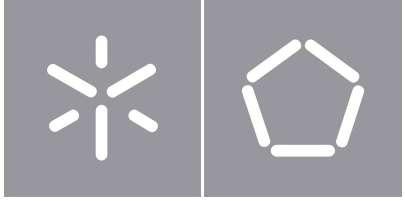
Universidade do Minho
Escola de Engenharia

João Manuel Silva de Amorim

Aplicação de monitorização de rede baseada em Blockchain

novembro 2023



Universidade do Minho
Escola de Engenharia

João Manuel Silva de Amorim

Aplicação de monitorização de rede baseada em Blockchain

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho efetuado sob a orientação de
José Carlos Bacelar Ferreira Junqueira Almeida
Helena Fernández López

novembro 2023

Direitos de Autor e Condições de Utilização do Trabalho por Terceiros

Este é um trabalho académico que pode ser utilizado por terceiros desde que respeitadas as regras e boas práticas internacionalmente aceites, no que concerne aos direitos de autor e direitos conexos.

Assim, o presente trabalho pode ser utilizado nos termos previstos na licença abaixo indicada.

Caso o utilizador necessite de permissão para poder fazer um uso do trabalho em condições não previstas no licenciamento indicado, deverá contactar o autor, através do RepositóriUM da Universidade do Minho.

Licença concedida aos utilizadores deste trabalho:

CC BY

https://creativecommons.org/licenses/by/4.0/

Agradecimentos

Depois de concluir esta etapa significativa da minha vida, desejo expressar minha gratidão a todas as pessoas que, de várias maneiras, ajudaram na realização desta dissertação. Quero transmitir os meus mais sinceros agradecimentos a cada um.

Em primeiro lugar, ao meu orientador, o Professor José Carlos Bacelar Ferreira Junqueira Almeida, e à minha coorientadora, a Professora Helena Fernández López, pela confiança e apoio prestados durante este período. Sem a orientação deles e suas análises críticas, não teria sido possível concluir este trabalho. Um enorme obrigado.

Quero expressar minha profunda gratidão à minha família pelo apoio e orientação que me deram ao longo da minha vida, especialmente durante este período. Agradeço aos meus amigos e namorada por estarem ao meu lado durante toda a minha jornada académica e por terem contribuído para a pessoa que sou hoje.

Por fim, quero manifestar minha gratidão à empresa dstelecom, especialmente ao Mário Oliveira e ao João Faria, bem como às equipas de Networking e de Inovação, pela confiança depositada e pelo constante suporte e sabedoria compartilhada ao longo deste projeto. A todos aqueles que não foram mencionados, mas que de alguma forma contribuíram para o sucesso desta jornada, o meu profundo agradecimento.

Declaração de Integridade

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

Universidade do Minho, Braga, novembro 2023

João Manuel Silva de Amorim

Resumo

Na era atual dos avanços tecnológicos, a blockchain emergiu como uma grande inovação, ganhando reconhecimento generalizado pelas suas capacidades de manutenção de registos seguros e transparentes. Ao mesmo tempo, a gestão e a comunicação eficazes de dados críticos da infraestrutura de tecnologias da informação (TI) continuam a ser uma necessidade premente. Esta dissertação aborda a intersecção destes domínios, com o objetivo de aproveitar o potencial da tecnologia blockchain para otimizar e proteger os processos de comunicação de dados. Patrocinada pela dstelecom, um operador de rede de fibra ótica neutra em Portugal, esta investigação investiga o papel transformador que a blockchain pode desempenhar na monitorização da rede. Com base numa análise abrangente da literatura, o estudo examina a capacidade de aplicação e os desafios da blockchain, apresentando um caso de estudo prático, que utiliza o Hyperledger Fabric, para destacar o seu potencial na monitorização da rede.

Palavras-chave Blockchain, Hyperledger Fabric, Monitorização de Rede

Abstract

In the present era of technological advancements, blockchain has emerged as a leading innovation, gaining widespread acclaim for its secure and transparent record-keeping capabilities. Concurrently, the effective management and reporting of critical data from Information Technology (IT) infrastructure remain a pressing need. This dissertation addresses the intersection of these domains, aiming to leverage the potential of blockchain technology to optimize and secure data reporting processes. Sponsored by dstelecom, a neutral fiber optic network operator in Portugal, this research delves into the transformative role blockchain can play in network monitoring. Through a comprehensive literature review, the study examines blockchain's applicability and challenges, presenting a practical case study using Hyperledger Fabric to highlight its potential in network monitoring.

Keywords Blockchain, Hyperledger Fabric, Network Monitoring

Conteúdo

Lista de Figuras

Lista de Tabelas

Acrónimos

BTS Base Transceiver Station.

DLT Distributed Ledger Tecnology.

FTTH Fiber-to-the-Home.

IBFT Istanbul Byzantine Fault Tolerant.

IoT Internet of Things.

IPC Inter-Process Comunication.

NMS NetNumem U31 Unified Management System.

P2P Peer-to-Peer.

PoA Proof-of-Authority.

PoC Proof-of-Concept.

PoET Proof-of-Elapsed-Time.

PON Passive Optical Network.

POP Point of Presence.

PoS Proof-of-Stake.

PoW Proof-of-Work.

ROE Repartidor Ótico Exterior.

Capítulo 1

Introdução

1.1 Contexto

Esta dissertação foi promovida pela empresa dstelecom, a qual propôs o tema e proporcionou assistência técnica. A dstelecom[1] é um operador neutro de redes de fibra ótica implantadas nos territórios de baixa densidade populacional de Portugal. Como tal, a empresa possui, como clientes, operadores de telecomunicações tais como a Altice, a Vodafone e a NOS, tendo a necessidade de apresentar dados sobre a disponibilidade da sua infraestrutura.

1.2 Motivação

A monitorização da rede é um aspeto crítico da infraestrutura de Tecnologias de Informação (TI) de qualquer organização. Envolve a monitorização contínua de dispositivos de rede, tais como *routers*, *switches* e servidores, para assegurar o seu bom funcionamento e detetar quaisquer avarias que possam surgir. A utilização de uma aplicação de monitorização de rede pode melhorar grandemente a eficiência e eficácia deste processo.

A *blockchain* é um livro de registos descentralizado e distribuído que permite o registo seguro e transparente das transações. A sua utilização na monitorização da rede pode proporcionar vários benefícios, tais como maior segurança, imutabilidade dos registos, e redução da dependência de uma autoridade central.

[1] https://www.dstelecom.pt/empresa/o-que-fazemos/

1.3 Principais Objetivos

Esta dissertação explora a utilização da tecnologia de *blockchain* para desenvolver uma aplicação de registo da informação proporcionada pelo sistema de monitorização da rede da dstelecom. Foi feita a análise da literatura relevante sobre o tema e discutidos os benefícios e desafios da utilização da *blockchain* neste contexto. Também se apresenta um caso de estudo de uma aplicação de monitorização de redes construída sobre *blockchain* e se fornece uma análise do seu desempenho e potenciais desenvolvimentos futuros.

Para alcançar esse objetivo, foi necessário fazer um estudo de tecnologias que asseguram a imutabilidade de dados armazenados e permitem uma auditoria a resultados obtidos a partir desses dados (por exemplo, a percentagem de tempo durante o qual a rede esteve operacional baseada no registo de incidências).

1.4 Estrutura da Dissertação

Este relatório de dissertação está dividido em oito capítulos principais.

Introdução. O primeiro capítulo descreve o contexto e a motivação da presente dissertação além dos objetivos que se pretendem atingir.

Estado da Arte. Neste capítulo, os termos e definições relacionados com este tópico serão revistos e discutidos, bem como trabalhos relacionados.

Problema e os seus desafios. Este terceiro capítulo discute o problema e os desafios que precisam de ser ultrapassados para alcançar os objetivos propostos. É também sugerida uma solução para o problema em termos de arquitetura do sistema recomendado.

Análise e Conceção. É no quarto capítulo que é feita uma análise sobre os requisitos e casos de uso da aplicação, é também idealizada a arquitetura da mesma.

Implementação. O capítulo 5, contém as tomadas de decisão efetuadas durante a implementação da aplicação, bem como uma explicação do funcionamento da mesma.

Deployment. Neste capítulo, são apresentadas as alterações que foram necessárias fazer para dar *deploy* da aplicação.

Testes. Este capítulo, apresenta uma descrição dos testes realizados à aplicação e os seus resultados.

Conclusões e trabalho futuro. Neste capítulo final faz-se uma revisão do que foi feito, reporta-se o atual estado do plano de trabalho proposto e são exibidas propostas para trabalho futuro.

Capítulo 2

Estado da arte

Neste capítulo serão abordadas as tecnologias relacionadas com o caso de estudo desta dissertação.

2.1 Distributed Ledger Tecnology

A **Distributed Ledger Tecnology** (**DLT**) representa uma estrutura de dados, o *ledger*, que está espalhada por diversas regiões, uma vez que esta se encontra armazenada em vários computadores. Esta tecnologia assenta numa rede distribuída, **Peer-to-Peer** (**P2P**), que valida transações dessa rede sem necessitar de um intermediário como autoridade central [Li and Kassem, 2021]. Em vez de uma autoridade central para validar as transações, estas são validadas pelos próprios nodos da rede através da utilização de um algoritmo de consenso. Este algoritmo permite que os nodos cheguem a um consenso sobre as transações e, consequentemente, a um consenso sobre o estado do *ledger* da rede. Desta forma, todos os nodos possuem o mesmo *ledger*.

Esta tecnologia é caracterizada por apenas se poder adicionar dados ao *ledger*. Os dados são de escrita única e por isso são imutáveis. O *ledger* é partilhado por toda a rede sendo distribuído pela mesma, ou seja, todos os nodos têm uma cópia integral ou parcial do *ledger* e partilham essa mesma cópia com outros nodos[1].

2.2 Blockchain

A *blockchain* corresponde a uma cadeia de blocos. Esta pode ter diferentes implementações, mas a maioria utiliza um *block header* e um *block data*. O *block header* é composto pelo número do bloco, a *hash* do bloco anterior, uma *hash* do *block data*, uma marca temporal (*timestamp*), o tamanho do bloco e o valor do *nonce*. Mediante a *blockchain*, o *nonce* pode ou não ser utilizado e este corresponde a um valor

[1] https://101blockchains.com/what-is-dlt/

que é alterado pelos mineradores com vista a cumprir as restrições da rede. O *block data* é composto pela lista de transações e pode conter outros dados [Yaga et al., 2019]. Através desta implementação, cada bloco está ligado ao bloco anterior formando uma cadeia, de onde deriva o nome da tecnologia.

A *blockchain* é uma **DLT** e, como tal, possui as características apresentadas anteriormente. Neste sistema é fácil de verificar a imutabilidade dos dados, pois uma vez que cada bloco possui a *hash* do bloco anterior, se algum dado do bloco for alterado vai resultar numa nova *hash*, que por consequência altera todas as *hash* dos próximos blocos [Yaga et al., 2019]. Nesta tecnologia são utilizados modelos de consenso para que todos os nodos da rede possuam o mesmo *ledger*, ou seja, para adicionar um novo bloco à rede é necessário que todos os nodos concordem no novo bloco a ser publicado. Para publicar novos blocos é necessário escolher as transações que vão ser incluídas nesse novo bloco, este processo é chamado de mineração, e quem decide que transações são incluídas no novo bloco são utilizadores específicos denominados de mineradores [Alharby and van Moorsel, 2018]. De acordo com [Bambara and Allen, 2018], existem 3 tipos de *blockchain*: pública, privada e consócio.

Blockchain Pública

Uma *blockchain* pública é normalmente denominada como "*permissionless blockchain*", uma vez que qualquer utilizador pode aceder à mesma e interagir sem ser necessário qualquer tipo de permissões. Este tipo de *blockchain* assemelha-se à internet pública e está presente na maioria das criptomoedas.

Como modelo de consenso, neste tipo de *blockchain*, o **Proof-of-Work** (**PoW**) e o **Proof-of-Stake** (**PoS**) são os mais utilizados. O **PoW** é um dos modelos de consenso mais comuns em *blockchain*, quando este é utilizado o mineradores têm de resolver um puzzle criptográfico complexo para adicionar o novo bloco à *blockchain* [Khan et al., 2020]. Segundo [Yaga et al., 2019], o primeiro a resolver o puzzle publica o novo bloco, a solução do puzzle prova que o utilizador realizou o trabalho, daí o nome **Proof-of-Work**. O **PoS** tem como pressuposto que os utilizadores que têm mais criptomoedas não irão prejudicar a rede [Khan et al., 2020]. De acordo com [Yaga et al., 2019], os utilizadores podem "apostar"a criptomoeda da rede, esta aposta é um investimento na rede e desta forma quanto maior a quantidade apostada maior a probabilidade de ser escolhido para publicar o novo bloco.

As *permissionless blockchains* são caracterizadas pelo seu anonimato, uma vez que os utilizadores que participam não necessitam de revelar a sua identidade. Além disso, estas *blockchains* são imutáveis, descentralizadas e transparentes, qualquer utilizador consegue aceder a uma cópia do *ledger* a qualquer altura. Outra característica que atrai utilizadores para a rede e que incentiva o bom comportamento são as recompensas que estas oferecem aos mineradores por descobrir ou validar novos blocos.

Habitualmente este género de *blockchain* utiliza modelos de consenso que gastam mais recursos, tanto computacionalmente como a nível energético. Como exemplos temos a Bitcoin[2] e Ethereum[3].

Blockchain Privada

A *blockchain* privada é parecida com uma internet privada, no sentido em que apenas os utilizadores com autorização têm acesso à mesma. Desta forma, as permissões dos utilizadores (leitura e escrita) na *blockchain* são geridas pela empresa ou organização que a controla. Geralmente, são utilizados modelos de consenso mais baratos computacionalmente, pois existe algum nível de confiança entre os participantes. Neste tipo de *blockchain* é possível revelar informação parcialmente com base na identidade do utilizador. Por exemplo, se ocorrer uma transação entre dois utilizadores, todos na rede conseguem ver que a transação ocorreu, mas apenas os envolvidos conseguem ver os detalhes da mesma. Neste tipo de *blockchain* existe uma maior possibilidade de alterar o estado do *ledger*, comparado com uma *blockchain* pública, pois uma vez que o tamanho da rede é menor, se a maioria do nós responsáveis pela publicação dos blocos forem mal-intencionados, podem alterar o estado do *ledger*. Porém, uma vez que se sabe a identidade de cada participante, se alguém apresentar um mau comportamento, esse utilizador pode ter o acesso revogado e podem ser tomadas medidas legais. Exemplos deste género de *blockchain* são: Corda[4] e MultiChain[5].

Blockchain de Consórcio

Uma *blockchain* de consórcio, é um tipo de *ledger* distribuído onde o modelo de consenso sobre o conteúdo do *ledger* é gerido por um grupo pré-determinado de nós. Por exemplo, um grupo de nove instituições financeiras, cada uma gerindo um nó, e das quais cinco (como o Supremo Tribunal dos Estados Unidos) devem assinar em cada bloco para que este seja considerado válido. O acesso à *blockchain* pode ser aberto ao público ou limitado aos membros participantes. Além disso, existem também abordagens híbridas em que o *hash* raiz dos blocos é tornado público, fornecendo ao mesmo tempo um API que permite um número limitado de consultas e fornecendo provas criptográficas de certas partes do estado da *blockchain* [Bambara and Allen, 2018]. No que diz respeito à imutabilidade, este tipo de *blockchain* tem o mesmo comportamento de uma *blockchain* privada. Como exemplos temos, o R3[6] e o Quorum[7]

2 https://bitcoin.org/en/

3 https://ethereum.org/en/

4 https://corda.net/

5 https://www.multichain.com/

6 https://www.r3.com/

7 https://consensys.net/quorum/

blockchain.

A Tabela 1 compara os diferentes tipos de *blockchain*.

Tabela 1: Comparação entre os diferentes tipos de *blockchain*, [Khettry et al., 2021]

Propriedade	Blockchain Pública	Blockchain de Consórcio	Blockchain Privada
Determinação do consenso	Todos os mineradores	Conjunto de nós selecionados	Uma organização
Permissão de leitura	Pública	Pode ser pública ou restrita	Pode ser pública ou restrita
Imutabilidade	Quase impossível de alterar	Pode ser alterada	Pode ser alterada
Eficiência	Baixa	Alta	Alta
Centralização	Não	Parcial	Sim
Processo de consenso	Permissionless	Permissioned	Permissioned

Agora que os conceitos-base e as diferentes tipologias de *blockchain* foram apresentadas, abordar-se uma comunidade que apresenta vários instrumentos de desenvolvimento em *blockchain* em diversas áreas da indústria.

2.3 Hyperldeger Foundation

A Hyperledger Foundation [8] é uma comunidade *open source*, liderado pela The Linux Foundation, que representa um local neutro para colaborações de desenvolvimento de software em *blockchain*. Esta comunidade tem como foco desenvolver ferramentas, bibliotecas e *frameworks* de desenvolvimento de software em *blockchain* para soluções empresariais. Os principais objetivos da mesma são:

- Criar *frameworks* de *distributed ledger* e código base para suportar transações comerciais, de qualidade empresarial e open source.
- Fornecer infraestruturas neutras, abertas e orientadas pela comunidade, apoiadas por uma governação técnica e empresarial.

[8] https://www.hyperledger.org/about

- Construir comunidades técnicas para desenvolver **Proof-of-Concept** (**PoC**) em *blockchains* e *distributed ledger*, casos de utilização, pistas de campo e implementações.
- Educar o público sobre a oportunidade do mercado para a tecnologia da *blockchain*.
- Promover a comunidade de comunidades adotando uma abordagem de conjunto de ferramentas com muitas plataformas e estruturas.

Atualmente existem seis projetos do Hyperledger graduados, ou seja, estes projetos já têm código base totalmente funcional, uma comunidade ativa e uma cobertura de testes proporcional aos outros projetos.

Hyperledger Aires

Este projeto foi desenhado para facilitar iniciativas e soluções de criação, transmissão e armazenamento de credenciais digitais verificáveis. O Hyperledger Aries[9] é uma biblioteca que fornece ferramentas partilhadas, reutilizáveis e interoperáveis para gestão de identidades. As principais características do Aries[10] são:

- Uma interface para criar, assinar e ler transações da *blockchain*.
- Utiliza um elemento de armazenamento criptográfico para guardar a informação com segurança.
- Um sistema de mensagens peer-to-peer criptografadas, que suporta interações fora do *ledger* entre clientes que usam vários protocolos de transporte.
- Suporta a troca de credenciais verificáveis em vários formatos.
- Protocolos que permitem implementar e instalar agentes Aries independentes, Aries Interop Profiles.
- Um conjunto de implementações da *framework* Aries pronto para produção (e várias provas de conceito) permitindo diferentes casos de utilização e instalações. As estruturas são implementações de casos de utilização específicos de agentes de Áries, tais como uma carteira móvel, um emissor/verificador de credenciais verificável de uma empresa, etc.
- Um conjunto de testes de agente para permitir testes contínuos de interoperabilidade de agentes e estruturas de agentes.

[9] https://www.hyperledger.org/use/aries

[10] https://wiki.hyperledger.org/display/ARIES/Hyperledger+Aries

Hyperledger Besu

O projeto Hyperledger Besu[11], escrito em Java, corresponde a um cliente Ethereum. Suporta diferentes algoritmos de consenso (**PoS**, **PoW**, **PoA** e **Istanbul Byzantine Fault Tolerant** (**IBFT**)) e pode ser utilizado na rede pública Ethereum, em rede teste (por exemplo, Rinkeby, Ropsten, e Görli) e em redes privadas. As suas principais características são:

- A Ethrereum Virtual Machine permite a implementação e execução de *smart contracts* através de transações dentro da Ethereum *blockchain*.
- Implementa diversos algoritmos de consenso tais como: **PoW**, **PoS**, **Proof-of-Authority** (**PoA**).
- Utiliza uma base de dados RocksDS *key-value* para armazenar dados da chain localmente.
- Rede *peer-to-peer*.
- Fornece APIs orientadas para o utilizador.
- Permite a monitorização do desempenho da rede e do nó.
- Habilidade de manter transações privadas entre as partes envolvidas.
- Permite que apenas nós ou contas específicas participem na rede através da atribuição de permissões a contas e nós.

Hyperledger Fabric

O Hyperledger Fabric [12] é uma *framework* para desenvolver aplicações de *permissioned distributed ledger*, concebida para ser utilizada em ambientes empresariais. Este apresenta as seguintes características[13]:

- Arquitetura modular: tem um desenho que permite que diferentes componentes da plataforma possam ser retiradas ou trocadas por outras alternativas.
- *Smart Contracts*: o Hyperledger Fabric suporta a implementação de *smart contracts*, denominados *chain codes*, este podem ser escritos em Go, Node.js e Java.
- Modelos de consenso *pluggables* que permitem aos desenvolvedores escolher o modelo de consenso que acham apropriado para a sua aplicação.

[11] https://wiki.hyperledger.org/display/BESU/Hyperledger+Besu

[12] https://wiki.hyperledger.org/display/fabric

[13] https://hyperledger-fabric.readthedocs.io/en/latest/whatis.html

- Privacidade e confidencialidade: o Fabric proporciona privacidade e confidencialidade através da sua arquitetura de canais e da sua funcionalidade de dados privados.
- Escalabilidade: suporta um grande número de transações e pode ser escalado para responder às necessidades de aplicações de larga escala.
- Interoperabilidade: O Fabric foi concebido para ser compatível com outras tecnologias de *distributed ledger* e pode ser integrado com os sistemas existentes.

Hyperledger Indy

Este projeto[14] foi criado para desenvolver sistemas de identidade descentralizada. O Indy oferece ferramentas, bibliotecas e componentes reutilizáveis para fornecer identidades digitais em *blockchains* ou outros *distributed ledgers*, para poderem ser interoperáveis entre domínios administrativos e aplicações. As suas principais características[15] são:

- *Distributed ledger* construído propositadamente para a identidade descentralizada.
- Desenhado para resistir a correlações.
- DIDs (Identificadores Descentralizados) os quais são globalmente únicos e resolvíveis (por um *ledger*) sem necessidade de qualquer autoridade de resolução centralizada.
- Os identificadores em pares criam relações seguras, de uma pessoa para outra pessoa, entre quaisquer duas entidades.
- Credencias verificáveis em formatos interoperáveis para troca de atributos e relações digitais (atualmente em processo de normalização no W3C).
- Protocolos de Zero Knowledge Proofs que permitem provar se parte ou todos os dados num conjunto de credenciais são verdadeiros sem revelar nenhuma informação adicional, inclusive o provedor.

Hyperledger Iroha

O Iroha[16] é um projeto desenvolvido com o intuito de complementar o Fabric, o Sawtooth e outros potenciais projetos, através do poder de uma *permissioned blockchain* com um modelo de consenso Crash Fault

[14] https://wiki.hyperledger.org/display/indy

[15] https://indy.readthedocs.io/en/latest/index.html

[16] https://wiki.hyperledger.org/display/iroha/Hyperledger+Iroha

Tolerant que permite criar aplicações rápidas, seguras e confiáveis. Este tipo de modelo de consenso permite o funcionamento da rede mesmo que alguns nós da mesma falhem [Baliga, 2017]. Este apresenta uma *framework* com um conjunto de comandos, permissões e *queries* pré definidos, que podem ser utilizados com várias bibliotecas para desenvolver aplicações para plataformas *mobile* e *desktop*. As principais características deste projeto[17] são:

- Manutenção e instalação simples.
- Variedade de bibliotecas para os desenvolvedores.
- Controlo de acessos baseado em *roles*/papéis desempenhados.
- Design modular.
- Gestão de bens e identidade.
- Algoritmo de consenso Crash Fault Tolerant de alto desempenho, Yet Another Consensus.

Hyperledger Sawtooth

Hyperledger Sawtooth[18] é uma estrutura *blockchain* open-source altamente modular e flexível que permite a construção, implementação e execução de aplicações e redes de *ledger* distribuído. A arquitetura separa o sistema central do domínio da aplicação, proporcionando aos *smart contracts* a capacidade de especificar as regras de negócio sem a necessidade de conhecimento técnico sobre a arquitetura subjacente do sistema central. Com segurança reforçada, essa plataforma é amplamente utilizada para fins corporativos e possui uma ampla gama de possíveis casos de uso, incluindo, mas não limitado a, **Internet of Things** (**IoT**) e Finanças. As principais características apresentadas[19] pela organização são:

- Algoritmos de consenso *pluggable*, é possível alterar o modelo de consenso em tempo real por transação.
- Inclui o modelo de consenso **Proof-of-Elapsed-Time** (**PoET**).
- Suporta a implementação de *smart contracts* em várias linguagens, Python, JavaScript, Go, C++, Java e Rust.
- Suporta *smart contracts* de rede Ethereum através da intregação do Hyperledger Burrrow.

[17] https://iroha.readthedocs.io/en/main/index.html

[18] https://sawtooth.hyperledger.org/docs/1.2/#try-hyperledger-sawtooth

[19] https://sawtooth.hyperledger.org/docs/1.2/#ethereum-contract-compatibility-with-seth

- Execução de transações paralelas para taxa de transferência adicional.

2.4 Trabalho Relacionado

Nesta secção são abordados projetos relacionados com caso de estudo desta dissertação, identificado como a criação de uma aplicação de *logging* em *blockchain*.

O projeto JusticeChain foi desenvolvido para o sistema judicial português e representa um *log system* para assegurar que as informações registadas não são alteradas e podem ser consultadas ou auditadas por indivíduos com as devidas autorizações. Neste projeto foi criada uma aplicação, utilizando o Hyperledger Fabric e o Hyperledger Composer, capaz de registar *logs* em *blockchain* e auditá-los. Este modelo apresenta duas principais componentes: a componente de cliente e a componente de *blockchain*. A componente de *blockchain* é responsável pelo registo dos *logs* e por apresentá-los. A componente de cliente é responsável pela criação de *logs* e envio dos mesmos para a componente de *blockchain* [Belchior et al., 2019].

[Yeung et al., 2022] implementaram um sistema de *logging* para registar o trabalho realizado em elevadores. Nomeadamente, as empresas de gestão de elevadores devem ter um diário de bordo para os seus elevadores. Os trabalhadores de empresas contratadas têm de registar neste diário trabalhos realizados e parâmetros do mesmo. Por fim, o Departamento de Serviços Elétricos e Mecânicos do Governo de Hong Kong tem de ter acesso a esse diário quando realiza uma auditoria. Foi utilizada uma arquitetura de consórcio, pois existem três organizações que contém a informação a ser armazenada. Deste modo, se fosse utilizada uma arquitetura privada, a empresa que controla a rede poderia auto beneficiar-se. Este projeto visa reduzir a perda de tempo ao registar e aceder a um diário de bordo físico, bem como evitar a alteração dos registos e a perda dos mesmos. A solução foi criar um *log book* digital, implementado em *blockchain*, e desenvolvido com o Hyperledger Fabric. Foram criados *smart contracts* que permitem o registo e o acessos dos *logs*, e foi utilizada uma arquitetura em que todas as entidades ficam com uma cópia do *log book* digital.

[Pourmajidi and Miranskyy, 2018] desenvolveram o Logchain que corresponde à criação de um *ledger* hierárquico capaz de armazenar *logs* de plataformas *cloud*. Esta solução transformou uma *blockchain* básica num *ledger* hierárquico. Através de uma API, os clientes conseguem enviar, recolher e verificar dados do armazenamento. A arquitetura deste sistema corresponde a duas *blockchains* sobrepostas, ou seja, existe uma super *blockchain* que possui a estrutura normal de uma *blockchain*, mas em cada bloco que a compõem em vez de termos uma parte do bloco com as transações, essa parte tem armazenada

outra *blockchain* circular. Por sua vez, essa *blockchain* circular possui as transações que, neste caso, representam os *logs*.

Capítulo 3

Análise e Conceção

Neste capítulo é feita uma análise ao problema que se quer resolver, às aplicações a desenvolver, são também descritos o levantamentos de requisitos, os casos de uso, a arquitetura das aplicações e a estrutura de dados utilizada. Foi utilizada a linguagem UML (Unified Modeling Language) para construir os diagramas de especificação dos sistemas.

3.1 Problema

Uma vez que a dstelecom aluga a sua infraestrutura de rede de fibra ótica a empresas de serviços de telecomunicações, estas representam os seus clientes. Deste modo, a dstelecom tem a necessidade de informar os períodos em que a rede está inativa aos seus clientes, os quais têm de confiar nas métricas apresentadas. Presentemente, estes dados são obtidos da seguinte maneira: a plataforma NetNumem U31 Unified Management System[1] (NMS) envia alertas acerca dos dispositivos da rede. Se o alerta corresponder a um dispositivo inativo, manualmente, os funcionários da dstelecom abrem um *ticket* através da plataforma OTRS[2]. Quando o *ticket* é fechado, os funcionários registam manualmente, numa base dados MySql, os tempos dos alertas. Todos os dias, esta base de dados é atualizada acerca dos clientes de toda a rede, sendo calculado o tempo de inatividade da rede do dia anterior. Estes processos dão abertura para o erro humano, o que pode diminuir a confiança dos clientes nos valores apresentados pela dstelecom.

Neste seguimento, surge a motivação para esta dissertação. Pretende-se criar um sistema capaz de resolver dois problemas: automatizar o processo anteriormente descrito e aumentar a confiança dos dados fornecidos ao cliente. Este sistema tem que registar os alertas referentes aos dispositivos da rede em blockchain e permitir consultar os mesmos, garantindo assim que os dados apresentados não são

[1] http://www.edgetech.lv/edge-technologies-main/software-solutions/zte-netnumen-u31/

[2] https://otrs.com/pt/home/

adulterados. Para além de registar a data e a hora dos alertas, é também necessário registar qual o tipo de alerta: ativo, não incidente, incidente com um responsável identificado, incidente com uma equipa destacada, ou incidente sem equipa destacada, tornando automático o processo de triagem de um alarme.

Para responder aos problemas apresentados, idealizou-se a criação de duas aplicações. Uma aplicação responsável por receber todos os alarmes e registá-los na *blockchain*. Esta aplicação também é encarregue por identificar o responsável de cada alarme e o seu tipo. A segunda aplicação terá o propósito de possibilitar consultar os alarmes presentes na *blockchain*, estas consultas vão permitir monitorizar diferentes atributos dos dispositivos da rede. A Figura 1, representa a solução descrita.

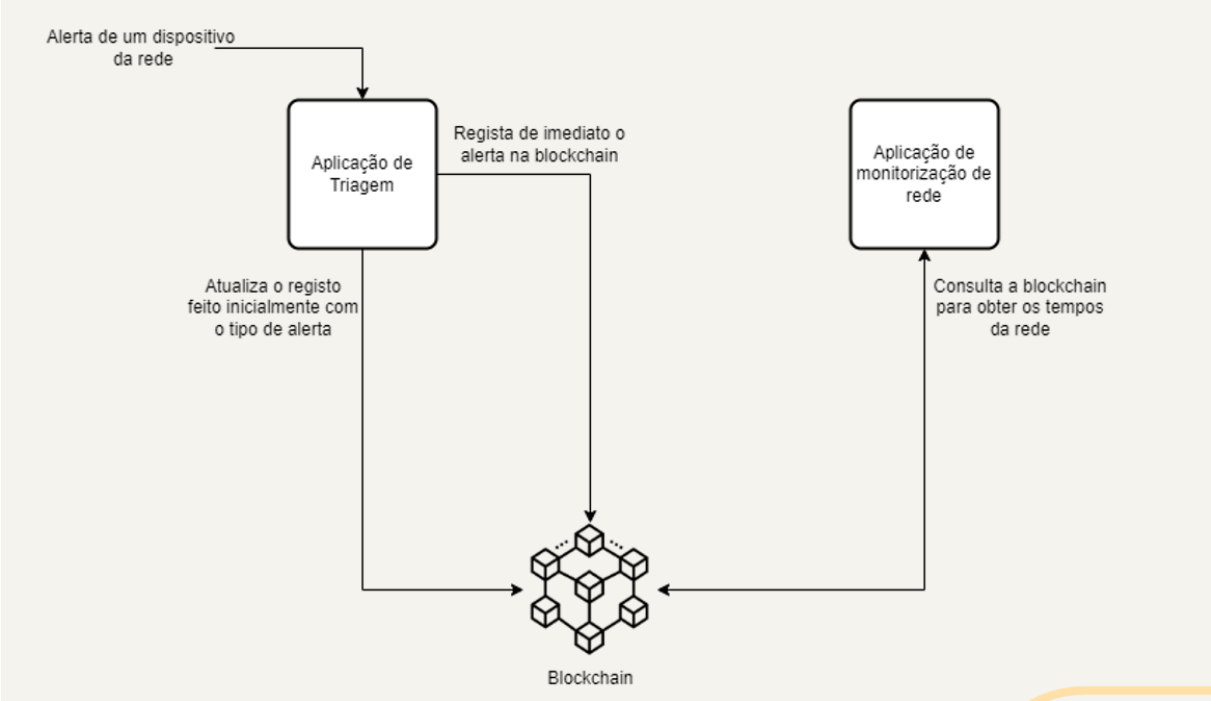


Figura 1: Representação Minimalista da Solução do Problema

3.2 Requisitos

Para desenvolver software de forma eficaz, é essencial analisar e estabelecer os requisitos. Este processo permite-nos definir como o software deve ser estruturado internamente e como os seus componentes devem ser organizados. Ao compreender e documentar os requisitos, podemos garantir que o software cumpre os objectivos desejados e funciona como pretendido. Este passo é crucial para criar uma base sólida e uma estrutura que permita que o software satisfaça as necessidades e expectativas identificadas dos seus intervenientes. De seguida, são apresentados os requisitos funcionais das duas aplicações, que

foram levantados em conjunto com a dstelecom, estes descrevem as funcionalidades e comportamentos que o sistema deve ser capaz de fazer.

De forma a classificar a prioridade de cada requisito foi escolhido um sistema rápido e fácil de utilizar, o método MoSCoW, segundo [Hudaib et al., 2018]. Neste sistema a classificação é feita da seguinte maneira[3]:

- **M** - Must Have, requisitos que devem estar obrigatoriamente implementados.
- **S** - Should Have, requisitos importantes mas não são críticos.
- **C** - Could Have, requisitos desejáveis mas menos importantes.
- **W** - Won't Have, requisitos que não vão ser implementados mas servem de referência futura.

3.2.1 Aplicação de Triagem

Após a análise da solução proposta, foram identificados os seguintes requisitos funcionais, no que diz respeito à Aplicação de Triagem, presentes na Tabela 2.

Tabela 2: Tabela de Requisitos Funcionais da Aplicação de Triagem

Identificador	RF-01
Título	Registo de alarme.
Descrição	O sistema deverá ser capaz de registar um alarme na blockchain.
Prioridade	**M**
Condições de Aceitação	O alarme registado deve ser igual ao alarme recolhido pelo sistema de monitorização.
Identificador	RF-02
Título	Registo do responsável do alarme
Descrição	O sistema deverá ser capaz de registar na blockchain caso o alarme seja provocado por alguma equipa a realizar trabalhos no local.
Prioridade	**S**
Condições de Aceitação	O responsável tem de corresponder a um funcionário presente na locken ou no Live Solutions.

[3] https://www.agilebusiness.org/dsdm-project-framework/moscow-prioririsation.html, visitado a 19/07/2023

Identificador	RF-03
Título	Identificar um incidente.
Descrição	O sistema deverá ser capaz de identificar se o alarme que está a ser tratado corresponde a um incidente ou faz parte de um incidente.
Prioridade	**M**
Condições de Aceitação	Para ser considerado um incidente é necessário existir pelo menos dois alarmes de inativos com uma diferença entre 20 minutos a 2 horas.
Identificador	RF-04
Título	Definir tempo de fim de um alarme.
Descrição	O sistema deverá ser capaz de registar o tempo de fim de um alarme na blockchain.
Prioridade	**M**
Condições de Aceitação	Para registar o tempo de fim de um alarme é necessário, que o respectivo alarme exista na blockchain sem tempo de fim definido e o tempo a registar tem de ser válido.
Identificador	RF-05
Título	Identificar um trabalho programado.
Descrição	O sistema tem de ser capaz de identificar que o alarme a ser tratado é um trabalho programado.
Prioridade	**S**
Condições de Aceitação	O alarme tem de possuir um **Repartidor Ótico Exterior** (**ROE**) que está presente na lista dos trabalhos programados e os dias têm de coincidir.
Identificador	RF-06
Título	Exibir incidentes.

Descrição	O sistema tem de ser capaz de mostrar no ecrã os incidentes que estão a ocorrer.
Prioridade	**S**
Condições de Aceitação	O utilizador tem de poder visualizar os incidentes organizados por **ROE**, Número de incidente, Data de Início, número de cliente de cada operadora.

3.2.2 Aplicação de Consulta

Através do método anterior, foram também levantados os requisitos funcionais da Aplicação de Consulta, estes são apresentados na Tabela 3.

Tabela 3: Tabela de Requisitos Funcionais da Aplicação de Consulta

Identificador	RF-01
Título	Consultar um incidente por nº de incidente.
Descrição	O Sistema deverá ser capaz de apresentar os detalhes de um incidente através do número do mesmo.
Prioridade	**M**
Condições de Aceitação	Os dados apresentados pelo Sistema têm de corresponder aos dados armazenados na blockchain.
Identificador	RF-02
Título	Consultar incidentes por **ROE**.
Descrição	O Sistema deverá ser capaz de apresentar uma lista de incidentes com o **ROE** fornecido pelo utilizador.
Prioridade	**S**
Condições de Aceitação	A lista de incidentes apresentada tem de corresponder aos incidentes armazenados na blockchain com o mesmo **ROE**.
Identificador	RF-03
Título	Consultar incidentes por data.

Descrição	O Sistema deverá ser capaz de apresentar uma lista de incidentes com a data (dia) fornecido pelo utilizador.
Prioridade	**S**
Condições de Aceitação	A lista de incidentes apresentada tem de corresponder aos incidentes armazenados na blockchain com a mesma data.
Identificador	RF-04
Título	Consultar incidentes por responsável.
Descrição	O Sistema deverá ser capaz de apresentar uma lista de incidentes com o responsável fornecido pelo utilizador.
Prioridade	**C**
Condições de Aceitação	A lista de incidentes apresentada tem de corresponder aos incidentes armazenados na blockchain com o mesmo responsável.
Identificador	RF-05
Título	Consultar incidentes por **Fiber-to-the-Home** (**FTTH**).
Descrição	O Sistema deverá ser capaz de apresentar uma lista com os eventos respetivos ao **FTTH** fornecido pelo utilizador.
Prioridade	**S**
Condições de Aceitação	A lista de acessos apresentada tem de corresponder aos acessos armazenados na blockchain com o mesmo **FTTH** .
Identificador	RF-06
Título	Consultar incidentes por **Point of Presence** (**POP**).
Descrição	O Sistema deverá ser capaz de apresentar uma lista de acessos com o **POP** fornecido pelo utilizador.
Prioridade	**C**
Condições de Aceitação	A lista de acessos apresentada tem de corresponder aos acessos armazenados na blockchain com o mesmo **POP**.

3.3 Casos de Uso

Um caso de uso é uma representação ou documentação que descreve uma interação ou cenário específico que envolve atores (indivíduos, grupos ou sistemas) e um sistema ou processo empresarial. Descreve uma sequência de ações ou passos que são dados para alcançar um resultado desejado ou um resultado de valor. Os casos de uso servem como meio de captar e ilustrar os requisitos funcionais de um sistema ou processo empresarial, proporcionando uma compreensão clara da forma como diferentes entidades interagem e colaboram para atingir objetivos ou tarefas específicas. Os casos de uso podem ser apresentados sob a forma de gráficos ou sob a forma escrita[4].

3.3.1 Aplicação de Triagem

Use case: Visualizar um incidente.

Descrição: O Sistema é capaz de mostrar um incidente corretamente.

Ator: Funcionário da dstelecom.

Pré-Condição: Um acesso de inatividade sem responsável entra no sistema e dá origem a um incidente.

Pós-Condição: O acesso é registado na blockchain e é mostrado o incidente no ecrã.

Fluxo normal:

1. Verifica que o acesso é de inatividade.
2. Sistema consulta cadastro de rede.
3. Verifica a que dispositivo corresponde o acesso.
4. Sistema consulta os trabalhos programados.
5. Verifica que não é um trabalho programado.
6. Regista o acesso na blockchain.
7. Regista o acesso localmente.
8. Verifica no armazenamento local que existem outros acessos do mesmo **ROE** inativos.
9. Sistema atribui um número de incidente e atualiza o armazenamento da blockchain.

[4] https://www.ibm.com/docs/pt-br/elms/elm/6.0?topic=cases-use

10. Consulta o Locken[5].
11. Verifica que não há um responsável.
12. Consulta o LiveSolutions[6].
13. Verifica que não há equipa destacada.
14. Mostra no ecrã o incidente identificado

Fluxo alternativo [O acesso é de atividade] (passo 1):

1.1 Verifica no armazenamento local se existe um **FTTH** igual.

1.2 Atualiza o registo com o tempo de fim.

Fluxo exceção [Não existe nenhum registo associado ao **FTTH** recebido] (passo 3):

3.1 Mostra mensagem de erro a informar que não existe dispositivo associado ao **FTTH** recebido.

Fluxo alternativo [O acesso é um trabalho programado] (passo 5):

5.1 Verifica que o acesso é um trabalho programado.

Fluxo alternativo [Não existem outros acessos com o mesmo **ROE**] (passo 8):

8.1 Verifica no armazenamento local que não existem outros acessos do mesmo **ROE**.

Fluxo alternativo [Existe responsável] (passo 11):

11.1 Verifica que existe responsável.

11.2 Atualiza o registo na blockchain com o devido responsável.

11.3 Avança para passo 14.

Fluxo alternativo [Existe equipa destacada] (passo 13):

13.1 Verifica que existe equipa destacada.

13.2 Atualiza o registo na blockchain com a devida equipa destacada.

13.3 Avança para passo 14.

[5] Plataforma de fechaduras inteligentes utilizada pela dstelecom.

[6] Plataforma de atribuição de trabalhos utilizada pela dstelecom.

3.3.2 Aplicação de Consulta

Na Figura 2 são representados os casos de uso da aplicação de consulta.

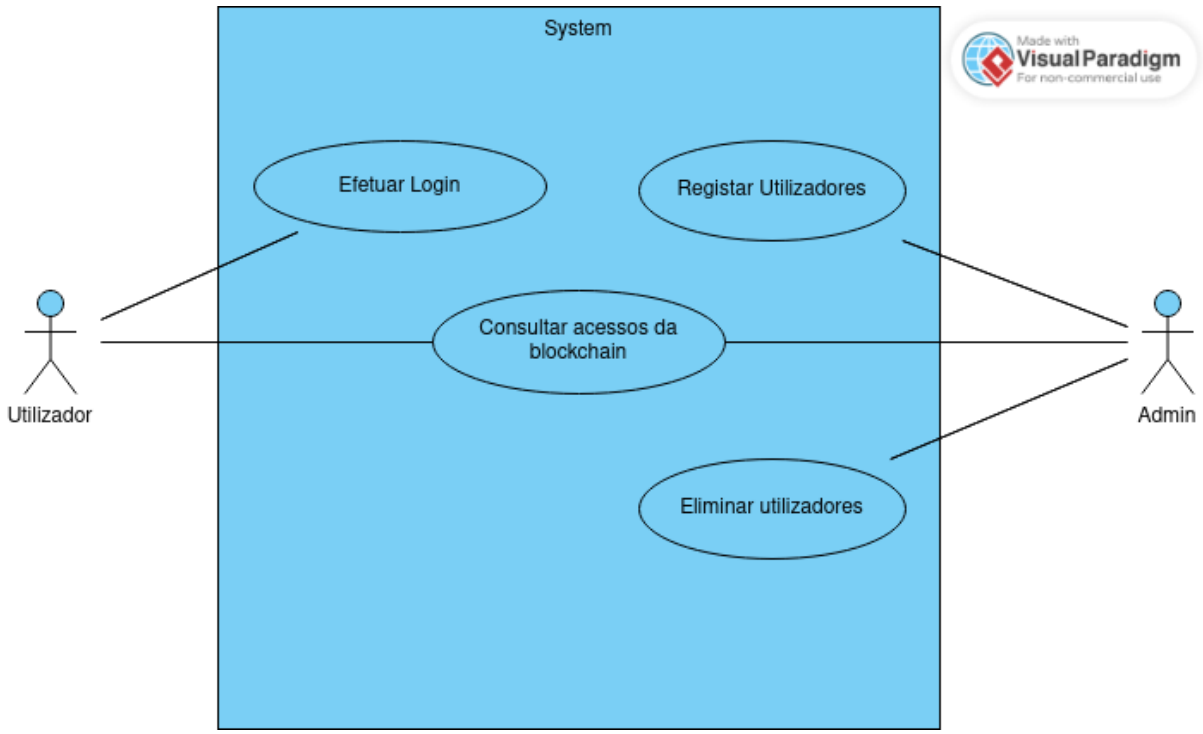


Figura 2: Diagrama de Casos de Uso da Aplicação de Consulta

3.4 Arquitetura Funcional

Nesta secção é apresentada e descrita a arquitetura funcional de cada uma das aplicações desenvolvidas.

3.4.1 Aplicação de Triagem

Conforme foi descrito na Secção 3.1, a solução passa por criar uma aplicação responsável por registar os alarmes na *blockchain* e fazer a triagem dos mesmos em tempo real, de modo a automatizar o processo atualmente feito pelos funcionários da dstelecom. Para ser possível registar um alarme, a aplicação deve ter acesso às seguintes plataformas: **NMS** para receber os alarmes da infraestrutura de rede, à Locken [7] para saber se existem algum técnico responsável pelo alarme, à Live Solutions [8] para saber se existe algum trabalho naquela zona. No caso, como este projeto é um protótipo e permitir o acesso a essas

[7] https://www.locken.eu/

[8] https://www.livesolutions.pt/

plataformas a partir da aplicação teria um custo financeiro adicional, o acesso às mesmas é simulado através de ficheiros estáticos, ou seja, a aplicação vai aceder a ficheiros estáticos que contêm a informação dessas plataformas, cada uma corresponde a um ficheiro independente.

A aplicação desenvolvida corresponde a uma aplicação Desktop, que comunica diretamente com a *blockchain* para proceder ao registo dos alarmes, os ficheiros referidos anteriormente são armazenados localmente pela aplicação, os dados são processados pela mesma e posteriormente são registados na blockchain, como se pode observar na Figura 3. Nesta estão representados 5 ficheiros:

- **Alarmes da Rede** - contém uma captura do **NMS** com todos os alarmes da rede da dstelecom durante um período de tempo.
- **Cadastro da Rede** - contém todos os dispositivos da rede.
- **Trabalhos Programados na Rede** - contém todos os trabalhos programados realizados e por realizar na rede com afetação de clientes.
- **Funcionários na Locken** - contém todos os funcionários, que estão ao momento da extração do ficheiro, que se encontram a operar no **ROE**.
- **Equipas na Live Solutions** - contém todas as equipas, que estão ao momento da extração do ficheiro, estão destacadas para realizar qualquer trabalho na rede tal como incidente, instalações ou suporte a clientes e manutenções na rede.

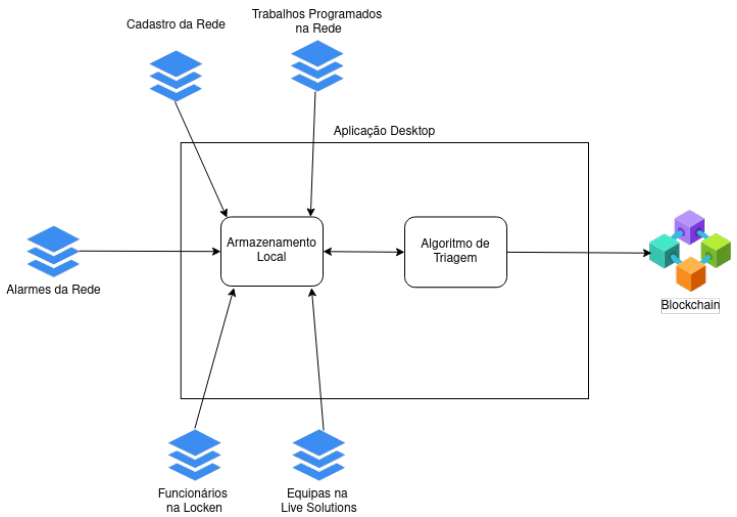


Figura 3: Esquema Representativo da Aplicação de Triagem

3.4.2 Aplicação de Consulta

No que diz respeito à Aplicação de Consulta, na Figura 4 está representada a arquitetura funcional utilizada. Esta aplicação corresponde a uma aplicação Web, que comunica com uma base de dados e a *blockchain*. Na base de dados são guardadas as credenciais dos utilizadores, a *password* sub a forma de *hash*, na *blockchain* estão armazenados os alarmes. O utilizador introduz como input a query que quer, o front-end envia o pedido ao back-end. Este, por sua vez, faz a query à *blockchain* e envia a resposta de volta ao front-end para ser mostrada ao utilizador.

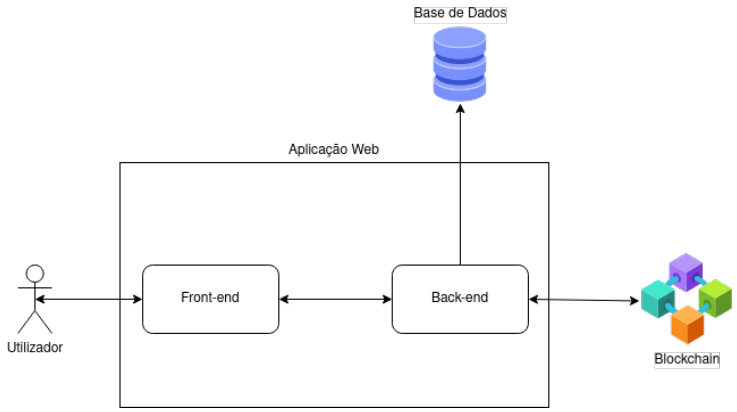


Figura 4: Esquema Representativo da Aplicação de Consulta

3.5 Diagrama de Atividade

Apesar da Aplicação de Triagem possuir apenas um *use case*, este é bastante complexo, conforme foi descrito na Secção 3.3.1. Desta forma, foi construído um diagrama de atividade para ajudar a entender melhor o ciclo de vida de um alarme (ou acesso) quando este é recebido pelo sistema. Como é visível na Figura 5, assim que um alarme é recebido pela aplicação esta verifica se este corresponde a um alarme de atividade ou de inatividade. Ou seja, há dois tipos de alarmes que queremos captar com a aplicação, os que indicam que um dispositivo da rede está inativo e os que indicam que está ativo. Mediante o tipo de acesso, o caminho que este segue no sistema é diferente. No caso de ser um acesso de atividade, parte-se do pressuposto que já existe um acesso de inatividade, e por isso o sistema procura na *blockchain* pelo respetivo acesso de inatividade e atualiza o seu dia e hora de fim com os dados do acesso que está a ser tratado pelo sistema.

No caso de ser de inatividade, então o tratamento do alarme já é mais complexo. Primeiramente, é consultado o ficheiro do cadastro de rede onde é utilizado do código do alarme (**FTTH**) para retirar

informação acerca do dispositivo a que este está associado. Da informação retirada do ficheiro, o campo mais importante é o identificador do **ROE** associado ao alarme, pois é através deste que vão ser retiradas informações dos outros ficheiros. Depois de obter essa informação, é consultado o ficheiro dos trabalhos programados para verificar se o identificador do **ROE** do alarme está na lista de trabalhos, se pertencer então o alarme é descartado. Se o alarme não pertencer aos trabalhos programados é registado de imediato na *blockchain*, são definidos dois *timeouts* associados ao alarme, um de vinte minutos, outro de duas horas. Após estes passos, o sistema verifica se existe outro acesso com o mesmo identificador do **ROE** registado, pois conforme essa informação o próximo passo a ser executado difere. Caso o sistema não tenha mais nenhum alarme com o mesmo identificador do **ROE** registado, então consulta a informação dos ficheiros do Locken e do Live Solutions, por esta ordem. Nestas consultas é procurado novamente pelo identificador do **ROE** associado ao alarme. Caso haja uma correspondência é atualizado na *blockchain* o registo do alarme com o seu respetivo responsável, que pode ser o nome do funcionário (obtido na Locken) ou o nº de ordem da equipa (obtido na Live Solutions).

Caso o sistema encontre registado outro alarme, ainda inativo, com o mesmo identificador do **ROE** verifica se a diferença entre o *timestamp* do alarme encontrado e do alarme que está a ser tratado é superior a vinte minutos e inferior a duas horas. Se estiver dentro desse intervalo então é considerado um incidente. Caso contrário não é considerado incidente e o processamento do alarme prossegue para identificar um responsável pelo mesmo. No cenário em que é identificado que se trata de um incidente, é então necessário verificar se o alarme que está a ser tratado pertence a um incidente que já está a ocorrer ou se corresponde a um novo incidente. Esta decisão é tomada mediante o número de incidente associado ao acesso encontrado pelo sistema. Se esse acesso tiver número de incidente então é atribuído o mesmo número de incidente ao alarme atual e é atualizado o seu registo na *blockchain*. Por outro lado, se não tiver número de incidente atribuído então o sistema atribui um número de incidente ao alarme atual e ao encontrado e atualiza os dois registos na *blockchain*.

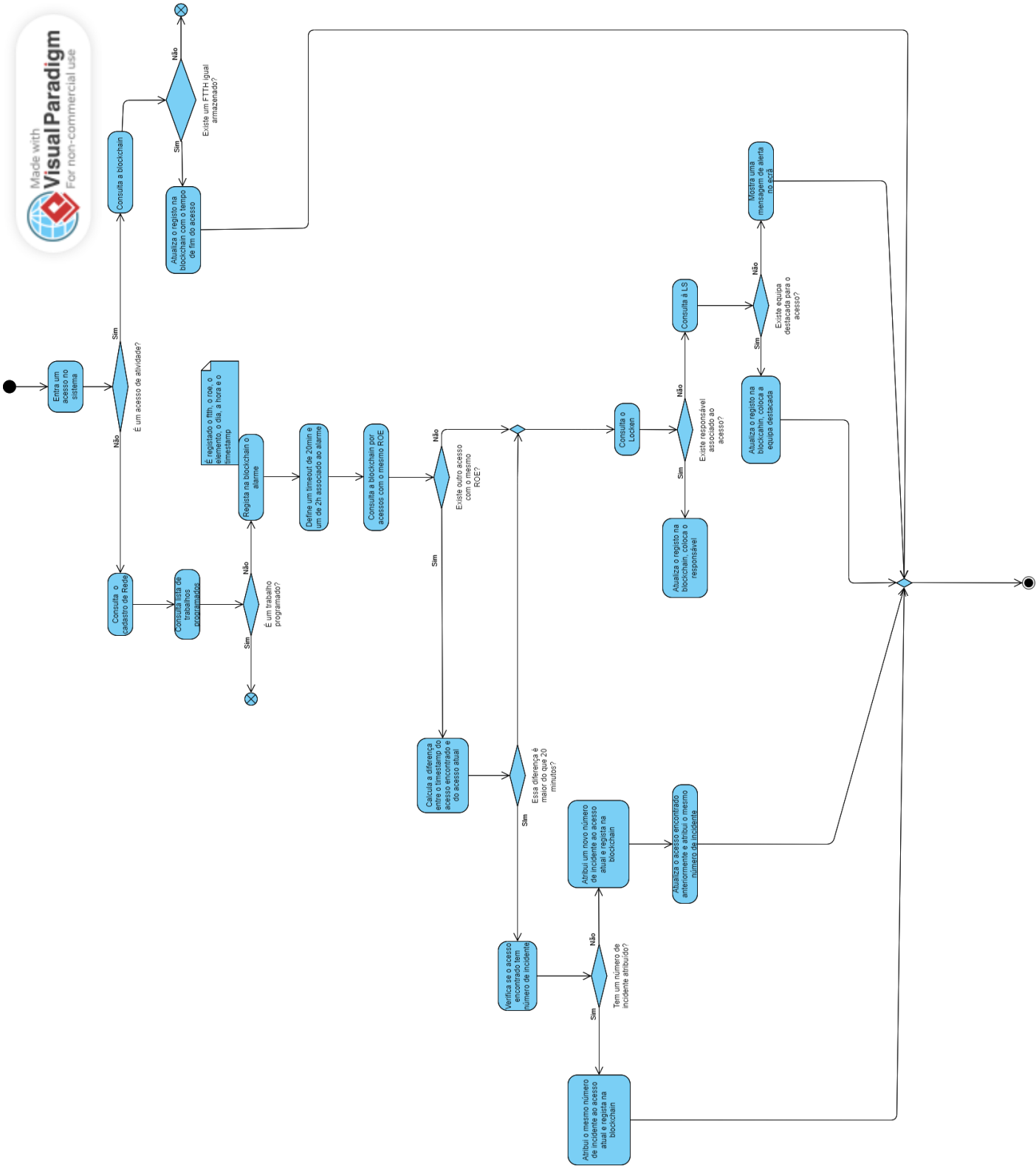


Figura 5: Diagrama de Atividade do Tratamento de um Acesso

Capítulo 4

Implementação

Neste capítulo é descrita a implementação da solução já apresentada, bem como as decisões que foram tomadas durante a mesma e a sua justificação.

4.1 Tecnologias

Na Figura 6 temos a representação das duas aplicações desenvolvidas e as respetivas tecnologias utilizadas nos seus componentes. Para criar a rede *blockchain* foi utilizado o Hyperledger Fabric. É nesta rede que são armazenadas as transações e os alarmes. A Aplicação de Triagem foi desenvolvida com Electron[1], uma ferramenta que permite desenvolver aplicações *desktop* com Javascript, HTML e CSS. Na Aplicação de Consulta foi utlizado MongoDB[2] para a base de dados, no backend foi utilizado Node.js[3] em conjunto com Expressjs[4] para construir a REST API e no frontend foi utilizado React para desenvolver a interface gráfica da aplicação.

1 https://www.electronjs.org/pt/docs/latest/

2 https://www.mongodb.com/pt-br

3 https://nodejs.org/en/docs

4 https://expressjs.com/

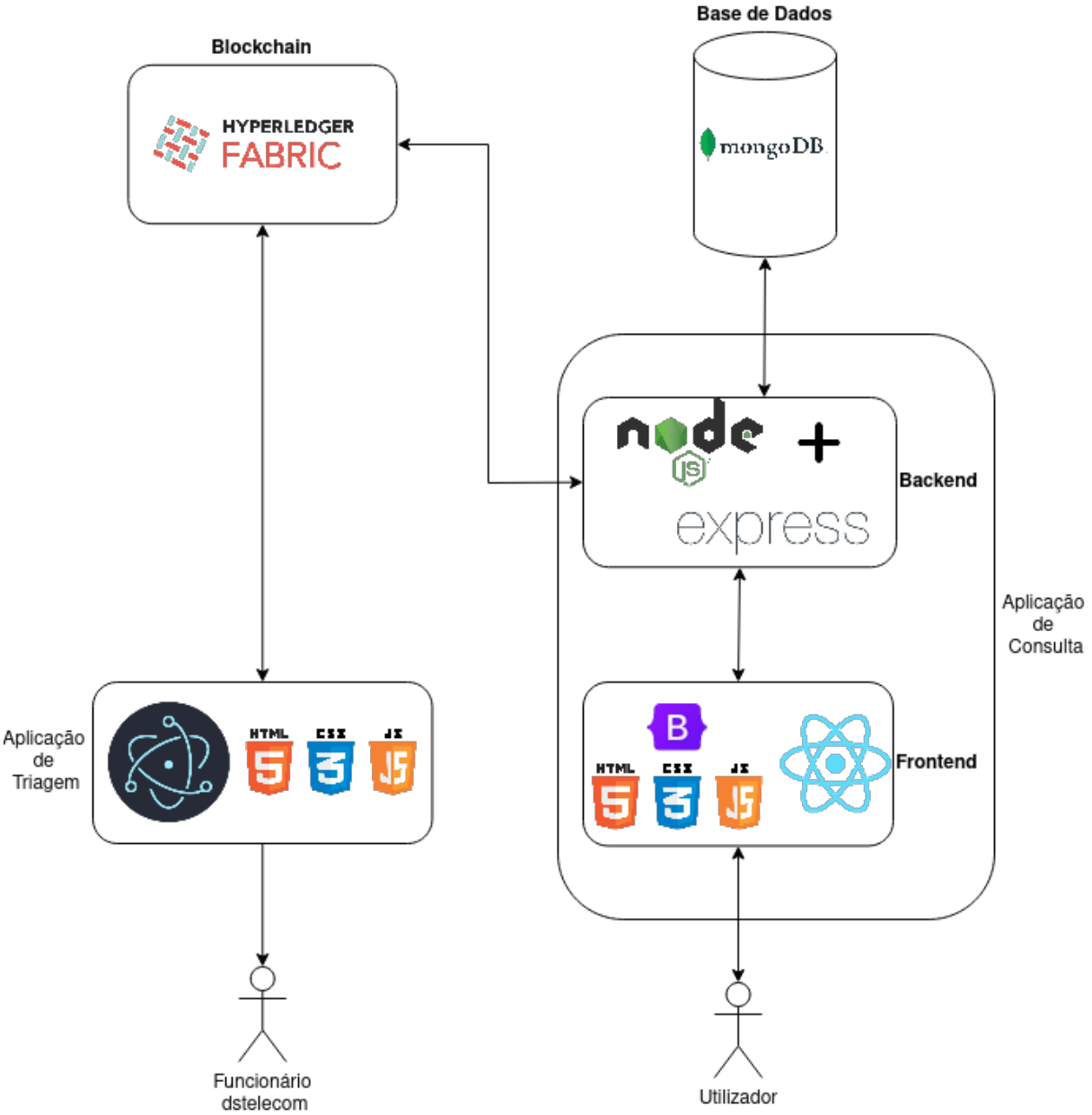


Figura 6: Tecnologias utilizadas no projeto

4.1.1 Hyperledger Fabric

Após uma primeira análise às tecnologias atuais, destaca-se o projeto Hyperledger Fabric, uma vez que este apresenta mais versatilidade e características, que foram consideradas, mais vantajosas para a solução do problema. Nomeadamente, a utilização de *smart contracts* em diversas linguagens de programação, que permitiu escolher a linguagem com o qual estamos mais confortáveis; permitir alterar o modelo de consenso utilizado na rede, que permite alterar este mesmo modelo ao longo do ciclo de vida rede conforme a rede evolui; a sua capacidade de suportar um número elevado de transações, pois são

processados inúmeros alarmes. Para além disso, o Fabric é uma ferramenta bastante utilizada e por isso apresenta uma comunidade maior, mais desenvolvida e mais sólida. Neste capítulo vão ser apresentadas e descritas as principais características do Fabric.

Estrutura de Rede utilizada no Fabric

Uma rede *blockchain* é uma infraestrutura que fornece serviços de *ledger* e de *smart contracts* a aplicações. As transações são geradas através dos *smart contracts* e são distribuidas por todos os *peers* da rede que os armazenam nas suas cópias do *ledger* sem sofrer alterações. Os utilizadores de aplicações podem ser utilizadores finais utilizando aplicações clientes ou administradores de rede de *blockchain*.

Pode ser formado um canal com várias organizações onde através da invocação de *chaincode* ocorrem transações. Quando este é configurado, são definidas as políticas acerca das permissões de cada organização. A configuração inicial não exclui a possibilidade de mais tarde alterar as políticas, caso as organizações concordem.. Para além disso, uma rede pode ter mais do que um canal e é possível associar componentes da rede a vários canais, por exemplo, um nó pertencente a uma organização pode participar em dois canais diferentes.

No Hyperledger Fabric, os termos "rede" e "canal" são sinónimos, pois ambos se referem coletivamente às organizações, componentes, políticas e processos que regem as interações entre organizações dentro de uma estrutura definida.

Ledger

No Fabric, o *ledger* ou livro-razão desempenha um papel crucial como repositório central de detalhes factuais essenciais sobre vários objetos de negócio. Funciona como uma base de dados que contém não só o estado atual desses objetos, incluindo os valores dos seus atributos, mas também mantém um histórico abrangente de todas as transações que contribuíram para esses valores atuais.

Na verdade, o *ledger* presente no Fabric contém duas partes separadas, o estado global (*World State*) e a *blockchain*, ver Figura 7, imagem retirada da documentação oficial do Fabric[5].

Estado Global

O estado global funciona como uma base de dados, e permite aos programas aceder diretamente ao valor atual de um objeto em vez de percorrer o histórico de transações para chegar a esse mesmo estado atual. O estado global começa vazio quando o *ledger* é criado, e pode ser criado novamente a partir da

[5] https://hyperledger-fabric.readthedocs.io/en/release-2.5/ledger/ledger.html

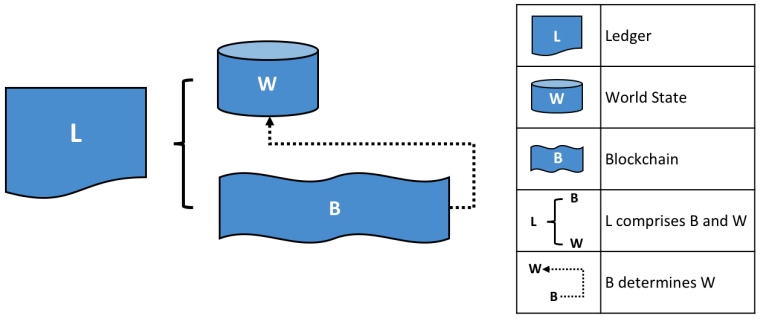


Figura 7: Representação do *Ledger*

blockchain a qualquer momento, pois todas as transações que representam uma mudança válida para o estado é registada na *blockchain*. Se transação adicionar ou alterar um par *key/value* ou mais então altera o estado global, caso contrário não altera. Isso pode ser extremamente conveniente. Por exemplo, quando um *peer* é criado, o estado global é gerado automaticamente. Além disso, se um *peer* falhar de forma inesperada, o estado global pode ser recriado antes que as transações sejam aceites. Na Figura 8, retirada da documentação oficial do Fabric[6], temos um exemplo representativo do estado global.

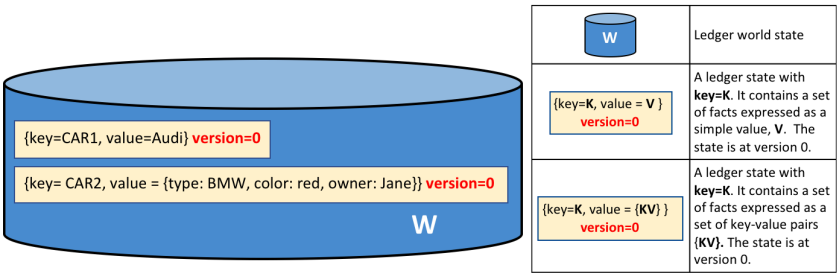


Figura 8: Exemplo do Estado Global

Blockchain

A *blockchain* serve como um arquivo detalhado que documenta toda a história de como os objetos dentro da rede atingiram as suas condições atuais. Atua como um livro de registo de todas as versões passadas do estado, captando todas as alterações que ocorreram ao longo do tempo. Estruturado como uma cadeia sequencial de blocos interligados, cada bloco contém uma série de transações. Estas transações

6 https://hyperledger-fabric.readthedocs.io/en/release-2.5/ledger/ledger.html

representam quer consultas sobre o estado global quer atualizações do mesmo.

Ao contrário do estado global, que se baseia numa base de dados, a *blockchain* é implementada como um ficheiro. Esta estrutura baseada em ficheiros garante que os registos históricos são preservados de forma segura e podem ser facilmente acedidos para efeitos de verificação e auditoria. Ao manter um registo claro e transparente de blocos interligados, a cadeia de blocos fornece um registo fiável e inviolável de todo o histórico de transações na rede. Esta conceção reforça a confiança, a responsabilidade e a integridade dos dados, tornando-a um componente fundamental no funcionamento de sistemas como o Hyperledger Fabric.

Na Figura 9, retirada da documentação oficial do Fabric[7], temos um exemplo da representação da *blockchain* no Hyperledger Fabric.

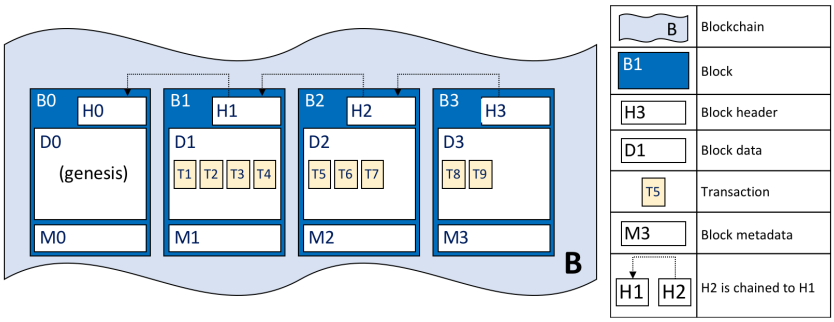


Figura 9: Representação da Blockchain

Transações

As transações guardadas nos blocos da *blockchain* possuem a seguinte estrutura, visível na Figura 10, imagem retirada da documentação oficial do Fabric[8]:

- **Cabeçalho** - este campo contém metadados importantes tais como o nome do *chaincode* e a sua versão, representado por H4.
- **Assinatura** - representado por S4, este campo é utilizado para verificar a integridade e autenticidade dos detalhes da transação, pois contém uma assinatura criptográfica criada pela aplicação cliente gerada pela sua chave privada.

7 https://hyperledger-fabric.readthedocs.io/en/release-2.5/ledger/ledger.html

8 https://hyperledger-fabric.readthedocs.io/en/release-2.5/ledger/ledger.html

- **Proposta** - codifica os *inputs* recebidos da aplicação para o *smart contract*, que dá origem a uma proposta de atualização do *ledger*, representado por P4.

- **Resposta** - representado por R4, corresponde a um conjunto de leitura e escrita (conjunto RW), que contém os valores antigos e os novos do estado global, é o output de um *smart contract* e é aplicado se a transação for válida.

- **Endorsements** - é uma lista de respostas de transações assinadas por todas as organizações necessárias para satisfazer a política de *endorsement*. Embora a transação em si contenha apenas uma resposta, são reunidos vários *endorsements*, a inclusão de respostas de todas as organizações necessárias garante que a transação recebeu os reconhecimentos necessários para ser considerada válida e capaz de atualizar o estado do mundo.

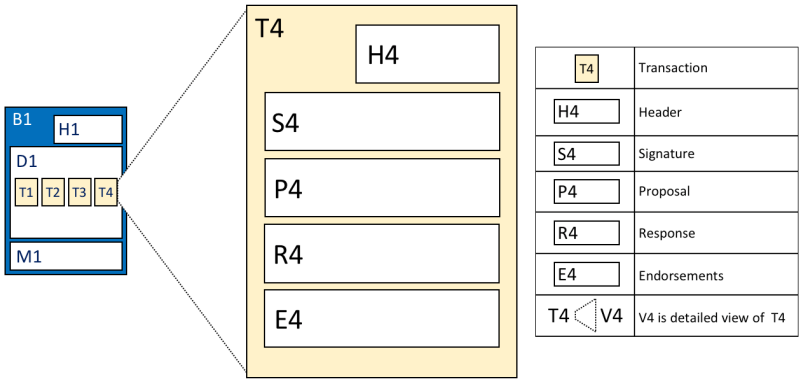


Figura 10: Representação da Estrutura de uma Transação

Atualmente, o Fabric oferece duas soluções para utilizar como estado global, LevelDB e CouchDB. O LevelDB corresponde a uma base de dados *key-value* embebida no processo do peer. O CouchDB é uma alternativa de base de dados externa, esta permite armazenar dados em formato JSON, emitir queries JSON aos dados e utiliza índices para suportar as queries. Por outro lado, o CouchDB corre num processo separado do *peer*. Uma vez que, as aplicações têm de ser capazes de fazer consultas aos acessos armazenados por diferentes campos, foi utilizado o CouchDB na nossa implementação do Fabric.

Canais

No Hyperledger Fabric, os canais atuam como espaços privados independentes, cada um com o seu próprio registo distinto. Operam como *blockchains* separadas que funcionam em paralelo. Este design garante que os dados e as transações dentro de um canal sejam isolados e privados de outros canais.

Dentro de cada canal existe um *ledger* dedicado que mantém um registo de todas as transações específicas desse canal. Além disso, o estado global também é separado para cada canal. Isto garante que os dados de um canal não interferem nem têm impacto nos dados de outro canal, proporcionando maior privacidade e segurança.

Apesar desta separação, o Fabric permite que as aplicações e os contratos inteligentes executados em diferentes canais comuniquem entre si. Esta comunicação permite a partilha de informações relevantes do *ledger* entre canais, quando necessário. Facilita a colaboração e a coordenação entre diferentes partes da rede, mantendo o isolamento essencial entre canais. Desta forma, os participantes podem aceder e trocar dados de forma seletiva, garantindo que a confidencialidade e a integridade são preservadas em toda a rede.

Chaincode

É crucial desenvolver um conjunto normalizado de contratos que inclua terminologia, dados, regulamentos, definições e processos típicos antes de as empresas poderem participar em transações. O modelo de negócio que orienta as interações entre as partes nas transações é moldado por estes contratos considerados como um todo. Podemos converter estes contratos em programas executáveis, também conhecidos como *smart contracts*, utilizando uma rede *blockchain*, abrindo um vasto leque de novas possibilidades. A capacidade de definir regras de governação para diferentes objetos de negócio utilizando *smart contracts* garante uma aplicação automatizada quando estes são implementados. Por exemplo, um *smart contract* pode garantir que o dinheiro seja disponibilizado de acordo com termos predefinidos, melhorando a eficiência do movimento de capital.

Muitos utilizadores do Hyperledger Fabric misturam frequentemente os termos *smart contract* e *chaincode*. Um *smart contract* normalmente define as regras para lidar com o ciclo de vida de um objeto de negócio dentro do estado global. Por outro lado, *chaincode* refere-se ao código responsável pela execução desses *smart contracts*. Uma vez criado o *chaincode*, ele é empacotado e publicado na rede *blockchain*. O *chaincode* rege essencialmente a forma como os *smart contracts* são preparados para serem implementados, enquanto os *smart contracts*, uma vez implementados, controlam a execução das transações.

4.1.2 Electron

Uma vez que a Aplicação de Triagem foi idealizada como uma aplicação *desktop*, foi necessário encontrar uma ferramenta que permitisse construir uma interface gráfica em Javascript, pois a aplicação foi desen-

volvida nessa linguagem. É nesse seguimento que surge então o Electron[9], esta framework tem como principais características:

- **Compatibilidade entre plataformas:** Permite criar aplicações desktop que funcionam em diferentes sistemas operativos como o Windows, o macOS e o Linux.
- **Conjunto de tecnologias Web:** Os programadores podem criar aplicações desktop utilizando ferramentas Web, como HTML, CSS e JavaScript.
- **Atualizações automáticas de software:** Através do módulo autoUpdater do Squirrel para o Electron, pode-se distribuir instantaneamente atualizações de software aos seus utilizadores do macOS e do Windows sempre que for lançada uma nova versão.
- **Distribuição na loja de aplicações:** O Electron tem suporte de para a Mac App Store (macOS), a Microsoft Store (Windows), ou a Snap Store (Linux), o que permite distribuir a aplicação a mais utilizadores.
- **Comunicação de falhas:** O módulo crashReporter permite reunir automaticamente informações dos seus utilizadores sobre falhas nativas e do JavaScript.

Para além destas características, esta ferramenta é *open-source* e possui uma grande e ativa comunidade.

4.2 Aplicação de Triagem

4.2.1 Estrutura de Dados

Como referido na Secção 3.5, é necessário armazenar os alarmes recebidos pelo sistema. Estes são armazenados no estado global da *blockchain* em formato JSON. Os alarmes recebidos no sistema, juntamente com os dados contidos nos ficheiros, contêm muitos campos, sendo que nem todos necessitam de ser guardados na *blockchain*. Desta forma, são apenas registados os campos necessários para permitir fazer a triagem do acesso e os campos que os técnicos da empresa devem consultar mais tarde, sendo descartados os restantes. No seguinte exemplo 4.1, temos uma representação em JSON da estrutura armazenada de um alarme, com o nome de cada campo e o seu tipo.

```
1 access = {
2       docType: "access",
```

9 https://www.electronjs.org/pt/

```
    id: number,
    ftth: string,
    roe: string,
    pop: string,
    elemento: string,
    position: string,
    operadora: string,
    incidente: string,
    dia_inicio: string,
    hora_inicio: string,
    timestamp_inicio: string,
    dia_fim: string,
    hora_fim: string,
    responsavel: string,
    inativo: string,
  }
```

Listagem 4.1: Estrutura de um alarme

4.2.2 Smart Contract

É através do *smart contract* que vai ser possível interagir com a base de dados da *blockchain*. É nele que vão ser definidas as transações que vão permitir registar, alterar e consultar um alarme. No Fabric existem transações de configuração, correspondem a transações de configuração da rede, estas são armazenadas em Blocos de Configuração, que apenas armazenam este tipo de transações. Para além destas, há transações que permitem alterar o estado global, é possível fazer 3 tipos de operações ao mesmo: **GET**, **PUT**, **DELETE**. As transações *GET* vão buscar informação ao estado global, as *PUT* inserem informação no estado global e as *DELETE* eliminam informação do estado global.

Um dos motivos para a criação deste projeto é a aumentar os níveis de confiança nos dados fornecidos pela dstelecom. Por esse motivo, não foram implementadas transações do tipo *DELETE*. Para além disso, apenas alguns campos são passíveis de atualização. De seguida, vão ser apresentados pedaços de código para demonstrar a implementação dessas transações.

Transações PUT

Este tipo de transações são responsáveis por introduzir dados no estado global. Como no Hyperledger Fabric não existem transações que atualizem diretamente os dados do estado global, temos de utilizar

as transações do tipo PUT para atualizar um alarme, ou seja, utiliza-se uma cópia do alarme existente altera-se o atributo desejado e depois reescreve-se o alarme existente com a cópia.

Na Listagem 4.2, é especificada a transação de criação de um alarme na *blockchain*, ou seja, é através desta transação que é inserido no estado global um alarme. Esta transação recebe os seguintes argumentos:

- **ctx** - corresponde a uma instância do chaincode.
- **ftth** - corresponde ao código identificador do dispositivo (**FTTH**).
- **roe** - corresponde ao código do **ROE** ao qual o dispositivo pertence.
- **pop** - corresponde ao código do **POP** ao qual o dispositivo pertence.
- **elemento** - corresponde ao código do elemento ao qual o dispositivo pertence.
- **operadora** - corresponde ao código da operadora ao qual o dispositivo pertence.
- **dia_inicio** - corresponde ao dia em que o alarme disparou.
- **hora_inicio** - corresponde à hora em que o alarme disparou.
- **timestamp** - corresponde ao timestamp do disparo do alarme.
- **position** - corresponde ao código da posição do dispositivo do alarme.

Primeiramente, é criada uma variável local com os respetivos argumentos recebidos e com os restantes campos inicializados a nulo. De seguida, essa variável é transformada numa *string* JSON e é introduzida no estado global juntamente com o seu id.

```
1  async CreateAccess(ctx, ftth, roe, pop, elemento, operadora, dia_inicio,
     hora_inicio, timestamp, position) {
2
3     // ==== Create access object and marshal to JSON ====
4     let access = {
5       docType: "access",
6       id: this.identifier,
7       ftth: ftth,
8       roe: roe,
9       pop: pop,
10      elemento: elemento,
```

```
      position: position,
      operadora: operadora,
      incidente: null,
      dia_inicio: dia_inicio,
      hora_inicio: hora_inicio,
      timestamp_inicio: timestamp,
      dia_fim: null,
      hora_fim: null,
      responsavel: null,
      inativo: true,
    };

    // === Save asset to state ===
    await ctx.stub.putState(this.identifier.toString(), Buffer.from(JSON.
   stringify(access)));
    this.identifier++;
  }
```

Listagem 4.2: Transação "CreateAccess"

A transação mostrada na Listagem 4.3 é responsável por atualizar o registo de um alarme, atribuindo-lhe um dia e hora de fim. Estes dados são recebidos como argumento, bem como o id do alarme para ser possível ir buscá-lo ao estado global. Apesar desta transação permitir alterar a data de fim de um acesso, para não permitir adulterar os tempos dos alarmes é verificado se esses campos estão preenchidos. Ou seja, a transação verifica se o registo no estado global possui os valores do dia de fim e da hora de fim a nulo, se esses dois atributos não estiverem a nulo, é lançado um erro e o alarme não é atualizado. Desta forma, é impossibilitada a alteração de alarmes que já tenham data de fim, o que não permite que esses dados sejam trocados por outros mais favoráveis. No caso de não existir este mecanismo, essa troca estaria, em última análise, espelhada no ledger, permitindo assim a identificação de tais situações.

```
async SetEndTime(ctx, id, dia, hora) {
    let accessAsBytes = await ctx.stub.getState(id);
    if (!accessAsBytes || !accessAsBytes.toString()) {
      throw new Error(`Asset ${id} does not exist`);
    }
    let accessUpdate = {};
    try {
      accessUpdate = JSON.parse(accessAsBytes.toString()); //unmarshal
    } catch (err) {
```

```
      let jsonResp = {};
      jsonResp.error = "Failed to decode JSON of: " + id;
      throw new Error(jsonResp);
    }
    if (accessUpdate.fim != null) {
      throw new Error(`Access ${id} has already ended`);
    }
    accessUpdate.dia_fim = dia;
    accessUpdate.hora_fim = hora; //set the end time
    accessUpdate.inativo = false; //indicate that the access is active

    let assetJSONasBytes = Buffer.from(JSON.stringify(accessUpdate));
    await ctx.stub.putState(id, assetJSONasBytes); //rewrite the access
  }
```

Listagem 4.3: Transação "SetEndTime"

Na Listagem 4.4, está representada a transação que introduz um número de incidente num alarme. Esta recebe como argumentos o id do alarme e o número de incidente que lhe queremos atribuir. À semelhança da transação anterior, na transação "SetIncidNumber"é verificado se o alarme tem algum número de incidente atribuído, pois se tiver não é permitido alterar novamente esse atributo do alarme. Se as condições forem todas respeitadas então é atualizado o acesso com o respetivo número de incidente.

```
async SetIncidNumber(ctx, id, number) {
    let accessAsBytes = await ctx.stub.getState(id);
    if (!accessAsBytes || !accessAsBytes.toString()) {
      throw new Error(`Access ${id} does not exist`);
    }
    let accessUpdate = {};
    try {
      accessUpdate = JSON.parse(accessAsBytes.toString()); //unmarshal
    } catch (err) {
      let jsonResp = {};
      jsonResp.error = "Failed to decode JSON of: " + id;
      throw new Error(jsonResp);
    }
    console.log(accessUpdate);
    if (accessUpdate.incidente != null) {
      throw new Error(`Alarm ${id} already has an incident number`);
    }
```

```
    accessUpdate.incidente = number; //set the incident number of the
    access

    let assetJSONasBytes = Buffer.from(JSON.stringify(accessUpdate));
    await ctx.stub.putState(id, assetJSONasBytes); //rewrite the asset
  }
```

Listagem 4.4: Transação "SetIncidNumber"

Por último, a transação mostrada na Listagem 4.5 é a transação responsável por atualizar o atributo "responsavel"de um acesso. Esta transação, assim como as duas anteriores, verifica se o atributo a atualizar se encontra a nulo, caso não esteja dá erro, e recebe como argumentos o id do acesso a atualizar e o responsável, que corresponde a uma string com o nome do funcionário ou da equipa responsável pelo alarme.

```
async SetResponsible(ctx, id, responsible) {
    let accessAsBytes = await ctx.stub.getState(id);
    if (!accessAsBytes || !accessAsBytes.toString()) {
      throw new Error(`Access ${id} does not exist`);
    }
    let accessUpdate = {};
    try {
      accessUpdate = JSON.parse(accessAsBytes.toString()); //unmarshal
    } catch (err) {
      let jsonResp = {};
      jsonResp.error = "Failed to decode JSON of: " + id;
      throw new Error(jsonResp);
    }
    // Perguntar ao mário se isto faz sentido
    if (accessUpdate.responsavel != null) {
      throw new Error(`Alarm ${id} already has a responsible`);
    }
    accessUpdate.responsavel = responsible; //set the person responsible
    for the access

    let assetJSONasBytes = Buffer.from(JSON.stringify(accessUpdate));
    await ctx.stub.putState(id, assetJSONasBytes); //rewrite the asset
  }
```

Listagem 4.5: Transação "SetResponsible"

Transações GET

É com este tipo de transações que vamos consultar informação sobre o estado global da rede. Nesse sentido, todas as *queries* identificadas através dos requisitos da secção 3.2.2 traduzem-se numa transação GET. Na Listagem 4.6, temos a transação "QueryAccessByRoe". Esta é responsável por fazer uma *query* ao estado global pelo **ROE** recebido. Ou seja, a transação recebe como argumentos a instância do *chaincode* e o **ROE** que quer consultar, de seguida é definido uma variável que vai correponder a uma *JSON string* com a *query*.

Como podemos ver nas linhas 4 e 5, são definidos os campos pelos quais queremos procurar no estado global, e na linha 6 é definido o índice a utilizar para fazer a *query*. Como resultado final, esta *query* devolve todos o alarmes armazenados que têm o mesmo **ROE** que o recebido como argumento.

```
1 async QueryAccessByRoe(ctx, roe) {
2     let queryString = {};
3     queryString.selector = {};
4     queryString.selector.docType = "access";
5     queryString.selector.roe = roe;
6     queryString.use_index = ["_design/indexRoeDoc", "indexRoe"];
7     return await this.GetQueryResultForQueryString(ctx, JSON.stringify(
    queryString));
8   }
```

Listagem 4.6: Transação "QueryAccessByRoe"

Todas as outras *queries* têm a sua estrutura igual à apresentada, sendo alterados os campos e o índice mediante a *query* que quer fazer. As restantes *queries* estão representadas nos apêndices A.

4.2.3 Algoritmo de Triagem

Nesta secção é explicado o *core* da aplicação, basicamente o algoritmo de triagem, corresponde à codificação do diagrama apresentado na Secção 3.5. Para facilitar a perceção, vamos dividir este algoritmo em passos.

Leitura dos ficheiros

O primeiro passo consiste em ler os vários ficheiros e armazenar localmente a informação necessária para o funcionamento do sistema. Na Listagem 4.8, é possível a criação das seguintes variáveis:

- **alarmesMap** - armazena todos os alarmes que estão inativos em tempo real.

- **alarmes** - armazena todos os alarmes de inatividade e atividade da infraestrutura de rede da dstelecom.
- **trabalhos** - armazena todos os trabalhos que estão programados na infraestrutura de rede da dstelecom.
- **locken** - armazena todos os funcionários da dstelecom que acederam a um **ROE**.
- **ordens** - é uma variável de auxílio, que armazena todas as ordens de trabalho que estão no sistema da Live Solutions.
- **equipas** - armazena todas as equipas presentes na variável "ordens"que estão destacadas para realizar trabalhos.
- **allFtths** - armazena todo o cadastro da infraestrutura de rede da dstelecom, a *key* é o **FTTH** e o *value* é a restante informação associada ao mesmo.

Após a inicialização das variáveis, é transferida a informação dos ficheiros para essas mesmas variáveis, como temos ficheiros de diferentes formatos foi necessário utilizar funções diferentes para extrair a informação.

A função "parseTxt"é responsável por ler o ficheiro de texto que corresponde à captura de alarmes recebido através do **NMS**. Adicionalmente esta função também faz uma filtragem armazenando na variável "alarmes"apenas os registo que correspondem a alarmes de inatividade e atividade. Para isso é utilizada uma expressão regular que captura todos os alarmes que tiverem a descrição "[GPON Alarm]ONU LOS(Loss of Signal)(35113)". Em adição é descartada ainda alguma informação desnecessária, uma vez que um alarme típico tem o seguinte formato 4.7:

```
1 ALARMID=1674122265354  ALARMDESC=[GPON Alarm]ONU LOS(Loss of Signal)(35113)
     DID=10.160.1.5  DIP=10.160.1.5  DNAME=N05A01\_01\_01  DTYPE=C300v1.3
   POSITION=10.160.1.5\_1\_1\_17\_7\_3  SEVERITY=Minor  HAPPENTIME
   =2023-06-14 09:00:40.000  ALARMTYPE=QoSAlarm  DESC=ONU ID=3,ONU Name=NPR
   -FTTH\_DST\_00609429,ONU Type=SAGEM-F5657,Authentication Value=
   znxeWZx006,ONU Registration Info: Password = znxeWZx006,Service Level=
   RESIDENCIAL  ENTITY\_ID=IP Address:10.160.1.5|1158624  EVENT\_CODE=35113
     PROBABLE\_CAUSE\_DESC=1.Fiber link failure. 2.ONU failure.  PROBABEL\
   _CAUSE\_CODE=351130000  SYSTEM\_TYPE=768  ACKSTATUS=Unacknowledged
```

Listagem 4.7: Formato de um Alarme

De todos estes dados apenas são guardados os campos: "POSITION","SEVERITY", "HAPPENTIME"e "Name".

A função "parseCSV"é semelhante à "parseTtxt", mas esta lê ficheiros do tipo .csv, mediante o ficheiro lido depois é apenas guardada a informação que nos é útil. No caso do "rota.csv", que é o documento com o cadastro de toda a rede, por cada linha lida são armazenados os seguintes campos: "FTTH", "ROE", "POP"e "DROP_ID", que corresponde ao código do elemento ao qual o dispositivo pertence. Estes dados do cadastro são armazenados numa variável do tipo Map, "allFtths". O valor do campo "FTTH"é utilizado como *key* e os restantes campos como *value*. No caso do "prog.csv"são armazenados, na variável "trabalhos", o dia, a hora e o local de cada linha do ficheiro, neste estão guardados os trabalhos programados. No "Locken.csv"estão presentes os dados dos funcionários que utilizaram o Locken. Dessa informação são armazenados, na variável "locken", o nome, o **ROE**, o dia e a hora de cada linha do ficheiro.

No caso do ficheiro "LS.xlsx"são extraídas os números de ordem, o dia e o **ROE** das equipas destacadas para resolver incidentes, estes dados são armazenados na variável "equipas".

```
let alarmesMap = new Map();
let alarmes = [];
let trabalhos = [];
let locken = [];
let ordens = [];
let equipas = [];
let allFtths = new Map();
...
await parseTxt("./src/registo2.txt", alarmes);
await parseCSV("./src/rota.csv")
...
await parseCSV("./src/prog.csv")
...
await parseCSV("./src/Locken.csv")
...
workbook = XLSX.readFile("./src/LS.xlsx");
worksheet = workbook.Sheets[workbook.SheetNames[0]];
ordens = XLSX.utils.sheet_to_json(worksheet);
...
```

Listagem 4.8: Leitura de todos os ficheiros

Identificar o tipo de alarme

Após a leitura dos ficheiros é executado um ciclo "for"para fazer a triagem de cada alarme presente no array "alarmes". A partir deste passo cada alarme repete as etapas seguintes. O segundo passo do programa passa por identificar o tipo do alarme que está a ser tratado.

```
1      ...
2      if (alarme.flag === "Minor" || alarme.flag === "Major" || alarme.flag
    === "Critical") {
3      ...
4      } else if (alarme.flag === "Restore") {
5      ...
6      }
7      ...
```

Listagem 4.9: Identificação do tipo de alarme

Como está representado em 4.9, o tipo do alarme é identifcado pela *flag* do mesmo, se for "Minor", "Major"ou "Critical", então é um alarme de inatividade, se a *flag* for "Restore"é de atividade. Caso o alarme não tenha nenhuma destas *flags* é descartado.

Alarme de inativade

Caso seja identificado que é um alarme de inatividade acede-se à variável "allFtths"para verificar se o **FTTH** do alarme a ser tratado existe no cadastro, se for encontrada uma correspondência, guarda-se os valores do **ROE**, **POP** e do elemento numa variável local de nome "acesso". Esta variável vai ser construída ao longo do processo de triagem para mais tarde ser introduzida no estado global. Posteriormente, verifica-se se o **ROE** do acesso está presente no array "trabalhos", se estiver considera-se que é um trabalho programado e descarta-se, se não estiver o acesso vai prosseguir com a triagem, representado em 4.10.

```
1          let ftthValue = allFtths.get(alarme.ftth);
2          console.log(ftthValue);
3          if (ftthValue.roe) {
4            acesso.roe = ftthValue.roe;
5            acesso.pop = ftthValue.pop;
6            acesso.elemento = ftthValue.elemento;
7          }
8
9          //VERIFICAR SE O ALARME PERTENCE A UM TRABALHO PROGRAMADO
10         for (const trabalho of trabalhos) {
```

```
11            if (trabalho.roe === acesso.roe && trabalho.dia === acesso.
    dia_inicio) {
12              isWork = true;
13              break;
14            }
15          }
```

Listagem 4.10: Verificação dos trabalhos programados

Quando o alarme prossegue a triagem, o sistema já sabe que se trata de um alarme de inatividade e que não está programado, então é necessário registá-lo no estado global da rede *blockchain*. Antes de fazer esse registo são atribuidos dois *timeouts* ao alarme, um de 20 minutos e outro de 2 horas. Estes *timeouts* servem para auxiliar a identificação de um incidente. Na Listagem 4.11 está codificada a atribuição dos *timeouts* e o registo do alarme através da transação "CreateAccess". É possível observar também, que da linha 7 à 14, é introduzido um objeto na variável "alarmesMap". Esta variável armazena todos os alarmes que estão inativos em tempo real e serve auxílio para a lógica dos *timeouts*.

```
1           //DEFINIR TIMEOUT DE 20 MINUTOS
2             let timer20 = setTimeout(await trigger20, 1200000, acessoTimer,
     contract);
3
4             //DEFINIR TIMEOUT DE 2 HORAS
5             let timer2 = setTimeout(trigger2, 7200000, acessoTimer,
    contract);
6
7             alarmesMap.set(acesso.ftth, {
8               id: acesso.id,
9               ftth: acesso.ftth,
10              position: acesso.position,
11              roe: acesso.roe,
12              timer20min: timer20,
13              timer2h: timer2,
14            });
15            await contract.submitTransaction(
16              "CreateAccess",
17              acesso.ftth,
18              acesso.roe,
19              acesso.pop,
20              acesso.elemento,
21              acesso.operadora,
```

```
            acesso.dia_inicio,
            acesso.hora_inicio,
            acesso.timestamp,
            acesso.position
          );
```

Listagem 4.11: Registo do alarme

Após o registo, o sistema vai verificar se o alarme pertence ou se deu origem a um incidente. É considerado um incidente quando dois alarmes pertencentes ao mesmo **ROE** estão inativos num período de tempo superior ou igual a 20 minutos e inferior a 2 horas. Há duas formas diferentes de identificar um incidente, através dos *timeouts*, que irá ser explicado posteriormente, ou através da hora de início presente nos alarmes. Para identificar que existe um incidente é necessário ter pelo menos dois acessos inativos durante o período de tempo já mencionado, ou seja, através do *timestamp* presente no alarme, que está a ser tratado, é possível calcular a diferença entre esse e os alarmes já registados. Como se pode observar em 4.12, o sistema começa por fazer uma *query* ao estado global, o resultado dessa *query* é o conjunto de alarmes registados com o mesmo **ROE** e o mesmo dia do alarme que está a ser tratado.

De seguida, para cada alarme do conjunto é calculada a diferença entre o seu *timestamp* e o do alarme em processamento. Se a diferença estiver dentro do intervalo de tempo mencionado, então é incrementada a variável local "down". Esta variável contabiliza o número de dispositivos que estão inativos e têm o mesmo **ROE** e o mesmo dia do acesso em processamento. Assim que o valor de "down"for maior ou igual que 2 significa que temos pelo menos dois dispositivos inativos há pelo menos 20 minutos, o que indica que estamos perante um incidente. Uma vez identificado que se trata de um incidente, é necessário verificar se este incidente já tem um número identificador atribuído ou não. Se o alarme inativo presente no estado global tiver um número de incidente atribuído então é atribuído o mesmo número de incidente ao alarme que está a ser tratado, se não é atribuído um novo número de incidente ao alarme que está a ser tratado e aos alarmes identificados que estão inativos.

```
        let alarmesMesmoROE = await contract.evaluateTransaction("
QueryAccessByRoeDate", acesso.roe, acesso.dia_inicio);
        alarmesMesmoROE = JSON.parse(alarmesMesmoROE.toString());
        if (alarmesMesmoROE.length > 1) {
            let down = 0;
            for (const record of alarmesMesmoROE) {
              let timestampRecord = record.Record.timestamp_inicio;
              let conta = Number(timestampRecord) - Number(acesso.
timestamp);
```

```
8                 //Ver se tem menos de duas horas e mais de 20 minutos.
9                 if (conta <= -1200000 && conta >= -7200000 && record.Record
  .inativo) {
10                  down = down + 1;
11                  if (down >= 2) {
12                    if (record.Record.incidente) {
13                      acesso.incidente = record.Record.incidente;
14                      await contract.submitTransaction("SetIncidNumber",
  acesso.id, acesso.incidente);
15                      ...
16                      break;
17                    } else {
18                      acesso.incidente = numeroIncidente;
19                      await contract.submitTransaction("SetIncidNumber",
  record.Record.id, acesso.incidente);
20                      await contract.submitTransaction("SetIncidNumber",
  acesso.id, acesso.incidente);
21                      numeroIncidente++;
22                      ...
23                      break;
24                    }
25                  }
26                }
27              }
28            }
```

Listagem 4.12: Verificação de incidente

Depois de se verificar que se trata de um incidente ou não, a aplicação verifica se existe algum responsável pelo alarme. Na Listagem 4.13 está representado esse processo. Primeiro verifica-se se algum funcionário presente no array "locken"tem o mesmo **ROE** e o mesmo dia do alarme. Encontrada uma correspondência é atualizado o registado no estado global com responsável pelo alarme, através da transação "SetResponsible", que neste caso é registado o nome do funcionário. Caso não haja nenhuma correspondência no array "locken", o sistema repete o processo para o array "equipas", se encontrar uma correspondência significa que existe uma equipa responsável pelo alarme e é atualizado o seu registo no estado global através da mesma transação.

```
1           //VERIFICAR SE EXISTE ALGUÉM NA LOCKEN RESPONSÁVEL PELO ALARME
2           if (!hasResponsible) {
```

```
            for (const func of locken) {
              if (func.roe === acesso.roe && func.dia === acesso.
dia_inicio) {
                hasResponsible = true;
                acesso.responsavel = func.Nome;
                await contract.submitTransaction("SetResponsible", acesso
.id, acesso.responsavel);
                break;
              }
            }
          }

          //VERIFICAR SE EXISTE ALGUÉM NA LS RESPONSÁVEL PELO ALARME
          if (!hasResponsible) {
            for (const equipa of equipas) {
              if (equipa.roe === acesso.roe && equipa.dia === acesso.
dia_inicio) {
                let responsavel = "Equipa LS, Ordem Nº: " + equipa.numero
;
                hasResponsible = true;
                acesso.responsavel = responsavel;
                await contract.submitTransaction("SetResponsible", acesso
.id, acesso.responsavel);
                break;
              }
            }
          }
```

Listagem 4.13: Verificação de responsável

No fim de todas estas etapas acaba a triagem do alarme é repetido o processo para o próximo alarme presente no array "alarmes".

Alarme de ativade

No caso de ser um alarme de atividade, ou seja, um alarme que indica que o dispositivo está novamente ativo, à partida vai existir um registo de inatividade do mesmo dispositivo. Deste modo, o sistema vai consultar o estado global para encontrar o registo de inatividade do dispositivo e em caso de sucesso atualiza o registo e coloca-lhe um dia e hora de fim através da transação "SetEndTime". Para além disso,

é necessário também fazer *clear* dos *timeouts* desse dispositivo e eliminá-lo da variável "alarmesMap", uma vez que ele já não se encontra inativo. Este procedimento está representado na Listagem 4.14

```
        acesso.dia_fim = alarme.dia;
        acesso.hora_fim = alarme.hora;
        let alarmesMesmoFTTH = null;
        if (alarme.ftth) {
          alarmesMesmoFTTH = await contract.evaluateTransaction("
QueryAccessByFtth", alarme.ftth);
          alarmesMesmoFTTH = JSON.parse(alarmesMesmoFTTH.toString());
        } else {
          alarmesMesmoFTTH = await contract.evaluateTransaction("
QueryAccessByPosition", alarme.position);
          alarmesMesmoFTTH = JSON.parse(alarmesMesmoFTTH.toString());
        }
        if (alarmesMesmoFTTH.length >= 1) {
          let ftthAlarme = alarmesMesmoFTTH[0].Record.ftth;
          if (alarmesMap.has(ftthAlarme)) {
            let alar = alarmesMap.get(ftthAlarme);
            if (roeMap.has(alar.roe)) {
              let value = roeMap.get(alar.roe);
              if (value.hasCliente(alar.id)) {
                value.delCliente(alar.id);
                if (value.counter < 1) {
                  roeMap.delete(alar.roe);
                } else {
                  roeMap.set(acessoTimer.roe, value);
                }
              }
            }
          }
          try {
            clearTimeout(alarmesMap.get(ftthAlarme).timer20min);
            clearTimeout(alarmesMap.get(ftthAlarme).timer2h);
          } catch (error) {
            console.log(error);
          }
          alarmesMap.delete(alarme.ftth);
          for (const alarm of alarmesMesmoFTTH) {
```

```
35              if (alarm.Record.dia_fim == null) {
36                await contract.submitTransaction("SetEndTime", alarm.Record
    .id, acesso.dia_fim, acesso.hora_fim);
37                break;
38              }
39            }
40          }
```

Listagem 4.14: Tratamento de um alarme de atividade

Lógica dos timeouts

No que diz respeito aos *timeouts*, estes têm um grande impacto na aplicação, pois no caso de ocorrerem incidentes em que vários dispositivos ficam inativos no mesmo instante, só é possível identificar que está a ocorrer um incidente através dos *timeouts*.

Para auxiliar o funcionamento do programa foram criadas duas classes: a Cliente e a TimeoutMap. A classe Cliente é composta por 5 atributos, o id, o dia, a hora, a operadora e o timestamp, todos estes atributos são correspondentes ao alarme, ou seja, são iguais ao que são registados no estado global. A classe TimeoutMap tem 3 atributos: o "counter"que representa o número de clientes em baixo, o "nIncid"que representa o número de incidente, o "clientes"que corresponde a um Map da classe Cliente e que representa o conjunto dos clientes que estão em baixo. Resumidamente, um TimeoutMap armazena um conjunto de clientes que se encontram inativos, registando o número de clientes inativos e o número de incidente a que pertencem.

Estas duas classes desempenham um papel fundamental, uma vez que no programa é criada a variável "roeMap", que corresponde a um mapa (*Map*) da classe TimeoutMap. Este mapa, cuja chave (*Key*) corresponde ao **ROE**, armazena os alarmes cujos timeouts mais curtos dispararam. Por outras palavras, a variável "roeMap"é usada para armazenar os alarmes que estão inativos há mais de 20 minutos e que pertencem ao mesmo **ROE**. Este agrupamento é essencial, pois um incidente só é considerado válido se houver pelo menos dois clientes inativos por pelo menos 20 minutos e se eles compartilharem o mesmo **ROE**.

Para cada alarme de inatividade são lançados dois *timeouts* um de 20 minutos e outro de 2 horas. Cada um dos *timeouts* possui uma função diferente para o cenário em que dispara. Quando dispara um timeout mais curto, é chamada a função "trigger20", presente na Listagem 4.15. Esta função é responsável por verificar na "roeMap"se existe alguma *key* com o mesmo **ROE** do alarme associado ao *timeout* disparado, se existir então ao *value* correspondente vai ser adicionado o alarme, depois de ser

feita essa atualização verifica-se se o "counter"é maior ou igual a 2. No caso de se verificar essa condição, estamos perante um incidente, o próximo passo é verificar se já existe um número de incidente associado e atualizar os registos dos alarmes.

```
    let newClient = new Cliente(
    acessoTimer.id,
    acessoTimer.dia_inicio,
    acessoTimer.hora_inicio,
    acessoTimer.operadora,
    acessoTimer.timestamp
  );
  if (roeMap.has(acessoTimer.roe)) {
    let value = roeMap.get(acessoTimer.roe);
    value.addCliente(newClient.id, newClient);
    if (value.counter >= 2) {
      if (value.nIncid) {
        //Atualiza a tabela com o novo cliente
        sendOldIncident(value.nIncid, newClient.operadora);
        //Define o número de incidente do alarme na blockchain
        await contract.submitTransaction("SetIncidNumber", newClient.id,
  value.nIncid);
      } else {
        value.setnIncid(numeroIncidente);
        let first = value.firstClient();
        let operadoras = new Array();
        operadoras.push(first.operadora);
        await contract.submitTransaction("SetIncidNumber", first.id,
  numeroIncidente);
        for (const cliente of value.getClientes().values()) {
          if (cliente.id != first.id) {
            operadoras.push(cliente.operadora);
            await contract.submitTransaction("SetIncidNumber", cliente.id,
  numeroIncidente);
          }
        }
        sendTimeoutIncident(numeroIncidente, acessoTimer.roe, first.dia,
  first.hora, operadoras);
        numeroIncidente = numeroIncidente + 1;
        roeMap.set(acessoTimer.roe, value);
```

```
      }
    }
  } else {
    let newEntry = new TimeoutMap();
    newEntry.addCliente(newClient.id, newClient);
    roeMap.set(acessoTimer.roe, newEntry);
  }
```

Listagem 4.15: Função "trigger20"

A função "trigger2"(4.16) é executada sempre que um *timeout* de 2 horas dispara. Esta função retira do "roeMap"o alarme associado ao *timeout* disparado. Se este corresponder ao último elemento do *value* em que está inserido, então esse *value* também é retirado do "roeMap". Esta ação só ocorre se o alarme não estiver inserido num incidente, pois se um dispositivo da rede está inativo há mais de 2 horas e não pertence a um incidente assume-se que esse dispositivo pode ser um possível desmantelamento (deixa de ser cliente da operadora e indiretamente da dstelecom) ou um problema com a rede interna do cliente. Se pertencer a um incidente então não podemos retirar do "roeMap"até estar ativo novamente.

```
  if (roeMap.has(acessoTimer.roe)) {
  let value = roeMap.get(acessoTimer.roe);
  if (!value.nIncid) {
    value.delCliente(acessoTimer.id);
    if (value.counter < 1) {
      roeMap.delete(acessoTimer.roe);
    } else {
      roeMap.set(acessoTimer.roe, value);
    }
  }
}
```

Listagem 4.16: Função "trigger2"

4.2.4 Interface Gráfica

As aplicações Electron[10] são compostas por dois processos principais: o processo principal e o processo de renderização.

O processo principal funciona como o esqueleto da aplicação Electron. Ele é responsável por interações a nível do sistema, gerar e manter janelas e controlar o tempo de vida do programa. É através do

[10] https://www.electronjs.org/pt/docs/latest/tutorial/process-model

processo principal que: é criada a janela da aplicação, gerido o seu aspeto e comportamento; são tratados os eventos do sistema, tais como, interações com menus, operações com ficheiros e comunicação com o sistema operativo; são geridas as diferentes janelas e a comunicação entre elas; é controlado o comportamento da aplicação, incluindo o tratamento dos eventos do ciclo de vida da aplicação, como por exemplo, fechar a janela, etc.

O processo de renderização é responsável por exibir a interface gráfica do utilizador (GUI) da aplicação Electron. Cada janela Electron executa o seu próprio processo de renderização, que é como uma instância separada de uma página da Web. O processo de renderização é isolado do processo principal e de outros processos de renderização para fins de segurança e estabilidade. No processo de renderização estão incluídas tarefas como: renderizar e apresentar o HTML, CSS e JavaScript que constituem a interface de utilizador da aplicação; tratar de interações do utilizador, como por exemplo, clicar em botões, preencher formulários, reagir a eventos.

Os dois processos comunicam entre si através do **Inter-Process Comunication**[11] (**IPC**). O **IPC** é a única forma de realizar muitas operações comuns, tais como aceder a uma API nativa a partir da sua interface de utilizador ou causar alterações nos seus conteúdos web a partir de menus nativos, porque os processos principal e de renderização têm tarefas separadas no paradigma de processos do Electron.

Em conclusão, cada processo de renderização mantém a interface do utilizador e coopera com o processo principal quando necessário, enquanto este controla todo o programa e comunica com o sistema operativo. Esta divisão de tarefas contribui para a estabilidade e segurança das aplicações Electron.

Processo Principal

Como referido anteriormente, o processo principal é o esqueleto da aplicação, no caso, este corresponde ao ficheiro "main.js"e é nele que está presente o algoritmo de triagem foi explicado na secção 4.2. Na Aplicação de Triagem o utilizador não tem nenhuma interação com a mesma, este apenas visualiza os incidentes a aparecerem no ecrã caso estes existam. Deste modo, apesar de o Electron permitir que os processos troquem mensagens entre eles, a aplicação apenas envia mensagens do processo principal para o de renderização, ou seja, sempre que identificar um incidente o processo principal envia uma mensagem ao processo de renderização e este atualiza a interface gráfica.

Para ser possível essa comunicação foram criadas 3 funções, presentes no 4.17, responsáveis por enviar mensagens ao processo de renderização: "sendOldIncident", "sendNewIncident"e "sendTimeoutIncident".

[11] https://www.electronjs.org/pt/docs/latest/tutorial/ipc

- **"sendNewIncident"** - Esta função é utilizada quando é identificado um novo incidente, envia para o processo de renderização o número de incidente, o **ROE** onde ocorre, o dia, a hora e a operadora do primeiro cliente a ficar indisponível.
- **"sendOldIncident"** - Esta função é utilizada quando é identificada a existência de um incidente, envia o número de clientes "em baixo"e a operadora dos mesmos.
- **"sendTimeoutIncident"** - Esta função é semelhante à "sendNewIncident", mas em vez de enviar apenas a operadora do primeiro cliente, envia a lista de operadoras dos clientes todos que estão indisponíveis.

```
function sendOldIncident(number, operadora) {
  mainWindow.webContents.send("old-incident", {
    number: number,
    operadora: operadora,
  });
}

function sendNewIncident(number, roe, data, hora, operadora) {
  mainWindow.webContents.send("new-incident", {
    number: number,
    roe: roe,
    data: data,
    hora: hora,
    operadora: operadora,
  });
}

function sendTimeoutIncident(number, roe, data, hora, operadoras) {
  mainWindow.webContents.send("timeout", {
    number: number,
    roe: roe,
    data: data,
    hora: hora,
    operadoras: operadoras,
  });
}
```

Listagem 4.17: Envio de mensagens para o processo de renderização

Processo de Renderização

Do lado do processo de renderização temos o ficheiro "renderer.js", neste ficheiro estão implementadas 3 funções, uma para cada tipo de mensagem recebida, responsáveis por alterar o ficheiro "index2.html". Resumidamente, aquilo que o utilizador vê é uma tabela com os incidentes identificados pelo sistema, a tabela inicialmente começa vazia e conforme os incidentes vão sendo identificados o processo principal envia mensagens ao processo de renderização e este a atualiza a tabela com os dados recebidos. Mediante a mensagem recebida do processo principal, é utilizada a função adequada que utiliza os dados presentes na mensagem para atualizar a tabela da interface gráfica.

4.3 Aplicação de Consulta

No que diz respeito à Aplicação de Consulta, foi utilizada uma aplicação web como base, pois a dstelecom tinha concorrentemente em desenvolvimento um projeto capaz de certificar documentos para a empresa em formato de Non-Fungible Tokens. Como a Aplicação de Consulta acaba por ser uma readaptação de outra aplicação, não foram tomadas quaisquer decisões em termos tecnológicos e arquiteturais, pelo que foi mantida a arquitetura original da aplicação e as tecnologias que a mesma utiliza, com exceção da componente de armazenamento de NFT's que foi descarta. Por este motivo, nesta secção vamos apenas abordar as funcionalidades implementadas e explicar que dados são armazenados na base de dados.

4.3.1 Base de Dados

Na Aplicação de Consulta existem utilizadores que vão interagir com a mesma e estes traduzem em funcionários diferentes, logo é necessário armazenar os utilizadores e os seus dados numa base de dados.

Utilizador

No que diz respeito aos utilizadores, as informações que são guardadas são as seguintes:

- **username** - Nome do utilizador na aplicação.
- **email** - Email do utilizador.
- **password** - Password do utilizador.
- **roles** - Array com os roles do utilizador.

Para possibilitar os armazenamento destas informações, foi necessário definir no MongoDB a coleção "User"com a seguinte estrutura:

```
{
    id: ObjectId,
    username: String,
    email: String,
    password: String,
    roles: []
}
```

Listagem 4.18: Estrutura da coleção "User"

Role

O role representa a empresa de cada utilizador, como a aplicação de consulta foi idealizada para possibilitar a utilização da mesma às empresas parceiras da dstelecom, foi definida a seguinte coleção:

```
{
    id: ObjectId,
    name: String
}
```

Listagem 4.19: Estrutura da coleção "Role"

Neste caso apenas precisamos de guardar o nome da empresa, daí a simplicidade desta coleção. Cada utilizador tem no array "roles"uma ou mais referências para esta coleção que se traduz nos respetivos nomes das empresas às quais pertence.

4.3.2 Backend

O *backend* é a parte responsável por toda a lógica por trás da Aplicação de Consulta, então, é nele que são implementadas as funcionalidades propostas na Secção 3.2.2. O *backend* foi implementado como uma REST API, deste modo, é permitido fazer pedidos HTTP do tipo GET, PUT, DELETE e POST ao servidor. Estes métodos são implementados através de rotas, neste caso existem 3 rotas distintas: a de utilizador, com o prefixo "/user"; a de autenticação, com o prefixo "/api/auth"; a de *query*, com o prefixo "/query". Nesta secção apenas são abordadas as rotas desenvolvidas de raiz, que correspondem às de *query*.

Dentro das rotas de *query* existem 3 sub-rotas, uma rota por cada empresa que interage com a aplicação. Para este protótipo, apenas foram consideras 3 empresas, a dstelecom, a NOS e a Vodafone,

pois estas duas correspondem aos clientes com mais dispositivos monitorizados da dstelecom. Isto traduz-se em rotas diferentes para cada empresa, os utilizadores da dstelecom utilizam rotas com o prefixo **"/query"**, os utilizadores da NOS utilizam rotas com o prefixo **"/query/nos"** e os utilizadores da Vodafone utilizam rotas com o prefixo **"/query/vdf"**.

Neste caso, cada rota corresponde a um tipo de *query*, ou seja, se o utilizador quiser fazer uma *query* por **FTTH** utiliza a rota "/query/ftth"e se quiser fazer uma *query* por **ROE** utiliza a rota "/query/roe". Para a dstelecom foram implementadas todas as *queries* propostas, enquanto que para as outras duas empresas não foi implementada a *query* por responsável. Uma vez que, essas duas empresas são externas elas não podem ter acesso à informação relativa ao responsável pelo alarme.

Na Listagem 4.20, temos a implementação da *query* por **FTTH** para a Vodafone, é possível verificar na primeira que são chamadas duas funções "verifyToken"e "isVodafone"que são responsáveis por verificar se o utlizador atual tem um JWToken válido e se o mesmo está registado na aplicação como um funcionário da Vodafone. Após essa verificação, é necessário verificar se o utilizador existe na base de dados, de seguida é feita a conexão à *blockchain*, posteriormente é executada a transação "QueryAccessByFtth", que corresponde a fazer um *query* por **FTTH**, e é aplicado pelo menos um filtro a resposta recebida antes de a enviar para o utilizador final, esse filtro corresponde retirar da resposta os alarmes que não pertencem à empresa do utilizador. Para além desse, podem ser aplicados outros filtros, nomeadamente o filtro de incidente e o filtro de não incidente, esses filtros servem para restringir o tipo de alarmes que são enviados na resposta. Se for aplicado o filtro de incidente, neste exemplo apenas são enviados como resposta os alarmes que pertencem a incidentes e com o **FTTH** pedido pelo utilizador. Se for aplicado o filtro de não incidente, neste exemplo apenas são enviados como resposta os alarmes que não pertencem a incidentes e com o **FTTH** pedido pelo utilizador.

```
1  router.get("/ftth", [authJwt.verifyToken, authJwt.isVodafone], async (req,
       res) => {
2    try {
3      //DETERMINAR O USER LOGGADO
4      const user = await User.findOne({ _id: req.userId });
5
6      if (!user) {
7        return res.send("User Not Found!");
8      }
9
10     const username = user.username;
11
```

```
12      //CONEXÃO Á BLOCKCHAIN
13      let networkObj = await network.connectToNetwork(username);
14
15      let uri = await network.invoke(networkObj, true, "QueryAccessByFtth", [
        req.query.ftth]);
16
17      let finalResponse = JSON.parse(uri.toString());
18      finalResponse = finalResponse.filter((ob) => ob.Record.operadora === "
        VDF");
19      if (req.query.filter === "incident") {
20        finalResponse = finalResponse.filter((ob) => ob.Record.incidente !==
        null);
21      } else if (req.query.filter === "no_incident") {
22        finalResponse = finalResponse.filter((ob) => ob.Record.incidente ===
        null);
23      }
24      res.send(finalResponse);
25    } catch (err) {
26      res.send(err);
27    }
28  });
```

Listagem 4.20: Rota da query por **FTTH**

Todas as outras rotas são implementadas da mesma forma, exceto a do número de incidente, pois nessa não é permitida a aplicação de filtros e estão presentes no apêndice A.

Capítulo 5

Deployment e Testes

5.1 Deployment

Neste secção são abordados os passos tomados para fazer *deploy* da rede *blockchain*. Uma vez que manter a rede local é um pouco custoso pois ainda consome alguma memória RAM, neste caso 8GB, isto acaba por limitar o desempenho da máquina onde estava a ser desenvolvida a aplicação. Após o desenvolvimento dos protótipos das aplicações para uma rede local, iniciou-se o processo de investigação para fazer o *deploy* da mesma. Foram encontradas duas soluções: criar um servidor que alojasse a rede ou utilizar uma plataforma externa que fornecesse a rede *blockchain* como serviço.

Optou-se por utilizar uma plataforma que oferece a rede *blockchain* como um serviço. Através de uma primeira pesquisa foram encontradas a Amazon Managed Blockchain[1] e a IBM Blockchain Platform[2]. Uma vez que estas soluções têm custos associados, procurou-se outra solução, tendo-se encontrado a Kaleido[3]. À semelhança das outras duas esta tem uma plataforma de *Blockchain-As-A-Service*, mas esta plataforma oferece um período de experimentação gratuito. A Kaleido tem integração com Etheruem, Corda e Hyperledger Fabric. Deste modo, criou-se uma rede Fabric para testar se os protótipos construídos são compatíveis com a plataforma e se é possível que estes comuniquem com a mesma. O plano de demonstração gratuita da Kaleido apenas permite criar uma rede com um máximo de 1 organização, 2 canais e 2 *nodes* (1 *orderer* e 1 *peer*), esta condição apesar de ser limitadora permite testar o funcionamento da aplicação. Para criar a rede foi seguida a documentação oficial da plataforma[4].

Após a criação da rede na plataforma, foi apenas necessário alterar na aplicação o código responsável pela conexão com a rede. Para conectar com a rede criada na Kaleido, foram utilizadas as seguintes linhas de código presentes na Listagem 5.1. Para esta adaptação, foi utilizado como base uma *sample* presente

1 https://aws.amazon.com/pt/managed-blockchain/

2 https://www.ibm.com/products/blockchain-platform-hyperledger-fabric

3 https://www.kaleido.io/

4 https://docs.kaleido.io/using-kaleido/quick-start-fabric/first-blockchain/

no github da própria plataforma[5], desse projeto exemplo também são utilizados os ficheiros presentes na diretoria "/nodejs/lib".

```
const KaleidoClient = require("./lib/kaleido");
const { Gateway } = require("fabric-network");
const Client = require("fabric-common");

const chaincodeName = process.env.CCNAME || "trial_app2";
const userId = process.env.USER_ID || "user001";
const useDiscovery = process.env.USE_DISCOVERY || "false";
```

Listagem 5.1: Conexão à Rede Kaleido na Aplicação de Triagem

5.2 Testes

Nesta secção são apresentados os testes feitos às aplicações criadas durante a realização desta dissertação. Os testes apresentados foram feitos manualmente para verificar se as funcionalidades estavam implementadas corretamente.

5.2.1 Testes de Aplicação de Triagem

Nesta subsecção vão ser apresentados os testes da Aplicação de Triagem. Estes testes, para além das funcionalidades da própria aplicação, também acabam por testar o *chaincode* e as transações que este permite. Para testar a aplicação foi utilizado um ficheiro de texto com uma captura de alarmes que foi alterado de forma a criar os cenários que se queria testar.

Para a maioria dos testes foi utilizada a rede local, pois uma vez que se tem de estabelecer uma rede nova a cada teste, esta é a opção mais rápida para o efeito. A rede utilizada é a *default* dos exemplos do Hyperledger Fabric, que consiste numa rede com 3 membros (ou organizações), onde cada membro possui um nodo da rede, sendo que um dos membros possui um *orderer node* e os restantes possuem um *peer node*.

Identificar um Incidente

Começou-se por testar a principal funcionalidade da aplicação, a identificação de um incidente. Para tal, criou-se um ficheiro texto para cada um dos cenários estudados com a equipa da empresa. No primeiro

[5] https://github.com/kaleido-io/kaleido-fabric-samples/tree/master

cenário, foram colocados 4 alarmes do mesmo **ROE** com uma diferença de pelo menos 20 minutos entre cada um, como mostrado na Listagem A.25 (Apêndice A).

Deste modo, o resultado esperado seria um incidente com 4 clientes, 2 da Vodafone, 1 da Nowo e 1 da blu, sendo que o primeiro cliente a estar em baixo é o da Vodafone às 10:27:39 do dia 14-06-2023, pelo que também é esperado que essa informação esteja presente. O resultado obtido está representado na Figura 11, pelo que se confirma que está correto. Em adição, se pesquisarmos através da Aplicação de Consulta pelo incidente número 1, podemos confirmar que os alarmes correspondem aos que estavam no ficheiro de texto, Figura 23 do Apêndice B.

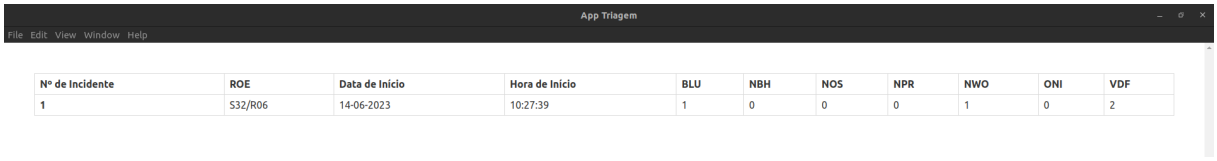


Figura 11: Resultado do Primeiro Cenário de Teste.

Como segundo cenário, alterou-se o ficheiro de texto anterior e acrescentou-se um alarme de "Restore"para o dispositivo do segundo alarme do ficheiro, visível na Listagem A.26 do Apêndice A. Desta forma, o resultado esperado é o mesmo do anterior, mas sem o cliente Nowo, o resultado obtido corresponde à Figura 12. Como podemos ver na Figura 24 do Apêndice B, o segundo alarme tem hora de fim, mas não tem número de incidente, enquanto que os restantes alarmes têm número de incidente e não têm hora de fim.

Figura 12: Resultado do Segundo Cenário de Teste.

O 3º cenário de teste foi criado para verificar se a aplicação consegue identificar os 2 tipos de incidentes. Estão presentes no ficheiro, A.27, os 4 alarmes do 1º cenário e a estes foram acrescentados 3 alarmes de dispositivos de um **ROE** diferente, 2 deles têm a mesma hora e o outro tem 5 minutos de atraso, ou seja, simula o cenário de dois dispositivos ficarem inativos ao mesmo tempo e o terceiro fica inativo 5 minutos depois. Para além desses, foram ainda acrescentados alarmes de "Restore"para fechar o 1º incidente. Deste modo, o resultado esperado é a identificação de dois incidentes, o 1º com 4 clientes tal como no 1º cenário e o 2º incidente com 3 clientes, 1 da NOS, 1 da NPR e 1 da Vodafone. Na Figura

13, encontramos o resultado obtido pela Aplicação de Triagem. Em complemento, nos anexos, temos a confirmação do resultado através da Aplicação de Consulta, nesse sentido, temos a Figura 25 com os alarmes do incidente nº 1 e temos a Figura 26 com os alarmes do incidente nº 2.

Figura 13: Resultado do Terceiro Cenário de Teste.

Como cenário final foi utilizada uma captura com dados reais registados na infraestrutura da rede da dstelecom num período de tempo de duas horas. Estes dados contém o registo dos alarmes onde se identificou um incidente no **ROE S32/R06**. Isto significa que, após a execução da Aplicação de Triagem, esta deve exibir no ecrã um incidente composto por seis clientes Vodafone, um cliente da blu e um cliente da Nowo. No fim da execução da aplicação o ecrã mostra a tabela visível na Figura 14. Para além deste

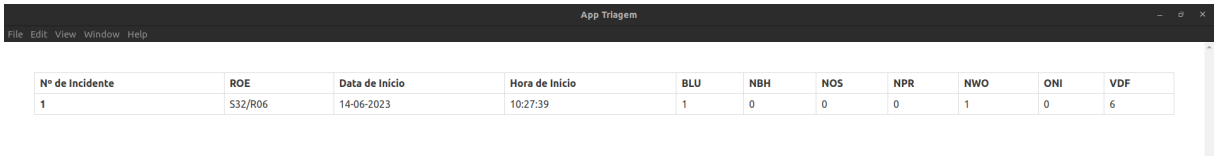


Figura 14: Resultado após a execução.

resultado, com este cenário também foi possível verificar a funcionalidade de encontrar o responsável por um alarme, com o auxílio da Aplicação de Consulta, é possível verificar na Figura 27 que existem alarmes com responsáveis atribuídos.

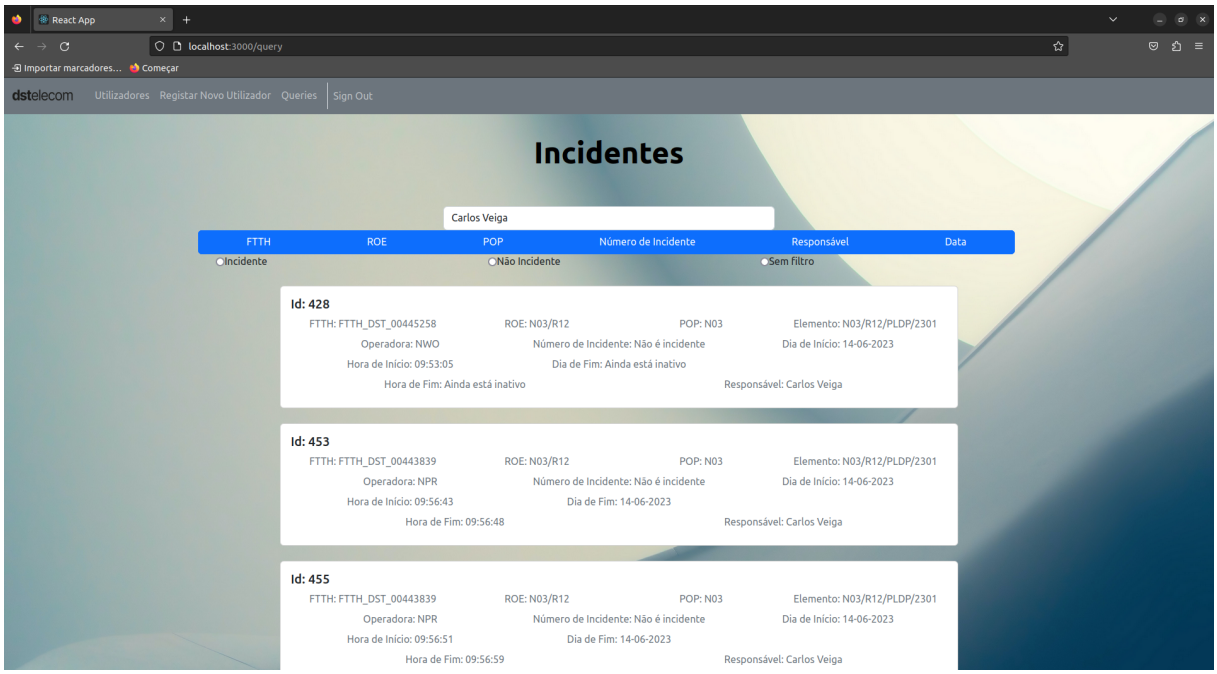


Figura 15: Alarmes com o Carlos Veiga como responsável.

Plataforma Kaleido

Para testar o desempenho da Kaleido teve-se em consideração o tempo de execução da Aplicação de Triagem. O teste executado compara o tempo de execução da aplicação em ambiente local, ou seja, com a rede na máquina local e em ambiente *deployed*, ou seja, com a rede na plataforma Kaleido. Neste teste utilizou-se a mesma captura do cenário final presente na Subsecção 5.2.1, os resultados obtidos encontram-se na Listagem 5.2.

```
1 LOCAL:
2 INICIO: Wed Aug 30 2023 14:16:30 GMT+0100 (Horário de Verão da Europa
    Ocidental)
3 FIM: Wed Aug 30 2023 16:02:24 GMT+0100 (Horário de Verão da Europa
    Ocidental)
4
5 DEPLOYED:
6 INICIO: Sat Sep 09 2023 17:52:20 GMT+0100 (Horário de Verão da Europa
    Ocidental)
7 FIM: Sat Sep 09 2023 18:11:35 GMT+0100 (Horário de Verão da Europa
    Ocidental)
```

Listagem 5.2: Resultados do Teste de Desempenho

Verifica-se uma diminuição significativa no tempo de execução da aplicação, uma vez que a solução

local demora 1 hora e 46 minutos a processar todos os alarmes da captura e a solução que utiliza a rede do Kaleido demora 19 minutos. Estas diferenças podem ser justificadas pela diferença da complexidade da rede, mas também pela diferença de utilização dos recursos da máquina onde é executada a aplicação. No caso da rede disponibilizada pela Kaleido, esta apenas possui uma organização com 2 nodos (*orderer* e *peer*), ou seja, quando é efetuada uma transação apenas um *peer* assina e valida a mesma. Quando a rede é maior, essa validação também tem de ser feita pelos outros *peers* o que vai causar um aumento no tempo de execução.

5.2.2 Testes de Aplicação de Consulta

Mediante o ator que interage com a aplicação, vão existir páginas diferentes, ou seja, o administrador tem páginas exclusivas, enquanto que os restantes utilizadores têm apenas acesso a 3 páginas: *home page*, *login* e de *queries*. Estas diferenças são visíveis pela *navbar* apresentada a cada tipo de ator, conforme mostrado nas Figuras 16 e 17.

Figura 16: Navbar do Administrador.

Figura 17: Navbar do Utilizador.

Utilizador

Começando pela *home page*, esta apresenta ao utilizador uma informação introdutória sobre a aplicação, conforme mostrado na Figura 18.

A página de *login*, mostrada na Figura 19, apresenta duas entradas de input para o utilizador inserir as suas credencias.

A página que possibilita a consulta do *world state* através de *queries* como representado na Figura 20. Nesta, o utilizador coloca na caixa de texto o parâmetro pelo qual quer fazer a consulta e, de seguida, carrega no botão do tipo de *query* que quer fazer. Para além disso, o utilizador pode ainda escolher o tipo de filtro a aplicar aos resultados, por defeito não é aplicado nenhum filtro. A Figura 20 foi obtida numa sessão de um utilizador criado como pertencente à empresa NOS, por isso a este não lhe é possível

Figura 18: Home Page.

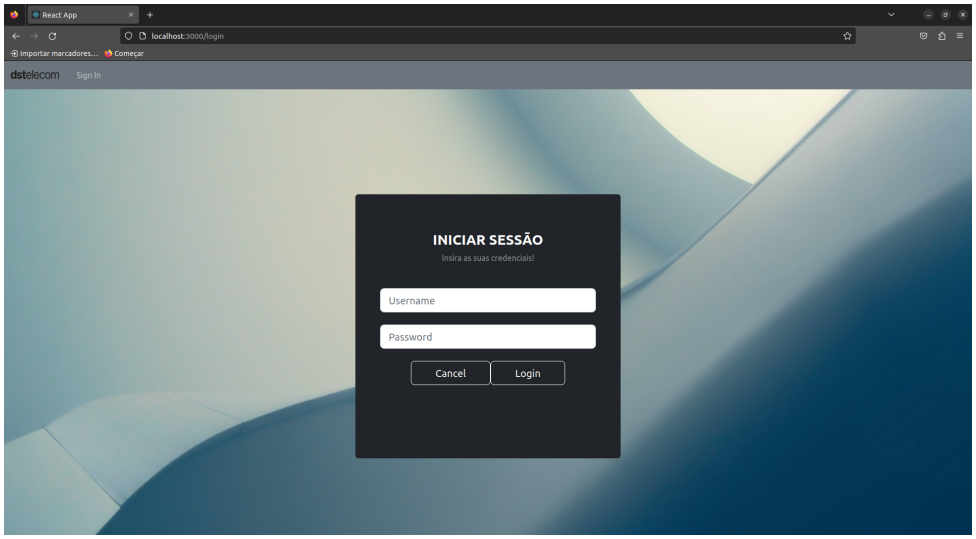


Figura 19: Página de Login.

fazer a *query* por responsável e não tem esse botão. Caso, o utilizador seja da dstelecom essa opção é disponibilizada. Para além disso, a informação do alarme mostrada ao utilizador também difere se este for da dstelecom ou de outra empresa, nomeadamente, os funcionários da dstelecom podem ver toda a informação dos alarmes e podem ver todos os alarmes, os funcionários de empresas externas apenas podem ver alarmes de dispositivos pertencentes à empresa e não podem visualizar o campo do responsável pelo alarme, nos anexos a Figura 27 e 28 demonstram essa diferença.

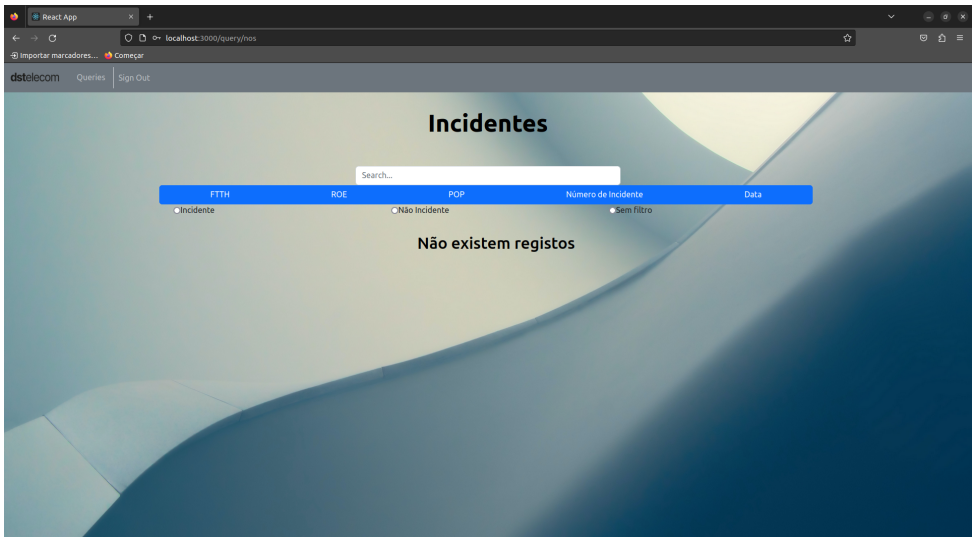


Figura 20: Página de Queries.

Administrador

Como referido anteriormente, o administrador tem acesso a páginas exclusivas, tais como, registar utilizadores e gestão de utilizadores. A página de *queries* no caso do administrador é igual à de um utilizador da dstelecom, ou seja, tem acesso a todos os tipos de consulta e a toda a informação dos alarmes. Na página de registo de utilizador é apresentado um formulário com as informações necessárias para registar um utilizador, após o administrador preencher os campos e submeter o pedido, o utilizador é registado na base de dados, ver Figura 21.

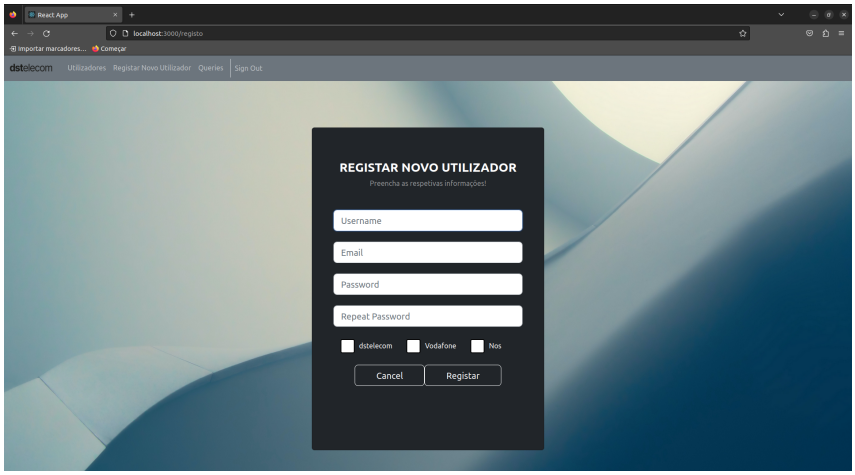


Figura 21: Página de Registo de Utilizadores.

Na página de gestão,ver Figura 22 , são apresentados todos os utilizadores registados na aplicação assim como as suas informações mais importantes, nomeadamente, nome, email e os respetivos *roles*

na aplicação. É ainda apresentado um botão que permite ao administrador apagar o utilizador da aplicação caso já não faça sentido este ter acesso à mesma, por exemplo no caso do despedimento de um funcionário.

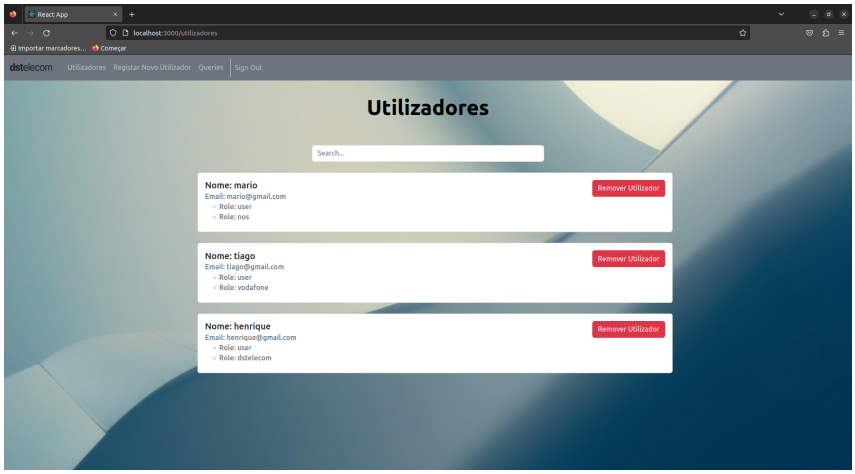


Figura 22: Página de Gestão de Utilizadores.

Capítulo 6

Conclusões e trabalho futuro

Este capítulo examina a dissertação na sua totalidade, apresenta os resultados e discute os obstáculos e soluções encontrados. Além disso, aborda aspetos que podem ser melhorados num trabalho futuro.

6.1 Conclusões

A primeira etapa desta dissertação consistiu em estudar os diversos tópicos que se pretendia abordar na mesma, começando por perceber o que é a *blockchain* e os conceitos básicos associados a esta tecnologia. Neste estudo, foi feita a distinção entre o diferentes tipos de *blockchain*, bem como uma comparação entre as mesmas. Ainda nesta fase, foram investigadas ferramentas que permitissem utilizar o tipo de *blockchain* desejado, e para além disso, foram estudados projetos já realizados com tecnologias semelhantes. No fim desta fase, decidiu-se a tecnologia a ser utilizada, Hyperledger Fabric, que além de suportar funcionalidades necessárias para o desenvolvimento desta dissertação, está em constante desenvolvimento por uma grande comunidade, foi usada em vários projetos relacionados, e é open-source e gratuita.

Após a conclusão da análise das tecnologias e trabalhos relacionados, o próximo passo foi o desenvolvimento das aplicações de triagem e consulta. Para começar, foi necessário definir uma lista de requisitos e prioridades de desenvolvimento. Em seguida, foram desenvolvidos modelos para definir as arquiteturas das aplicações e como os seus componentes se conectariam. Esta fase foi vital para o desenvolvimento de software, pois qualquer falha no desenho técnico e funcional da aplicação a desenvolver poderia resultar no seu insucesso posterior.

A fase mais desafiadora de toda a dissertação, a implementação, foi abordada após a definição da arquitetura da solução. O primeiro passo foi definir todas as tecnologias necessárias para implementar as funcionalidades descritas, incluindo frameworks e linguagens de programação. Com base na priorização dos requisitos, as funcionalidades relacionadas aos requisitos obrigatórios foram desenvolvidas primeiro.

Foi nesta fase que foram encontradadas a maiores dificuldades, porque foram utilizadas muitas tecnologias com as quais não tinha experiência. Durante a implementação foram realizados testes unitários, que isolaram partes do código para garantir que a Aplicação de Triagem e o *chaincode* funcionavam corretamente. Posteriormente, foi necessário testar as aplicações como um todo para garantir, que as mesmas funcionavam corretamente e cumpriam o seu propósito.

Em resumo, o sistema desenvolvido mostrou ser uma solução adequada, pois cumpriu com todos os requisitos propostos e passou em todos os testes realizados.

6.2 Perspetiva de trabalho futuro

Os objetivos estabelecidos para a realização desta dissertação foram atingidos. Contudo, há margem para introduzir aperfeiçoamentos ou funcionalidades adicionais. A seguir, serão propostas algumas ideias para trabalhos futuros:

- Fazer *deploy* da Aplicação de Consulta, permitindo aos devidos funcionários e empresas o acesso à mesma, e habilitá-la a comunicar com a rede criada no Kaleido ou com outra rede futuramente criada.

- Alterar a Aplicação de Triagem para tratar os alarmes em tempo real, integrando os vários sistemas, dos quais o processamento dos alarmes depende, com aplicação de modo à mesma ter acesso em tempo real aos alarmes que são disparados na rede, bem como às informações atualizadas dos funcionários, das equipas e do cadastro.

- Desenvolver uma rede para produção, ou seja, incluir na rede as empresas com quem a dstelecom deseja partilhar os dados.

- Permitir a recolha de novos alarmes tais como alarmística de **Base Transceiver Station** (**BTS**) e de **Passive Optical Network** (**PON**) e realizar a integração a novas aplicações para recolha de alarmística tais como Altiplano e MEO.

Bibliografia

Maher Alharby and Aad van Moorsel. The impact of profit uncertainty on miner decisions in blockchain systems. *Electronic Notes in Theoretical Computer Science*, 340:151–167, 2018.

Arati Baliga. Understanding blockchain consensus models. *Persistent*, 4(1):14, 2017.

Joseph J Bambara and Paul R Allen. Blockchain. *A practical guide to developing business, law and technology solutions. New York City: McGraw-Hill Professional*, 2018.

Rafael Belchior, Miguel Correia, and André Vasconcelos. Justicechain: Using blockchain to protect justice logs. In *OTM Confederated International Conferences"On the Move to Meaningful Internet Systems"*, pages 318–325. Springer, 2019.

Amjad Hudaib, Raja Masadeh, Mais Haj Qasem, Abdullah Alzaqebah, et al. Requirements prioritization techniques comparison. *Modern Applied Science*, 12(2):62, 2018.

Dodo Khan, Low Tang Jung, Manzoor Ahmed Hashmani, and Ahmad Waqas. A critical review of blockchain consensus model. In *2020 3rd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, pages 1–6. IEEE, 2020.

Akash Raj Khettry, Karthik R Patil, and Abhilash C Basavaraju. A detailed review on blockchain and its applications. *SN Computer Science*, 2(1):1–9, 2021.

Jennifer Li and Mohamad Kassem. Applications of distributed ledger technology (dlt) and blockchain-enabled smart contracts in construction. *Automation in construction*, 132:103955, 2021.

William Pourmajidi and Andriy Miranskyy. Logchain: Blockchain-assisted log storage. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 978–982, 2018. doi: 10.1109/CLOUD.2018.00150.

Dylan Yaga, Peter Mell, Nik Roby, and Karen Scarfone. Blockchain technology overview. *arXiv preprint arXiv:1906.11078*, 2019.

Wai-Keung Yeung, M.K. Tong Stephen, Y.K. Wong Jerry, Mentor Cheung, K.Y. Cheung Ashley, Graham Lui, and K.K. Cheng Kevin. Implementation of digital log-book system for lifts and escalators based on blockchain technology. In *2022 IEEE International Conference on Blockchain (Blockchain)*, pages 356–361, 2022. doi: 10.1109/Blockchain55522.2022.00056.

Parte I
Apêndices

Apêndice A
Listagens

A.1 Queries

```
async QueryAccessByPop(ctx, pop) {
    let queryString = {};
    queryString.selector = {};
    queryString.selector.docType = "access";
    queryString.selector.pop = pop;
    queryString.use_index = ["_design/indexPopDoc", "indexPop"];
    return await this.GetQueryResultForQueryString(ctx, JSON.stringify(
   queryString)); //shim.success(queryResults);
  }
```

Listagem A.1: Transação "QueryAccessByPop"

```
async QueryAccessByDate(ctx, day) {
    let queryString = {};
    queryString.selector = {};
    queryString.selector.docType = "access";
    queryString.selector.dia_inicio = day;
    queryString.use_index = ["_design/indexDateDoc", "indexDate"];
    return await this.GetQueryResultForQueryString(ctx, JSON.stringify(
   queryString)); //shim.success(queryResults);
  }
```

Listagem A.2: Transação "QueryAccessByDate"

```
async QueryAccessByRoeDate(ctx, roe, dia_inicio) {
    let queryString = {};
    queryString.selector = {};
    queryString.selector.roe = roe;
    queryString.selector.dia_inicio = dia_inicio;
    queryString.use_index = ["_design/indexRoeDateDoc", "indexRoeDate"];
    return await this.GetQueryResultForQueryString(ctx, JSON.stringify(
   queryString)); //shim.success(queryResults);
  }
```

Listagem A.3: Transação "QueryAccessByRoeDate"

```
async QueryAccessByResponsible(ctx, responsible) {
    let queryString = {};
    queryString.selector = {};
    queryString.selector.responsavel = responsible;
    queryString.use_index = ["_design/indexResponsibleDoc", "
   indexResponsible"];
```

```
    return await this.GetQueryResultForQueryString(ctx, JSON.stringify(
    queryString)); //shim.success(queryResults);
  }
```

Listagem A.4: Transação "QueryAccessByResponsible"

```
async QueryAccessByFtth(ctx, ftth) {
    let queryString = {};
    queryString.selector = {};
    queryString.selector.docType = "access";
    queryString.selector.ftth = ftth;
    queryString.use_index = ["_design/indexFtthDoc", "indexFtth"];
    return await this.GetQueryResultForQueryString(ctx, JSON.stringify(
    queryString)); //shim.success(queryResults);
  }
```

Listagem A.5: Transação "QueryAccessByFtth"

```
async QueryAccessByPosition(ctx, position) {
    let queryString = {};
    queryString.selector = {};
    queryString.selector.docType = "access";
    queryString.selector.position = position;
    queryString.use_index = ["_design/indexPositionDoc", "indexPosition"];
    return await this.GetQueryResultForQueryString(ctx, JSON.stringify(
    queryString)); //shim.success(queryResults);
  }
```

Listagem A.6: Transação "QueryAccessByPosition"

```
async QueryAccessByIncidNumber(ctx, incid) {
    let queryString = {};
    queryString.selector = {};
    queryString.selector.docType = "access";
    queryString.selector.incidente = incid;
    queryString.use_index = ["_design/indexIncidDoc", "indexIncid"];
    return await this.GetQueryResultForQueryString(ctx, JSON.stringify(
    queryString)); //shim.success(queryResults);
  }
```

Listagem A.7: Transação "QueryAccessByIncidNumber"

A.2 Rotas

A.2.1 Funcionários dstelecom

```
router.get("/ftth", [authJwt.verifyToken, authJwt.isDstelecom], async (req,
    res) => {
  try {
    //DETERMINAR O USER LOGGADO
    const user = await User.findOne({ _id: req.userId });

    if (!user) {
      console.log("User not found!");
      return res.send("User Not Found!");
    }
```

```
    const username = user.username;

    //CONEXÃO Á BLOCKCHAIN
    let networkObj = await network.connectToNetwork(username);

    let uri = await network.invoke(networkObj, true, "QueryAccessByFtth", [req.query.ftth]);

    let finalResponse = JSON.parse(uri.toString());
    if (req.query.filter === "incident") {
      finalResponse = finalResponse.filter((ob) => ob.Record.incidente !== null);
    } else if (req.query.filter === "no_incident") {
      finalResponse = finalResponse.filter((ob) => ob.Record.incidente === null);
    }
    res.send(finalResponse);
  } catch (err) {
    res.send(err);
  }
});
```

Listagem A.8: Rota da query por FTTH

```
router.get("/incident", [authJwt.verifyToken, authJwt.isDstelecom], async (req, res) => {
  try {
    //DETERMINAR O USER LOGGADO
    const user = await User.findOne({ _id: req.userId });

    if (!user) {
      console.log("User not found!");
      return res.send("User Not Found!");
    }

    const username = user.username;

    //CONEXÃO Á BLOCKCHAIN
    let networkObj = await network.connectToNetwork(username);
    let uri = await network.invoke(networkObj, true, "QueryAccessByIncidNumber", [req.query.number]);
    res.send(uri.toString());
  } catch (err) {
    res.send(err);
  }
});
```

Listagem A.9: Rota da query por Número de Incidente

```
router.get("/roe", [authJwt.verifyToken, authJwt.isDstelecom], async (req, res) => {
  try {
    //DETERMINAR O USER LOGGADO
    const user = await User.findOne({ _id: req.userId });

    if (!user) {
      console.log("User not found!");
      return res.send("User Not Found!");
```

```
    }

    const username = user.username;

    //CONEXÃO Á BLOCKCHAIN
    let networkObj = await network.connectToNetwork(username);

    let uri = await network.invoke(networkObj, true, "QueryAccessByRoe", [
   req.query.roe]);
    let finalResponse = JSON.parse(uri.toString());
    if (req.query.filter === "incident") {
      finalResponse = finalResponse.filter((ob) => ob.Record.incidente !==
   null);
    } else if (req.query.filter === "no_incident") {
      finalResponse = finalResponse.filter((ob) => ob.Record.incidente ===
   null);
    }
    res.send(finalResponse);
  } catch (err) {
    res.send(err);
  }
});
```

Listagem A.10: Rota da query por ROE

```
router.get("/pop", [authJwt.verifyToken, authJwt.isDstelecom], async (req,
   res) => {
  try {
    //DETERMINAR O USER LOGGADO
    const user = await User.findOne({ _id: req.userId });

    if (!user) {
      console.log("User not found!");
      return res.send("User Not Found!");
    }

    const username = user.username;

    //CONEXÃO Á BLOCKCHAIN
    let networkObj = await network.connectToNetwork(username);

    let uri = await network.invoke(networkObj, true, "QueryAccessByPop", [
   req.query.pop]);
    let finalResponse = JSON.parse(uri.toString());
    if (req.query.filter === "incident") {
      finalResponse = finalResponse.filter((ob) => ob.Record.incidente !==
   null);
    } else if (req.query.filter === "no_incident") {
      finalResponse = finalResponse.filter((ob) => ob.Record.incidente ===
   null);
    }
    res.send(finalResponse);
  } catch (err) {
    res.send(err);
  }
});
```

Listagem A.11: Rota da query por POP

```
router.get("/date", [authJwt.verifyToken, authJwt.isDstelecom], async (req, res) => {
  try {
    //DETERMINAR O USER LOGGADO
    const user = await User.findOne({ _id: req.userId });

    if (!user) {
      console.log("User not found!");
      return res.send("User Not Found!");
    }

    const username = user.username;

    //CONEXÃO Á BLOCKCHAIN
    let networkObj = await network.connectToNetwork(username);

    let uri = await network.invoke(networkObj, true, "QueryAccessByDate", [req.query.date]);
    let finalResponse = JSON.parse(uri.toString());
    if (req.query.filter === "incident") {
      finalResponse = finalResponse.filter((ob) => ob.Record.incidente !== null);
    } else if (req.query.filter === "no_incident") {
      finalResponse = finalResponse.filter((ob) => ob.Record.incidente === null);
    }
    res.send(finalResponse);
  } catch (err) {
    res.send(err);
  }
});
```

Listagem A.12: Rota da query por Data

```
router.get("/responsible", [authJwt.verifyToken, authJwt.isDstelecom], async (req, res) => {
  try {
    //DETERMINAR O USER LOGGADO
    const user = await User.findOne({ _id: req.userId });

    if (!user) {
      console.log("User not found!");
      return res.send("User Not Found!");
    }

    const username = user.username;

    //CONEXÃO Á BLOCKCHAIN
    let networkObj = await network.connectToNetwork(username);

    let uri = await network.invoke(networkObj, true, "QueryAccessByResponsible", [req.query.responsavel]);
    let finalResponse = JSON.parse(uri.toString());
    if (req.query.filter === "incident") {
      finalResponse = finalResponse.filter((ob) => ob.Record.incidente !== null);
    } else if (req.query.filter === "no_incident") {
      finalResponse = finalResponse.filter((ob) => ob.Record.incidente === null);
```

```
    }
    res.send(finalResponse);
  } catch (err) {
    res.send(err);
  }
});
```

Listagem A.13: Rota da query por Responsável

A.2.2 Funcionários Vodafone

```
router.get("/incident", [authJwt.verifyToken, authJwt.isVodafone], async (
    req, res) => {
  try {
    //DETERMINAR O USER LOGGADO
    const user = await User.findOne({ _id: req.userId });

    if (!user) {
      console.log("User not found!");
      return res.send("User Not Found!");
    }

    const username = user.username;

    //CONEXÃO Á BLOCKCHAIN
    let networkObj = await network.connectToNetwork(username);
    let uri = await network.invoke(networkObj, true, "
    QueryAccessByIncidNumber", [req.query.number]);
    let finalResponse = JSON.parse(uri.toString());
    finalResponse = finalResponse.filter((ob) => ob.Record.operadora === "
    VDF");
    // finalResponse = finalResponse.map((ob) => {
    //   return (ob.Record.responsavel = null);
    // });
    res.send(finalResponse);
  } catch (err) {
    res.send(err);
  }
});
```

Listagem A.14: Rota da query por Número de Incidente

```
router.get("/roe", [authJwt.verifyToken, authJwt.isVodafone], async (req,
    res) => {
  try {
    //DETERMINAR O USER LOGGADO
    const user = await User.findOne({ _id: req.userId });

    if (!user) {
      console.log("User not found!");
      return res.send("User Not Found!");
    }

    const username = user.username;

    //CONEXÃO Á BLOCKCHAIN
    let networkObj = await network.connectToNetwork(username);
```

```
15
16      let uri = await network.invoke(networkObj, true, "QueryAccessByRoe", [
        req.query.roe]);
17      let finalResponse = JSON.parse(uri.toString());
18      finalResponse = finalResponse.filter((ob) => ob.Record.operadora === "
        VDF");
19      if (req.query.filter === "incident") {
20        finalResponse = finalResponse.filter((ob) => ob.Record.incidente !==
        null);
21      } else if (req.query.filter === "no_incident") {
22        finalResponse = finalResponse.filter((ob) => ob.Record.incidente ===
        null);
23      }
24      res.send(finalResponse);
25    } catch (err) {
26      res.send(err);
27    }
28  });
```

Listagem A.15: Rota da query por ROE

```
1   router.get("/pop", [authJwt.verifyToken, authJwt.isVodafone], async (req,
        res) => {
2     try {
3       //DETERMINAR O USER LOGGADO
4       const user = await User.findOne({ _id: req.userId });
5
6       if (!user) {
7         console.log("User not found!");
8         return res.send("User Not Found!");
9       }
10
11      const username = user.username;
12
13      //CONEXÃO Á BLOCKCHAIN
14      let networkObj = await network.connectToNetwork(username);
15
16      let uri = await network.invoke(networkObj, true, "QueryAccessByPop", [
        req.query.pop]);
17      let finalResponse = JSON.parse(uri.toString());
18      finalResponse = finalResponse.filter((ob) => ob.Record.operadora === "
        VDF");
19      if (req.query.filter === "incident") {
20        finalResponse = finalResponse.filter((ob) => ob.Record.incidente !==
        null);
21      } else if (req.query.filter === "no_incident") {
22        finalResponse = finalResponse.filter((ob) => ob.Record.incidente ===
        null);
23      }
24      res.send(finalResponse);
25    } catch (err) {
26      res.send(err);
27    }
28  });
```

Listagem A.16: Rota da query por POP

```
1   router.get("/date", [authJwt.verifyToken, authJwt.isVodafone], async (req,
        res) => {
```

```
  try {
    //DETERMINAR O USER LOGGADO
    const user = await User.findOne({ _id: req.userId });

    if (!user) {
      console.log("User not found!");
      return res.send("User Not Found!");
    }

    const username = user.username;

    //CONEXÃO Á BLOCKCHAIN
    let networkObj = await network.connectToNetwork(username);

    let uri = await network.invoke(networkObj, true, "QueryAccessByDate", [
   req.query.date]);
    let finalResponse = JSON.parse(uri.toString());
    finalResponse = finalResponse.filter((ob) => ob.Record.operadora === "
   VDF");
    if (req.query.filter === "incident") {
      finalResponse = finalResponse.filter((ob) => ob.Record.incidente !==
   null);
    } else if (req.query.filter === "no_incident") {
      finalResponse = finalResponse.filter((ob) => ob.Record.incidente ===
   null);
    }
    res.send(finalResponse);
  } catch (err) {
    res.send(err);
  }
});
```

Listagem A.17: Rota da query por Data

```
router.get("/responsible", [authJwt.verifyToken, authJwt.isVodafone], async
    (req, res) => {
  try {
    //DETERMINAR O USER LOGGADO
    const user = await User.findOne({ _id: req.userId });

    if (!user) {
      console.log("User not found!");
      return res.send("User Not Found!");
    }

    const username = user.username;

    //CONEXÃO Á BLOCKCHAIN
    let networkObj = await network.connectToNetwork(username);

    let uri = await network.invoke(networkObj, true, "
   QueryAccessByResponsible", [req.query.responsavel]);
    let finalResponse = JSON.parse(uri.toString());
    finalResponse = finalResponse.filter((ob) => ob.Record.operadora === "
   VDF");
    if (req.query.filter === "incident") {
      finalResponse = finalResponse.filter((ob) => ob.Record.incidente !==
   null);
    } else if (req.query.filter === "no_incident") {
```

```
      finalResponse = finalResponse.filter((ob) => ob.Record.incidente ===
    null);
    }
    res.send(finalResponse);
  } catch (err) {
    res.send(err);
  }
});
```

Listagem A.18: Rota da query por Responsável

A.2.3 Funcionários NOS

```
router.get("/ftth", [authJwt.verifyToken, authJwt.isNos], async (req, res)
    => {
  try {
    //DETERMINAR O USER LOGGADO
    const user = await User.findOne({ _id: req.userId });

    if (!user) {
      console.log("User not found!");
      return res.send("User Not Found!");
    }

    const username = user.username;

    //CONEXÃO Á BLOCKCHAIN
    let networkObj = await network.connectToNetwork(username);

    let uri = await network.invoke(networkObj, true, "QueryAccessByFtth", [
    req.query.ftth]);

    let finalResponse = JSON.parse(uri.toString());
    finalResponse = finalResponse.filter((ob) => ob.Record.operadora === "
    NOS" || ob.Record.operadora === "NPR");
    if (req.query.filter === "incident") {
      finalResponse = finalResponse.filter((ob) => ob.Record.incidente !==
    null);
    } else if (req.query.filter === "no_incident") {
      finalResponse = finalResponse.filter((ob) => ob.Record.incidente ===
    null);
    }
    res.send(finalResponse);
  } catch (err) {
    res.send(err);
  }
});
```

Listagem A.19: Rota da query por FTTH

```
router.get("/incident", [authJwt.verifyToken, authJwt.isNos], async (req,
    res) => {
  try {
    //DETERMINAR O USER LOGGADO
    const user = await User.findOne({ _id: req.userId });

    if (!user) {
```

```
      console.log("User not found!");
      return res.send("User Not Found!");
    }

    const username = user.username;

    //CONEXÃO Á BLOCKCHAIN
    let networkObj = await network.connectToNetwork(username);
    let uri = await network.invoke(networkObj, true, "
    QueryAccessByIncidNumber", [req.query.number]);
    let finalResponse = JSON.parse(uri.toString());
    finalResponse = finalResponse.filter((ob) => ob.Record.operadora === "
    NOS" || ob.Record.operadora === "NPR");
    res.send(finalResponse);
  } catch (err) {
    res.send(err);
  }
});
```

Listagem A.20: Rota da query por Número de Incidente

```
router.get("/roe", [authJwt.verifyToken, authJwt.isNos], async (req, res)
    => {
  try {
    //DETERMINAR O USER LOGGADO
    const user = await User.findOne({ _id: req.userId });

    if (!user) {
      console.log("User not found!");
      return res.send("User Not Found!");
    }

    const username = user.username;

    //CONEXÃO Á BLOCKCHAIN
    let networkObj = await network.connectToNetwork(username);

    let uri = await network.invoke(networkObj, true, "QueryAccessByRoe", [
    req.query.roe]);
    let finalResponse = JSON.parse(uri.toString());
    finalResponse = finalResponse.filter((ob) => ob.Record.operadora === "
    NOS" || ob.Record.operadora === "NPR");
    if (req.query.filter === "incident") {
      finalResponse = finalResponse.filter((ob) => ob.Record.incidente !==
    null);
    } else if (req.query.filter === "no_incident") {
      finalResponse = finalResponse.filter((ob) => ob.Record.incidente ===
    null);
    }
    res.send(finalResponse);
  } catch (err) {
    res.send(err);
  }
});
```

Listagem A.21: Rota da query por ROE

```
router.get("/pop", [authJwt.verifyToken, authJwt.isNos], async (req, res)
    => {
```

```
  try {
    //DETERMINAR O USER LOGGADO
    const user = await User.findOne({ _id: req.userId });

    if (!user) {
      console.log("User not found!");
      return res.send("User Not Found!");
    }

    const username = user.username;

    //CONEXÃO Á BLOCKCHAIN
    let networkObj = await network.connectToNetwork(username);

    let uri = await network.invoke(networkObj, true, "QueryAccessByPop", [
   req.query.pop]);
    let finalResponse = JSON.parse(uri.toString());
    finalResponse = finalResponse.filter((ob) => ob.Record.operadora === "
   NOS" || ob.Record.operadora === "NPR");
    if (req.query.filter === "incident") {
      finalResponse = finalResponse.filter((ob) => ob.Record.incidente !==
   null);
    } else if (req.query.filter === "no_incident") {
      finalResponse = finalResponse.filter((ob) => ob.Record.incidente ===
   null);
    }
    res.send(finalResponse);
  } catch (err) {
    res.send(err);
  }
});
```

Listagem A.22: Rota da query por POP

```
router.get("/date", [authJwt.verifyToken, authJwt.isNos], async (req, res)
   => {
  try {
    //DETERMINAR O USER LOGGADO
    const user = await User.findOne({ _id: req.userId });

    if (!user) {
      console.log("User not found!");
      return res.send("User Not Found!");
    }

    const username = user.username;

    //CONEXÃO Á BLOCKCHAIN
    let networkObj = await network.connectToNetwork(username);

    let uri = await network.invoke(networkObj, true, "QueryAccessByDate", [
   req.query.date]);
    let finalResponse = JSON.parse(uri.toString());
    finalResponse = finalResponse.filter((ob) => ob.Record.operadora === "
   NOS" || ob.Record.operadora === "NPR");
    if (req.query.filter === "incident") {
      finalResponse = finalResponse.filter((ob) => ob.Record.incidente !==
   null);
    } else if (req.query.filter === "no_incident") {
```

```
      finalResponse = finalResponse.filter((ob) => ob.Record.incidente === null);
    }
    res.send(finalResponse);
  } catch (err) {
    res.send(err);
  }
});
```

Listagem A.23: Rota da query por Data

```
router.get("/responsible", [authJwt.verifyToken, authJwt.isNos], async (req, res) => {
  try {
    //DETERMINAR O USER LOGGADO
    const user = await User.findOne({ _id: req.userId });

    if (!user) {
      console.log("User not found!");
      return res.send("User Not Found!");
    }

    const username = user.username;

    //CONEXÃO Á BLOCKCHAIN
    let networkObj = await network.connectToNetwork(username);

    let uri = await network.invoke(networkObj, true, "QueryAccessByResponsible", [req.query.responsavel]);
    let finalResponse = JSON.parse(uri.toString());
    finalResponse = finalResponse.filter((ob) => ob.Record.operadora === "NOS" || ob.Record.operadora === "NPR");
    if (req.query.filter === "incident") {
      finalResponse = finalResponse.filter((ob) => ob.Record.incidente !== null);
    } else if (req.query.filter === "no_incident") {
      finalResponse = finalResponse.filter((ob) => ob.Record.incidente === null);
    }
    res.send(finalResponse);
  } catch (err) {
    res.send(err);
  }
});
```

Listagem A.24: Rota da query por Responsável

A.3 Cenários de Teste

```
   U31_02 2023-06-14 10:27:40
** 13435.0 REPT
   ALARMID=1674122276351  ALARMDESC=[GPON Alarm]ONU LOS(Loss of Signal)
   (35113)  DID=10.165.1.32  DIP=10.165.1.32  DNAME=S32A01_01_01  DTYPE=
   C300v1.3  POSITION=10.165.1.32_1_1_2_1_51  SEVERITY=Minor  HAPPENTIME
   =2023-06-14 10:27:39.000  ALARMTYPE=QoSAlarm  DESC=ONU ID=51,ONU Name=
   VDF-FTTH_DST_00608799,ONU Type=HUAWEI-HS8247W,Authentication Value=
```

```
   CODtrrr240,ONU Registration Info: Password = CODtrrr240,Service Level=
   RESIDENCIAL  ENTITY_ID=IP Address:10.165.1.32|4613851  EVENT_CODE=35113
    PROBABLE_CAUSE_DESC=1.Fiber link failure. 2.ONU failure.
   PROBABEL_CAUSE_CODE=351130000  SYSTEM_TYPE=768  ACKSTATUS=Unacknowledged
;

   U31_02 2023-06-14 10:28:10
** 13521.0 REPT
   ALARMID=1674122276409  ALARMDESC=[GPON Alarm]ONU LOS(Loss of Signal)
   (35113)  DID=10.165.1.32  DIP=10.165.1.32  DNAME=S32A01_01_01  DTYPE=
   C300v1.3  POSITION=10.165.1.32_1_1_2_1_47  SEVERITY=Minor  HAPPENTIME
   =2023-06-14 10:48:11.000  ALARMTYPE=QoSAlarm  DESC=ONU ID=47,ONU Name=
   NWO-FTTH_DST_00571239,ONU Type=ZTE-F680,Authentication Value=jwatmoq231,
   ONU Registration Info: Password = jwatmoq231,Service Level=RESIDENCIAL
   ENTITY_ID=IP Address:10.165.1.32|4613874  EVENT_CODE=35113
   PROBABLE_CAUSE_DESC=1.Fiber link failure. 2.ONU failure.
   PROBABEL_CAUSE_CODE=351130000  SYSTEM_TYPE=768  ACKSTATUS=Unacknowledged
;

   U31_02 2023-06-14 10:29:45
** 13991.0 REPT
   ALARMID=1674122276731  ALARMDESC=[GPON Alarm]ONU LOS(Loss of Signal)
   (35113)  DID=10.165.1.32  DIP=10.165.1.32  DNAME=S32A01_01_01  DTYPE=
   C300v1.3  POSITION=10.165.1.32_1_1_2_1_21  SEVERITY=Minor  HAPPENTIME
   =2023-06-14 11:09:44.000  ALARMTYPE=QoSAlarm  DESC=ONU ID=21,ONU Name=
   VDF-FTTH_DST_00337216,ONU Type=ZTE-F612C,Authentication Value=pTiaAkN201
   ,ONU Registration Info: Password = pTiaAkN201,Service Level=RESIDENCIAL
    ENTITY_ID=IP Address:10.165.1.32|4613976  EVENT_CODE=35113
   PROBABLE_CAUSE_DESC=1.Fiber link failure. 2.ONU failure.
   PROBABEL_CAUSE_CODE=351130000  SYSTEM_TYPE=768  ACKSTATUS=Unacknowledged
;

   U31_02 2023-06-14 10:34:03
** 14780.0 REPT
   ALARMID=1674122277378  ALARMDESC=[GPON Alarm]ONU LOS(Loss of Signal)
   (35113)  DID=10.165.1.32  DIP=10.165.1.32  DNAME=S32A01_01_01  DTYPE=
   C300v1.3  POSITION=10.165.1.32_1_1_2_1_37  SEVERITY=Major  HAPPENTIME
   =2023-06-14 11:34:03.000  ALARMTYPE=QoSAlarm  DESC=ONU ID=37,ONU Name=
   BLU-EMP-FTTH_DST_00161342,ONU Type=ZTE-F601,Authentication Value=
   QlpQXbw086,ONU Registration Info: Password = QlpQXbw086,Service Level=
   EMPRESARIAL  ENTITY_ID=IP Address:10.165.1.32|4614164  EVENT_CODE=35113
    PROBABLE_CAUSE_DESC=1.Fiber link failure. 2.ONU failure.
   PROBABEL_CAUSE_CODE=351130000  SYSTEM_TYPE=768  ACKSTATUS=Unacknowledged
;
```

Listagem A.25: Primeiro cenário de teste

```
   U31_02 2023-06-14 10:27:40
** 13435.0 REPT
   ALARMID=1674122276351  ALARMDESC=[GPON Alarm]ONU LOS(Loss of Signal)
   (35113)  DID=10.165.1.32  DIP=10.165.1.32  DNAME=S32A01_01_01  DTYPE=
   C300v1.3  POSITION=10.165.1.32_1_1_2_1_51  SEVERITY=Minor  HAPPENTIME
   =2023-06-14 10:27:39.000  ALARMTYPE=QoSAlarm  DESC=ONU ID=51,ONU Name=
   VDF-FTTH_DST_00608799,ONU Type=HUAWEI-HS8247W,Authentication Value=
   CODtrrr240,ONU Registration Info: Password = CODtrrr240,Service Level=
   RESIDENCIAL  ENTITY_ID=IP Address:10.165.1.32|4613851  EVENT_CODE=35113
    PROBABLE_CAUSE_DESC=1.Fiber link failure. 2.ONU failure.
   PROBABEL_CAUSE_CODE=351130000  SYSTEM_TYPE=768  ACKSTATUS=Unacknowledged
;
```

```
5
6     U31_02 2023-06-14 10:28:10
7  ** 13521.0 REPT
8     ALARMID=1674122276409  ALARMDESC=[GPON Alarm]ONU LOS(Loss of Signal)
      (35113)  DID=10.165.1.32  DIP=10.165.1.32  DNAME=S32A01_01_01  DTYPE=
      C300v1.3  POSITION=10.165.1.32_1_1_2_1_47  SEVERITY=Minor  HAPPENTIME
      =2023-06-14 10:48:11.000  ALARMTYPE=QoSAlarm  DESC=ONU ID=47,ONU Name=
      NWO-FTTH_DST_00571239,ONU Type=ZTE-F680,Authentication Value=jwatmoq231,
      ONU Registration Info: Password = jwatmoq231,Service Level=RESIDENCIAL
      ENTITY_ID=IP Address:10.165.1.32|4613874  EVENT_CODE=35113
      PROBABLE_CAUSE_DESC=1.Fiber link failure. 2.ONU failure.
      PROBABEL_CAUSE_CODE=351130000  SYSTEM_TYPE=768  ACKSTATUS=Unacknowledged
9  ;
10
11    U31_02 2023-06-14 10:28:10
12 ** 13521.0 REPT
13    ALARMID=1674122276409  ALARMDESC=[GPON Alarm]ONU LOS(Loss of Signal)
      (35113)  DID=10.165.1.32  DIP=10.165.1.32  DNAME=S32A01_01_01  DTYPE=
      C300v1.3  POSITION=10.165.1.32_1_1_2_1_47  SEVERITY=Restore  HAPPENTIME
      =2023-06-14 10:49:11.000  ALARMTYPE=QoSAlarm  DESC=ONU ID=47,ONU Name=
      NWO-FTTH_DST_00571239,ONU Type=ZTE-F680,Authentication Value=jwatmoq231,
      ONU Registration Info: Password = jwatmoq231,Service Level=RESIDENCIAL
      ENTITY_ID=IP Address:10.165.1.32|4613874  EVENT_CODE=35113
      PROBABLE_CAUSE_DESC=1.Fiber link failure. 2.ONU failure.
      PROBABEL_CAUSE_CODE=351130000  SYSTEM_TYPE=768  ACKSTATUS=Unacknowledged
14 ;
15
16    U31_02 2023-06-14 10:29:45
17 ** 13991.0 REPT
18    ALARMID=1674122276731  ALARMDESC=[GPON Alarm]ONU LOS(Loss of Signal)
      (35113)  DID=10.165.1.32  DIP=10.165.1.32  DNAME=S32A01_01_01  DTYPE=
      C300v1.3  POSITION=10.165.1.32_1_1_2_1_21  SEVERITY=Minor  HAPPENTIME
      =2023-06-14 11:09:44.000  ALARMTYPE=QoSAlarm  DESC=ONU ID=21,ONU Name=
      VDF-FTTH_DST_00337216,ONU Type=ZTE-F612C,Authentication Value=pTiaAkN201
      ,ONU Registration Info: Password = pTiaAkN201,Service Level=RESIDENCIAL
       ENTITY_ID=IP Address:10.165.1.32|4613976  EVENT_CODE=35113
      PROBABLE_CAUSE_DESC=1.Fiber link failure. 2.ONU failure.
      PROBABEL_CAUSE_CODE=351130000  SYSTEM_TYPE=768  ACKSTATUS=Unacknowledged
19 ;
20
21    U31_02 2023-06-14 10:34:03
22 ** 14780.0 REPT
23    ALARMID=1674122277378  ALARMDESC=[GPON Alarm]ONU LOS(Loss of Signal)
      (35113)  DID=10.165.1.32  DIP=10.165.1.32  DNAME=S32A01_01_01  DTYPE=
      C300v1.3  POSITION=10.165.1.32_1_1_2_1_37  SEVERITY=Major  HAPPENTIME
      =2023-06-14 11:34:03.000  ALARMTYPE=QoSAlarm  DESC=ONU ID=37,ONU Name=
      BLU-EMP-FTTH_DST_00161342,ONU Type=ZTE-F601,Authentication Value=
      QlpQXbw086,ONU Registration Info: Password = QlpQXbw086,Service Level=
      EMPRESARIAL  ENTITY_ID=IP Address:10.165.1.32|4614164  EVENT_CODE=35113
       PROBABLE_CAUSE_DESC=1.Fiber link failure. 2.ONU failure.
      PROBABEL_CAUSE_CODE=351130000  SYSTEM_TYPE=768  ACKSTATUS=Unacknowledged
24 ;
```

Listagem A.26: Segundo cenário de teste

```
1     U31_02 2023-06-14 10:27:40
2  ** 13435.0 REPT
3     ALARMID=1674122276351  ALARMDESC=[GPON Alarm]ONU LOS(Loss of Signal)
      (35113)  DID=10.165.1.32  DIP=10.165.1.32  DNAME=S32A01_01_01  DTYPE=
```

```
   C300v1.3  POSITION=10.165.1.32_1_1_2_1_51  SEVERITY=Minor  HAPPENTIME
   =2023-06-14 10:27:39.000  ALARMTYPE=QoSAlarm  DESC=ONU ID=51,ONU Name=
   VDF-FTTH_DST_00608799,ONU Type=HUAWEI-HS8247W,Authentication Value=
   CODtrrr240,ONU Registration Info: Password = CODtrrr240,Service Level=
   RESIDENCIAL  ENTITY_ID=IP Address:10.165.1.32|4613851  EVENT_CODE=35113
    PROBABLE_CAUSE_DESC=1.Fiber link failure. 2.ONU failure.
   PROBABEL_CAUSE_CODE=351130000  SYSTEM_TYPE=768  ACKSTATUS=Unacknowledged
;

   U31_02 2023-06-14 10:30:03
** 14780.0 REPT
   ALARMID=1674122277378  ALARMDESC=[GPON Alarm]ONU LOS(Loss of Signal)
   (35113)  DID=10.165.1.32  DIP=10.165.1.32  DNAME=S32A01_01_01  DTYPE=
   C300v1.3  POSITION=10.165.1.32_1_1_2_1_36  SEVERITY=Major  HAPPENTIME
   =2023-06-14 10:30:03.000  ALARMTYPE=QoSAlarm  DESC=ONU ID=37,ONU Name=
   NOS-FTTH_DST_00030550,ONU Type=ZTE-F601,Authentication Value=QlpQXbw086,
   ONU Registration Info: Password = QlpQXbw086,Service Level=EMPRESARIAL
   ENTITY_ID=IP Address:10.165.1.32|4614164  EVENT_CODE=35113
   PROBABLE_CAUSE_DESC=1.Fiber link failure. 2.ONU failure.
   PROBABEL_CAUSE_CODE=351130000  SYSTEM_TYPE=768  ACKSTATUS=Unacknowledged
;

   U31_02 2023-06-14 10:29:45
** 13991.0 REPT
   ALARMID=1674122276731  ALARMDESC=[GPON Alarm]ONU LOS(Loss of Signal)
   (35113)  DID=10.165.1.32  DIP=10.165.1.32  DNAME=S32A01_01_01  DTYPE=
   C300v1.3  POSITION=10.165.1.32_1_1_2_1_20  SEVERITY=Minor  HAPPENTIME
   =2023-06-14 10:30:03.000  ALARMTYPE=QoSAlarm  DESC=ONU ID=21,ONU Name=
   VDF-FTTH_DST_00023791,ONU Type=ZTE-F612C,Authentication Value=pTiaAkN201
   ,ONU Registration Info: Password = pTiaAkN201,Service Level=RESIDENCIAL
    ENTITY_ID=IP Address:10.165.1.32|4613976  EVENT_CODE=35113
   PROBABLE_CAUSE_DESC=1.Fiber link failure. 2.ONU failure.
   PROBABEL_CAUSE_CODE=351130000  SYSTEM_TYPE=768  ACKSTATUS=Unacknowledged
;

   U31_02 2023-06-14 10:28:10
** 13521.0 REPT
   ALARMID=1674122276409  ALARMDESC=[GPON Alarm]ONU LOS(Loss of Signal)
   (35113)  DID=10.165.1.32  DIP=10.165.1.32  DNAME=S32A01_01_01  DTYPE=
   C300v1.3  POSITION=10.165.1.32_1_1_2_1_47  SEVERITY=Minor  HAPPENTIME
   =2023-06-14 10:48:11.000  ALARMTYPE=QoSAlarm  DESC=ONU ID=47,ONU Name=
   NWO-FTTH_DST_00571239,ONU Type=ZTE-F680,Authentication Value=jwatmoq231,
   ONU Registration Info: Password = jwatmoq231,Service Level=RESIDENCIAL
   ENTITY_ID=IP Address:10.165.1.32|4613874  EVENT_CODE=35113
   PROBABLE_CAUSE_DESC=1.Fiber link failure. 2.ONU failure.
   PROBABEL_CAUSE_CODE=351130000  SYSTEM_TYPE=768  ACKSTATUS=Unacknowledged
;

   U31_02 2023-06-14 10:34:03
** 14780.0 REPT
   ALARMID=1674122277378  ALARMDESC=[GPON Alarm]ONU LOS(Loss of Signal)
   (35113)  DID=10.165.1.32  DIP=10.165.1.32  DNAME=S32A01_01_01  DTYPE=
   C300v1.3  POSITION=10.165.1.32_1_1_2_1_25  SEVERITY=Major  HAPPENTIME
   =2023-06-14 10:35:03.000  ALARMTYPE=QoSAlarm  DESC=ONU ID=37,ONU Name=
   NPR-FTTH_DST_00039790,ONU Type=ZTE-F601,Authentication Value=QlpQXbw086,
   ONU Registration Info: Password = QlpQXbw086,Service Level=EMPRESARIAL
   ENTITY_ID=IP Address:10.165.1.32|4614164  EVENT_CODE=35113
   PROBABLE_CAUSE_DESC=1.Fiber link failure. 2.ONU failure.
```

```
   PROBABEL_CAUSE_CODE=351130000  SYSTEM_TYPE=768  ACKSTATUS=Unacknowledged
;


   U31_02 2023-06-14 10:29:45
** 13991.0 REPT
   ALARMID=1674122276731  ALARMDESC=[GPON Alarm]ONU LOS(Loss of Signal)
   (35113)  DID=10.165.1.32  DIP=10.165.1.32  DNAME=S32A01_01_01  DTYPE=
   C300v1.3  POSITION=10.165.1.32_1_1_2_1_21  SEVERITY=Minor  HAPPENTIME
   =2023-06-14 11:09:44.000  ALARMTYPE=QoSAlarm  DESC=ONU ID=21,ONU Name=
   VDF-FTTH_DST_00337216,ONU Type=ZTE-F612C,Authentication Value=pTiaAkN201
   ,ONU Registration Info: Password = pTiaAkN201,Service Level=RESIDENCIAL
    ENTITY_ID=IP Address:10.165.1.32|4613976  EVENT_CODE=35113
   PROBABLE_CAUSE_DESC=1.Fiber link failure. 2.ONU failure.
   PROBABEL_CAUSE_CODE=351130000  SYSTEM_TYPE=768  ACKSTATUS=Unacknowledged
;

   U31_02 2023-06-14 10:34:03
** 14780.0 REPT
   ALARMID=1674122277378  ALARMDESC=[GPON Alarm]ONU LOS(Loss of Signal)
   (35113)  DID=10.165.1.32  DIP=10.165.1.32  DNAME=S32A01_01_01  DTYPE=
   C300v1.3  POSITION=10.165.1.32_1_1_2_1_37  SEVERITY=Major  HAPPENTIME
   =2023-06-14 11:34:03.000  ALARMTYPE=QoSAlarm  DESC=ONU ID=37,ONU Name=
   BLU-EMP-FTTH_DST_00161342,ONU Type=ZTE-F601,Authentication Value=
   QlpQXbw086,ONU Registration Info: Password = QlpQXbw086,Service Level=
   EMPRESARIAL  ENTITY_ID=IP Address:10.165.1.32|4614164  EVENT_CODE=35113
    PROBABLE_CAUSE_DESC=1.Fiber link failure. 2.ONU failure.
   PROBABEL_CAUSE_CODE=351130000  SYSTEM_TYPE=768  ACKSTATUS=Unacknowledged
;

   U31_02 2023-06-14 10:27:40
** 13435.0 REPT
   ALARMID=1674122276351  ALARMDESC=[GPON Alarm]ONU LOS(Loss of Signal)
   (35113)  DID=10.165.1.32  DIP=10.165.1.32  DNAME=S32A01_01_01  DTYPE=
   C300v1.3  POSITION=10.165.1.32_1_1_2_1_51  SEVERITY=Restore  HAPPENTIME
   =2023-06-14 11:40:39.000  ALARMTYPE=QoSAlarm  DESC=ONU ID=51,ONU Name=
   VDF-FTTH_DST_00608799,ONU Type=HUAWEI-HS8247W,Authentication Value=
   CODtrrr240,ONU Registration Info: Password = CODtrrr240,Service Level=
   RESIDENCIAL  ENTITY_ID=IP Address:10.165.1.32|4613851  EVENT_CODE=35113
    PROBABLE_CAUSE_DESC=1.Fiber link failure. 2.ONU failure.
   PROBABEL_CAUSE_CODE=351130000  SYSTEM_TYPE=768  ACKSTATUS=Unacknowledged
;

   U31_02 2023-06-14 10:28:10
** 13521.0 REPT
   ALARMID=1674122276409  ALARMDESC=[GPON Alarm]ONU LOS(Loss of Signal)
   (35113)  DID=10.165.1.32  DIP=10.165.1.32  DNAME=S32A01_01_01  DTYPE=
   C300v1.3  POSITION=10.165.1.32_1_1_2_1_47  SEVERITY=Restore  HAPPENTIME
   =2023-06-14 11:40:55.000  ALARMTYPE=QoSAlarm  DESC=ONU ID=47,ONU Name=
   NWO-FTTH_DST_00571239,ONU Type=ZTE-F680,Authentication Value=jwatmoq231,
   ONU Registration Info: Password = jwatmoq231,Service Level=RESIDENCIAL
   ENTITY_ID=IP Address:10.165.1.32|4613874  EVENT_CODE=35113
   PROBABLE_CAUSE_DESC=1.Fiber link failure. 2.ONU failure.
   PROBABEL_CAUSE_CODE=351130000  SYSTEM_TYPE=768  ACKSTATUS=Unacknowledged
;

   U31_02 2023-06-14 10:29:45
** 13991.0 REPT
```

```
49    ALARMID=1674122276731  ALARMDESC=[GPON Alarm]ONU LOS(Loss of Signal)
      (35113)  DID=10.165.1.32  DIP=10.165.1.32  DNAME=S32A01_01_01  DTYPE=
      C300v1.3  POSITION=10.165.1.32_1_1_2_1_21  SEVERITY=Restore  HAPPENTIME
      =2023-06-14 11:41:44.000  ALARMTYPE=QoSAlarm  DESC=ONU ID=21,ONU Name=
      VDF-FTTH_DST_00337216,ONU Type=ZTE-F612C,Authentication Value=pTiaAkN201
      ,ONU Registration Info: Password = pTiaAkN201,Service Level=RESIDENCIAL
       ENTITY_ID=IP Address:10.165.1.32|4613976  EVENT_CODE=35113
      PROBABLE_CAUSE_DESC=1.Fiber link failure. 2.ONU failure.
      PROBABEL_CAUSE_CODE=351130000  SYSTEM_TYPE=768  ACKSTATUS=Unacknowledged
50 ;
51
52    U31_02 2023-06-14 10:34:03
53 ** 14780.0 REPT
54    ALARMID=1674122277378  ALARMDESC=[GPON Alarm]ONU LOS(Loss of Signal)
      (35113)  DID=10.165.1.32  DIP=10.165.1.32  DNAME=S32A01_01_01  DTYPE=
      C300v1.3  POSITION=10.165.1.32_1_1_2_1_37  SEVERITY=Restore  HAPPENTIME
      =2023-06-14 11:42:03.000  ALARMTYPE=QoSAlarm  DESC=ONU ID=37,ONU Name=
      BLU-EMP-FTTH_DST_00161342,ONU Type=ZTE-F601,Authentication Value=
      QlpQXbw086,ONU Registration Info: Password = QlpQXbw086,Service Level=
      EMPRESARIAL  ENTITY_ID=IP Address:10.165.1.32|4614164  EVENT_CODE=35113
       PROBABLE_CAUSE_DESC=1.Fiber link failure. 2.ONU failure.
      PROBABEL_CAUSE_CODE=351130000  SYSTEM_TYPE=768  ACKSTATUS=Unacknowledged
55 ;
```

Listagem A.27: Terceiro cenário de teste

Apêndice B
Detalhes dos resultados

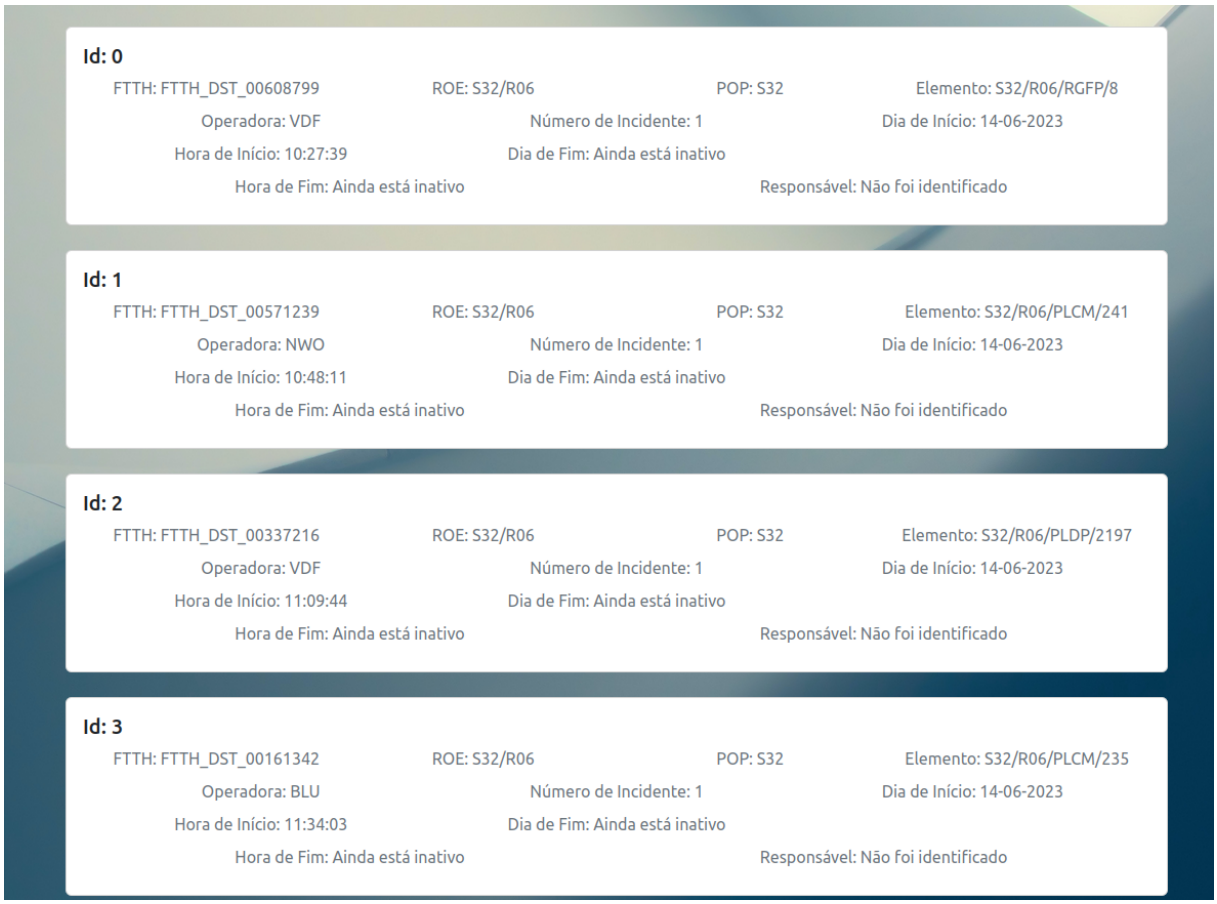


Figura 23: Resultado Auxiliar do Primeiro Cenário de Teste.

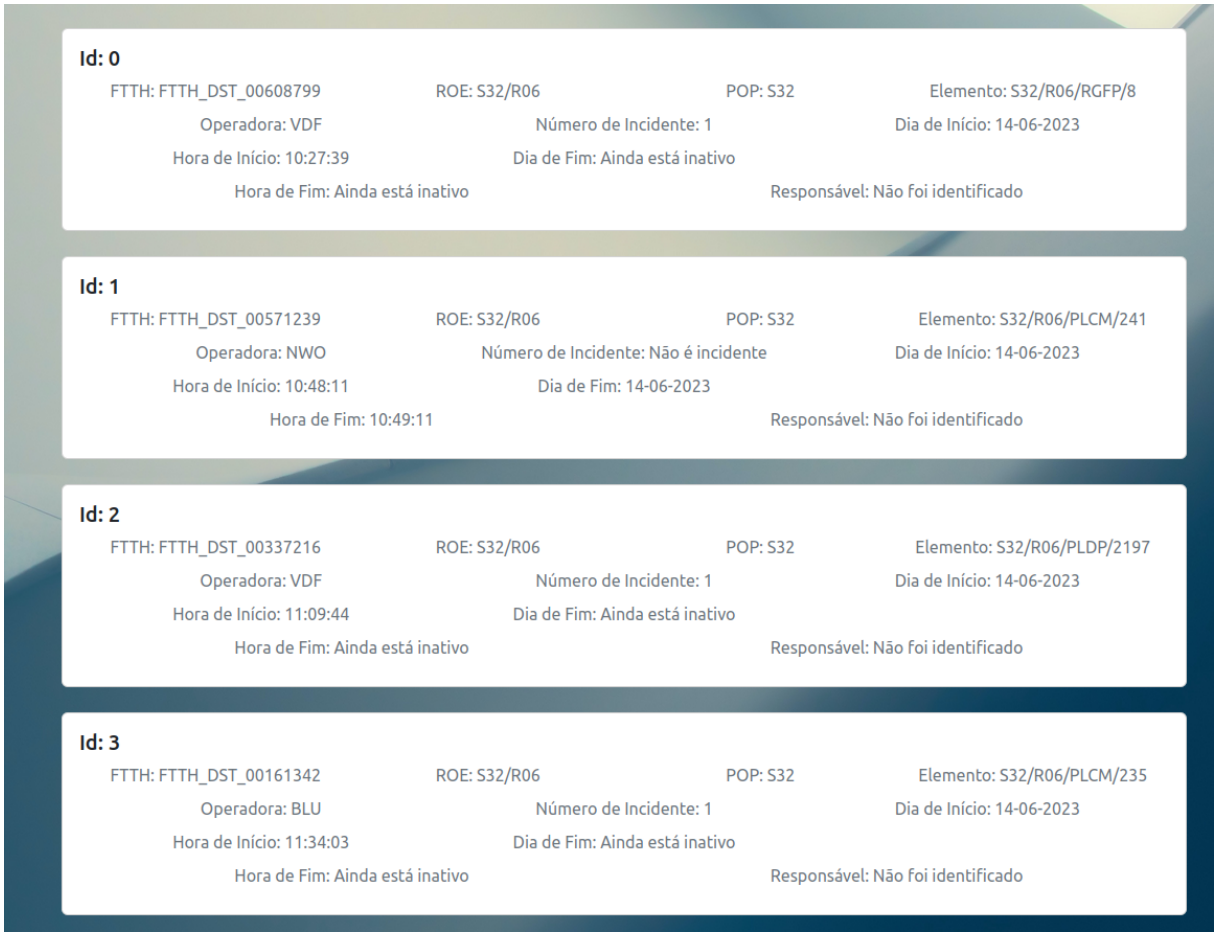


Figura 24: Resultado Auxiliar do Segundo Cenário de Teste.

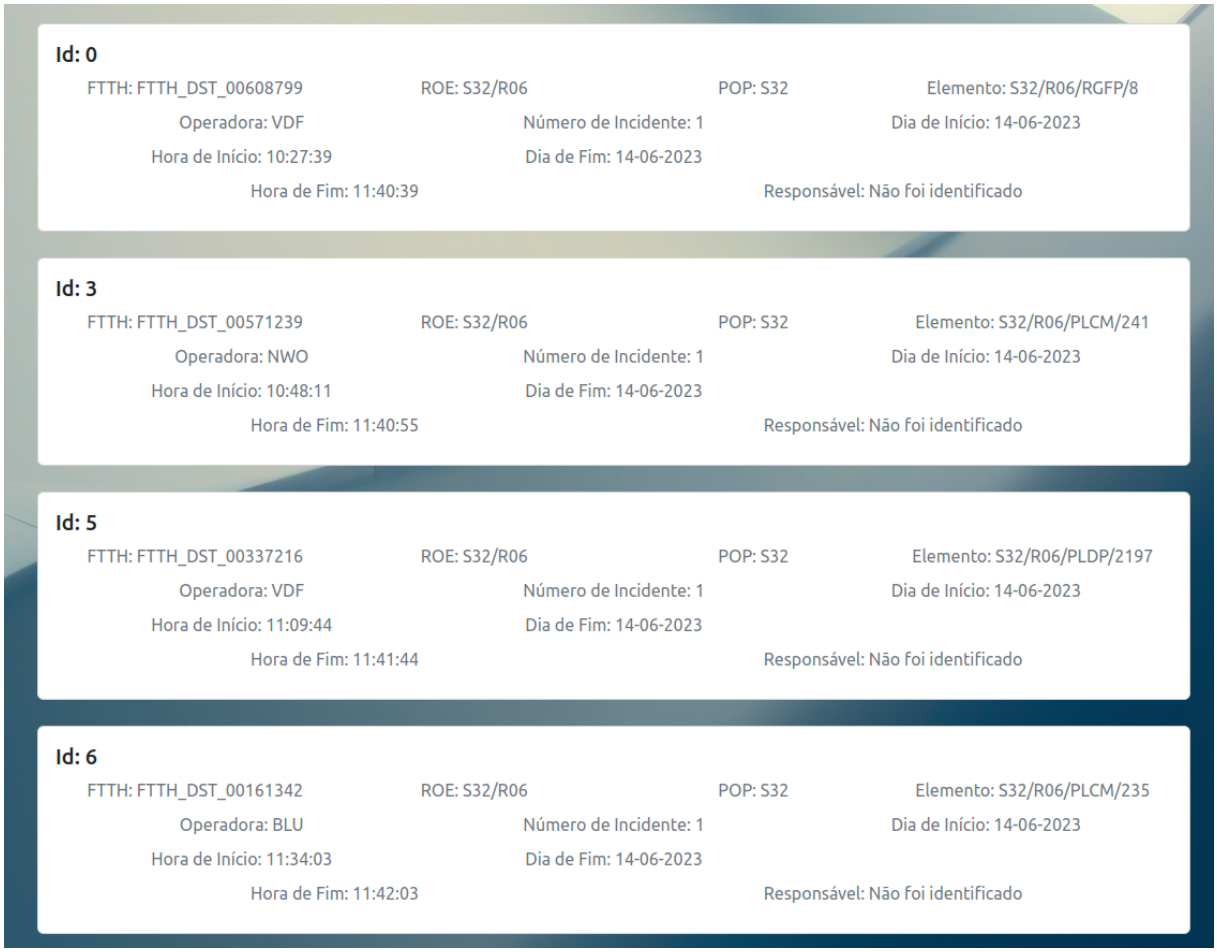


Figura 25: Resultado Auxiliar do Terceiro Cenário de Teste, Incidente N° 1.

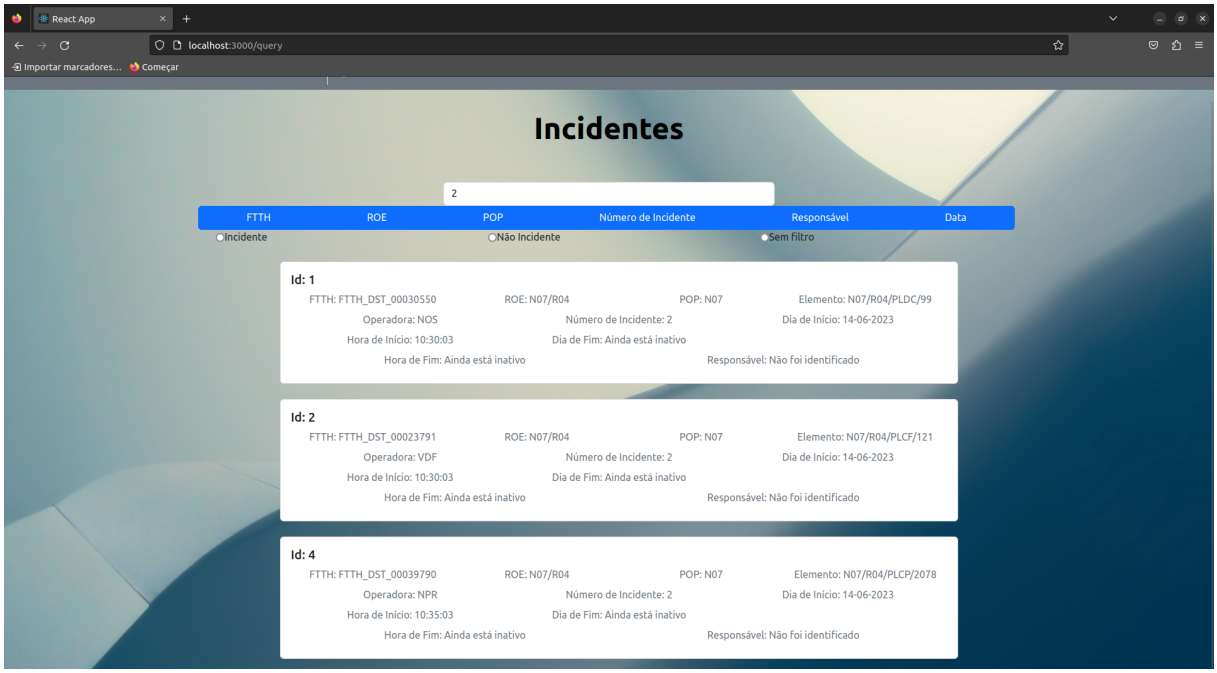


Figura 26: Resultado Auxiliar do Terceiro Cenário de Teste, Incidente N° 2.

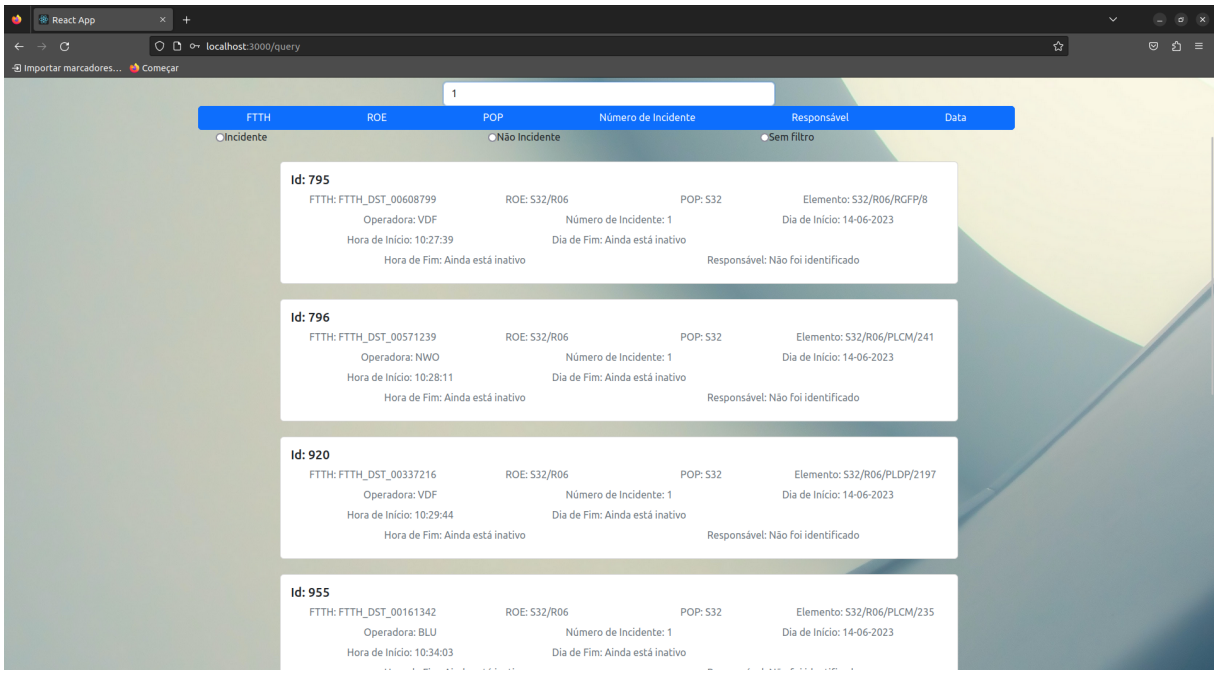


Figura 27: Consulta do Incidente Nº 1, utilizador dstelecom.

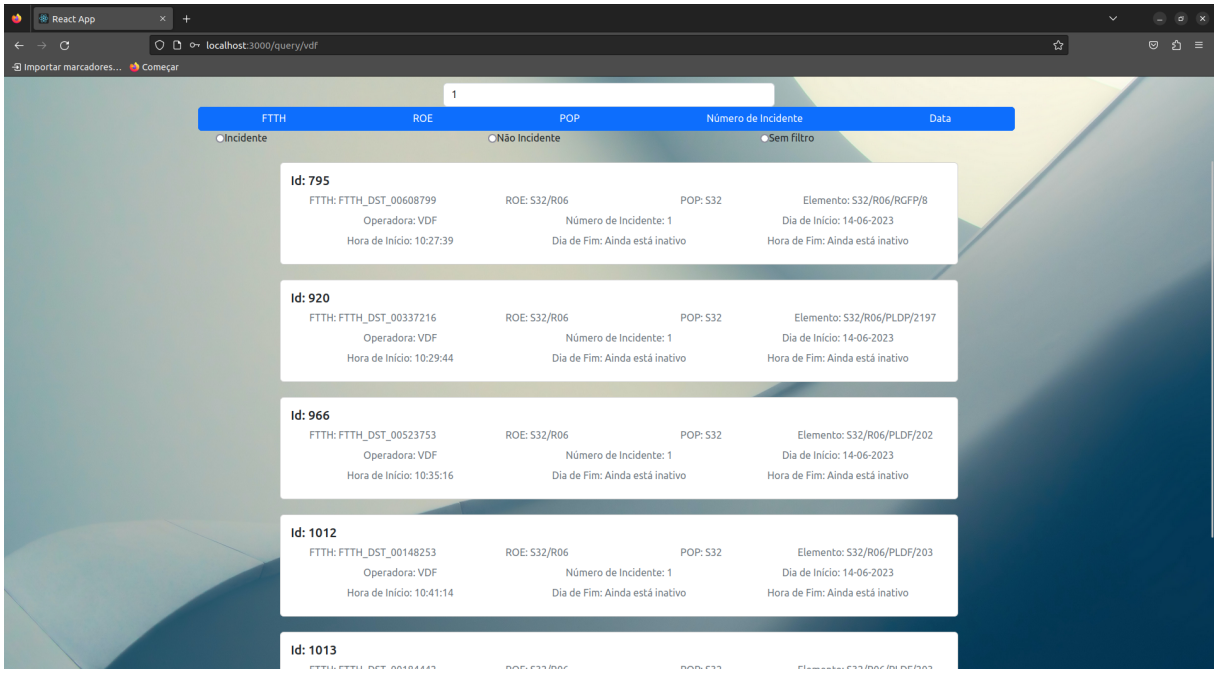


Figura 28: Consulta do Incidente Nº 1, utilizador Vodafone.

Esta dissertação foi realizada com o apoio da dstelecom, que forneceu financiamento através de uma bolsa.