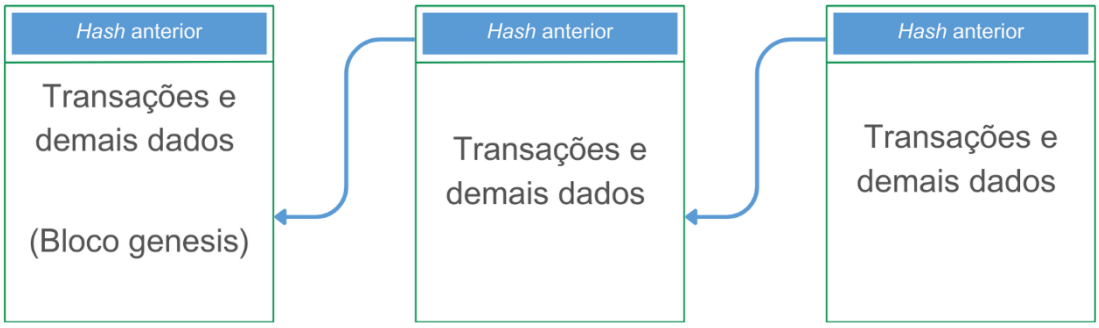

Figura 9 - Estrutura genérica de uma *blockchain*

Fonte: Mastering Blockchain (BASHIR, 2018)

2.5.1.1 Chaves públicas e privadas

Cada usuário em uma *blockchain* deve ter uma *private key.* Esta é basicamente um número aleatório de 256 bits em formato hexadecimal. Uma chave pública pode ser obtida utilizando a chave privada como referência através de algum algoritmo de conversão, como é o caso do *Elliptic Curve Digital Signature Algorithm* (ECDSA). É impossível que um usuário consiga efetuar transações em uma *Blockchain* sem seu par de chaves (pública e privada), pois estas funcionam como uma espécie de documento, onde a chave pública é compartilhada com todos os nós da rede e deve ser utilizada para toda transação realizada, já a chave privada funciona somente como verificação da identidade do usuário e jamais deve ser compartilhada (BASHIR, 2018).

2.5.1.2 Endereço

Um endereço pode ser definido como uma chave única de identificação dos *peers* participantes. O mesmo usuário pode usar sempre o mesmo endereço ou um endereço a cada transação. No *Bitcoin*, o endereço é gerado rodando a chave privada em uma função ECDSA chamada de secp256k1 a fim de gerar a chave pública, logo é gerado um *hash* da chave pública pelas funções SHA256 ou RIPEMD160, então é inserido o valor 00 ao *hash* da chave pública e o valor roda em outra função de verificação, gerando uma forma comprimida da *public key* do usuário (BASHIR, 2018).

2.5.1.3 Transação

A transação representa qualquer ação inserida na *blockchain*. Pode ser a criação de um usuário, alteração nas permissões dos *peers*, transferência de valor de um usuário a outro ou criação de um registro qualquer. Esta é considerada como a unidade fundamental da *Blockchain*.

2.5.1.4 Blocos

Um bloco é composto por informações da transação e múltiplos outros elementos, como o *hash* do bloco anterior, a data e horário da transação (*timestamp*) e *nonce*.

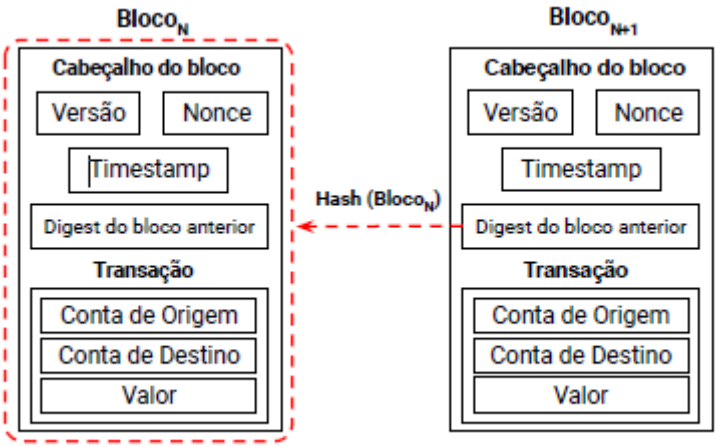

Figura 10 - estrutura do bloco de uma *Blockchain*

Fonte: Introdução às tecnologias dos blockchains e das criptomoedas (CHERVINSKY; KREUTZ, 2019)

2.5.1.5 Nó

Cada nó na rede pode possuir uma função diferente. Um nó é responsável por propor transações, as quais são validadas por outro conjunto de nós obedecendo o protocolo de consenso estabelecido.

2.5.1.6 Protocolo de consenso

O protocolo de consenso em sistemas distribuídos deve assegurar três propriedades fundamentais: vivacidade, segurança e consistência dos dados no *ledger*. A vivacidade implica que o sistema precisa ser robusto, operando sem interrupções e recuperando-se de eventuais falhas. Já a segurança visa garantir que os pares não comprometidos rejeitem dados fraudulentos. Por sua vez, a consistência exige que todos os nós saudáveis mantenham ou eventualmente cheguem a uma ordem e estado global uniformes.

Existem diversos algoritmos de consenso para diferentes cenários, entre os quais se destacam o PBFT (*Practical Byzantine Fault Tolerance* ou Tolerância

a Falhas Bizantinas Prático) e o PoET (*Proof of Elapsed Time* ou Prova por Tempo Decorrido).

O PBFT, empregado em redes permissionadas, é baseado em votação, exigindo a aprovação dos nós participantes para qualquer nova atualização. Esse protocolo é indicado para redes onde é possível definir manualmente os nós participantes e os direitos de voto. O funcionamento do PBFT, conforme descrito por Castro e Liskov (1999), envolve três fases principais: pré-preparação, preparação e confirmação. Um cliente envia uma solicitação a um nó, que distribui mensagens de pré-preparação aos demais pares. Na fase de preparação, essas mensagens são retransmitidas por *multicast* a todos os nós. Ao receber 2f mensagens de preparação, um nó combina essas mensagens com a de pré-preparação e emite uma confirmação. Quando 2f + 1 confirmações são recebidas, o estado é atualizado para confirmado, a operação é executada e uma resposta é enviada ao cliente. O objetivo do PBFT é garantir a operação contínua do sistema, mesmo que alguns nós apresentem falhas arbitrárias. No entanto, o protocolo possui limitações práticas, incluindo sobrecarga de rede, que afeta a escalabilidade e o desempenho. Estudos demonstraram (MILLER et al., 2016) que o PBFT pode ser suscetível a ataques de agendamento, resultando em atrasos ou interrupções no consenso.

Por outro lado, o PoET permite que qualquer nó participe da validação dos blocos em redes não-permissionadas, selecionando aleatoriamente nós com base em um tempo de espera antes da proposta de um bloco. Esse método facilita a criação de redes maiores com numerosos nós.

2.5.1.7 Contrato inteligente

Um contrato inteligente é um programa que executa transações automaticamente, à medida que algumas condições definidas pelo usuário são satisfeitas. Esse projeto tem como objetivo principal criar um *smart contract*, que realiza transações de acordo com a leitura de leitores RFID.

2.6 HYPERLEDGER IROHA

2.6.1 Conceito

O Hyperledger Iroha é um DLT (*Distributed Ledger Tecnology* ou Tecnologia de Livro Razão Distribuído) permissionado criado com base no sistema do Ethereum. As diferenças entre o Iroha e o Ethereum é que no Iroha não há a presença de criptomoedas e os usuários podem realizar outros tipos de transações de acordo com suas permissões adquiridas. Esse sistema foi inspirado no método organizacional *kaizen*, o qual, de acordo com SHANG, 2013 possui os seguintes princípios:

1. Zero defeito;
2. Esquemas de sugestão
3. Implantação de políticas
4. Atividades de pequenos grupos

A metodologia *kaizen* é um processo de melhoramento contínuo japonês o qual promove competitividade empresarial quando usado de forma correta.

2.6.2 Funcionamento

O HL Iroha 1 utiliza um sistema de consenso chamado de YAC (*Yet Another Consensus* ou mais um consenso*)* cujo funcionamento semelhante ao BFT. Este sistema é robusto pois possui um sistema de CFT (*Crash Fault Tolerance*), porém sua grande desvantagem é que o armazenamento dos blocos é realizado por um servidor PostgreSQL. Já o HL Iroha 2 possui a implementação de um sistema de metadados e utiliza o método de consenso tolerante a falhas bizantinas (BFT) chamado de *Sumeragi,* que significa “imperador” se traduzido para o português. Este método também possui um sistema de CFT. Este algoritmo é inspirado no B-Chain (IROHA, 2024).

Duan et al. (2014) apresenta que no algoritmo B-Chain não há um serviço de ordenamento responsável por encapsular e enviar as propostas de transação aos nós da rede. A cada rodada um líder é escolhido através de um algoritmo e

este cria o cálculo do novo bloco e envia aos participantes para realizarem a votação.

São necessários $3f + 1$ *peers* para que f falhas bizantinas sejam toleradas, portanto caso haja 4 nós na rede somente 1 poderá ser malicioso sem que comprometa o estado da rede.

Os *peers* podem ser votantes ou normais. Os votantes, como o próprio nome já diz, participam da votação, enquanto os normais não participam, porém verificam todos os dados recebidos.

Figura 11 - Modelo de dados no HL Iroha 2

Os dados da blockchain no Iroha 1 eram armazenados em um container com o PostgreSQL, o que foi atualizado na versão 2. O sistema de armazenamento é gerenciado por uma ferramenta chamada *Kura,* a qual divide os dados em dois volumes:

- *Block store*: responsável por registrar todas as transações de forma criptografada e inviolável;
- *World State View:* armazenado na memória RAM do dispositivo e é calculado sempre que o Hyperledger Iroha é iniciado, o que economiza armazenamento, mas consome mais memória volátil.

O Iroha 2 possui um sistema de permissionamento de contas, onde somente as contas com as permissões necessárias podem realizar cada tipo de transação (IROHA, 2024).

2.6.3 *Sumeragi*

Como dito anteriormente, a cadeia de blocos necessita ao menos $3f + 1$ *peers* para que a rede funcione apropriadamente e somente f são tolerados. Portanto, os nós são organizados em grupos para que os agentes maliciosos sejam sempre alocados em um grupo de baixa prioridade contendo f nós.

É realizada uma divisão entre grupo α e β, onde α contém $2f + 1$ e β contém f. O primeiro nó do grupo α é chamado de *sumeragi* ou líder e o último é chamado de *proxy tail*. Todos os nós possuem uma reputação e assim que a esta é definida como maliciosa, o *peer* é movido ao grupo β.

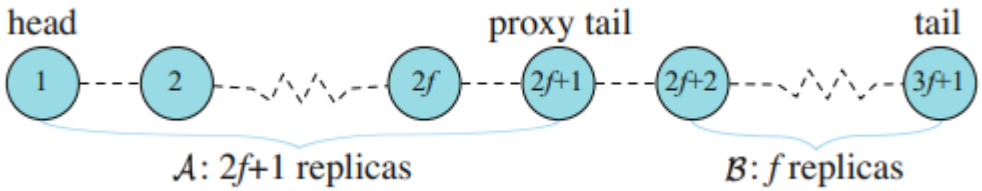

Figura 12 - configuração dos *peers* no Sumeragi

Fonte: DUAN et al. 2014.

Todo recebimento de proposta de transação deve ser confirmado pelo líder aos demais participantes como forma de provar que o líder não é malicioso. Caso um nó encaminhe uma transação ao *sumeragi* e este cria um bloco vazio ou o deixa de enviar, um novo líder deve ser eleito imediatamente (Duan et al. 2014).

O *proxy tail* é o último nó o qual é aguardado pelo líder após o envio do bloco, quando ocorre algum problema na confirmação do bloco um novo grupo α é selecionado para que o bloco seja rapidamente adicionado (caso válido).

3 METODOLOGIA E DESENVOLVIMENTO

O presente trabalho apresenta a criação de um modelo de dados distribuído utilizando a leitura de *tags* de RFID do tipo NFC por meio de celulares integrado a um *Hyperledger*. Foi feita uma análise do desempenho das tecnologias *Hyperledger* disponíveis com o objetivo de implementar a que mais atende os requisitos de projeto. A metodologia adotada para o desenvolvimento do trabalho foi a de pesquisa aplicada focando na conversão de conceitos teóricos em aplicações práticas e funcionais.

3.1 REQUISITOS DE PROJETO

Com o objetivo de atender as condições de um banco de dados para armazenar o histórico de registros de leitura de *tags* RFID do tipo NFC por celulares, foram definidos os seguintes requisitos de projeto:

- Obter os dados globais da rede a qualquer momento;
- Acessar os dados remotamente via celular;
- Leitura/gravação da *tag* via NFC;
- Alterar o estado atual da *tag* simulando um sistema de produção;
- Possuir um ambiente descentralizado;
- Visualizar o fluxo sem interferir no processo;

De acordo com os requisitos de projeto, foram necessárias as seguintes implementações:

1) Criação de um programa cliente para aquisição e envio de informações à *hyperledger* com suporte à tecnologia MQTT,
2) Implementação de um aplicativo móvel com suporte à NFC e MQTT e um sistema de comunicação MQTT para transferir mensagens entre a *hyperledger* e o aplicativo.

Para a escolha do banco de dados, este deve ser descentralizado, acessível, seguro e imutável. Logo, deve ser um *hyplerledger* de baixo custo, com uma comunidade ativa e com suporte a metadados.

3.2 ANÁLISE DE DESEMPENHO DE HYPERLEDGER

Foram identificadas 3 soluções que atenderam os requisitos do banco de dados *hyperledger*. São eles o *Hyperledger Sawtooth*, *Hyperledger Fabric* e o *Hyperledger Iroha*.

3.2.1 Hyperledger Sawtooth

Dentre os *Hyperledgers* citados, o *Sawtooth* foi o primeiro avaliado para implementação pela simplicidade, a criação de redes não permissionadas e o conhecimento de projetos previamente realizados com essa blockchain. Porém ao buscar as ferramentas necessárias da *blockchain*, como API's e apoio da comunidade, foi percebido que o *Sawtooth* está em desuso e não possui atualização constante de suas bibliotecas. Recentemente, o *Hyperledger Sawtooth* foi movido para o status de *end of life*, portanto foi decidido que não seria mais uma opção.

3.2.2 *Hyperledger Fabric*

Após o *Sawtooth*, o *Fabric* foi avaliado como a segunda melhor opção, pois é uma *blockchain* desenvolvida para uso em indústrias, possui uma arquitetura modular e configurável e sua plataforma é permissionada. Em consequência de o *Hyperledger Fabric* ser um sistema empresarial, este possui um sistema de canais, pois é necessário criar uma anonimidade para as transações entre empresas por motivos de preços negociados e proteção de dados.

Os contratos inteligentes do *Fabric* rodam dentro de um *Docker container* e podem ser programados em linguagens de programação convencionais, como Java, GO e Node.js, ao contrário de outras *blockchains* em que seus *smart contracts* são programados em *Solidity*. O Fabric é um dos projetos mais ativos do *Hyperledger* e sua comunidade está crescendo cada vez mais.

Um dos motivos por este não ter sido escolhido como melhor opção é que sua versão mais atualizada (2.5) não possui suporte para diversas ferramentas como o *Ethereum Virtual Machine* para realizar transações em *Ethereum* e há conflito de versões de softwares dados como requisito. o *Fabric* trabalha com mais camadas de aplicação, o que reflete em uma implementação mais custosa e complexa que os demais *Hyperledgers*.

3.2.3 Hyperledger Iroha

Por fim, o *Hyperledger Iroha* foi a última *blockchain* a ser avaliada. Seu método de consenso tem uma melhor performance em relação ao *Fabric* pois permite transações com uma menor latência e funciona bem com a falha de *peers* na rede. Tem um sistema de permissões robusto, que permite uma série de permissões para determinados comandos. A comunidade da *blockchain* é bastante ativa e as bibliotecas são atuais.

O *Iroha* foi desenvolvido com o objetivo de ser uma *blockchain* de implementação simples, sem a necessidade da criação de *smart contracts* complicados e seu foco é a aplicação em IoT, tendo SDK nas linguagens Kotlin/Java, Javascript, Python e IOS Swift, o que facilita o desenvolvimento do projeto.

Há duas versões disponíveis para desenvolvimento, o *Iroha* 1 e o *Iroha* 2. No início da pesquisa foi concluído que o *Iroha* 1 seria mais viável pois ainda havia pouco conteúdo sobre o *Iroha* 2 e seu funcionamento. Um aplicativo web foi desenvolvido, mas devido à limitação de haver somente a possibilidade de realizar transações monetárias, o projeto foi migrado para o *Iroha* 2 com o uso de metadados. Assim, fica mais fácil aplicar o conteúdo deste projeto em

situações práticas como o rastreamento de uma peça na indústria em seus inúmeros processos de fabricação.

3.3 ARQUITETURA DO PROJETO

O projeto foi dividido em quatro etapas, as quais serão detalhadas nos próximos subtópicos. Estas são:

1. Configuração e execução do *Hyperledger Iroha 2*;
2. Desenvolvimento do Processador de Transações para comunicação com o banco de dados distribuído;
3. Aplicativo para leitura de *tags* NFC por celular e interface com o usuário;
4. Configuração de um *broker* MQTT para possibilitar a troca de informações entre Processador de Transações e aplicativo.

O fluxograma do projeto está descrito na Figura 13. O usuário realiza a leitura ou a gravação de alguma informação na *tag* e este é considerado como o estado atual. Ao consultar a informação da etiqueta RFID, o aplicativo envia a informação via MQTT ao processador de transações, o qual recebe e compara os dados atuais com os dados da *tag* previamente registrados na *blockchain*, caso as informações sejam diferentes, o programa registra o novo estado no banco de dados e retorna o status da transação ao usuário via MQTT.

O registro funciona da mesma forma, porém há uma alteração do *payload* da *tag* antes de iniciar a comunicação com o *Iroha*.

Em resumo, toda comunicação é realizada via MQTT, pois é um método de troca de mensagens de baixa latência e baixo custo computacional.

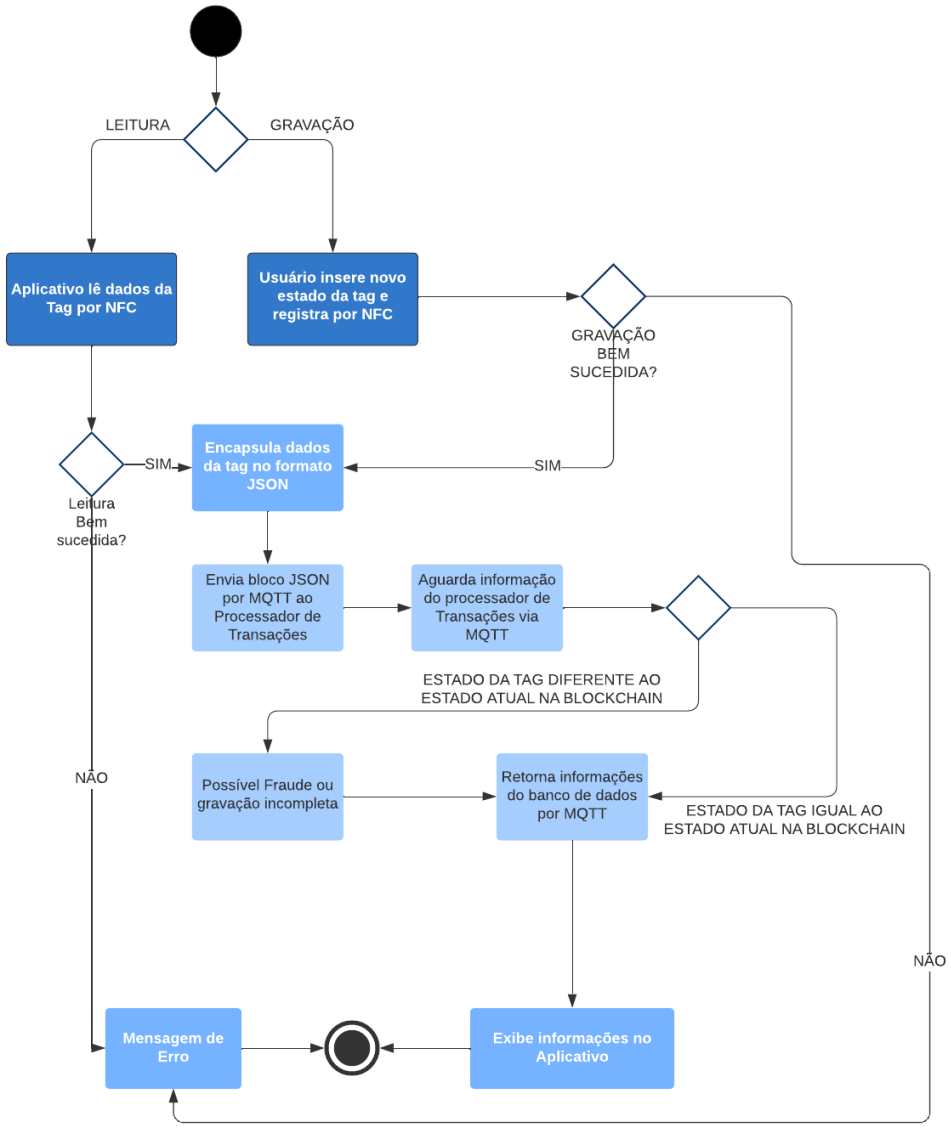

Figura 13 - Fluxograma de funcionamento do Sistema

3.3.1 Configuração do *Hyperledger Iroha 2*

Para executar o *Iroha*, é necessário utilizar a ferramenta *Docker*, que atribui imagens a *containers* como uma espécie de máquina virtual. Cada *peer* deve ser executado dentro de um *container* com o *Hyperledger Iroha* instalado.

Antes de iniciar o *Iroha* de acordo com as *instruções* do Apêndice A, é necessário configurar o arquivo do bloco gênesis, pois sem ele não é possível criar nenhum outro bloco. O arquivo é escrito em formato JSON e nele são

inseridas quaisquer transações que o usuário desejar realizar como registro de contas e domínios, registro e transferência de ativos, entre outros.

No exemplo da Figura 14 há duas transações no formato JSON. A criação do domínio chamado *garden_of_live_flowers* e da conta de chave pública "ed0120E9F632D3034BAB6BB26D92AC8FD93EF878D9C5E69E01B61B4C47101884EE2F99" e domínio *garden_of_live_flowers*.

Figura 14 - exemplo de transação no bloco genesis

Após a configuração do bloco *genesis*, deve ser realizada a configuração do arquivo YAML contendo as informações dos *peers*, pois este arquivo servirá como executável para iniciar os *containers* do *Docker*. As informações registradas nesse arquivo são:

- Nome do *peer*;
- Chaves pública e privada;
- Endereço de comunicação P2P;
- Endereço de comunicação com a API;
- Chave pública do bloco *genesis*;
- Nome, endereço de comunicação P2P e chave pública dos *peers* confiáveis da rede;
- Portas de comunicação utilizadas.

Na Figura 15, temos o exemplo de configuração do *peer* com o nome *irohad0*, onde todos seus parâmetros foram configurados corretamente. A variável *CHAIN* representa o código de nome atribuído à *blockchain* criada.

```
irohad0:
    image: hyperledger/iroha:dev
    environment:
      CHAIN: 00000000-0000-0000-0000-000000000000
      PUBLIC_KEY: ed0120A98BAFB0663CE08D75EBD506FEC38A84E576A7C9B0897693ED4B04FD9EF2D18D
      PRIVATE_KEY: 802620A4DFC16789FBF9A588525E4AC7F791AC51B12AEE8919EACC03EB2FC31D32C692
      P2P_ADDRESS: 0.0.0.0:1337
      API_ADDRESS: 0.0.0.0:8080
      GENESIS_PUBLIC_KEY: ed01204164BF554923ECE1FD412D241036D863A6AE430476C898248B8237D77534CFC4
      TRUSTED_PEERS:
'[{"address":"irohad2:1339","public_key":"ed01204EE2FCD53E1730AF142D1E23951198678295047F9314B4006B0CB61850B1DB10"},
{"address":"irohad1:1338","public_key":"ed01209897952D14BDFAEA780087C38FF3EB800CB20B882748FC95A575ADB9CD2CB21D"},
{"address":"irohad3:1340","public_key":"ed0120CACF3A84B8DC8710CE9D6B968EE95EC7EE4C93C85858F026F3B4417F569592CE"}]'
      GENESIS_PRIVATE_KEY: 80262082B3BDE54AEBECA4146257DA0DE8D59D8E46D5FE34887DCD8072866792FCB3AD
      GENESIS: /tmp/genesis.signed.scale
      TOPOLOGY:
'[{"address":"irohad2:1339","public_key":"ed01204EE2FCD53E1730AF142D1E23951198678295047F9314B4006B0CB61850B1DB10"},
{"address":"irohad1:1338","public_key":"ed01209897952D14BDFAEA780087C38FF3EB800CB20B882748FC95A575ADB9CD2CB21D"},
{"address":"irohad0:1337","public_key":"ed0120A98BAFB0663CE08D75EBD506FEC38A84E576A7C9B0897693ED4B04FD9EF2D18D"},
{"address":"irohad3:1340","public_key":"ed0120CACF3A84B8DC8710CE9D6B968EE95EC7EE4C93C85858F026F3B4417F569592CE"}]'
    ports:
    - 1337:1337
    - 8080:8080
```

Figura 15 - configuração do peer irohad0

Após todas essas configurações, o arquivo YAML criado deve ser executado através do método *docker compose up*. No arquivo *docker-compose.yml* foram configurados *4 peers: irohad0, irohad1, irohad2* e *irohad3*, os quais utilizam as portas lógicas de comunicação P2P 1337, 1338, 1339 e 1340 e as portas para comunicação com a API 8080, 8082, 8084 e 8085, respectivamente (a porta 8083 foi reservada para uso do Broker MQTT e a 8081 ao *Metro Bundler* para testar o aplicativo no dispositivo em tempo real).

Após seguir os passos do Apêndice A para iniciar o *Iroha*, resulta na seguinte impressão no prompt de comando:

```
irohad3-1  | 2024-10-29T22:29:08.672586Z  INFO consensus: iroha_core::sumeragi::main_loop: Sumeragi initialized peer_
irohad2-1  | 2024-10-29T22:29:08.672569Z  INFO consensus: iroha_core::sumeragi::main_loop: Sumeragi initialized peer_
irohad0-1  | 2024-10-29T22:29:08.672576Z  INFO consensus: iroha_core::sumeragi::main_loop: Sumeragi initialized peer_
irohad1-1  | 2024-10-29T22:29:08.672590Z  INFO consensus: iroha_core::sumeragi::main_loop: Sumeragi initialized peer_
r_id=ed0120CACF3A84B8DC8710CE9D6B968EE95EC7EE4C93C85858F026F3B4417F569592CE@@0.0.0.0:1340 role=ValidatingPeer
r_id=ed01204EE2FCD53E1730AF142D1E23951198678295047F9314B4006B0CB61850B1DB10@@0.0.0.0:1339 role=ProxyTail
r_id=ed0120A98BAFB0663CE08D75EBD506FEC38A84E576A7C9B0897693ED4B04FD9EF2D18D@@0.0.0.0:1337 role=Leader
r_id=ed01209897952D14BDFAEA780087C38FF3EB800CB20B882748FC95A575ADB9CD2CB21D@@0.0.0.0:1338 role=ObservingPeer
```

Figura 16 – HL Iroha inicializado

3.3.2 *Broker* MQTT

A fim de que haja uma conexão rápida entre App de Celular e Processador de transações, é necessário a criação de um *broker* seguro e que seja de fácil acesso por dispositivos móveis. Com esse objetivo foi utilizado um *broker* da plataforma EMQX hospedado em *WebSocket* autenticado por usuário e senha.

Com o auxílio da ferramenta MQTTX foi possível monitorar em tempo real os tópicos *irohaMqtt/ClientListener* e *irohaMqtt/ClientResponse* de comunicação criados para trabalharem com o *Iroha*.

Figura 17 - parâmetros de configuração do broker MQTT

De acordo com a Figura 17 e já mencionado, foram estabelecidos dois tópicos de comunicação, o *irohaMqtt/ClientListener* e o *irohaMqtt/ClientResponse*. O primeiro é responsável por enviar transações ou solicitações ao processador de transações e o segundo é responsável por enviar informações do processador de transações ao aplicativo, para que não haja conflito de informações.

3.3.3 Processador de transações

O processador de transações é o nome dado ao programa que possui a função de se comunicar com o dispositivo móvel e o *Hyperledger Iroha* simultaneamente. Este necessita sua execução a todo momento e utiliza duas *threads* para que uma seja responsável por subscrever ao tópico

irohaMqtt/ClientListener e assim que receber alguma transação no formato correto, consulta ou registra no *Hyperledger Iroha* e retorna a mensagem através da outra *thread* pelo tópico *irohaMqtt/ClientResponse*.

Este utiliza o próprio código da *tag* ID como referência para buscar na *blockchain*, uma função dentro da biblioteca do *iroha* converte o valor da *tag* ID para uma *public key* (não há função para calcular o caminho inverso por motivos de segurança) e é registrada uma conta com o valor dessa chave e o domínio de nome “MAPL”. Assim, o *payload* da *tag* recebido via MQTT é tratado como um ativo de nome igual ao *payload* e sempre que o estado é alterado o ativo é removido e inserido um novo.

As trocas de mensagens são todas no formato JSON, o que facilita na identificação da informação de cada campo. Cada mensagem foi encapsulada no formato da Figura 18.

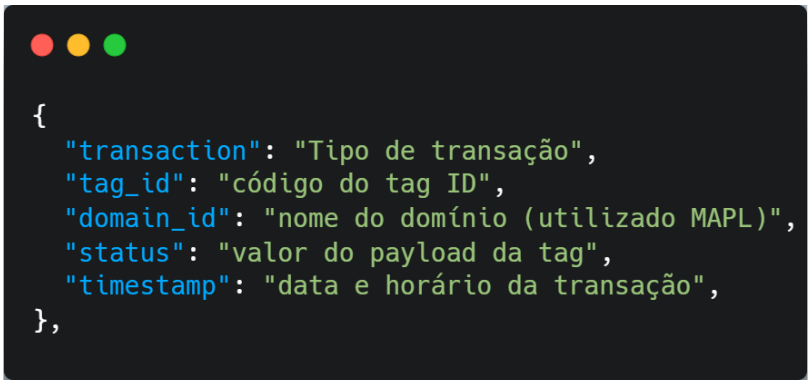

Figura 18 - formato da mensagem JSON

Foram registrados 4 tipos de transações para a aplicação, os quais são detalhados na Tabela 2.

Tabela 2 - funções registradas no processador de transações

Nome	Função
RegisterDomain	Registrar um novo domínio.
RegisterAccount	Registrar uma nova *tag.*
Consulta	Consultar o último estado registrado da *tag.*
RegisterStatus	Registrar um novo estado da *tag* na *blockchain.*

A Figura 19 apresenta o fluxograma de processamento de transações. O processador de transações trabalha sempre subscrito ao tópico *irohaMQTT/ClientListener*, quando uma mensagem é publicada este verifica se é uma transação válida de acordo com o cabeçalho da mensagem e os tipos de transação registrados, caso negativo retorna uma mensagem de erro e caso positivo verifica se é alguma das 4 transações conforme Tabela 2. Se for uma das 4 transações, o processador de transações insere ou consulta a informação de acordo com a mensagem recebida e retorna o status por MQTT pelo tópico *irohaMQTT/ClientResponse* ao usuário.

Caso a transação seja uma consulta e a *tag* não esteja cadastrada, a aplicação registra no banco de dados e retorna a informação ao aplicativo.

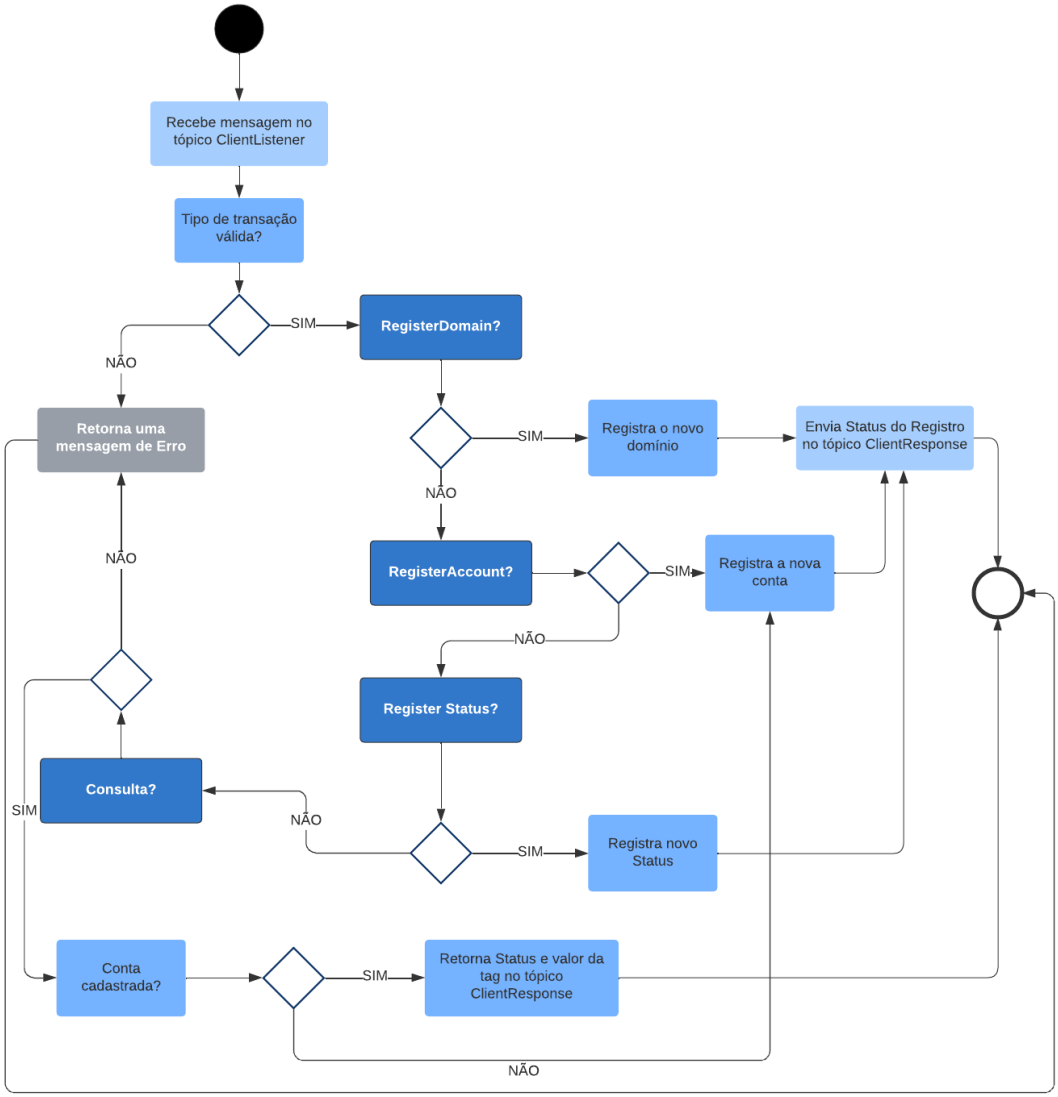

Figura 19 - Fluxograma processador de transações

3.3.4 Aplicativo Móvel

O aplicativo móvel foi desenvolvido a partir da biblioteca *React Native* em *Javascript* pois possui um desenvolvimento mais simples que nas linguagens nativas *Java/Kotlin* ou *Switf iOS*. O projeto foi realizado todo voltado para a plataforma Android pois a Apple não fornece formas de desenvolvimento gratuito com o uso de NFC.

Foi utilizada a biblioteca *react-native-nfc-manager* para comunicação do dispositivo com a *tag* e a *react_native_mqtt* para comunicação com o *broker MQTT*. A biblioteca responsável pelo NFC possui compatibilidade com várias tecnologias conforme a Tabela 3. Embora haja várias tecnologias disponíveis, foi utilizada somente a NDEF, pois é mais simples de trabalhar e é suportada nas duas plataformas (caso seja utilizado em projetos futuros com iOS, apesar de ter suporte limitado a tecnologia NFC).

Tabela 3 - Tecnologias NFC suportadas por tipo de Sistema Operacional de Celular

Tecnologias NFC	Android	iOS
Ndef	✅	✅
NfcA	✅	✅
IsoDep	✅	✅
NfcB	✅	❌
NfcF	✅	❌
NfcV	✅	❌
MifareClassic	✅	❌
MifareUltralight	✅	❌
MifareIOS	❌	✅
Iso15693IOS	❌	✅
FelicaIOS	❌	✅

A aplicação possui duas funções, a de consulta e a de registro. Uma somente lê o conteúdo e o ID da *tag* e envia por MQTT ao processador de transações para obter uma consulta do último estado registrado. Caso a *tag* ainda não tenha sido registrada, o programa registra no *Iroha* e envia uma mensagem de retorno referente a transação.

A segunda tela altera o *payload* da *tag* para qualquer valor inserido pelo usuário e envia a atualização ao *Iroha* via MQTT assim que a gravação for bem-sucedida. Em seguida, o processador de transações envia uma mensagem de retorno informando o status da transação.

3.4 RESULTADOS

O resultado obtido foi o aplicativo com a interface da Figura 20.

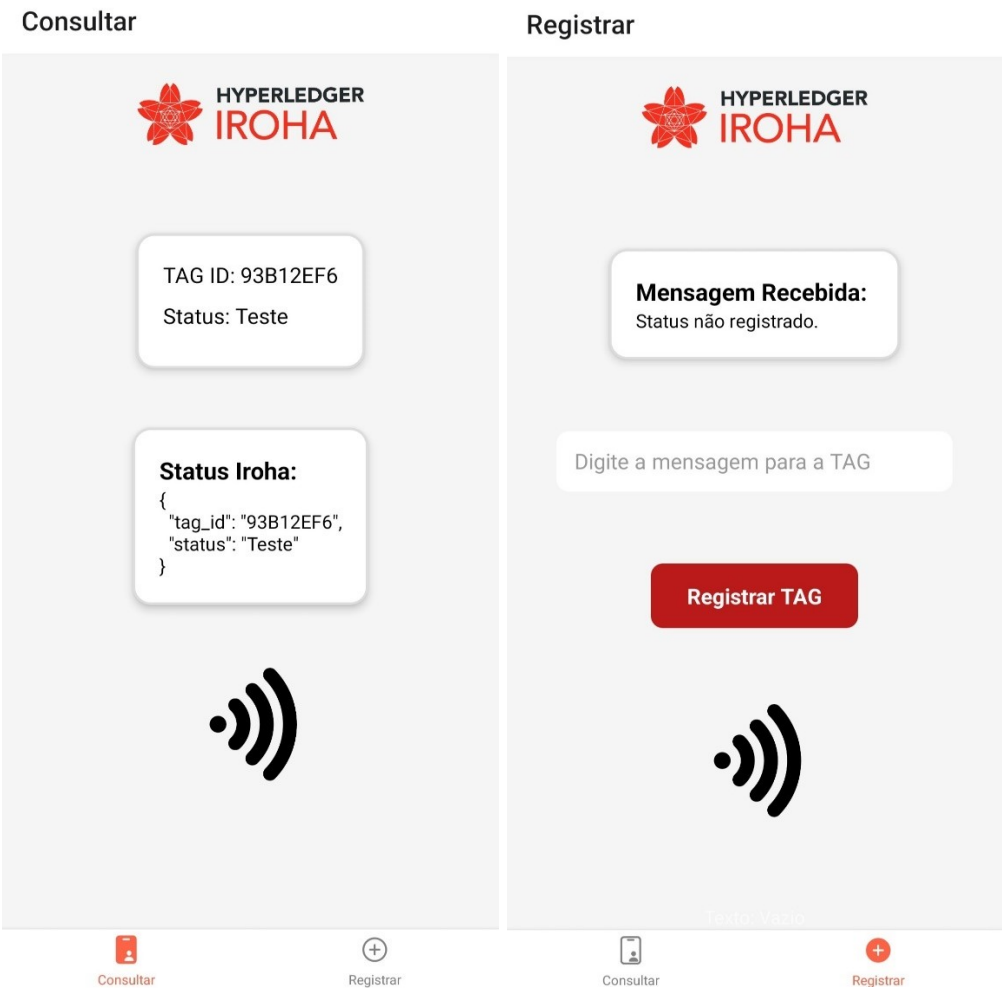

Figura 20 - Interface do App desenvolvido

O aplicativo, juntamente com o processador de transações, resultou em um sistema de armazenamento de dados históricos da *tag*. A integração entre as tecnologias NFC, MQTT e *Hyperledger Iroha* se mostrou extremamente eficiente na confecção do sistema pois todas as partes se complementaram. O dispositivo móvel funciona como um sensor e transmissor da informação da etiqueta ao banco de dados, o broker MQTT funciona como mediador da comunicação e o processador de transações funciona como um atuador cuja função é registrar ou consultar informações no *Hyperledger Iroha*.

Como o desenvolvimento da *blockchain* ainda é recente, não há SDK disponível para todas as funções desejadas como capturar o valor do ativo de cada conta ou inserir metadados em um ativo para descrições mais detalhadas. Contudo, o resultado atendeu todos os requisitos de projeto e foi possível visualizar na prática como o RFID pode otimizar a logística em sistemas de produção devido a sua acessibilidade (qualquer *smartphone* compatível com a tecnologia NFC pode funcionar como sensor), velocidade de transmissão de informações, segurança e confiabilidade em integração com o *Hyperledger Iroha*.

4 CONCLUSÃO

Este trabalho realizou a simulação da integração de uma blockchain no contexto de produção, destacando as possibilidades que essa tecnologia oferece. A principal vantagem é a capacidade de criar um banco de dados compartilhado entre diferentes empresas, no qual nenhuma delas assume individualmente a responsabilidade de hospedá-lo, já que ele pode ser gerenciado por uma entidade externa em ambiente de nuvem.

Essa abordagem abre caminho para uma colaboração eficiente entre fornecedores e outras empresas, permitindo que múltiplos participantes utilizem a estrutura para gerenciar suas operações de forma integrada. O Hyperledger Iroha 2, apesar de ainda estar em desenvolvimento, mostrou-se promissor ao oferecer funcionalidades fundamentais, como armazenamento de dados, controle granular de permissões e imutabilidade, assegurando um elevado nível de segurança e rastreabilidade. Além disso, o sistema proporciona transparência, já que todos os participantes têm acesso aos dados gravados, sem possibilidade de alterações, o que fortalece a confiança e a integridade do processo.

Apesar de ter sido utilizado somente uma máquina, é possível a criação de um ambiente descentralizado para o envio de dados de uma *tag* RFID. Foi criado um sistema com 4 *peers* (mínimo necessário) para testes, mas é possível a criação de vários nós desde que contenham uma *public* e *private key* e tenham acesso ao servidor no qual a *blockchain* está localizada.

A principal limitação no desenvolvimento do projeto foi utilizar uma *blockchain* com bibliotecas atuais, pois, entre os *Hyperledgers* considerados, somente o Iroha 2 possui uma comunidade ativa que segue atualizando e aperfeiçoando os pacotes da ferramenta.

Pode-se concluir que se pode tirar proveito de tecnologias como Blockchain como ferramenta no mundo corporativo que garantam confiabilidade e competitividade a empresa. O *Hyperledger Iroha* conta com um sistema seguro de inclusão de informações, onde estas são armazenadas em todos os

participantes (nós) da rede, e estas são armazenadas em um formato criptografado e impossível de alterar.

Apesar de ter sido desenvolvido um modelo, este projeto abre espaço para um leque de possibilidades a serem desenvolvidos e aplicados de forma prática em situações reais como um sistema de rastreamento de peças de usinagem, gerenciamento de contas, entre outros.

4.1 TRABALHOS FUTUROS

Embora o projeto tenha alcançado seus objetivos, ainda existem alguns projetos de melhoria que podem ser futuramente colocadas em prática. Entre elas destacam-se:

- Implementar *smartcontracts;*
- Desenvolver uma aplicação que utilize uma PNRD (*Petri Net Inside RFID Database* ou Rede de Petri Inserida em Base de Dados RFID) (Tavares *et* Saraiva, 2010) para inserção de dados da *tag* RFID em uma *blockchain;*
- Desenvolver o aplicativo utilizando metadados para armazenamento de informações;
- Aplicar o projeto em um sistema de produção real, como sistema de apoio a trilhas;
- Utilizar um servidor para aplicação de *peers* reais.

5 REFERÊNCIAS

WANT, Roy. An introduction to RFID technology. **IEEE pervasive computing**, v. 5, n. 1, p. 25-33, 2006.

COSKUN, Vedat; OZDENIZCI, Busra; OK, Kerem. The survey on near field communication. Sensors, v. 15, n. 6, p. 13348-13405, 2015.

ANDROID. Conectar-se usando NFC. Developer Android. Disponível em: https://developer.android.com/develop/connectivity/nfc?hl=pt-br. Acesso em: 26 out. 2024.

DOMDOUZIS, Konstantinos; KUMAR, Bimal; ANUMBA, Chimay. Radio-Frequency Identification (RFID) applications: A brief introduction. **Advanced Engineering Informatics**, v. 21, n. 4, p. 350-355, 2007.

THANAPAL, Pandi; PRABHU, J.; JAKHAR, Mridula. A survey on barcode RFID and NFC. In: **IOP Conference Series: Materials Science and Engineering**. IOP Publishing, 2017. p. 042049.

REJEB, Abderahman; KEOGH, John G.; TREIBLMAIER, Horst. Leveraging the internet of things and blockchain technology in supply chain management. **Future Internet**, v. 11, n. 7, p. 161, 2019.

MENTZER, John T. et al. Defining supply chain management. **Journal of Business logistics**, v. 22, n. 2, p. 1-25, 2001.

HESSEL, F. Implementando RFID na Cadeia de Negócios. 2. Ed. Porto Alegre, RS: Editora EDIPUCRS, 2012.

CHINELATTO, C. Tecnologia de Identificação por Radiofrequencia - RFID aplicada em sistemas de gerenciamento de armazéns. 2010.

SILVA, C. E. A. D. Desenvolvimento de biblioteca para aplicações de pnrd e pnrd invertida embarcadas em arduino. Repositorio Universidade Federal de Uberlândia, p. 46–54, 2017.

PAULA, Roger Henrique Carrijo de. Integração da PNRD com banco de dados distribuído Hyperledger Sawtooth. 2021. 120 f. Trabalho de Conclusão de Curso (Graduação em Engenharia Mecatrônica) – Universidade Federal de Uberlândia, Uberlândia, 2021.

VOWELS, Susan A. Understanding rfid (radio frequency identification). In: Ubiquitous and Pervasive Computing: Concepts, Methodologies, Tools, and Applications. IGI global, 2010. p. 54-64.

VAN STEEN, Maarten; TANENBAUM, Andrew S. Distributed systems. Leiden, The Netherlands: Maarten van Steen, 2017.

BASHIR, I. *Mastering Blockchain*. 2. ed. [S.l.]: Packt Publishing, 2018.

LANTZ, L.; CAWREY, D. *Mastering Blockchain*. 1. ed. [S.l.]: O'Reilly Media, Inc., 2020.

CHERVINSKY, J. O. M.; KREUTZ, D. Introdução às tecnologias dos blockchains e das criptomoedas. **Revista Brasileira de Computação Aplicada.** Alegrete – RS, v.11, n.3, pp.12–27, 2019.

MICHAEL, J.; COHN, A. L. A. N.; BUTCHER, Jared R. Blockchain technology. The Journal, v. 1, n. 7, p. 1-11, 2018.

SORAMITSU CO. Hyperledger Iroha v2.0 Documentation, 2024. Disponível em https://github.com/hyperledger-iroha/iroha/blob/main/docs/source/iroha_2_whitepaper.md. Acesso em: 30/10/2024.

SHANG, Gao; SUI PHENG, Low. Understanding the application of Kaizen methods in construction firms in China. Journal of Technology Management in China, v. 8, n. 1, p. 18-33, 2013.

MURATOV, Fedor et al. **YAC: BFT Consensus Algorithm for Blockchain**, 2018. Disponível em: < https://arxiv.org/pdf/1809.00554.pdf>. Acesso em: 29 de março de 2024.

GOODRICH, Rebecca S. MQTT–HOW IT WORKS. 2022.

MQTT. MQTT: The standard for IoT messaging. Disponível em: https://mqtt.org/. Acesso em: 26 de outubro 2024.

REACT. REACT: Describing the UI. Disponível em: https://react.dev/learn/describing-the-ui. Acesso em 28 de outubro 2024.

BODUCH, Adam. React and react native. Packt Publishing Ltd, 2017.

PEREIRA NUNES, Vinicius. AppIrohaMqttNfc: aplicação para comunicação via MQTT e NFC com Iroha. 2024. Disponível em: <https://github.com/MAPL-UFU/MQTT-NFC-Iroha.git>. Acesso em: 29 out. 2024.

Apêndices

APÊNDICE A – CONFIGURAÇÃO HYPERLEDGER IROHA

Antes de instalar o *Hyperledger Iroha* é necessário realizar algumas pré-instalações:

- *Docker*
- *Docker Compose*
- *Rust*
- *OpenSsl*

Em seguida, crie uma pasta chamada *Git* (recomendado para usuários macOS) e clone o repositório do *iroha 2* nessa pasta. A versão selecionada foi a v.2.0.0-pre-rc.22.0 pois é a versão sobre a qual o SDK do *Iroha-Python* foi desenvolvido.

```
mkdir ~/Git
cd Git
git clone https://github.com/hyperledger-iroha/iroha.git --branch v2.0.0-pre-rc.22.0
```

Após a clonagem do repositório, acesse a pasta criada e faça a *build* dos binários do *Hyperledger Iroha* com os seguintes códigos.

```
cd iroha
cargo build
docker build . -t hyperledger/iroha:v2.0.0-pre-rc.22.0
```

Nesse momento serão criados vários arquivos e possíveis erros de versão surgirão. Ao se deparar com algum erro verifique sua versão do *rustc* e *cargo* (devem ter a mesma versão), em seguida confira as atualizações disponíveis dos pacotes no arquivo *Cargo.toml*.

Após a finalização, altere os arquivos *configs/swarm/Docker-compose.yml* e *genesis.JSON* para a forma que desejar e insira o código a seguir para iniciar a *blockchain*:

```
docker compose -f configs/swarm/docker-compose.yml up
```

Caso de falha na sincronização dos *peers*, finalize a execução do programa (Ctrl + C) e altere o arquivo *configs/swarm/docker-compose.yml*. Substitua as linhas que contém *image: hyperledger/iroha:dev* para *image: hyperledger/iroha:v.2.0.0-pre-rc.22.0*.

```
#substitua em docker-compose.yml
image: hyperledger/iroha:dev
#para
image: hyperledger/iroha:v.2.0.0-pre-rc.22.0
```

Feito isso sua *blockchain* estará pronta para realizar transações.

Caso queira instalar o Python SDK para o *iroha 2*, primeiramente instale *maturin*, *pip* e *Python3* e siga os comandos:

```
git clone https://github.com/hyperledger-iroha/iroha-python.git --branch main
cd iroha-python
maturin build
```

Um arquivo com formato *whl* será criado em *target/wheels*, instale esse arquivo com *pip*.

```
#Altere iroha-0.1.0-cp312-cp312-manylinux_2_34_x86_64.whl para o nome do arquivo gerado
pip install --break-system-packages target/wheels/iroha-0.1.0-cp312-cp312-manylinux_2_34_x86_64.whl

#Verifique se a instalação foi concluída corretamente
python -c "import iroha; print(dir(iroha))"
```

APÊNDICE B – INSTALAÇÃO DO APLICATIVO

Caso queira testar o aplicativo em seu dispositivo *smartphone*, clone o repositório *https://github.com/MAPL-UFU/MQTT-NFC-Iroha.git* para a pasta de sua preferência e navegue até ela.

Tenha instalado em sua máquina:

- *Node.js;*
- *Android Studio;*
- *Android SDK;*
- *Android Virtual Device;*
- *Android Debug Bridge;*

Configure a variável ANDROID_HOME:

```
export ANDROID_HOME=$HOME/Android/Sdk
export PATH=$PATH:$ANDROID_HOME/emulator
export PATH=$PATH:$ANDROID_HOME/platform-tools
```

Instale as dependências do aplicativo e execute.

```
npm install
npx react-native start
```

Inicie o *broker* MQTT de sua preferência desde que seja hospedado em *WebSocket* e execute o programa *ProgramaCliente/mqtt_broadcast.py* e inicie as transações.