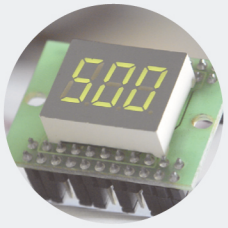
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






Implementação de Garantias de Integridade e de Não Repudiação no Sistema de Gestão de Equipamento do DEE

GONÇALO OSÓRIO MORAIS
Setembro de 2023

POLITÉCNICO
DO PORTO

Politécnico do Porto
Instituto Superior de Engenharia do Porto

Implementação de Garantias de Integridade e de Não Repudiação no Sistema de Gestão de Equipamento do DEE

Gonçalo Osório Morais

Mestrado em Engenharia Electrotécnica e de Computadores
Área de Especialização em Sistemas e Planeamento Industrial

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto

Setembro, 2023

Esta dissertação satisfaz, parcialmente, os requisitos que constam da Ficha de Unidade Curricular de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Electrotécnica e de Computadores, Área de Especialização em Sistemas e Planeamento Industrial.

Candidato: Gonçalo Osório Morais, Nº 1180823, 1180823@isep.ipp.pt

Orientação Científica: Francisco José Dias Pereira, fdp@isep.ipp.pt

Coorientação Científica: Jorge Manuel Estrela da Silva, jes@isep.ipp.pt

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
Rua Dr. António Bernardino de Almeida, 431, 4200–072 Porto

Setembro, 2023

"Science can amuse and fascinate us all, but it is engineering that changes the world."

Isaac Asimov

Agradecimentos

Gostaria de começar por agradecer aos meus orientadores, o professor Francisco José Dias Pereira e o professor Jorge Manuel Estrela da Silva, por toda a disponibilidade e apoio demonstrados desde o início da realização da presente dissertação, bem como ao longo da realização da Bolsa de Iniciação à Investigação realizada durante este ano letivo no DEE.

Aos meus pais e ao meu irmão Nuno por todos os valores que me transmitiram, pela formação, educação e por todo o apoio que me deram ao longo da minha vida.

À minha namorada Catarina, por toda a paciência, incentivo, carinho, apoio e ajuda, principalmente nos momentos mais difíceis.

Ao meu avô Lindolfo, por todas as memórias que construímos e pelos valores que me transmitiu.

À minha tia Luísa, por toda a preocupação e apoio.

Aos meus amigos, por todos os momentos partilhados e entreajuda durante este percurso académico, bem como nos restantes anos.

Ao meu colega e amigo Ângelo Pinheiro, pela colaboração na realização da Bolsa de Iniciação à Investigação.

Resumo

O Departamento de Engenharia Eletrotécnica (DEE) tem vindo a desenvolver uma aplicação informática dedicada à gestão do equipamento do departamento. Uma vez que esta aplicação é desenvolvida e administrada por várias pessoas do próprio departamento, sentiu-se a necessidade de implementar mecanismos que reforcem as garantias de integridade dos dados, assim como de não repudiação das operações efetuadas pelos utilizadores da aplicação.

Assim, nesta dissertação, foram consideradas e analisadas diversas abordagens para alcançar esse fim. Foram estudados e comparados diferentes algoritmos utilizados no processo de assinatura digital; os diferentes tipos de redes *blockchain* existentes e as plataformas que permitem a sua implementação; bem como outras tecnologias emergentes, tais como bases de dados imutáveis, com especial ênfase no caso particular do sistema de gestão de base de dados (SGBD) Immudb.

O desenvolvimento da solução proposta exigiu a análise da aplicação existente no DEE. Nesse processo, foram também adicionadas novas funcionalidades à aplicação, cuja integração estava pendente, tais como um sistema de reporte e acompanhamento da resolução de problemas no equipamento e um sistema de requisição de equipamento. Os resultados dessa tarefa são apresentados neste documento.

Numa primeira abordagem ao problema, foi implementada uma solução baseada no envio de mensagens de correio eletrónico como confirmação das operações, que o utilizador poderá usar como prova da operação efetuada. Contudo, esta solução apresenta várias limitações, nomeadamente a dificuldade de implementação de um esquema de verificação da integridade global dos dados para os gestores do sistema, além de não impedir a eventual repudiação da autoria das operações registadas na base de dados.

A segunda solução implementada baseia-se em dois aspetos chave: na assinatura digital de todos os pedidos de inserção, edição ou eliminação de dados; na replicação de dados do servidor principal, que recorre ao SGBD MariaDB para a gestão da base de dados, para uma base de dados imutável armazenada no SGBD Immudb.

Por fim, foi analisado o impacto destas alterações no desempenho da aplicação.

Palavras-Chave: Bases de dados, imutabilidade, integridade, não repudiação, Immudb, assinatura digital.

Abstract

The Department of Electrical Engineering (DEE) has been developing a computer application dedicated to managing the department's equipment. Since this application is developed and administered by several people in the department, there was a need to implement mechanisms to reinforce guarantees of data integrity and non-repudiation of operations carried out by the users of the application.

Therefore, this dissertation considered and analyzed several approaches to achieve this goal. Different algorithms used in the digital signature process were studied and compared; the different types of existing blockchain networks and the platforms that enable their implementation; as well as other emerging technologies, such as immutable databases, with special emphasis on the particular case of the Immudb database management system (DBMS).

The development of the proposed solution required an analysis of DEE's existing application. In the process, new functionalities were also added to the application, which integration was pending, such as a system for reporting and monitoring the resolution of equipment problems and an equipment requisition system. The results of this task are presented in this document.

In a first stage of the problem, a solution was implemented based on sending e-mails as a confirmation of operations, which the user could use as proof of the operation carried out. However, this solution has some limitations, like the difficutly of implementing a global data integrity verification scheme for system managers, as well as not preventing the possible repudiation of the authorship of such operations recorded in the database.

The second solution implemented is based on two key aspects: the digital signature of all requests to insert, edit or delete data of the database; the replication of data from the main server, which uses the MariaDB DBMS to manage the database, to an immutable database stored in the Immudb DMBS.

Finally, the impact of these changes on the application's performance was analyzed.

Keywords: Database, immutability, integrity, non-repudiation, Immudb, digital signature.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

API	Interface de Programação de Aplicações
BFT	*Byzantine Fault Tolerance*
CA	*Certificate Authority*
CAP	*Consistency, Availability, Partition Tolerance*
CFT	*Crash Fault Tolerance*
CSV	*Comma-separated values*
DCS	Descentralização, Consistência, Escalabilidade
DEE	Departamento de Engenharia Electrotécnica
DoS	*Denial-of-Service*
DSS	*Digital Signature Standard*
ECDSA	*Elliptic Curve Digital Signature Algorithm*
FIPS	*Federal Information Processing Standard*
HMAC	*Hash-based Message Authentication Code*
HTTP	*Hypertext Transfer Protocol*
ID	Identificador
IoT	*Internet of Things*
ISEP	Instituto Superior de Engenharia do Porto
JSON	*JavaScript Object Notation*
JWT	*JSON Web Token*
MD5	*Message Digest 5*
OAS	*OpenAPI Specification*
OSN	*Ordering Service Nodes*

PBFT *Practical Byzantine Fault Tolerance*

PKI *Public Key Infrastructure*

PoS *Proof of Stake*

PoW *Proof of Work*

QLDB *Quantum Ledger Database*

RACE *Research and Development in Advanced Communications Technologies in Europe*

RAML *RESTful API Modeling Language*

REST *Representational State Transfer*

RIPEMD-160 *RACE Integrity Primitives Evaluation Message Digest 160*

RSA *Rivest-Shamir-Adleman*

SDK *Kit* de Desenvolvimento de *Software*

SHA1 *Secure Hash Algorithm 1*

SIGEDEE Sistema de Gestão de Equipamento do DEE

TA *Trust Authority*

URL *Uniform Resource Locator*

W3C *World Wide Web Consortium*

WADL *Web Application Description Language*

Capítulo 1

Introdução

1.1 Contextualização

O trabalho desenvolvido no âmbito desta dissertação está relacionado com os tópicos mais gerais da garantia da integridade de dados e de não repudiação de ações nos sistemas informáticos. Neste trabalho, dá-se especial foco ao problema da garantia de integridade dos dados face a potenciais ações maliciosas ou fraudulentas, sendo abordados alguns pontos comuns ao problema da segurança de dados. Nomeadamente, pretende-se estudar e aplicar abordagens para o desenvolvimento de sistemas de apoio à gestão de uma organização que possam ser administrados por múltiplos utilizadores, com fortes garantias de imunidade relativamente a eventuais tentativas de adulteração dos dados por parte desses mesmos utilizadores. Sublinhe-se que num cenário destes, o conjunto de administradores pode incluir utilizadores com conflito de interesses relativamente aos dados armazenados no sistema. Dessa forma, a construção do sistema deverá garantir que a veracidade e consistência dos dados ficam acima de suspeitas.

O problema geral da garantia de integridade dos dados em sistemas distribuídos tem sido um tópico de investigação ativo ao longo dos tempos, havendo atualmente soluções com um apreciável grau de sucesso para domínios específicos, como é o caso dos protocolos BitTorrent e InterPlanetary File System e da tecnologia *blockchain*.

Como caso de estudo desta dissertação, será usada uma aplicação informática que tem vindo a ser desenvolvida no Departamento de Engenharia Electrotécnica (DEE) do Instituto Superior de Engenharia do Porto (ISEP). Esta aplicação, denominado

de SIGEDEE, é dedicada à gestão do equipamento do DEE, tendo tido uma forte contribuição de vários estudantes do DEE para o seu desenvolvimento, sob orientação da direção do DEE.

O sistema atual está dividido em duas componentes principais: *frontend*, responsável pela interface com o utilizador, e o *backend*, responsável pelo processamento e armazenamento da informação. Os serviços do *backend* estão acessíveis através de uma API do estilo REST. O sistema atual permite a consulta do inventário a qualquer utilizador autenticado. A inserção e edição de registos de equipamentos está restrita a determinados tipos de utilizadores. A aplicação está alojada numa máquina virtual do DEE que se pretende que possa ser administrada por um ou mais técnicos e docentes do DEE.

Esta dissertação surge com a necessidade de melhorar o SIGEDEE, estudando novas tecnologias que possam acrescentar valor a este sistema, avaliando os seus prós e contras.

1.2 Definição do Problema

Como mencionado acima, pretende-se que a máquina onde a aplicação em estudo (o SIGEDEE) está alojada possa ser administrada por diferentes docentes e técnicos do DEE, que, por sua vez, serão também utilizadores da aplicação. Do ponto de vista técnico e hipotético, estes utilizadores têm condições para fazer alterações temporárias a executáveis e configurações do sistema. Têm também a possibilidade de alterar dados das tabelas da base de dados, assim como os seus *logs*, sendo estas ações difíceis de detetar. Assim, uma vez que os utilizadores têm noção que um administrador pode facilmente contornar o mecanismo de autenticação e alterar os dados da base de dados, surge a possibilidade de um utilizador repudiar a autoria de operações registadas na base de dados ou de alegar a ausência de registo de operações previamente efetuadas. Deste modo, decidiu-se alterar o SIGEDEE, tornando-o num sistema onde, comprovadamente, qualquer alteração seja difícil de realizar, ou, caso aconteça, que a mesma seja facilmente detetada.

Apesar de o SIGEDEE conter também informações sobre fornecedores e instalações, o seu foco principal é realizar a gestão do equipamento, como computadores, multímetros, entre outros, sendo considerada fundamental a adição de novas funcionalidades relacionadas com a gestão dos equipamentos do DEE, concentrando-as todas numa única plataforma. Assim, de modo a facilitar o processo de reporte e resolução de avarias, era necessário integrar um sistema de *helpdesk*, que permitisse facilmente reportar quaisquer problemas que surjam nos equipamentos do departamento, assim como acompanhar o processo de resolução dos mesmos. O registo destes processos poderá ser uma ferramenta útil para o processo de avaliação de desempenho dos técnicos, desde que se garanta consenso relativamente à integridade

dos dados entre os utilizadores do sistema. Por outro lado, o SIGEDEE não permitia também acompanhar os processos de requisição de equipamentos realizados por técnicos e docentes do departamento, sendo essa uma funcionalidade adicionalmente pretendida. Neste caso, é importante que o requisitante não possa repudiar o registo de levantamento de um equipamento, assim como o responsável pelo mesmo não possa repudiar o registo da sua devolução. De uma forma geral, é importante que todos os utilizadores possam ser creditados e responsabilizados pelas suas ações no âmbito da utilização do SIGEDEE.

1.2.1 Objetivos

Assim, através da realização desta dissertação pretende-se resolver os problemas anteriormente identificados, introduzindo as seguintes funcionalidades:

- Integração de um sistema de *helpdesk* com o registo de inventário, de modo a ser possível referenciar o equipamento na altura em que é registado um novo *ticket*. Além disso, este sistema deverá ser usado como métrica de avaliação do desempenho dos técnicos, avaliando o tempo de resolução de cada *ticket*, existindo um comprovativo de cada alteração de um *ticket*, desde que é criado, até ser finalizado;
- Registos de requisição de equipamento com o respetivo seguimento do processo (autorização, entrega, devolução);
- Criação de um sistema de transações (de criação ou atualização de registos) que garanta a não-repudiação das operações efetuadas e que evite que os registos armazenados sejam corrompidos, ou um sistema que seja capaz de detetar tal acontecimento;
- Documentação da Interface de Programação de Aplicações (API) para utilização de serviços de *backend*.

1.2.2 Resultados esperados

Assim, numa fase final pretende-se que a plataforma em estudo esteja muito mais evoluída. Isto é, pretende-se obter um sistema de *backend* otimizado, com novas funcionalidades implementadas e com cópias de segurança que garantam a não-repudiação e o registo de todas as transações relevantes (inserções e edições dos registos) na API.

Além disso, é também esperado que o SIGEDEE inclua um sistema de *helpdesk*, que permita reportar avarias de equipamentos e acompanhar a evolução da resolução das mesmas, e um sistema de requisição de equipamentos, que permita aos utilizadores efetuar pedidos de empréstimos de equipamentos para uma instalação do departamento.

1.3 Organização da Dissertação

Esta dissertação está divida em cinco capítulos. No Capítulo 2, é efetuada uma revisão bibliográfica sobre as tecnologias estudadas durante a realização da tese. De seguida, no Capítulo 3, é descrita a aplicação em estudo, bem como as funcionalidades adicionadas à mesma. No Capítulo 4, são descritas as alterações implementadas na estrutura do sistema em estudo, de modo a ser possível atingir os objetivos pretendidos. No Capítulo 5, é analisado o impacto destas alterações no desempenho do *backend*. Por último, são apresentadas as conclusões deste trabalho no Capítulo 6.

Capítulo 2

Revisão Bibliográfica

Ao longo deste capítulo é efetuada uma revisão bibliográfica acerca das tecnologias utilizadas durante a realização da tese em estudo.

2.1 Assinatura Digital

Uma assinatura digital assegura a privacidade, integridade, autenticidade e a não repudiação dos dados utilizando um algoritmo criptográfico, sendo possível verificar se os dados de uma transação foram adulterados [1]. As assinaturas digitais devem ser verificáveis e a sua falsificação não deve ser praticável. Para criar uma assinatura digital, são necessários três parâmetros:

- Algoritmo de geração de chaves - este algoritmo é responsável por criar duas chaves: uma privada, utilizada para assinar as mensagens, e uma pública, à qual se recorre para desencriptar a mensagem e verificar se a mesma não foi corrompida;
- Algoritmo de assinatura - sistema que irá produzir a assinatura digital nos dados enviados, utilizando a chave privada;
- Algoritmo de verificação - recebe a assinatura, a mensagem e a chave pública. Posteriormente, valida a assinatura da mensagem utilizando a chave pública e retorna um valor booleano.

A vantagem de utilizar uma assinatura digital é a capacidade de validar a autenticidade de uma mensagem de forma eficaz, recorrendo a um sistema de chaves

públicas, uma *Public Key Infrastructure* (PKI). Neste método, o remetente da mensagem assina-a com a sua chave privada antes de a enviar e, posteriormente, o recetor será capaz de validar a sua autenticidade, utilizando a chave pública de quem produziu a mensagem. Uma PKI é utilizada para gerir as chaves públicas, estabelecendo uma ligação entre as identidades dos sujeitos (nome, *e-mail*) e as suas chaves públicas. Normalmente, este processo é efetuado através do registo e emissão de certificados, sendo estes da responsabilidade de entidades competentes (*Certificate Authority* (CA)) [2]. Estes certificados possuem a data em que foram criados, bem como a sua data de expiração. Assim, a fiabilidade de uma chave pública pode depender da validade de todos os certificados numa cadeia de certificados. Estes certificados podem também tornar-se inválidos mesmo antes de expirarem, caso, por exemplo, a chave secreta da assinatura digital, correspondente à pública, seja comprometida [1].
Uma assinatura digital pode ser divida em dois processos [3]:

1. Assinatura e Encriptação

    - *Hashing* - num primeiro momento, é calculado um pequeno resumo da mensagem a enviar, sendo uma representação única da mesma, assegurando a sua integridade. A assinatura digital é aplicada ao resumo da mensagem, gerando um código único;
    - Encriptação - nesta etapa, o resumo da mensagem, obtido na fase anterior, é encriptado utilizando a chave privada do remetente. Esta é utilizada para assinar o resumo da mensagem, garantindo, assim, a não repudiação da mesma. O resumo da mensagem original pode ser recuperado utilizando a chave pública correspondente;
    - Agrupar - a mensagem original, a assinatura da mensagem e a chave pública do remetente são agrupadas numa única unidade;

2. Desencriptação e Verificação

    - Desagregar - a mensagem recebida é agora repartida, obtendo a mensagem original, a mensagem assinada e a chave pública do remetente;
    - *Hashing* - nesta fase, a mensagem assinada recebida é introduzida na função de *hash* utilizada pelo remetente para calcular o resumo da mensagem;
    - Desencriptação - posteriormente, o resumo da mensagem recebido é desencriptado utilizando a chave pública do remetente, obtendo-se assim o resumo da mensagem original gerado pelo emissor;
    - Comparação - por último, o resumo da mensagem recebida é comparado com o resumo desencriptado no passo anterior. Se estes forem iguais,

então a mensagem não foi adulterada. Este processo é apenas possível utilizando o par de chaves correspondente.

Utilizada no processo de criação de uma assinatura digital, uma função *hash* é responsável por converter uma mensagem de tamanho variável num valor *hash* de tamanho único. Existem diversos algoritmos para esta finalidade. A Tabela 2.1 resume as características de alguns algoritmos *hash*.

Tabela 2.1: Algoritmos *hash* [3]

Nome do Algoritmo	**Tipo e Características**	**Tamanho da *hash***
Secure Hash Algorithm 1 (SHA1)	Aprovada pela *Federal Information Processing Standard* (FIPS); Outras versões: SHA256, SHA384, SHA512	160 bits; Outras versões: 256 bits, 384 bits, 512 bits, respetivamente
Message Digest 5 (MD5)	Podem ser utilizadas como *Hash-based Message Authentication Code* (HMAC)	128 bits
RACE Integrity Primitives Evaluation Message Digest 160 (RIPEMD-160)	Desenvolvido como parte do *Research and Development in Advanced Communications Technologies in Europe* (RACE)	160 bits
TIGER Hash	Desenvolvido para operações eficientes em plataformas 64-bit	192 bits

Além disso, existem ainda algoritmos de assinatura digital, certificados pela FIPS. A Tabela 2.2 resume as características de alguns algoritmos de assinatura digital.

Tabela 2.2: Algoritmos de assinatura digital [3]

Nome do Algoritmo	**Tipo e Características**	**Tamanho Mínimo da Chave**
Digital Signature Standard (DSS)	Assinatura digital baseada em SHA1	1024 bits
Rivest-Shamir-Adleman (RSA) *Digital Signature*	Assinatura digital previamente registada	1024 bits
Elliptic Curve Digital Signature Algorithm (ECDSA)	Assinatura digitl baseada na chave de curva elíptica, utiliza chaves mais pequenas do que outros algoritmos	160 bits

Um exemplo de um *software* dedicado a encriptação ou assinatura de dados numa a aplicação *web*, é a *Web Cryptography API* [4]. Desenvolvida pelo *World Wide Web Consortium* (W3C) [5], esta é uma API *Javascript* destinada a executar operações criptográficas em aplicações *web*, tais como efetuar o *hashing* de uma mensagem,

gerar e verificar assinaturas digitais, assim como encriptar e desencriptar dados, sendo aceite pela maioria dos *browsers* [6]. Esta API é capaz de armazenar o par de chaves no browser, recorrendo à IndexedDB [7], uma base de dados *key-value* que é executada no *browser*.

2.2 *Blockchain*

Introduzida em 2008 por Satoshi Nakamoto como sendo a base de uma das criptomoedas mais reconhecidas [8], a Bitcoin [9], uma *blockchain* (ou uma rede de blocos em português) é um sistema de base de dados descentralizado onde os registos armazenados são protegidos por criptografia e os nós da rede efetuam transações entre si (*peer-to-peer*) sem a necessidade de existir uma autoridade central [10]. Os registos que esta armazena são imutáveis, ou seja, a probabilidade de um registo ser adulterado é bastante reduzida. Caso tal aconteça, a validade da *blockchain* é destruída. Assim, os registos apenas podem ser atualizados através da inserção de um registo completamente novo. sendo mantido de forma segura um registo de todas as transações efetuadas na rede de forma ordenada (*ledger*), que é replicado por todos os nós da rede [11]. Apenas podem ser adicionados novos blocos à cadeia após ser atingido um consenso. Numa *blockchain*, cada bloco da rede contém o seu *hash* e o do bloco anterior, criando uma ligação encriptada entre os dois blocos [2].

Uma rede *blockchain* possui algumas características chave [12]:

- Descentralização - uma rede *blockchain* permite que as transações sejam validadas entre dois nós sem autenticação, jurisdição ou intervenção de uma entidade central, reduzindo o custo do serviço e eliminando o *bottleneck* do desempenho da rede;
- Imutabilidade - uma *blockchain* consiste numa cadeia de blocos interligados, em que cada ligação contém o valor *hash* do bloco anterior. Assim, qualquer modificação efetuada nos blocos anteriores, irá criar uma inconsistência com todos os blocos posteriormente gerados;
- Não repudiação - a assinatura criptográfica da transação não pode ser negada por quem a inicia;
- Transparência - cada nova transação é validada e armazenada na *blockchain*, estando disponível para todos os nós da rede;
- Pseudónimo - este tipo de sistemas são capazes de garantir um certo nível de privacidade, tornando os endereços anónimos. Contudo, só garante privacidade até um determinado nível, visto que os endereços são facilmente rastreáveis por dedução;

- Rastreamento - cada transação armazenada na rede contém o seu *timestamp* (registado quando a transação ocorre). Desta forma, os utilizadores são capazes de verificar e rastrear as origens dos registos após analisarem os dados da *blockchain* com os *timestamps* correspondentes.

Tal como ilustrado na Figura 2.1, na base da *blockhain*, temos as transações assinadas e efetuadas entre diferentes nós. Estas transações declaram um acordo entre vários participantes, o que pode envolver a transferência de ativos físicos ou digitais, a realização de uma tarefa, etc. Pelo menos um participante assina a transação e esta é partilhada pelos restantes nós. Alguns nós são responsáveis por verificar a validade das transações e se estas devem ou não ser inseridas na rede. No segundo patamar, temos o consenso. Dependendo do tipo de *blockchain*, existem diversos mecanismos de consenso, que serão explorados na Secção 2.2.4. Temos ainda a interface computacional, que permite que a *blockchain* ofereça mais funcionalidades, e a parte administrativa, que adapta a arquitetura da *blockchain* de forma a cobrir as interações humanas no mundo real [13].

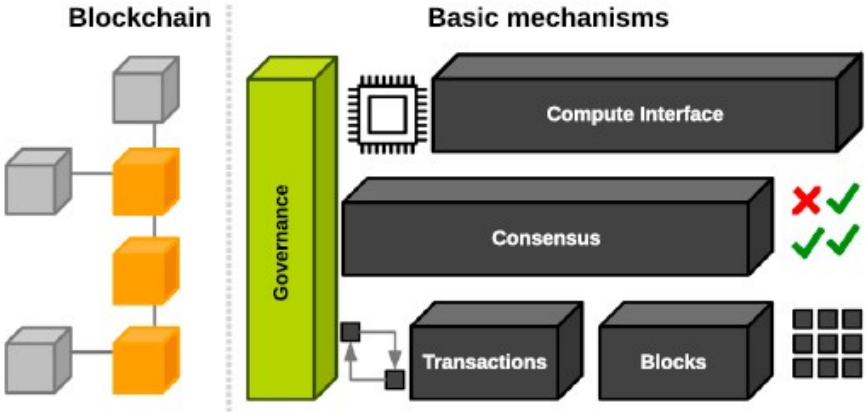


Figura 2.1: Arquitetura da *blockchain* [13]

Alguns autores classificam as aplicações da *blockchain* como financeiras ou não financeiras, visto que uma elevada percentagem das redes *blockchain* são dedicadas ao mercado das criptomoedas. Por outro lado, alguns autores classificam-nas de acordo com as versões da *blockchain*: *blockchain* 1.0, que inclui as aplicações que permitem as transações de criptomoedas; *blockchain* 2.0, onde foram introduzidos os *smart contracts* e um conjunto de aplicações além das criptomoedas e, por último, a *blockchain* 3.0, que abrange diversas áreas além das duas anteriores, como, por exemplo, administração, saúde e *Internet of Things* (IoT) [13]. Assim, a *blockchain* está presente em diversos ramos [13], tais como: aplicações financeiras, administração, IoT [14], gestão de cuidados de saúde [15], gestão das cadeias de abastecimento, gestão de dados e muitas mais aplicações, tal como demonstrado na Figura 2.2.

Figura 2.2: Áreas de aplicação da *blockchain* [13]

O aumento da relevância da *blockchain* é comprovado pelo crescente número de artigos publicados ao longo dos últimos anos, tendo sido publicados apenas 10 artigos no final de 2014, enquanto no ano de 2022 foram publicados mais de 8000 artigos [16], em diversas áreas.

2.2.1 Diferenças entre *blockchain* e bases de dados relacionais

As redes *blockchain* apresentam algumas diferenças relativamente às bases de dados relacionais [17], tais como:

- O rendimento de uma rede *blockchain*, isto é o número de transações efetuadas por segundo, diminui com o aumento da capacidade de processamento dos nós que a constituem;
- Normalmente, a latência das transações numa rede *blockchain* é superior;
- As transações efetuadas numa rede *blockchain* exigem que seja efetuado um controlo através de mecanismos de consenso que sejam eficazes e consistentes. Por vezes, estes processos exigem um elevado poder computacional;
- A maioria das redes *blockchain* não suporta pedidos complexos, como, por exemplo, consulta de dados onde é efetuado o cruzamento entre várias tabelas, o que é possível nas bases de dados relacionais;
- Ao contrário das bases de dados convencionais, a *blockchain* oferece um sistema descentralizado;
- As transações efetuadas numa rede *blockchain* são imutáveis. Isto é possível devido a mecanismos que desabilitam a possibilidade de eliminar ou editar dados existentes.

2.2.2 Como funciona uma *blockchain*

Uma *blockchain* funciona como uma base de dados descentralizada, que apenas aceita inserção de dados e armazena um registo de todas as transações efetuadas na rede de forma imutável, sendo extremamente complicado adulterar os registos. Tal como demonstrado na Figura 2.3, uma rede *blockchain* é constituída por uma cadeia de blocos interligados através dos seus valores *hash* [10].

Cada bloco, com exceção do primeiro (*genesis block*), aponta para o bloco imediatamente anterior (*parent block*) através de uma referência inversa, o valor *hash* do seu *parent block*. Ou seja, o bloco *i* contém o valor *hash* do bloco *i-1* [12].

Existem três fatores importantes a considerar numa rede de blocos: uma estrutura ordenada de blocos interligados, consenso e assinatura digital.

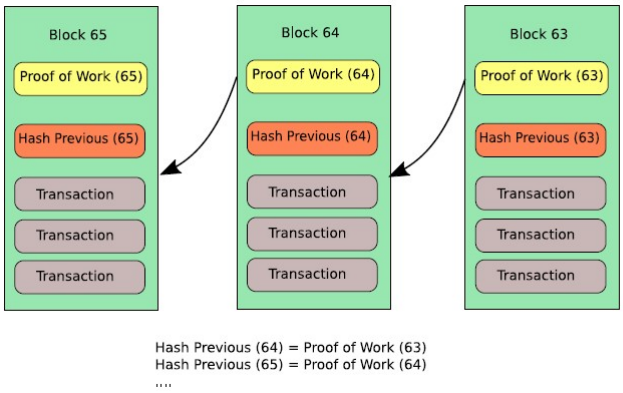


Figura 2.3: Constituição dos blocos de uma *blockchain* [10]

Estrutura ordenada de blocos interligados

Os sistemas *blockchain* utilizam técnicas criptográficas para garantir a integridade dos *ledgers*, ou seja, a capacidade de detetar se os dados da rede foram adulterados. Existem dois níveis de proteção de integridade:

1. A integridade dos registos é protegida por uma árvore de Merkle, que é caraterizada como sendo uma árvore de pesquisa binária com os seus nós interligados através de uma função *hash* [2]. Qualquer alteração de estado provoca a criação de um novo *hash* raíz. Neste caso, as folhas da árvore contêm o estado da rede e os nós internos armazenam os valores *hash* dos seus descendentes;

2. O histórico do bloco é protegido, visto que os blocos da rede são imutáveis. A técnica aqui utilizada é ligar os blocos através de uma cadeia de apontadores *hash*: o bloco *n+1* possui o valor *hash* do bloco *n*. Desta forma, qualquer alteração ao valor do bloco *n* irá invalidar imediatamente todos os blocos seguintes.

Ao combinar a árvore de Merkle com os apontadores *hash*, a *blockchain* oferece um modelo de dados seguro e eficiente que monitoriza todas as alterações efetuadas [11].

Consenso

Numa *blockchain*, quando um novo bloco é submetido na rede, cada nó tem a opção de adicionar esse bloco à sua cópia da rede ou então rejeitá-lo. O consenso numa rede é atingido quando a maioria dos nós aceitam adicionar o bloco, evitando, assim, tentativas desonestas de inserção de blocos ou ataques maliciosos na rede.
De forma a garantir o correto funcionamento da rede, é necessário que haja um algoritmo de consenso eficaz e seguro, que seja tolerante a falhas e garanta que todos os nós possuem em simultâneo uma cadeia de blocos semelhante e que a rede não depende de uma identidade autoritária central para evitar que nós maliciosos

sejam capazes de interferir de forma negativa no processo de atingir um consenso. Ou seja, todas as mensagens transmitidas entre os nós devem ser aprovadas pela maioria dos participantes da rede. Um bom mecanismo de consenso possui duas características importantes [2]:

- Persistência - garante uma resposta consistente do sistema, independentemente do estado de uma transação;
- *Liveness* - todos os nós acabarão por, eventualmente, concordar numa decisão ou valor.

Caso o consenso seja atingido, isto é quando todos os nós da rede, ou uma percentagem deles, validam uma transação, o *miner* inclui a transação validada num bloco que contém informações sobre quando foi aprovado (dia e hora), fazendo com que este se torne rastreável. Este bloco é novamente transferido para a rede e, após ser validado e verificar a ligação com o bloco anterior, será adicionado à rede [10]. A Figura 2.4 ilustra todo este processo. Quaisquer alterações desautorizadas num bloco previamente gerado são facilmente detetáveis, uma vez que o valor *hash* do bloco alterado será significativamente diferente do seu valor original [12].

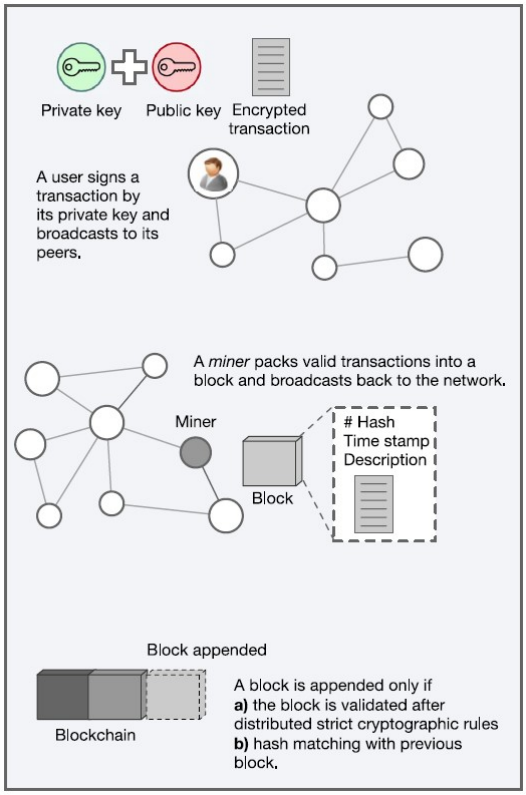


Figura 2.4: Processo de criação de um bloco numa rede *blockchain* [10]

Além destes três fatores, uma rede descentralizada deve respeitar as propriedades do modelo *Consistency, Availability, Partition Tolerance* (CAP). Este modelo defende que um sistema distribuído apenas pode possuir duas das três seguintes propriedades [2]:

- Consistência (*Consistency*) - todos os nós têm o mesmo registo de dados mais atualizado;
- Disponibilidade (*Availability*) - cada pedido recebe sempre uma resposta;
- Partição tolerante a falhas (*Parition tolerance*) - mesmo que alguns nós falhem, a rede continuará operacional.

Uma rede descentralizada tem que possuir sempre a característica de ser tolerante às falhas de partições. Logo, apenas será possível selecionar a disponibilidade ou a consistência da rede. Se se optar pela disponibilidade, não existe garantia de que os dados obtidos para leitura sejam os mais atualizados e formar-se-á um sistema AP. Por outro lado, se for eleita a consistência, será um sistema CP e este estaria indisponível durante a partição de dados, o que poderia dificultar o processo de atingir o consenso [17].
Além do teorema CAP, existe também o teorema Descentralização, Consistência, Escalabilidade (DCS), onde, tal como no anterior, apenas duas propriedades podem ser garantidas em simultâneo [17]:

- Descentralização - não existe uma única identidade de confiança a controlar a rede, não havendo, assim, um único ponto de falha. Neste caso, qualquer nó se pode juntar à rede;
- Consistência - tal como acontece em CAP, os nós da *blockchain* terão acesso aos mesmos dados em simultâneo. Ou seja, dois nós distintos que efetuem o mesmo pedido, deverão receber resultados iguais;
- Escalabilidade - o desempenho da *blockchain* deve aumentar com o aumento do número de nós na rede e, consequentemente, o aumento dos recursos computacionais.

2.2.3 Tipos de *blockchain*

Existem dois tipos principais de redes *blockchain*: públicas (*permissionless*) e privadas (*permissioned*). As redes do primeiro tipo são totalmente descentralizadas e garantem os mesmos privilégios aos nós que constituem a rede. Por outro lado, as redes *permissioned* são parcial ou totalmente descentralizadas e são geridas por determinados nós de confiança (*Trust Authority* (TA)) e são, normalmente, estabelecidas em ambientes privados. Apesar de todos os participantes terem a capacidade

de efetuar transações, apenas um determinado conjunto de nós pode participar no processo de consenso da rede [18].

As redes *permissionless* são públicas, onde qualquer utilizador se pode juntar à rede, ou tornar-se *miner* no caso das criptomoedas, criando, assim, mais desafios para o design do sistema, uma vez que qualquer um dos nós da rede se poderá comportar de forma maliciosa com o intuito de manipular os ativos da rede [14]. Todos os participantes destas redes têm permissão para efetuar transações ou contratos, assim como a capacidade de participarem no processo de consenso, de modo a decidir se um bloco tem, ou não, as condições necessárias para ser validado e posteriormente adicionado à rede [2]. Estas redes são normalmente utilizadas para a mineração de criptomoedas [13].

Por outro lado, as redes *permissioned* dividem-se em dois tipos distintos:

- *Blockchain* privada - enquanto todos os nós da rede podem consultar as transações efetuadas, as permissões de escrita são restritas a apenas uma organização de confiança [2]. As suas principais aplicações são em gestão de bases de dados, auditoria e soluções exigentes no desempenho [13];

- *Blockchain* de consórcio (*Consortium Blockchain*) - nestas redes, qualquer nó pode visualizar as transações efetuadas. Contudo, existem restrições nas permissões de escrita. Por exemplo, apenas um grupo predefinido de nós da rede pode influenciar ou controlar o processo de se atingir o consenso [2]. Assim, é possível afirmar que a *blockchain* de consórcio é uma combinação das redes públicas e privadas, uma vez que partilha o nível de escalabilidade e privacidade das redes privadas, mas, neste caso, um conjunto de nós é selecionado como líderes para verificar as transações na rede, ao invés do que acontece nas redes privadas, onde apenas uma única identidade realiza esta ação. Estas características geram então um ambiente parcialmente descentralizado. Este tipo de redes é utilizada nos setores bancário e industrial [13].

A Tabela 2.3 resume as características dos diferentes tipos de *blockchain* existentes.

Assim, geralmente, as redes *blockhain* apresentam várias diferenças entre si que se devem ter em consideração [12]:

- As redes públicas são totalmente descentralizadas, enquanto as redes privadas ou de consórcio são parcialmente descentralizadas ou controladas totalmente por uma ou várias organizações;

- É praticamente impossível adulterar as transações efetuadas numa rede pública, visto que todos os nós possuem uma cópia da *blockchain*. Por outro lado, as organizações dominantes nas redes *permissioned* têm a capacidade de modificar a rede, uma vez que a controlam;

Tabela 2.3: Classificação de *blockchains* [2]

Tipos	**Descrição**	**Nº de TA**	**Velocidade de consenso**	Cenários
Blockchain Pública	Qualquer pessoa se pode juntar a esta rede e é mundialmente acessível	0	Lenta	Cenários globalmente descentralizados
Blockchain Privada	Direitos de escrita são restritos a uma organização	1	Rápida	Partilha e gestão de informação numa organização
Blockchain de Consórcio	Controlada por nós predefinidos na rede	≥ 1	Ligeiramente rápida	Negócios entre determinadas organizações

- As redes *permissionless* são capazes de garantir totalmente a não repudiação, a transparência e a rastreabilidade das transações, enquanto as restantes redes apenas o conseguem fazer parcialmente, ou no pior cenário não são capazes de o fazer;
- As redes privadas ou de consórcio conseguem ter uma maior escalabilidade do que as redes públicas, uma vez que são controladas por uma única ou várias organizações, o que facilita o processo de atingir um consenso;
- As redes públicas apresentam uma menor flexibilidade, visto que as configurações das redes *permissioned* são mais ajustáveis.

A Tabela 2.4 resume as características dos diferentes tipos de *blockchain* existentes.

Tabela 2.4: Comparação dos diferentes tipos de redes *blockchain* [12]

Características	**Pública**	**Privada**	**Consórcio**
Descentralização	Descentralizada	Centralizada	Parcialmente descentralizada
Imutabilidade	Imutável	Alterável	Parcialmente imutável
Não repudiação	Não recusável	Recusável	Parcialmente recusável
Transparência	Transparente	Opaca	Parcialmente transparente
Escalabilidade	Fraca	Superior	Boa
Flexibilidade	Fraca	Superior	Boa
Permissão	*Permissioned*	*Permissionless*	*Permissionless*
Exemplos	Bitcoin, Ethereum	GemOS, Multichain	Hyperledger, Ethereum

2.2.4 Mecanismos de consenso

Tal como referido anteriormente, os nós de uma rede *blockchain* participam num mecanismo para chegar a um consenso, decidindo se certos dados devem ou não ser inseridos na rede.

Assim, entende-se por consenso o procedimento efetuado para atingir um acordo entre as diferentes máquinas num sistema descentralizado. O algoritmo de consenso é aplicado para garantir a fiabilidade do sistema, garantindo o correto funcionamento do mesmo quando esteja na presença de elementos maliciosos ou num sistema com tolerância ao erro. Estes algoritmos são baseados num sistema de partilha de mensagens ou de memória.

Atingir o consenso pode ser um processo fácil e direto em determinados cenários, como, por exemplo, quando todo o sistema é irrepreensível (*faultless*) ou não existe nenhuma falha no mesmo, garantindo que todas as entidades irão receber as mensagens de forma correta. Tal acontece também quando o sistema se comporta de forma síncrona, ou seja, espera-se que as mensagens sejam todas recebidas num determinado período previamente definido [18].

Assim, os mecanismos de consenso são caracterizados por:

- Segurança - este parâmetro garante que nada de errado acontecerá. Geralmente, um algoritmo de consenso é seguro quando pelo menos um nó confiável produz um resultado válido, levando os restantes nós a produzir ou receber o mesmo resultado;

- *Liveliness* - garante que irá, eventualmente, acontecer algo positivo. A *liveliness* é garantida se todos os nós benignos que participem neste processo produzam um valor e todos os pedidos corretos serão, eventualmente, processados;

- Tolerância a falhas - um algoritmo de consenso garante tolerância a falhas se for resiliente às falhas de alguns nós que participam no consenso em qualquer momento. Enquanto os nós maliciosos forem limitados, é possível finalizar este processo de forma correta [19]. Tipicamente, num sistema distribuído são considerados três tipos de falhas [18]:

  - Falha de um nó (*Node Failure*) - um dos nós da rede deixa de funcionar subitamente ou fica indisponível durante a comunicação. Assim, não se espera receber qualquer tipo de mensagens desse nó. Este problema pode ser causado por falhas de *hardware* ou de *software*;

  - Falhas de partição (*Partitioned Faults*) - esta falha surge quando é quebrada uma ligação na rede, resultando numa partição da mesma. Este erro pode dificultar todo o processo de consenso;

– Falhas Bizantinas (*Byzantine Faults*) - este é o tipo de erro mais difícil de controlar num ambiente descentralizado. Neste tipo de falhas, uma identidade começa a ter um comportamento malicioso na rede, fazendo com que os restantes nós não atinjam um consenso. Enquanto nas duas falhas anteriores é possível determinar os efeitos que as mesmas terão na rede, neste caso é mais difícil de prever, uma vez que depende no quão maliciosa a identidade é e as ações que tentará realizar. Por outro lado, numa falha bizantina um nó também pode surgir como ativo e inativo de forma inconsistente, fazendo com que os restantes nós tenham dificuldade em considerá-lo inativo para o processo de consenso.

Qualquer algoritmo de consenso em ambientes assíncronos deve sacrificar uma das três propriedades dependendo das suas necessidades. Uma vez que a tolerância a falhas é essencial nas redes *blockchain*, as aplicações tendem a sacrificar ou a sua segurança ou a sua *liveness* [19].

Existem diversos mecanismos de consenso para os diferentes tipos de *blockchain*.

Mecanismos de consenso numa *blockchain permissionless*

As redes *blockchain* estão sujeitas a ataques Sybil e *Denial-of-Service* (DoS). Nos ataques Sybil, um utilizador mal intencionado tenta preencher a rede com clientes controlados por si, o que o irá ajudar a controlar toda a rede, comprometendo o algoritmo. Por outro lado, nos ataques DoS, um nó malicioso irá enviar dados para um ou mais nós, atrapalhando o normal funcionamento das transações na rede.

De modo a dificultar este tipo de ataques, foram desenvolvidos os seguintes mecanismos de consenso, geralmente utilizados em redes públicas:

1. *Proof of Work* (PoW)
   O algoritmo PoW foi desenvolvido em 1992 por Dwork e Naor com o intuito de combater os *e-mails* de spam [20]. Os sistemas *blockchain* baseados em PoW devem ter algumas características, tais como:

   - Assimetria - a tarefa deverá ser relativamente complicada, mas viável para quem solicita o serviço;
   - A tarefa deve ser fácil de verificar pelo prestador de serviços. Assim, o cliente será desencorajado a adulterar o trabalho, enquanto o prestador consegue facilmente verificar a veracidade do trabalho, devido à assimetria do mesmo [18].

   Num sistema com algoritmo PoW, quanto mais mensageiros existirem, menor será a probabilidade de todos eles serem maliciosos. Assim, mesmo que existam

nós maliciosos, o tempo necessário para adulterar os dados aumentará substancialmente. Apesar deste protocolo ser eficaz na resolução dos problemas bizantinos, possui três limitações:

- O protocolo é extremamente ineficiente, uma vez que o processo de obtenção de um resultado válido (*proof of work*) é muito exigente a nível computacional e apresenta uma baixa probabilidade de sucesso;
- A sua segurança é oriunda essencialmente das recompensas da mineração (*mining*), que são incentivos para atrair *miners* para a rede;
- Os participantes da rede podem possuir diferentes capacidades computacionais, originando diferentes probabilidades de sucesso para cada nó [2].

No caso da Bitcoin, os *miners* que participem no processo de consenso necessitam de apresentar provas de que efetuaram o trabalho antes de proporem um novo bloco na rede. A probabilidade de obter um PoW é bastante reduzida. Deste modo, nenhum *miner* será capaz de controlar a rede exclusivamente. Caso alguém pretenda adulterar as informações presentes num bloco, terá que efetuar mais trabalho do que o trabalho coletivo de todos os blocos presentes na cadeia mais longa, o que é computacionalmente custoso. Assim, o sistema de Bitcoin é à prova de adulteração e é complicado atacar o mesmo.

Outro grande problema dos sistemas que utilizam este algoritmo é a possibilidade de surgir um monopólio. Tal acontece quando um determinado *miner* possui servidores com elevado processamento de mineração, ganhando assim o controlo da rede. Deste modo, este será capaz de gerar um grande número de blocos, acabando por controlar todo o fluxo de transações daquela rede. Assim, os restantes nós serão desencorajados a utilizar esta rede, sendo controlada por apenas alguns *miners* com elevadas capacidades de processamento [18]

2. *Proof of Stake* (PoS)
   O mecanismo PoS é uma versão melhorada do algoritmo PoW, com o intuito de reduzir o seu consumo energético. Tal como no algoritmo anterior, um *miner* é selecionado para criar um bloco e adicioná-lo à *blockchain*. Contudo, em PoS, a seleção deste nó é efetuada com base na quantidade de ativos que cada nó possui. Quanto mais ativos um nó possuir, maior a probabilidade de ser selecionado. O *miner* é selecionado de forma aleatória, para que nenhum nó seja capaz de prever a sua vez, resolvendo o problema de monopólio presente nos sistemas PoW. Estes sistemas são mais rápidos em comparação com os sistemas PoW, uma vez que neste caso não é necessário resolver problemas complexos a nível computacional [18].

Mecanismos de consenso numa *blockchain permissioned*

Numa rede *blockchain permissioned*, por vezes recorre-se à utilização de *smart contracts* para garantir o consenso. A implementação destes não é obrigatória, mas é ideal quando várias organizações estão a trabalhar na mesma rede, e não confiam umas nas outras. Esta tecnologia foi introduzida em plataformas como Ethereum e Hyperledger Fabric.

Os *smart contracts* são contratos celebrados entre os elementos participantes e são implementados no topo da *blockchain*, sendo as suas cláusulas são convertidas em programas computacionais executáveis. Os *inputs*, *outputs* e estados afetados pela execução do *smart contract* são acordados por cada nó [11]. A execução de cada declaração do contrato é registada como uma transação imutável que será armazenada na rede. Estes garantem um controlo de acesso adequado e a aplicação do contrato. Os desenvolvedores de *smart contracts* têm a possibilidade de atribuir diferentes permissões de acesso a cada função do contrato. Assim que estejam satisfeitas determinadas condições do contrato, a respetiva função será executada. O ciclo de vida dos *smart contracts* consistem em quatro fases, tal como ilustrado na Figura 2.5:

1. Criação do *smart contract*
   Primeiramente, as partes interessadas discutem os parâmetros do contrato (obrigações, direitos e proibições). Assim que se chegar a um acordo, o contrato será então convertido num *smart contract*, sendo escrito em linguagem computacional, incluindo linguagem declarativa e lógica. O processo de desenvolvimento do *smart contract* envolve o seu desenvolvimento, implementação e, por último, uma fase de validação do mesmo, através da realização de alguns testes;

2. Implementação do *smart contract*
   Os *smart contracts* validados podem então ser implementados no topo da rede. Os contratos são armazenados na *blockchain* e não podem ser alterados devido à imutabilidade da mesma. Qualquer alteração que se pretenda efetuar, implica a criação de um novo contrato. Assim que os *smart contracts* estiverem implementados na rede, todos os nós terão acesso a estes;

3. Execução do *smart contract*
   Uma vez que sejam atingidas as condições descritas no contrato, os procedimentos contratuais serão automaticamente executados. Visto que um *smart contract* consiste num conjunto de declarações com ligações lógicas, quando uma condição é válida, a declaração correspondente será automaticamente realizada, fazendo, assim, com que seja ordenada uma nova transação e validada pelos *miners* da rede;

4. Conclusão do *smart contract*
   Após o *smart contract* ser concluído, as transações efetuadas durante a sua execução são armazenadas na *blockchain*, bem como os estados atualizados de todos os participantes do contrato, terminando assim o seu ciclo [12].

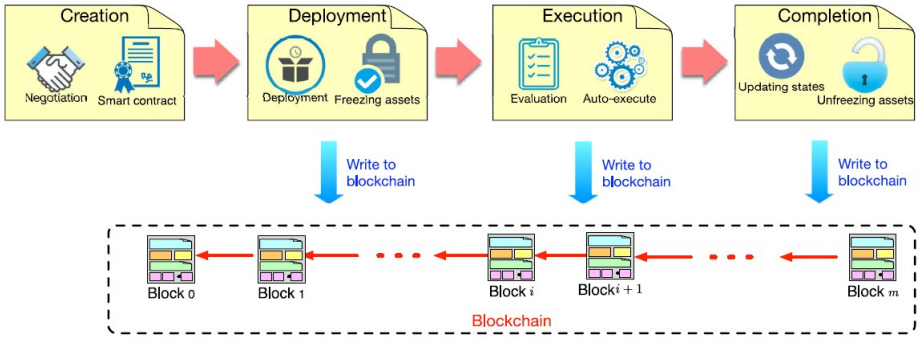


Figura 2.5: Ciclo de vida de um *smart contract* [12]

De forma a atingir o consenso, é utilizado o mecanismo de replicação, uma vez que se trata de uma rede fechada e todos os nós são conhecidos, evitando a sobrecarga do *mining* [18]. De seguida, serão apresentados alguns mecanismos de consenso aplicados em rede de *blockchain* privadas:

1. Paxos
   Proposto por Lamport em 1998 [21], Paxos é um algoritmo de consenso desenvolvido com o objetivo de escolher apenas um valor sob erro ou falha da rede. Os nós são divididos em três categorias tal como demonstra a Figura 2.6: *proposers*, *acceptors* e *learners*. Nestas redes, todos os nós são *learners* que recebem o resultado do consenso.

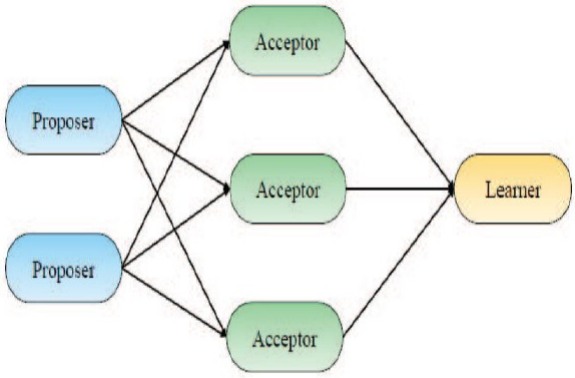


Figura 2.6: Tipos de nós em Paxos [18]

Este protocolo funciona da seguinte forma:

- Primeiramente, o *proposer* prepara uma proposta, atribuindo-lhe um número (número da proposta) e envia-a para os *acceptors*. O número da proposta segue uma linha cronológica e o maior número é considerado o mais recente;
- De seguida, cada *acceptor* compara o número da proposta recebida no ponto anterior com os valores recebidos previamente de todas as propostas enviadas pelo *proposer*. Se o número recebido for superior aos restantes, a proposta será aceite. Caso contrário, a mesma será rejeitada. Assim, o *acceptor* prepara uma mensagem de resposta a informar se aceita ou não a proposta efetuada e enviando o número da proposta, que é o valor mais elevado recebido pelo *acceptor*, e os valores aceites, que são os valores já aceites provenientes de outro *proposer*;
- Posteriormente, é efetuada uma votação baseada numa decisão maioritária. O *proposer* verifica se a maioria dos *acceptors* rejeitou a proposta. Caso isto aconteça, o *proposer* atualiza a proposta recebida inicalmente com o número da proposta mais recente. Caso contrário, o *proposer* verifica ainda se a maioria dos *acceptors* já possui valores aceites de outro *proposer*. Se tal acontecer, o valor do *proposer* não poderá ser selecionado;
- Finalmente, no caso da proposta ser válida, o *proposer* envia uma mensagem a todos os *acceptors* com o número da proposta e um valor proposto por si. Assim que o *acceptor* aceitem um valor, informam os *learner nodes*, de modo a que todos os nós tenham conhecimento do valor aceite.

Se mais do que (N/2 - 1) *acceptors* falharem, então nenhum *proposer* receberá respostas suficientes para formar uma conclusão, não sendo possível chegar a um consenso. Além disso, quantos mais nós existirem na rede, mais mensagens serão trocadas ao longo da rede (Multi-Paxos) [18].

2. *Practical Byzantine Fault Tolerance* (PBFT)
   Apesar de ser considerado uma propriedade e não um mecanismo, o *Byzantine Fault Tolerance* (BFT) é algo relevante nas *blockchain*. O BFT é a capacidade de uma rede descentralizada atingir o consenso da forma pretendida e correta, apesar da existência de nós maliciosos. Este é baseado no problema dos generais bizantinos, onde um General envia uma mensagem de ataque para um grupo de Tenentes e uma mensagem de retirada para outro grupo de Tenentes (Figura 2.7), dificultando a tomada de decisão. Também existe a possibilidade de o General enviar a mesma mensagem para os dois grupos de Tenentes, e um desses grupos ser malicioso, adulterando a mensagem recebida (Figura 2.8) [22].

   O modelo BFT assume os seguintes parâmetros:

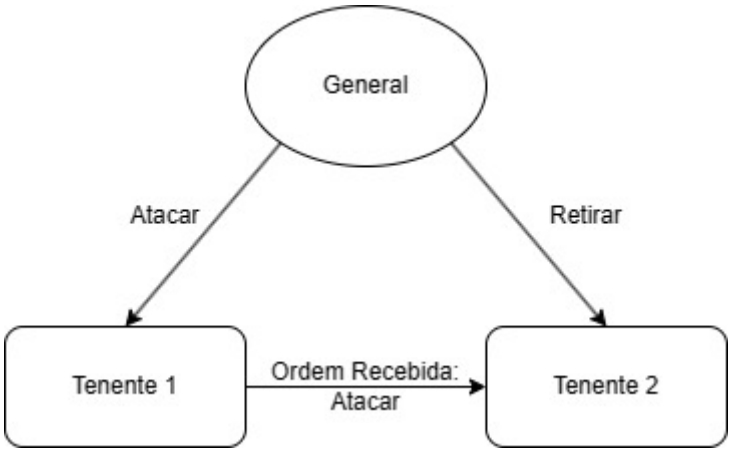


Figura 2.7: General como traidor (adaptado de [22])

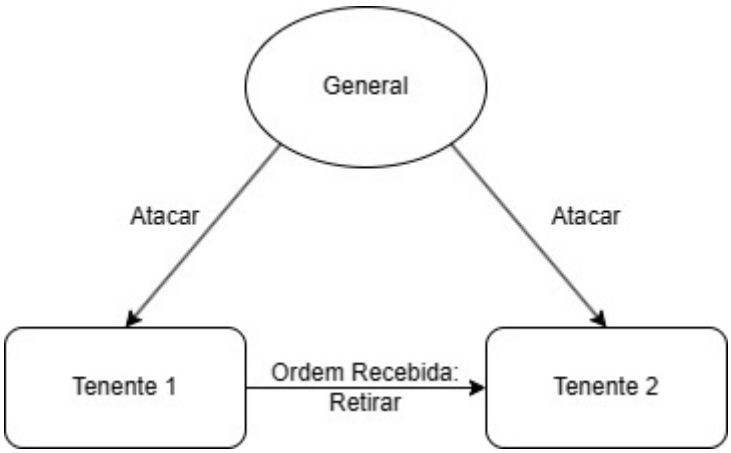


Figura 2.8: Tenente 1 como traidor (adaptado de [22])

- Total número de nós N com um máximo de f nós maliciosos;
- Ambiente fechado, ou seja, o recetor sabe sempre a identidade do emissor;
- Rede totalmente conectada;
- Meio de comunicação fiável;
- Sistema síncrono.

Os sistemas BFT são tipicamente desenvolvidos utilizando replicação, garantindo consenso na presença de f nós maliciosos, quando existem *2f+1* tenentes além do general [18].

Contudo, como os sistemas reais se comportam de forma assíncrona, não é garantido que a mensagem será recebida num determinado período. Assim, foi desenvolvido, para sistemas assíncronos, o algoritmo PBFT [23]. O modelo bizantino possui três tipos de nós - os clientes, um líder (General) e os seus seguidores (os Tenentes) - e é executado em cinco fases distintas (Figura 2.9):

- Fase de pré-preparação - Após um cliente efetuar um pedido, o general atribuiu-lhe uma sequencia de números e distribui-o pela rede. A mensagem contém a assinatura digital e o resumo da mensagem para verificação. Os tenentes confirmam a validade do bloco, analisando a assinatura digital e o resumo da mensagem do mesmo;
- Assim que os tenentes aceitem a pré-preparação da mensagem, esta entra na fase de preparação, sendo distribuída pelo resto da rede. Uma vez mais, o general e os tenentes verificam novamente a mensagem de preparação antes de a aceitarem;
- As mensagens são confirmadas quando *2f* mensagens de preparação de diferentes cópias de segurança coincidem com as mensagens de pré-preparação correspondentes. Assim, o total de *2f + 1* votos (um da réplica primária) da réplica não defeituosa ajuda o sistema a chegar a um consenso.

Num sistema que utilize PBFT, são necessárias *3f + 1* elementos para se atingir um consenso. Este algoritmo providencia um elevado rendimento, baixa latência e é mais eficiente que o protocolo PoW, uma vez que consome menos energia a verificar as transações. Além disso, o modelo PBFT garante privacidade, assegurando um sistema de transmissão de mensagens inviolável. Aqui, a autenticidade dos nós é verificada através de assinaturas digitais.
Apesar de todas as suas vantagens, o algoritmo PBFT apresenta algumas limitações. A sobrecarga provocada pela constante difusão de mensagens na rede dificulta a gestão da rede para além de um determinado número de réplicas. Além disso, este protocolo é suscetível de sofrer ataques Sybil.

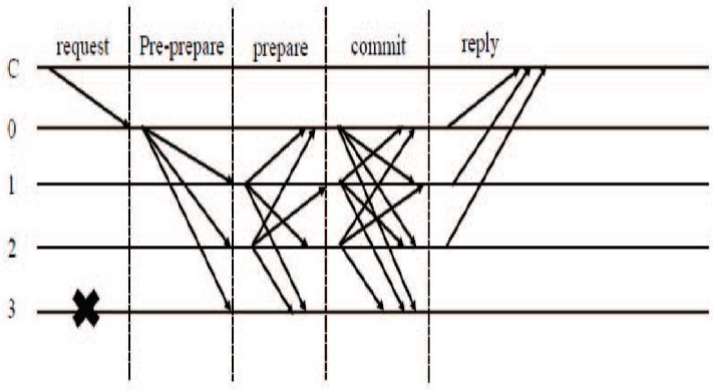


Figura 2.9: Fases de execução do algoritmo PBFT

Este protocolo é utilizado por diversas redes *blockchain* privadas, tais como o Hyperledger Fabric [18].

3. Raft
   O algoritmo Raft foi desenvolvido com o intuito de ser uma melhor alternativa ao protocolo Paxos. Neste, os nós escolhem em conjunto um líder e os restantes tornam-se seguidores.
   O líder é responsável pela replicação das transações para os seguidores. Deste modo, os registos fluem apenas num sentido, desde o líder até aos seguidores. Ao contrário do que acontece no protocolo Paxos, cada nó pode assumir um de três estados (líder, candidato e seguidor) a qualquer momento, tal como demonstra a Figura 2.10. De forma a manter a sua autoridade, o líder envia sinais de forma periódica para os seguidores [18].

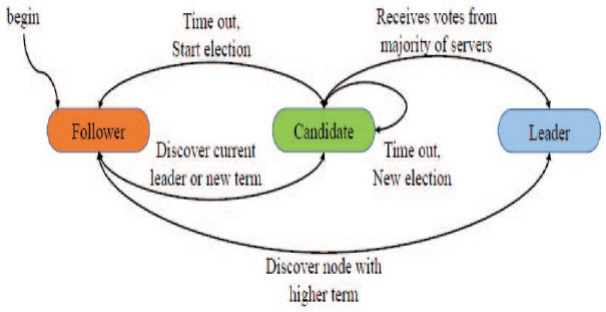


Figura 2.10: Modelo de alteração de estado no algoritmo Raft [18]

O algoritmo Raft segue a seguinte lógica:

- Numa primeira fase, existe um conjunto de nós seguidores que procuram um líder. Se ao fim de um determinado intervalo de tempo não encontrarem nenhum, é iniciado o processo de eleição de um novo líder. Nesta

fase, alguns dos seguidores voluntariam-se para se tornarem líderes. Posteriormente, os nós candidatos enviam uma mensagem para os seguidores do sistema a pedir o seu voto: *Request_vote: (term,index)*, onde *term* é igual ao último número conhecido a candidatar se + 1 e *index* corresponde à transação confirmada disponível para o candidato;

- Quando um nó recebe o pedido, compara o *term* e o *index* recebidos na mensagem com os valores correspondentes já conhecidos. Posteriormente, tal como em Paxos, os seguidores votam num dos candidatos e o líder selecionado será aquele que receber a maioria dos votos. A Figura 2.11 ilustra o processo de eleição de um líder no algoritmo Raft;

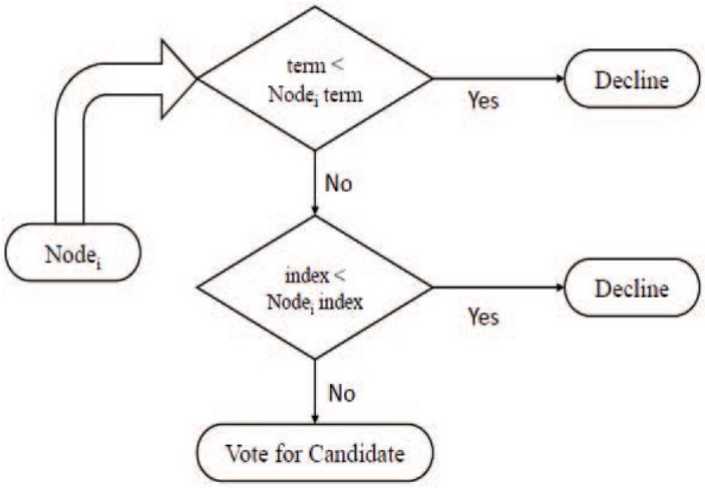


Figura 2.11: Processo de eleição de um líder no algoritmo Raft [18]

- Finalmente, o líder eleito irá propor valores que serão escolhidos pelos seguidores de forma a atingir o consenso na rede.

O algoritmo Raft apresenta uma maior eficiência e clareza, em comparação com os protocolos Paxos e PBFT . Este algoritmo não suporta nós bizantinos e aguenta uma taxa de erro de até 50% dos nós [18].

A Tabela 2.5 e a Tabela 2.6 comparam os mecanismos de consenso apresentados, com base nas suas características e no seu desempenho, respetivamente.

2.2.5 Plataformas *blockchain permissioned*

Atualmente, existem diversas plataformas *blockchain*, sendo algumas das mais conhecidas a Bitcoin e a Ethereum [24]. Contudo, nesta dissertação apenas serão abordadas algumas plataformas destinadas a redes *blockchain permissioned*: Ethereum,

Tabela 2.5: Comparação das características dos mecanismos de consenso apresentados [18]

Características	**PoW**	**PoS**	**Paxos**	**BFT e suas variantes**	**Raft**
Modelo de Confiança	Não Confiável	Não Confiável	Semi Confiável	Semi Confiável	Semi Confiável
Tipo de *Blockchain*	*Permissionless*	Ambos	*Permissioned*	*Permissioned*	*Permissioned*
Grau de Descentralização	Elevado (Completamente Descentralizada)	Elevado	Reduzido	Reduzido	Reduzido
Escalabilidade	Elevada	Elevada	Reduzida	Reduzida	Moderada
Recompensa	Sim	Sim	Maioritariamente Não	Maioritariamente Não	Maioritariamente Não

Tabela 2.6: Comparação do desempenho dos mecanismos de consenso apresentados [18]

Atributos de desempenho	**PoW**	**PoS**	**Paxos**	**BFT e suas variantes**	**Raft**
Crash Fault Tolerance (CFT)	50%	50%	50%	33%	50%
BFT	50%	50%	NA	33%	NA
Tempo de Resposta	10 Minutos	1 Minuto		1 Segundo	
Consumo Enrgético	Elevado	Melhor do que PoW	Elevado	Moderado	Elevado
Taxa de Transferência de Dados Por Transação	Muito Reduzida	Reduzida	Muito Eleavada	Média	Muito Elevada
Latência da Transação	Muito Elevada	Elevada		Muito Reduzida	

Hyperledger Fabric [25], Corda [26], Quorum [27] e Parity [28]. Alguns estudos realizados demonstram que o Hyperledger Fabric é a plataforma mais otimizada nesta lista.

Em [11], os autores referem que as plataformas Ethereum e Parity são mais resilientes a falhas dos nós da sua rede, contudo são mais vulneráveis a falhas de segurança e que os principais *bottlenecks* das plataformas Hyperledger Fabric e Ethereum são os protocolos que estes usam para atingir o consenso. Neste mesmo artigo, foram efetuados testes para comparar o desempenho entre as plataformas Hyperledger Fabric, Ethereum e Parity. Os testes realizados demonstram uma superioridade do Hyperledger Fabric relativamente à taxa de execução, enquanto o Parity é o que apresenta uma menor latência, tal como ilustra a Figura 2.12.

Além disso, os resultados apresentados na Figura 2.13 demonstram que o desempenho das plataformas Hyperledger Fabric e Ethereum pioram com o aumento de nós, diminuindo o número de transações por segundo e aumentando a sua latência.

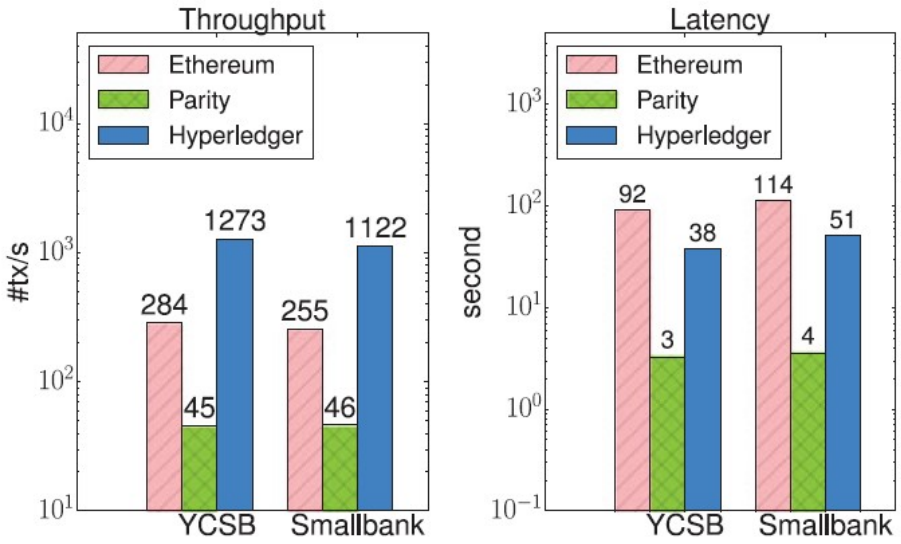


Figura 2.12: Comparação da taxa de execução (transações por segundo) e latência entre Hyperledger Fabric, Ethereum e Parity [11]

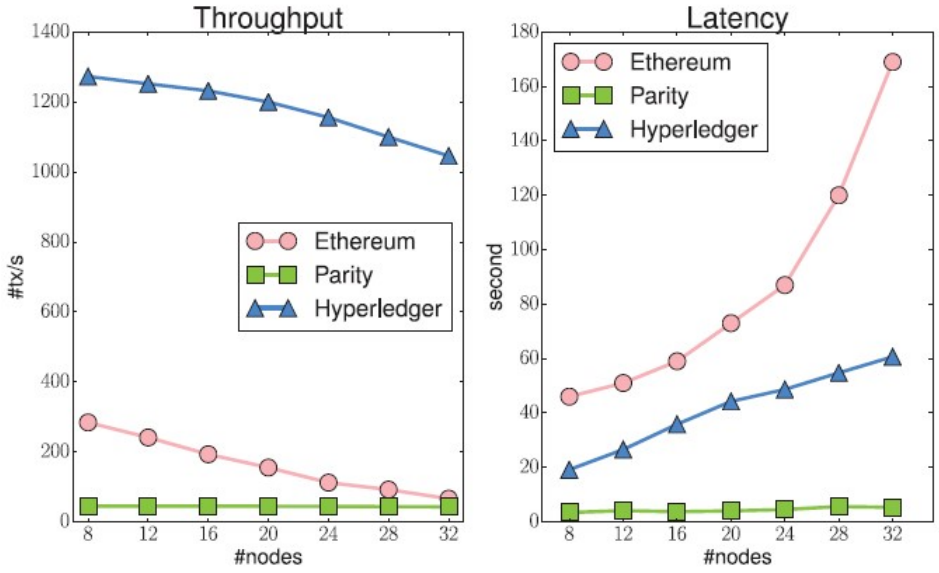


Figura 2.13: Comparação da taxa de execução (transações por segundo) e latência entre Hyperledger Fabric, Ethereum e Parity, com o aumento de nós [11]

Finalmente, no artigo referido é ainda demonstrado que o Hyperledger Fabric é a plataforma mais eficiente, necessitando de menos tempo de execução, enquanto consome menos recursos (Figura 2.14) e que apresenta um maior número de transações por segundo em ações de leitura (Figura 2.15) e de escrita (Figura 2.16), sendo a plataforma que menos espaço no disco ocupa (Figura 2.17).

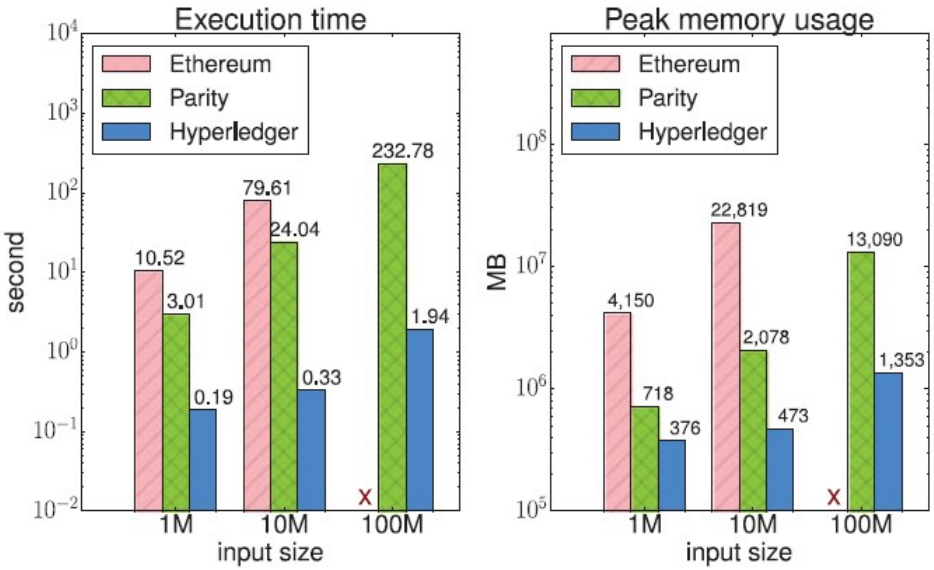


Figura 2.14: Comparação de tempo de execução e uso de memória entre Hyperledger Fabric, Ethereum e Parity, com o aumento de nós ('X' indica um erro *Out-of-Memory*)[11]

A superioridade do Hyperledger Fabric é comprovada num outro artigo que compara o desempenho de quatro plataformas *blockchain permissioned*: Hyperledger Fabric, Ethereum, Corda e Quorum. Os resultados apresentados neste artigo demonstram mais uma vez que o Hyperledger Fabric é a plataforma que apresenta uma maior taxa de execução (Figura 2.18) e uma menor latência (Figura 2.19) [29].

Assim, nesta dissertação foi efetuado um estudo mais aprofundado da plataforma Hyperledger Fabric, uma vez que, através do estudo de artigos sobre plataformas *blockchain* privadas, se concluiu que esta seria a opção mais eficaz.

Hyperledger Fabric

Detido pela Linux Foundation [30], o Hyperledger Fabric é um dos muitos projetos da Hyperledger [31] e é uma plataforma *open source* dedicada ao desenvolvimento de redes *blockchain* privadas (*permissioned*), onde o armazenamento de dados é efetuado seguindo o modelo *key-value* [32]. Ao contrário de outras redes como a Bitcoin, aqui não existe nenhuma criptomoeda e o acesso à rede é restrito aos membros da

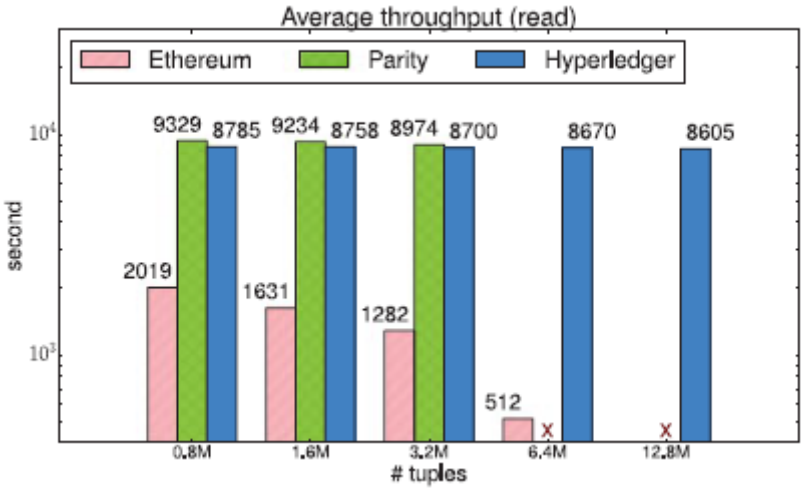


Figura 2.15: Comparação da taxa de execução (transações por segundo) na leitura de dados entre Hyperledger Fabric, Ethereum e Parity, com o aumento de nós [11]

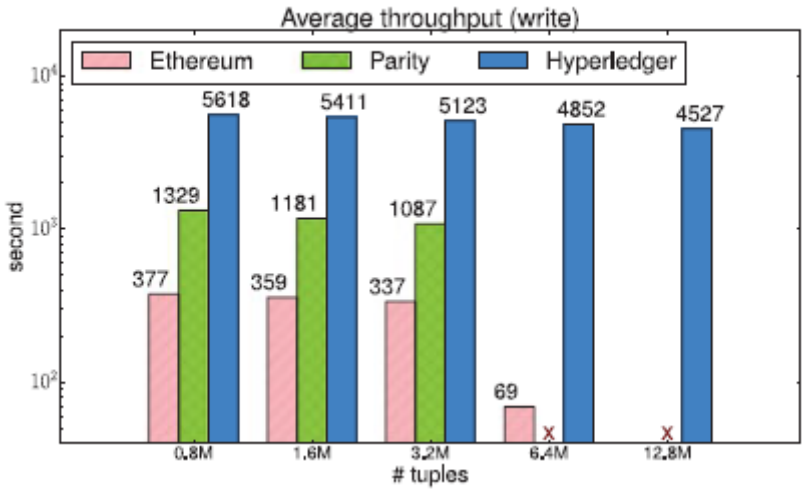


Figura 2.16: Comparação da taxa de execução (transações por segundo) na escrita de dados entre Hyperledger Fabric, Ethereum e Parity, com o aumento de nós [11]

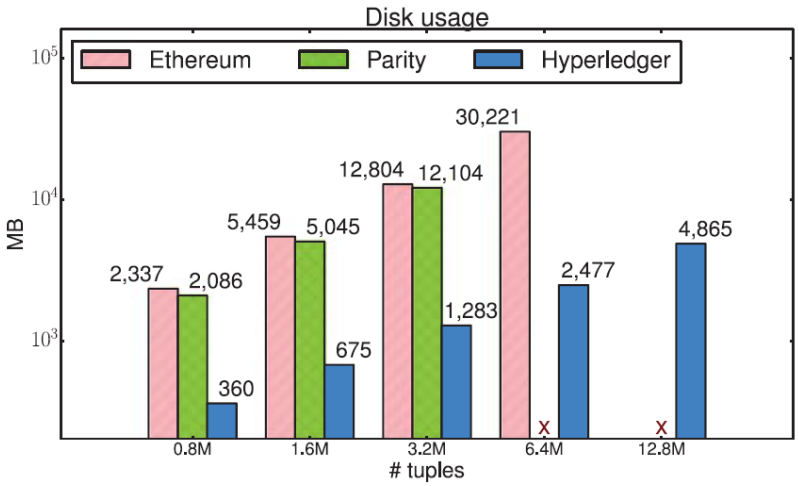


Figura 2.17: Comparação do uso do disco entre Hyperledger Fabric, Ethereum e Parity, com o aumento de nós [11]

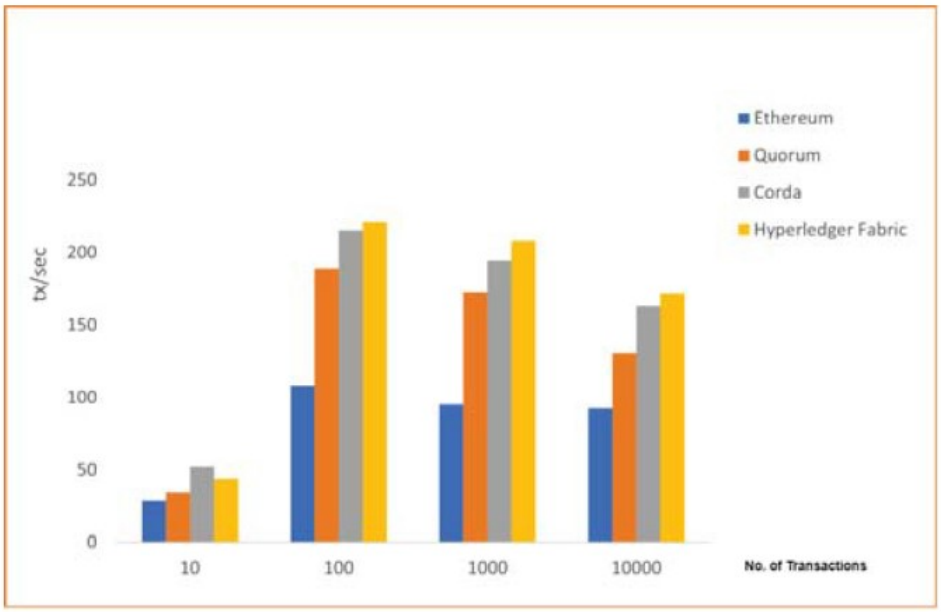


Figura 2.18: Comparação da taxa de execução (transações por segundo) entre Hyperledger Fabric, Ethereum, Quorum e Corda [29]

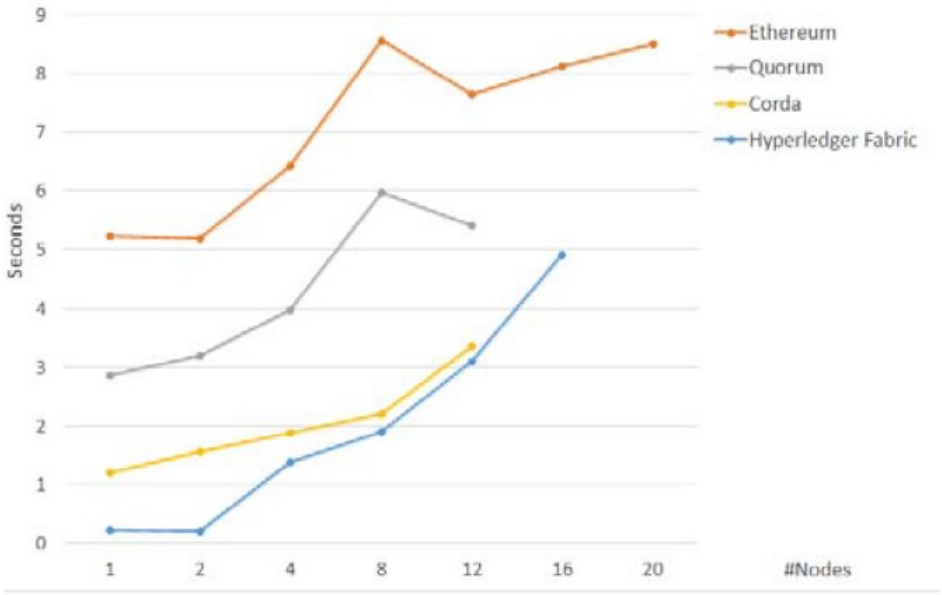


Figura 2.19: Comparação da latência Hyperledger Fabric, Ethereum, Quorum e Corda [29]

mesma, não havendo a possibilidade de qualquer pessoa se juntar à rede, tal como aconteceria numa rede pública. Esta plataforma utiliza o algoritmo de consenso PBFT e as transações são controladas através de *smart contracts* [33].

Uma *blockchain* do Hyperledger Fabric consiste num conjunto de nós que formam uma rede. Como se trata de uma plataforma privada, todos os nós que participam na rede têm uma identidade, que é providenciada por um *Membership Service Provider*. Nesta plataforma, os nós podem assumir um de três papéis [34] e comunicam entre si através do *framework* gRPC [35]:

- Clientes - apresentam propostas de transações para execução, ajudam a gerir a fase de execução e, finalmente, transmitem as transações para ordenação;
- Pares (*peers*) - executam as propostas de transação e validam as transações. Todos eles possuem uma cópia da rede. Nem todos executam as propostas de transação, apenas um conjunto deles realiza esta ação, sendo estes nós denominados de *endorsing peers*;
- *Ordering Service Nodes* (OSN) - a principal função destes nós é requisitar as transações dos blocos e transmiti-las para os restantes nós, para que as mesmas possam ser validadas. A utilização de apenas um *ordering node* não é recomendada num cenário real, uma vez que este seria um ponto de rutura único da rede [34].

A Figura 2.20 ilustra um exemplo de uma possível rede da plataforma Hyperledger Fabric.

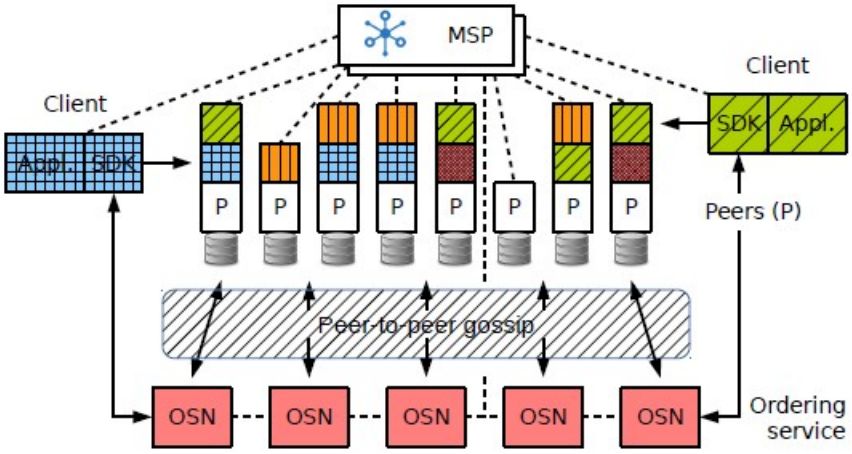


Figura 2.20: Exemplo de uma rede Hyperledger Fabric [34]

A privacidade das transações efetuadas entre o cliente e a rede é obtida através de um mecanismo de isolamento denominado de canal. Um canal é uma rede de comunicação privada dentro da rede principal e, assim, apenas os nós que constituem esse canal possuem uma cópia do mesmo [33]. A Figura 2.21 demonstra um exemplo desta situação, onde a organização A tem acesso ao canal 1, a organização C tem acesso ao canal 2, enquanto a organização B tem acesso a ambos os canais da rede.

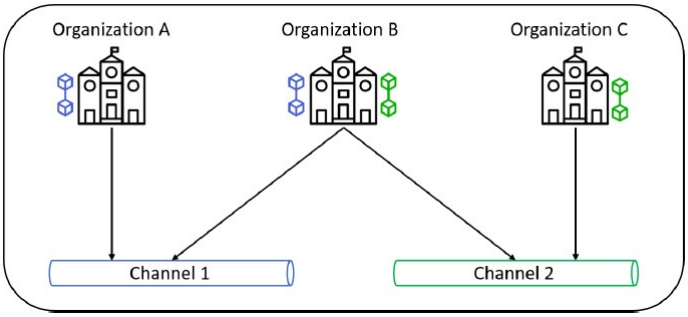


Figura 2.21: Exemplo de canais numa rede Hyperledger Fabric [36]

Ao contrário dos *smart contracts* da rede Ethereum, que são redigidos em Solidity, a plataforma Hyperledger Fabric suporta o desenvolvimento de *smart contracts* utilizando linguagens de programação mais convencionais, como, por exemplo, Go (também conhecido como Golang), Node.js, Java e Python. Ao suportar estas linguagens, os desenvolvedores podem poupar o tempo e esforço necessário para aprender novas linguagens de programação. Contudo, um dos grandes problemas de utilizar linguagens de programação mais convencionais neste cenário é a falta de

restrições ou determinadas características que permitem redigir *smart contracts* de forma correta e segura [37]. O Hyperledger Fabric recorre ao *software* Docker para executar os *smart contracts*. Os contentores do Docker suportam aplicações com baixa sobrecarga, sacrificando o isolamento das mesmas, visto que são executadas no mesmo sistema operativo e não em máquinas virtuais diferentes, cada uma com o seu sistema operativo [32]. Além disso, o Hyperledger Fabric distingue-se das demais graças à sua nova arquitetura de transações *execute-order-validate*. Esta nova arquitetura substitui o tradicional *order-execute* (Figura 2.22) utilizado pelas restantes plataformas. Numa arquitetura *order-execute*, primeiro, as transações são filtradas com base no algoritmo de consenso. Posteriormente, na fase de execução, cada nó executa as transações de forma sequencial com base na mesma ordem e, finalmente, os estados são atualizados em todos os nós. Esta fase de execução tem um impacto negativo no desempenho da rede, uma vez que os nós têm que percorrer e executar todas as transações, aumentado a latência.

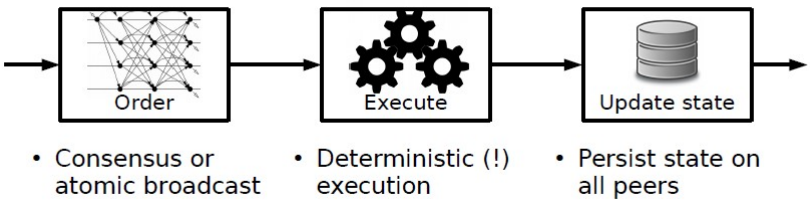


Figura 2.22: Arquitetura *Order-Execute* [34]

Por outro lado, na arquitetura *execute-order-validate*, o processo inicia-se quando é proposta uma nova transação e termina quando a mesma é adicionada à rede, passando pelas três seguintes fases (Figura 2.23):

1. Proposta (*Execute*) - Esta fase começa quando o cliente envia uma proposta de transação para os nós responsáveis por esta fase e o seu principal objetivo é aprovar as mesmas. Cada nó verifica se a proposta está bem estruturada, se a aplicação não está a tentar duplicar uma transação que já exista, se a assinatura do requerente é válida e se o mesmo tem as permissões necessárias para efetuar as operações pedidas. Posteriormente, cada nó executa o *chaincode* individualmente e gera uma resposta transacional baseada nos resultados obtidos, assinando a mesma. Finalmente, a transação de resposta à proposta recebida é enviada para a aplicação. Esta primeira fase termina após o cliente receber um determinado número de respostas;

2. *Ordering and Packaging* - Nesta fase, as transações são agrupadas em blocos, que são da responsabilidade do *orderer*. Primeiramente, o cliente envia as respostas aprovadas da proposta de transação para os *orderers*. Posteriormente,

estes últimos colocam as transações autorizadas num bloco numa ordem específica. É importante salientar que o número de transações por bloco é decidida pela configuração do canal e afeta diretamente a latência do mesmo.

3. Validação - A última fase é iniciada quando o *orderer* envia o bloco para os restantes nós do canal. Cada nó irá validar as transações do bloco separadamente e de forma determinística, uma vez que todos os nós validam o bloco de forma idêntica, fazendo com que cada nó fique com uma cópia igual da rede.

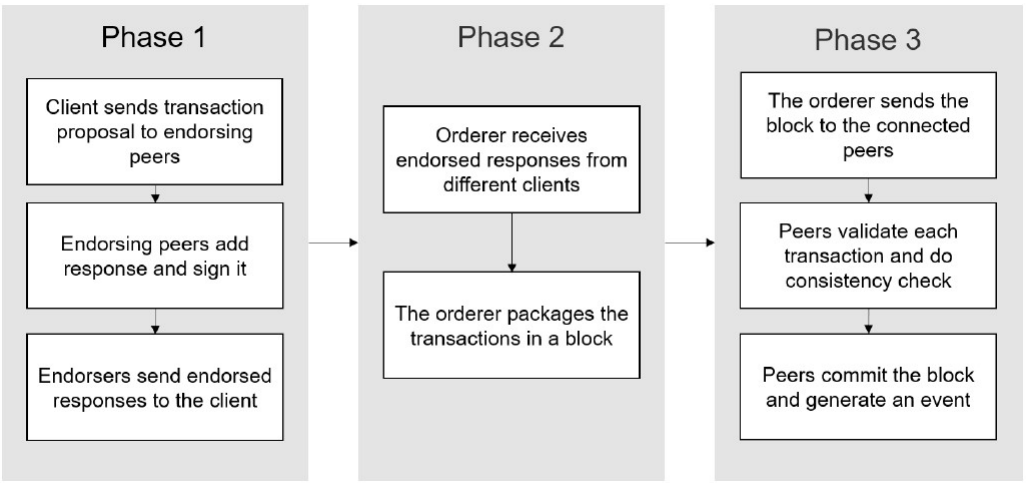


Figura 2.23: Arquitetura *Execute-Order-Validate* [36]

A latência de uma rede desenvolvida na plataforma Hyperledger Fabric aumenta com o aumento das transações realizadas. O número de blocos e de transações por bloco afeta também o rendimento da rede, sendo que este aumenta com o aumento do tamanho do bloco, pois mais transações num bloco implica que mais transações sejam validadas simultaneamente. Também é possível observar que o aumento do *batch-timeout* aumenta a latência, uma vez que cada bloco terá de esperar que atinja o *timeout* definido mesmo que já tenha recebido todas as transações [36].

Contudo, apesar de todos estes aspetos positivos, a plataforma Hyperledger Fabric é vulnerável a ataques DoS. Em [38] foram efetuados testes para avaliar a vulnerabilidade da plataforma a este tipo de ataques e os resultados indicam que os ataques DoS têm um impacto significativo na eficiência da rede, diminuindo o número de transações por segundo (Figura 2.24) e aumentando a latência da rede (Figura 2.25), consoante o incremento do número de nós sob ataque.

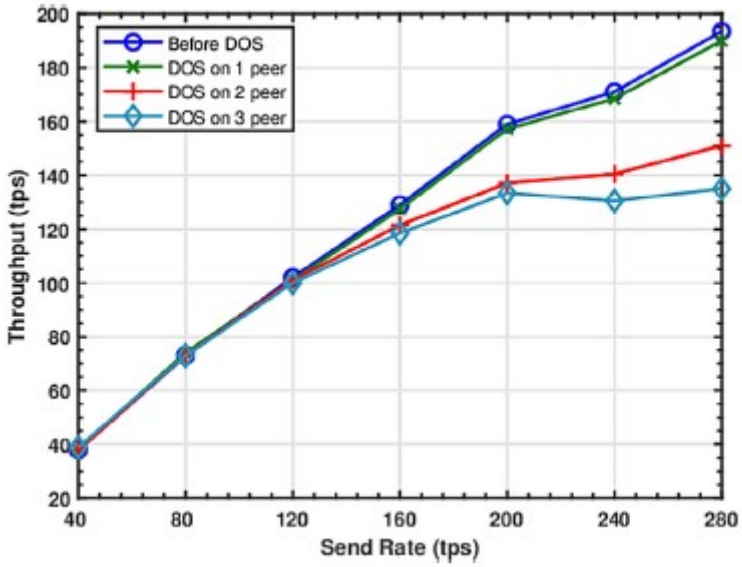


Figura 2.24: Influência do ataque DoS no número de transações por segundo do Hyperledger Fabric [38]

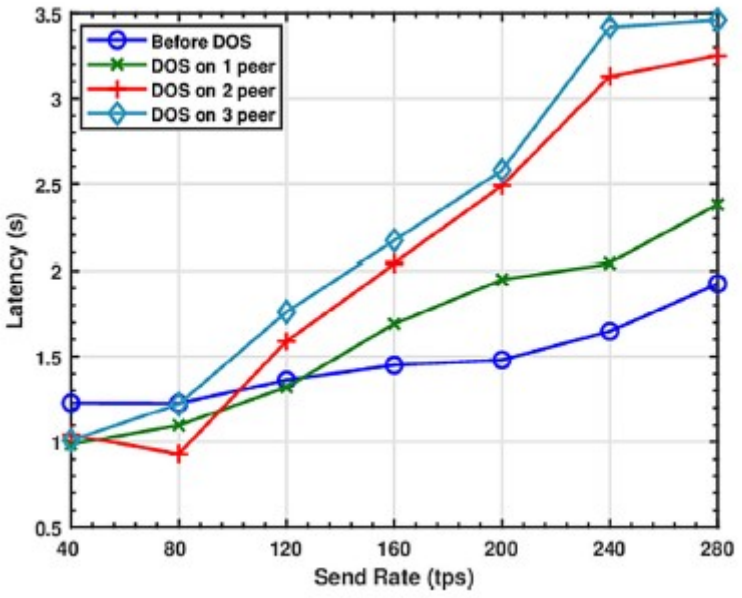


Figura 2.25: Influência do ataque DoS na latência do Hyperledger Fabric [38]

2.3 Bases de Dados Imutáveis

Com a crescente adoção de sistemas *blockchain* em diferentes indústrias, recentemente têm vindo a surgir algumas alternativas na forma de bases de dados imutáveis. Uma base de dados imutável é uma base de dados segura capaz de cumprir os critérios necessários para uma efetuar uma auditoria adequada e que apenas deve poder ser lida por um número restrito de utilizadores autorizados. Esta não deve permitir a edição ou eliminação de transações. Ou seja, apenas podem ser adicionados novos registos à base de dados, não sendo possível editar ou eliminar registos armazenados, preservando o seu estado original [39].

Nesta secção, serão apresentadas algumas soluções de bases de dados imutáveis, tais como o BigchainDB e o Immudb. Além destas, existem outras soluções de bases de dados imutáveis que não serão aprofundadas nesta dissertação, tais como Amazon *Quantum Ledger Database* (QLDB) [40], Dolt [41], TerminusDB [42] e Trillian [43].

2.3.1 BigchainDB

O BigchainDB [44] é um *software* que combina as propriedades da *blockchain* com as propriedades das bases de dados. Assim, este *software* caracteriza-se como sendo descentralizado e imutável.

O BigchainDB utiliza a plataforma Tendermint [45] para todas as operações relacionadas com consenso e *networking*. Cada nó da rede possui a sua própria base de dados MongoDB [46] e comunicam entre si através de protocolos Tendermint (Figura 2.26). Deste modo, o sistema é tolerante a falhas bizantinas (BFT), ou seja, até um terço dos nós podem falhar sem afetar o normal funcionamento da rede. Além disso, se um agente malicioso tentar obter permissões de administração de uma base de dados MongoDB local, no pior cenário, apenas conseguirá corromper ou eliminar os dados presentes nessa mesma base de dados, não afetando os restantes nós.

Uma vez que seja efetuada a inserção de dados, estes não poderão ser alterados ou eliminados. Caso tal aconteça, será automaticamente detetado. O BigchainDB utiliza algumas estratégias de forma a garantir a imutabilidade:

- Este *software* não fornece nenhuma API que permita editar ou eliminar dados armazenados;
- Cada nó possui uma cópia de todos os dados numa base de dados MongoDB local. Assim, se um nó for corrompido, os restantes nãos serão afetados;
- Todas as transações efetuadas são assinadas criptograficamente. Se os dados de uma transação inserida na rede forem alterados, a sua assinatura também irá mudar, o que será facilmente detetado, a não ser que a chave pública também seja alterada. Contudo, este último aspeto também deverá ser identificado

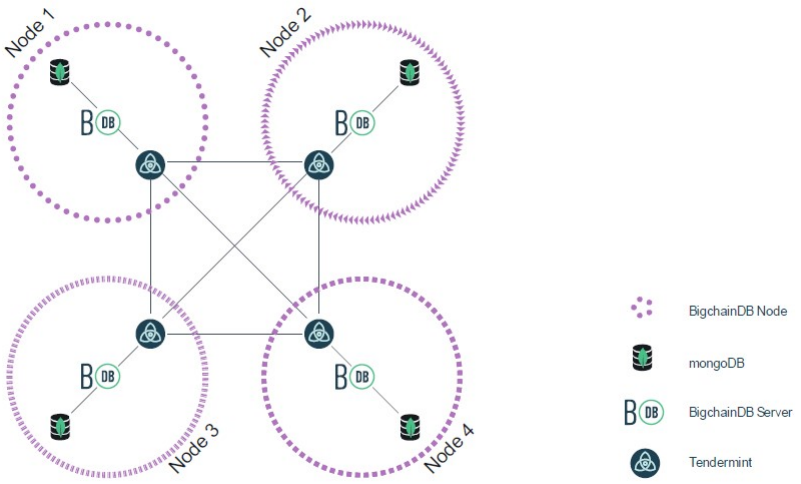


Figura 2.26: Comunicações numa rede BigchainDB com quatro nós [47]

pois cada bloco de transações é assinado por um nó, e as chaves públicas de todos os nós são conhecidas.

Apesar de não possuir testes que comprovem o seu desempenho, a organização alega que uma vez que este *software* utiliza Tendermint, o seu desempenho deverá ser elevada. As redes baseadas em Tendermint tendem a demorar apenas alguns segundos para que uma transação seja incluída num bloco. É também importante salientar que, visto que uma rede BigchainDB é gerida por uma entidade que controla os seus membros, os ataques Sybil não são um problema [47].

2.3.2 Immudb

Desenvolvido pela Codenotary [48], o Immudb [49] é um sistema de gestão de base de dados escrito em Go [50] e imutável: o seu histórico é preservado e não poderá ser alterado sem que os utilizadores se apercebam. Esta tecnologia pode ser utilizada como uma base de dados *key-value* ou relacional (SQL).

O Immudb garante a imutabilidade dos dados armazenados recorrendo a uma árvore de Merkle. O seu servidor efetua a gestão global das árvores de Merkle e a manipulação dos dados associados. O servidor possui um *thread* que calcula de forma contínua os valores da árvore de Merkle através dos dados subjacentes, de modo a prevenir uma possível corrupção dos dados. Também possui mecanismos de geração de provas e de assinatura de raiz. A Figura 2.27 ilustra a arquitetura do Immudb.

O servidor possui uma API baseada em gRPC, através da qual os clientes e auditores se conectam com a finalidade de se poderem autenticar, inserir dados, acompanhar o desempenho da rede e avaliar a sua consistência. O Immudb utiliza

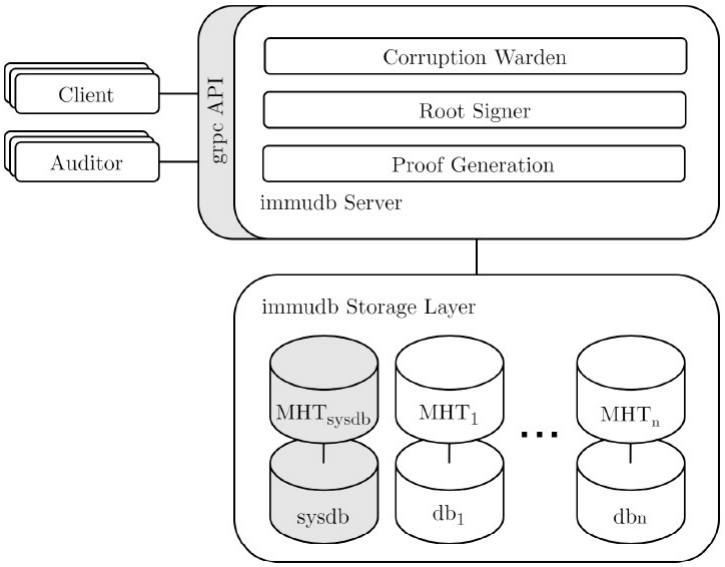


Figura 2.27: Arquitetura do Immudb [51]

os algoritmos SHA-256 e ECDSA, certificados pela FIPS, para realizar o *hashing* da árvore de Merkle e efetuar a assinatura de raíz, respetivamente.

A sua camada de armazenamento inclui uma ou mais bases de dados, além da base de dados *sysdb* e da árvore de Merkle que armazena os metadados do servidor. Cada base de dados é composta pela camada de armazenamento, que contém todos os dados inseridos, e uma árvore de Merkle, que protege o armazenamento contra adulterações não detectadas. Atualmente, o Immudb utiliza o sistema Badger [52] como motor de armazenamento, contudo, o servidor e a camada de armazenamento do Immudb podem ser considerados agnósticos em relação a este.

Caso este *software* seja utilizado como uma base de dados *key-value*, existem alguns comandos importantes:

- *get (Key k)* - retorna o valor mais recente do par *key-value (k,v)* associado à chave k;
- *safeGet (Key k, int i)* - retorna o mesmo valor do comando anterior, mas também exige uma prova de inclusão de *(k,v)* e de consistência entre a última árvore de Merkle conhecida pelo servidor e a árvore após a inserção do registo. A Figura 2.28 ilustra um exemplo deste comando;
- *getByIndex (int i)* - retorna o par *key-value* referente ao registo inserido número i;
- *history (Key k)* - retorna a lista de todos os pares *key-value* inseridos associados à chave k;

- *set (Key k, Value v)* - insere um registo no formato *key-value (k, v)* e adiciona o seu *hash* à árvore de Merkle;
- *safeSet (Key k, Value v, int i)* - efetua a mesma ação do comando anterior, mas, além disso, exige uma prova de inclusão de *(k,v)* e de consistência entre o estado da árvore de Merkle após a inserção do registo e o estado da mesma após a inserção de *xi*.

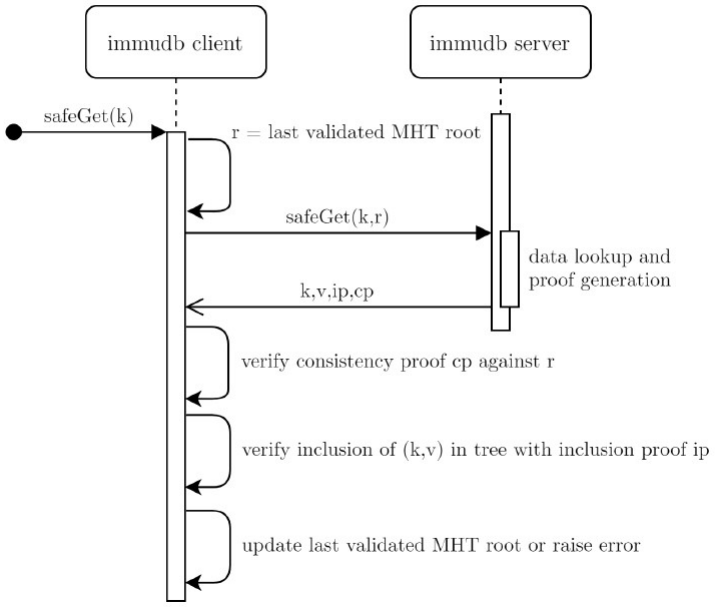


Figura 2.28: Exemplo de execução do comando *safeGet()* [51]

Cada auditor e cliente é responsável por manter a sua própria cópia do último estado válido da árvore de Merkle que recebeu e comparar e validar periodicamente a árvore de Merkle do servidor em relação à sua e atualizar o seu estado interno ou reportando uma falha, caso a mesma ocorra.

A assinatura de raiz presente no Immudb fornece garantias mais fortes de irrefutabilidade, mas apenas são úteis em aplicações em que os utilizadores podem trabalhar de forma adversa. Uma vez que o algoritmo de assinatura ECDSA é exigente a nível computacional, utilizar o Immudb sem recorrer a esta característica, pode melhorar o rendimento da rede [51].

Todas estas características promissoras do Immudb, fizeram que a empresa responsável pelo projeto (Codenotary) fosse considerada a *startup open-source* com maior crescimento no último quarto do ano 2021 [53].

2.4 Documentação da API

Uma vez que a *Web* API do SIGEDEE segue os princípios de uma arquitetura do estilo *Representational State Transfer* (REST) [54], esta análise centrou-se em ferramentas mais direcionadas para esse tipo de *Web* APIs.

Atualmente, existem diversas aplicações destinadas à documentação de uma API, tais como a *RESTful API Modeling Language* (RAML) [55], Apache Avro IDL [56] e a *Web Application Description Language* (WADL) [57]. Contudo, nesta dissertação, apenas serão abordadas duas soluções: Postman e Swagger.

2.4.1 Postman

O Postman [58] é um *software* dedicado ao desenvolvimento e teste de APIs. Esta aplicação é capaz de simular os diferentes pedidos *Hypertext Transfer Protocol* (HTTP) (*GET*, *POST*, *PUT*, *PATCH* e *DELETE*) [59]. Além disso, é ainda capaz de efetuar a documentação de uma API e testar o seu desempenho, medindo o tempo de execução de cada pedido ou então simulando tráfego real, ou seja, simulando que a API está a ser acedida por diversos utilizadores em simultâneo. Estas funções permitem avaliar o desempenho da API num cenário real [58].

2.4.2 Swagger

Outra ferramenta reconhecida no que toca a documentação de APIs é o Swagger [60]. Este *software* permite descrever a estrutura de uma API REST, de modo a que um computador a entenda. Assim, ao ler a estrutura da API, o Swagger será capaz de gerar automaticamente a sua documentação.

Para que tal seja possível, a API deverá estar desenvolvida de acordo com a estrutura *OpenAPI Specification* (OAS) [61], permitindo descrever diversos aspetos da API, incluindo [60]:

- *Endpoints* disponíveis e as operações de cada um;
- Os *inputs* e *outputs* de cada operação. Por exemplo, os *inputs* podem ser uma estrutura de dados *JavaScript Object Notation* (JSON) a enviar para realizar uma determinada operação na API;
- Métodos de autenticação;
- Informações sobre a licença, termos de utilização e outros temas.

Este *software* fornece ainda diferentes aplicações para facilitar o processo de documentar uma API [62]:

- Swagger Editor - possibilita a criação da documentação de uma API REST através de um *browser*, apresentando, em simultâneo, o documento Swagger e os resultados obtidos consoante as configurações da API;

- Swagger UI - permite visualizar e testar a API REST definida pelo Swagger através de qualquer *browser*. As funções de teste incluídas permitem definir os *inputs* de uma operação, realizar essa mesma operação e obter os resultados obtidos após a executar;
- Swagger Codegen - por último, este gera um *Kit* de Desenvolvimento de *Software* (SDK) em diversas linguagens de programação, tais como Java, Objective-C, PHP e Python, que corresponda às características da API previamente definida.

Capítulo 3

Sistema de Gestão de Equipamento do DEE

Numa fase inicial da realização da presente dissertação, foi necessário estudar a aplicação de modo a entender o seu funcionamento. Nesse processo, foram também adicionadas novas funcionalidades para os utilizadores finais, apesar de o foco principal desta dissertação ser a implementação de um mecanismo que garanta a não repudiação e a integridade dos dados armazenados no SIGEDEE. Neste capítulo é explicado o modo de funcionamento do Sistema de Gestão de Equipamento do DEE (SIGEDEE) e são apresentadas as suas novas funcionalidades.

O sistema original permitia consultar, adicionar e editar informações sobre os equipamentos (computadores, multímetros, entre outros), os respetivos fornecedores e as instalações afetas ao DEE. Uma das funcionalidades disponibilizadas é a inserção de múltiplos equipamentos num único pedido, através da submissão de um ficheiro no formato XLSX (Microsoft Excel) ou *Comma-separated values* (CSV). O sistema original possuía também um mecanismo que permitia reverter alterações, através da criação de uma cópia do registo original após cada edição, que poderia ser selecionada pelo autor da alteração, ou pela direção do DEE, para reverter um registo para um estado anterior.

O SIGEDEE é baseado num *backend* cujas funcionalidades são acedidas através de uma API com arquitetura REST. O *backend* foi maioritariamente desenvolvido com recurso à linguagem de programação PHP e ao sistema de gestão de bases de dados MariaDB, estando alojado numa máquina com o sistema operativo Fedora 38.

No que se segue, no contexto da descrição da API do SIGEDEE, o endereço deste servidor é referido como *Uniform Resource Locator* (URL).

Existem três tipos de utilizadores que podem aceder a esta aplicação, cada um com acesso a um diferente conjunto de funcionalidades:

- Docentes, apenas com permissões de consulta de dados.
- Técnicos do DEE, com acesso a funcionalidades para gestão de equipamento em instalações sob sua responsabilidade;
- Elementos da direção do DEE, com acesso a todas as funcionalidades da aplicação;

A página de autenticação do SIGEDEE permite o *login* com base nas credenciais geridas pelo DEE (Figura 3.1) ou com base no envio de uma ligação para o endereço de *e-mail* institucional do utilizador (Figura 3.2). Em ambos os casos, é gerado um *JSON Web Token* (JWT), com um período de validade limitado e contendo informações referentes ao utilizador. Cada chamada à API do *backend* deverá incluir um JWT válido, garantindo deste modo que o respetivo utilizador foi devidamente autenticado.

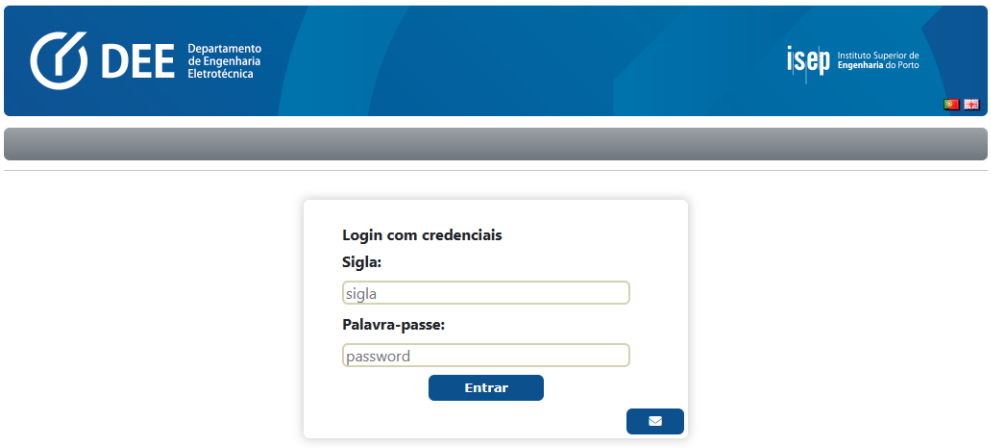


Figura 3.1: *Login* com sigla e palavra-passe no SIGEDEE

No caso de tentativa de *login* através das credenciais do DEE, o JWT só é gerado, e o *login* concretizado, no caso da verificação da palavra chave ser bem sucedida. No caso de tentativa de *login* através de endereço de *e-mail*, caso o endereço esteja registado na base de dados, o JWT é gerado e incluído num *link* de acesso ao *frontend* da aplicação. Por sua vez, este *link* é imediatamente enviado para o endereço de *e-mail* especificado, tal como ilustrado na Figura 3.3.

Nas secções que se seguem, são descritas as funcionalidades adicionadas ao SIGEDEE.

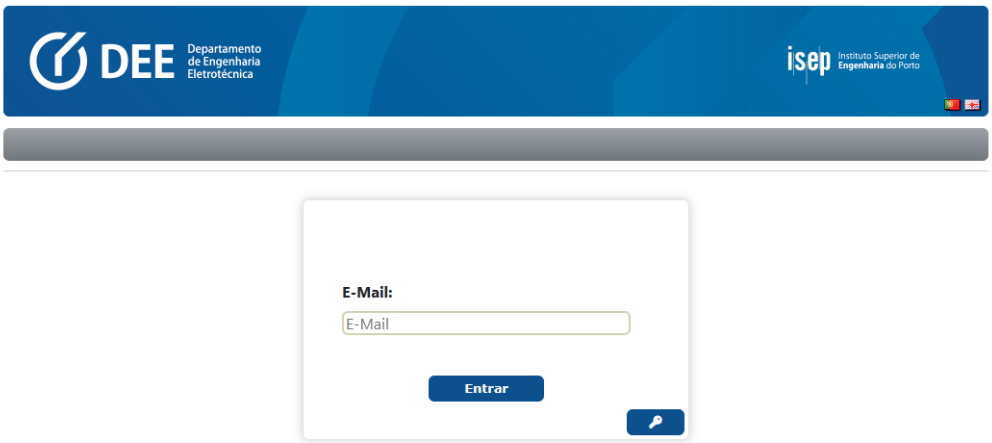


Figura 3.2: *Login* com endereço de *e-mail* no SIGEDEE

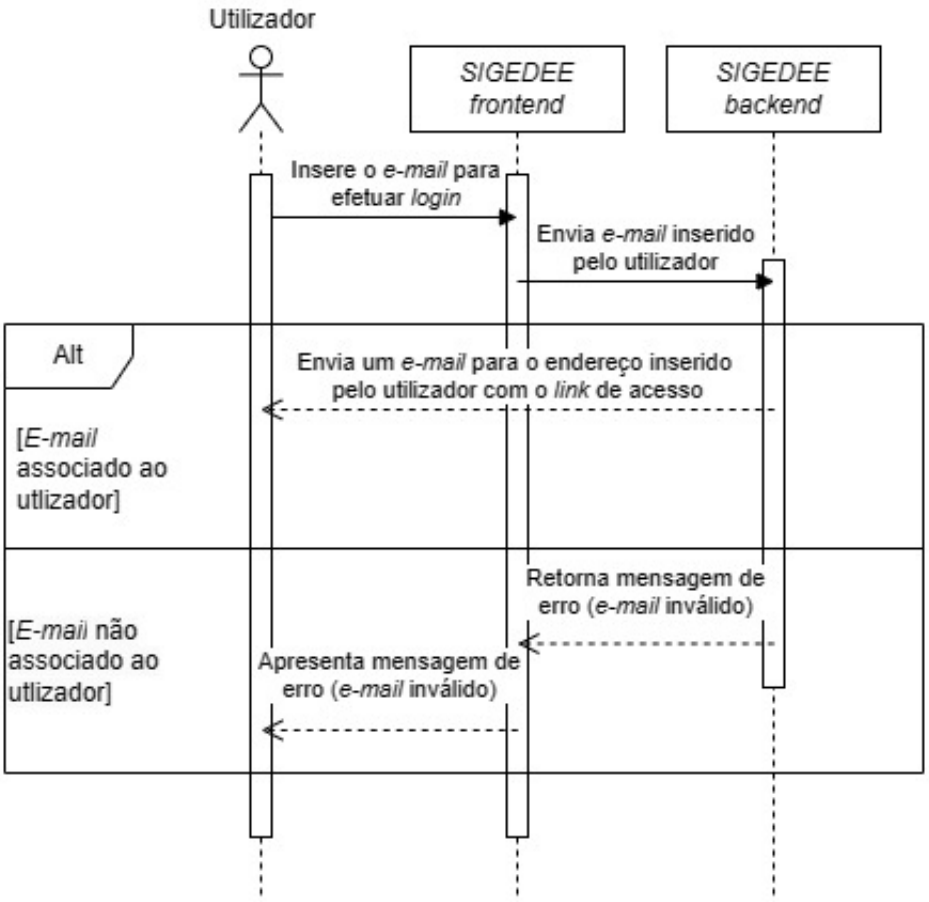


Figura 3.3: Processo de *login* com *e-mail*

3.1 Sistema de *Helpdesk*

Uma vez que a finalidade da aplicação em estudo é apoiar a gestão dos equipamentos do DEE, pretendia-se integrar na mesma um sistema de *helpdesk* que permitisse reportar avarias nos equipamentos, assim como acompanhar o processo de resolução das mesmas. Adicionalmente, tal sistema poderá ser útil no processo de avaliação do desempenho dos técnicos do departamento, permitindo uma mais fácil contabilização dos processos resolvidos e o tempo médio de resolução. Previamente, tinha já sido desenvolvido o protótipo de uma plataforma *helpdesk*, ainda não integrada no SIGEDEE. Assim, decidiu integrar-se no SIGEDEE uma versão melhorada do sistema de *helpdesk*, seguindo a lógica do modelo inicial.

Um pedido de *helpdesk* é designado habitualmente por *ticket*. Todos os utilizadores estão autorizados a efetuar este tipo de pedidos. Após a sua criação, um ticket pode passar por diferentes estados:

- *analise* - a aguardar análise; o *ticket* assume este estado assim que é criado;
- *processamento* - indica que o problema se encontra em tratamento; o *ticket* pode ser colocado neste estado por elementos da direção ou técnicos responsáveis pela resolução do problema;
- *resolvido* - indica que o problema foi resolvido; tal como no caso anterior, o *ticket* apenas pode ser dado como resolvido por elementos da direção ou técnicos responsáveis pela resolução do problema;
- *cancelado* - caso o autor do *ticket* deseje cancelar o pedido; apenas pode passar a este estado a pedido do respetivo autor e enquanto estiver a aguardar análise.

A Figura 3.4 ilustra os possíveis estados de um *ticket* no sistema implementado.

De seguida explica-se em mais detalhe as operações disponíveis no *backend* para a gestão dos *tickets*.

3.1.1 Criação de *ticket*

A criação de um *ticket* é efetuada através de um pedido do tipo *POST* para o *endpoint URL/helpdesk* da API REST do DEE, contendo os dados necessários (Tabela 3.1) no formato JSON.

Após a o registo do *ticket* na base de dados, o *backend* envia três *e-mails* a confirmar que a avaria foi reportada: um para o utilizador que reportou a avaria, outro para a direção do DEE e o terceiro para o técnico responsável pela instalação onde o equipamento se encontra.

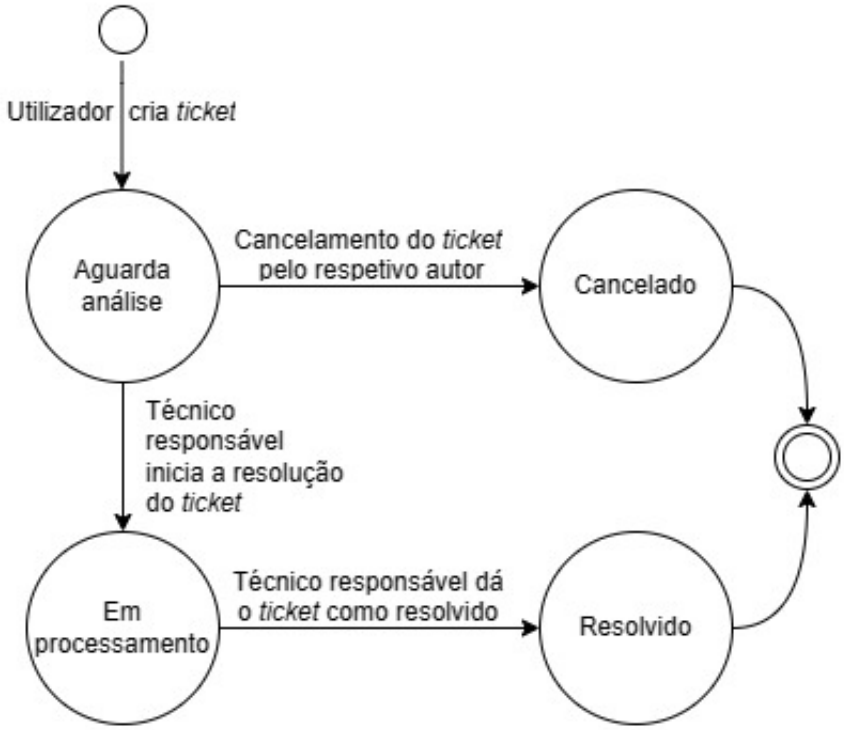


Figura 3.4: Possíveis estados de um *ticket* no sistema de *helpdesk* do SIGEDEE

3.1.2 Consulta de *tickets*

No *frontend* atualmente implementado, todos os utilizadores têm acesso a uma listagem dos *tickets* por si criados. Além disso, os elementos da direção têm acesso também às avarias reportadas por todos os utilizadores. Adicionalmente, os técnicos e os restantes docentes conseguem consultar os *tickets* referentes aos equipamentos das instalações da sua responsabilidade.

Esses dados são obtidos através de pedidos do tipo *GET* para o *endpoint* *URL/helpdesk* com os parâmetros *filtro_tipo* e *filtro_pedido*. Estes parâmetros referem-se ao estado e tipo do *ticket*, respetivamente. O parâmetro *filtro_pedido* pode assumir os valores *own* e *all*, que indicam, respetivamente, se o utilizador pretende consultar apenas os *tickets* criados por si ou todos os *tickets* associados a instalações sob sua responsabilidade. O parâmetro *filtro_tipo* pode assumir um dos seguintes valores, referentes aos estados anteriormente descritos:

- *analise* - todos os *tickets* que aguardam análise;
- *processamento* - todos os *tickets* que se encontram em processo de resolução;
- *resolvido* - todos os *tickets* que tenham sido concluídos;
- *cancelado* - todos os *tickets* que tenham sido cancelados.

Tabela 3.1: Dados a enviar para criar um *ticket*

Variável	**Tipo de Variável**	**Descrição**
id_equipamento	*int*	Identificador do equipamento avariado
titulo_problema	*text*	Título a atribuir ao *ticket*
descricao_problema	*text*	Breve descrição da avaria detetada no equipamento

Existe ainda a possibilidade de limitar os resultados obtidos, enviando também o parâmetro *limite*, indicando o número de *tickets* que se pretenda consultar. Caso se pretenda obter informações detalhadas de um determinado *ticket*, deverá ser indicado o respetivo Identificador (ID), um número único para cada *ticket* (*GET URL/helpdesk/{id_ticket}*).

3.1.3 Alteração do estado do *ticket*

Assim que é criado, o *ticket* assume o estado *Aguarda Análise* com prioridade *Média*. Posteriormente, assumirá um dos três restantes estados possíveis, previamente apresentados: *Em processamento*, *Resolvido* ou *Cancelado*. A gestão dos *tickets* é da responsabilidade dos técnicos responsáveis pela instalação onde se encontra o equipamento. Apesar disso, todos os utilizadores poderão cancelar *tickets* criados por si, enquanto estes não entram na fase de processamento. Além disso, os elementos da direção têm a capacidade de alterar a prioridade do *ticket*, estando previstos três níveis de prioridade. Para tal, deverá ser efetuado um pedido do tipo *PUT* à API (*PUT /URL/helpdesk/id_ticket/priority*) contendo o novo valor da prioridade em formato JSON (*{"prioridade":"{prioridade}"}*), tal como indicado na Tabela 3.2.

Tabela 3.2: Diferentes níveis de prioridade de um *ticket*

Prioridade do *ticket*	**Parâmetros a enviar**
Baixa	{"id_prioridade":"Baixa"}
Média	{"id_prioridade":"Média"}
Alta	{"id_prioridade":"Alta"}

A Tabela 3.3 apresenta um resumo dos pedidos a serem efetuados à API para colocar o ticket em cada um dos possíveis estados, assim como os tipos de utilizadores com autorização para efetuar o respetivo pedido.

Tabela 3.3: Alteração do estado de um *ticket*

Estado desejado para o *Ticket*	**Utilizadores Autorizados**	**Pedido API**
Aguarda análise	Qualquer utilizador	*POST /URL/helpdesk*
Cancelado	Autor do *ticket*	*PUT /URL/helpdesk/id_ticket/cancelled*
Em processamento	Técnico responsável pela instalação do equipamento	*PUT /URL/helpdesk/id_ticket/processing*
Resolvido	Técnico responsável pela instalação do equipamento	*PUT /URL/helpdesk/id_ticket/fixed*

Tal como no caso anterior, sempre que o estado do *ticket* for atualizado, a direção do DEE e o utilizador que reportou a avaria serão notificados por *e-mail*, sendo que a direção do DEE será informada sobre quem efetuou tal alteração.

3.1.4 Trocas de mensagens adicionais associadas a um *ticket*

Os utilizadores que participam no reporte ou resolução de uma avaria, isto é, elementos da direção, o utilizador que cria o *ticket* e o técnico responsável pela sua resolução, têm a possibilidade de inserir comentários ou questões associadas ao respetivo *ticket*, desde o momento em que este é criado e até 30 dias após ter sido encerrado (cancelado ou resolvido). Essas mensagens são transmitidas para todos os intervenientes via *e-mail*.

Para efetuar esta ação, o cliente deverá enviar as observações que pretende inserir no formato JSON ({{"mensagem":"*{observacoes}*") para a API (*POST /URL/helpdesk/{id_ticket}/obs*. Sempre que é submetido um pedido deste tipo, o *backend* regista a mensagem na base de dados e faz o envio do respetivo *e-mail* a todos os intervenientes.

3.2 Registo de Movimentação de Equipamentos

A segunda funcionalidade adicionada ao SIGEDEE foi o registo de movimentação de equipamentos. A finalidade desta funcionalidade é facilitar o processo de movimentação de equipamento entre diferentes locais, por parte dos técnicos do DEE, ou a sua requisição pelos utilizadores do sistema. Esta funcionalidade permite que

todos os movimentos sejam rastreados, disponibilizando a localização esperada de cada equipamento.

3.2.1 Pedidos de movimento

Os docentes, assim como os técnicos, caso o equipamento em questão não esteja afeto a uma instalação da sua responsabilidade, necessitam de efetuar um pedido de movimento. Enquanto os docentes podem requisitar um equipamento para qualquer sala, os técnicos apenas o podem fazer para aquelas que sejam da sua responsabilidade. Os docentes possuem uma maior seleção de instalações para onde podem requisitar um equipamento, uma vez que este pode ser necessário para lecionar uma aula. Por outro lado, os técnicos têm a liberdade de movimentar equipamentos entre instalações da sua responsabilidade, para não sobrecarregar a plataforma com pedidos de movimento. Contudo, estes movimentos devem também ser registados.

Para criar um pedido de movimentação de equipamento no *backend*, deverá ser efetuado um pedido *POST* para o *endpoint URL/movimentos*, enviando os dados necessários no formato JSON, descritos na Tabela 3.4.

Tabela 3.4: Dados a enviar para criar um pedido de movimentação de equipamento

Variável	Tipo de Variável	Descrição
id_equipamento	*int*	ID do equipamento que se pretende pedir emprestado
nova_localizacao	*text*	Instalação para onde se pretende mover o equipamento

Após receber o pedido, o *backend* irá automaticamente efetuar algumas verificações para inserir um novo registo de requisição na base de dados, como obter a instalação atual do equipamento e o estado em que se encontra, bem como verificar se um técnico está a tentar mover um equipamento para uma instalação da sua responsabilidade. Assim que seja inserido um pedido de empréstimo de um equipamento na base de dados, será enviado um *e-mail* de confirmação para o requerente e outro para a direção do DEE de modo a notificá-la sobre o mesmo. A Figura 3.5 explica o processo de criação de pedido de movimentação de equipamento na base de dados.

3.2.2 Consulta de movimentos

Tal como no caso anterior, os três tipos de utilizadores poderão consultar os pedidos de empréstimo, com algumas restrições, consoante o seu cargo. Os elementos

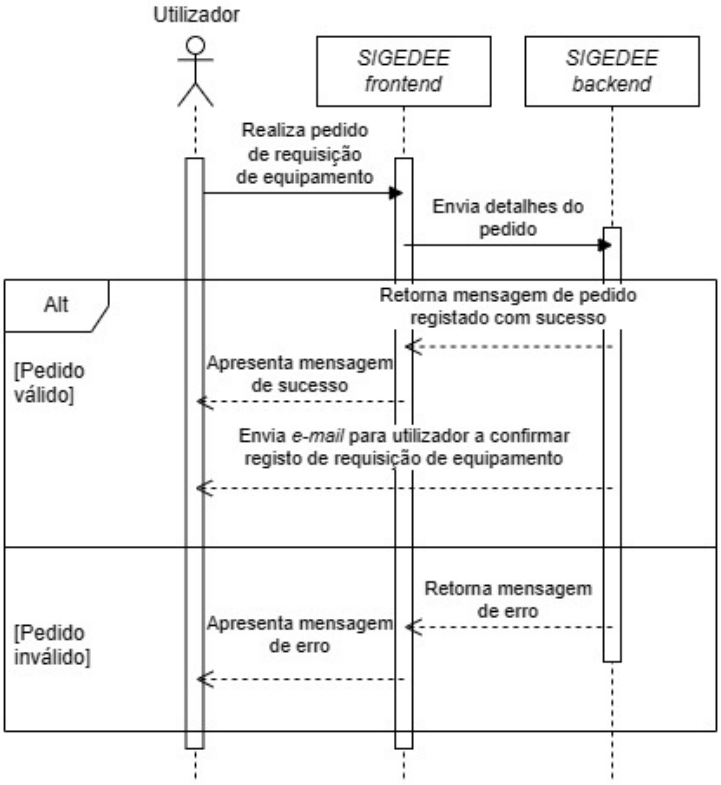


Figura 3.5: Criação de pedido de movimentação de equipamento

da direção podem visualizar todos os pedidos de empréstimo efetuados no DEE, enquanto os técnicos têm acesso aos pedidos de empréstimo de equipamentos que estejam localizados em instalações da sua responsabilidade. Além disso, tal como os docentes, os técnicos também são capazes de consultar os pedidos de empréstimo realizados por si.

Caso pretenda consultar estes dados, o cliente deve endereçar um pedido *GET* para o *endpoint URL/movimentos* da API do DEE, enviando, novamente, como parâmetros as variáveis *filtro_tipo* e *filtro_pedido*, que se referem ao estado e tipo da requisição de equipamento, respetivamente. A variável *filtro_tipo* segue a mesma lógica do sistema *helpdesk*, ou seja, irá retornar pedidos de empréstimo realizados pelo utilizador (*own*), ou então de todos os utilizadores no caso da direção e pedidos de empréstimo de equipamentos de instalações da sua responsabilidade, no caso dos técnicos (*all*). Por outro lado, a variável *filtro_pedido* pode assumir agora cinco estados:

- *analise* - retorna os pedidos de movimento no estado de análise. Sempre que é efetuado um pedido, este assume automaticamente este estado. Estes pedidos apenas estão disponíveis para os elementos da direção e para os restantes utilizadores, caso o pedido tenha sido feito por si;

- *aprovado* - retorna os pedidos que foram aprovados. A decisão de aprovar um pedido de empréstimo é tomada por elementos da direção;
- *rejeitado* - retorna os pedidos que foram rejeitados. A decisão de rejeitar um pedido de empréstimo é da responsabilidade dos elementos da direção. Apenas os elementos da direção e os utilizadores que efetuaram o pedido de empréstimo têm acesso a estes resultados;
- *emprestado* - retorna os equipamentos que já foram emprestados. O equipamento é declarado como emprestado pelo técnico responsável pela instalação onde o equipamento se encontrava;
- *devolvido* - retorna os equipamentos que já foram devolvidos. O equipamento é declarado como devolvido pelo técnico responsável pela instalação onde o equipamento se encontrava, indicando o estado em que o mesmo foi devolvido.

Tal como no sistema *helpdesk*, existe também a possibilidade de limitar o número de resultados obtidos, seguindo a mesma lógica. Além disso, é também possível visualizar um pedido de empréstimo de um equipamento de forma detalhada, efetuando para tal um pedido à API indicando o ID pretendido (*GET URL/movimentos/{id_movimento}*).

3.2.3 Processar pedido de movimento

Após ser inserido um pedido de empréstimo de um equipamento, o mesmo poderá assumir diferentes estados, sendo estes alterados por diferentes tipos de utilizador consoante a sua fase, tal como descrito na Secção 3.2.2, sendo eles *Em análise*, *Aprovado*, *Rejeitado*, *Emprestado* ou *Devolvido*. Numa fase inicial, o pedido de movimento estará em análise e a direção do DEE irá decidir se aprova ou rejeita o mesmo. Caso este seja aprovado, o requerente receberá um *e-mail* contendo um código de levantamento que deverá indicar, no ato de levantamento do equipamento, ao técnico responsável pela instalação onde o equipamento se encontrava localizado quando o pedido foi efetuado, de modo a ser possível efetuar o empréstimo do mesmo. Após o equipamento ser declarado como emprestado, o requerente receberá um novo *e-mail*, desta vez contendo o código de devolução que deverá indicar ao técnico no ato de devolução do equipamento. Finalmente, o técnico indicará o estado do equipamento aquando da sua devolução.

Não podem ser atualizados vários pedidos de movimento em simultâneo e um pedido de movimentação de equipamento apenas pode ser aprovado, ou o equipamento emprestado, no caso de não existir nenhum processo de requisição do mesmo equipamento. Ou seja, se o equipamento pretendido já tiver sido emprestado, ou um pedido de empréstimo aceite, não é possível aceitar outro pedido.

A Tabela 3.5 descreve os diferentes estados que um pedido de movimento pode assumir, quem os pode atualizar, que pedido deverá ser efetuado pelo cliente à API e os dados a enviar no formato JSON, caso seja necessário.

Tabela 3.5: Diferentes estados de um movimento

Estado *Ticket*	**Permissões de Edição (Tipo de Utilizador)**	**Pedido API**	**Dados a Enviar (JSON)**
Em análise	Direção, técnicos e docentes	*POST /URL/movimento*	
Aprovado	Direção	*PUT /URL/movimento/id_movimento/accept*	
Rejeitado	Direção	*PUT /URL/movimento/id_movimento/decline*	
Emprestado	Técnico responsável pela instalação onde se encontrava o equipamento quando foi efetuado pedido de empréstimo	*PUT /URL/movimento/id_movimento/loan*	*{"codigo_movimento": "{codigo_movimento}"}, em que "{codigo_movimento}"é o código de levantamento do equipamento*
Devolvido	Técnico responsável pela instalação onde se encontrava o equipamento quando foi efetuado pedido de empréstimo	*PUT /URL/movimento/id_movimento/return*	*{"id_estado_final": "{ID}", "codigo_movimento": "{codigo_movimento}"}*, em que "{ID}"é o valor *int* referente ao estado em que o equipamento foi devolvido e "{codigo_movimento}"é o código de devolução do equipamento

Tal como no caso anterior, sempre que o estado do pedido de movimento for alterado, o utilizador que efetuou o pedido será notificado por *e-mail*. Caso a decisão tomada indique que o pedido foi aceite ou que o equipamento foi emprestado ou devolvido, o técnico responsável pela instalação original do equipamento receberá também um *e-mail*. Assim que o equipamento seja emprestado, a sua localização será atualizada na base de dados

A Figura 3.6 ilustra o funcionamento do sistema de registos de requisição de equipamentos do DEE.

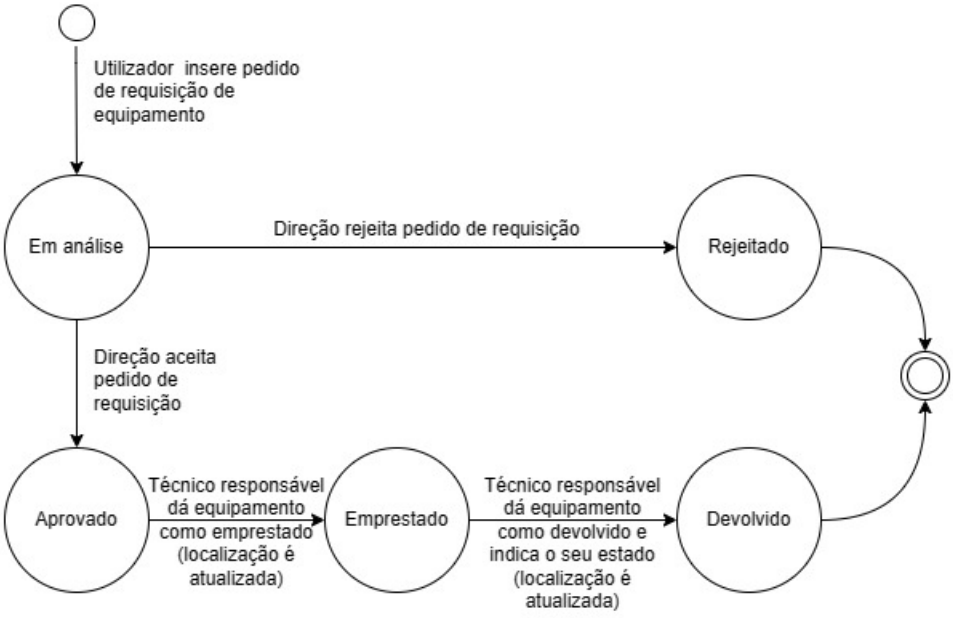


Figura 3.6: Funcionamento do sistema de registos de requisição de equipamentos do DEE

3.3 Sistema de Gestão de Utilizadores

Apesar de não ter sido considerado como um objetivo numa fase inicial da dissertação, ao longo da realização da mesma revelou-se ser necessário introduzir um sistema de gestão de utilizadores do SIGEDEE, de modo a ser possível criar novos utilizadores ou editar os seus dados de uma forma simples. Através deste serviço, é possível inserir um novo utilizador, alterar o tipo de utilizador ou desativá-lo. Apenas os elementos da direção do DEE têm acesso a este serviço.

3.3.1 Inserção de novo utilizador

A API do SIGEDEE fornece também a possibilidade de registar novos utilizadores, efetuando um pedido *POST* para o *endpoint URL/utilizadores*, enviando os dados necessários no formato JSON, tal como descrito na Tabela 3.6.

Após receber os dados enviados pelo cliente, o *backend* irá automaticamente completar os dados em falta correspondentes ao utilizador que se pretende inserir, como o seu *e-mail*. De seguida, antes de inserir o novo utilizador, irá verificar se o tamanho da sigla inserida corresponde ao tipo de utilizador selecionado, abortando o processo de inserção, no caso de ambos não corresponderem. Ou seja, no caso de ser um técnico, a sua sigla deverá ser composta por quatro letras. Por outro lado, caso se trate de um docente, a sigla deverá ser constituída por três letras apenas.

A Figura 3.7 ilustra o processo de registo de um novo utilizador no SIGEDEE.

Tabela 3.6: Dados a enviar para registar um utilizador

Variável	Tipo de Variável	Descrição
nome_apelido_utilizador	*text*	Nome e apelido do utilizador
sigla	*text*	Sigla do utilizador (3 letras no caso de ser um elemento da direção ou um docente; 4 letras no caso de ser um técnico)
id_tipo_utilizador	*int*	Tipo de utilizador (direção, técnico, docente)
gabinete	*text*	Número de instalação onde se localiza o gabinete do utilizador
ext_telefonica	*int*	Extensão telefónica do utilizador

3.3.2 Consultar informações sobre os utilizadores

Para consultar as informações de todos os utilizadores registados no SIGEDEE, o cliente deve efetuar um pedido *GET* para o *endpoint URL/utilizadores* da API do DEE. Tal como nos casos anteriores, pode limitar os resultados obtidos enviando como parâmetro a variável limite, indicando o número de utilizadores que pretende consultar. Além disso, existe também a possibilidade de consultar os dados detalhados de um determinado utilizador, indicando o respetivo ID (*GET URL/utilizadores/{id_utilizador}*.

3.3.3 Editar informações de um utilizador

Tal como referido previamente, é também possível alterar o tipo de utilizador ou desativá-lo.

Alterar tipo de um utilizador

Para alterar o tipo de um utilizador, isto é, alterar a sua função no DEE (direção, docente, técnico), o cliente deverá efetuar um pedido *PUT* para o *endpoint URL/utilizadores/{id_utilizador}/tipo* e enviar em formato JSON, o ID referente ao novo tipo de utilizador (*{"novo_tipo_utilizador":{id_tipo_utilizador}}*).

Após receber o pedido, o *backend* irá obter o tipo de utilizador inicial e a sigla do utilizador em causa. Tal como no processo de registo de um utilizador, aqui também é verificado se a sigla corresponde ao novo tipo de utilizador pretendido. Isto é, um docente não pode ser convertido num técnico e vice-versa. Esta verificação é também efetuada quando se pretende reativar um utilizador.

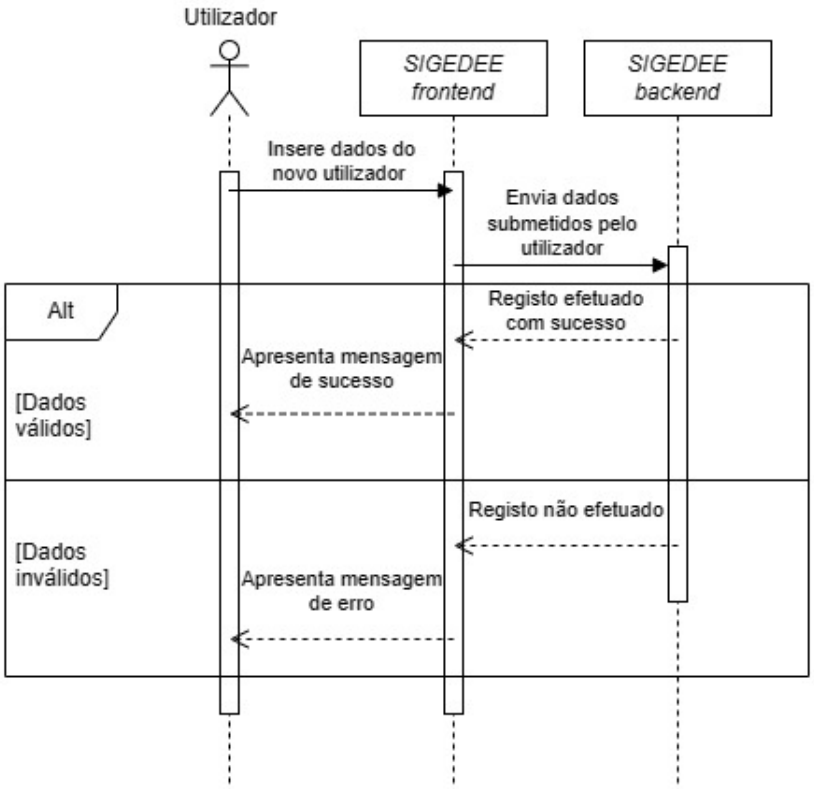


Figura 3.7: Registo de um utilizador no SIGEDEE

Por outro lado, é também exigido que exista sempre, pelo menos, um elemento da direção registado. Deste modo, sempre que é efetuado um pedido para despromover algum utilizador da direção, é verificado se existe pelo menos um elemento da direção registo. Caso não exista, a alteração não será efetuada. A Figura 3.8 ilustra o processo de verificação efetuado quando se pretende alterar o tipo de utilizador de um elemento da direção do DEE.

Desativar um utilizador

Caso pretenda desativar um utilizador, o cliente deverá efetuar um pedido *PUT* para a API, indicando o ID do utilizador que pretende eliminar (*PUT URL/utilizadores/{id_utilizador}/desativar*). Após um utilizador ser desativado, este não será capaz de aceder ao SIGEDEE.

3.4 Documentação da API e da Base de Dados

De modo a possuir um registo completo de todas as funcionalidades do SIGEDEE, procedeu-se à documentação da API do SIGEDEE e das suas bases de dados.

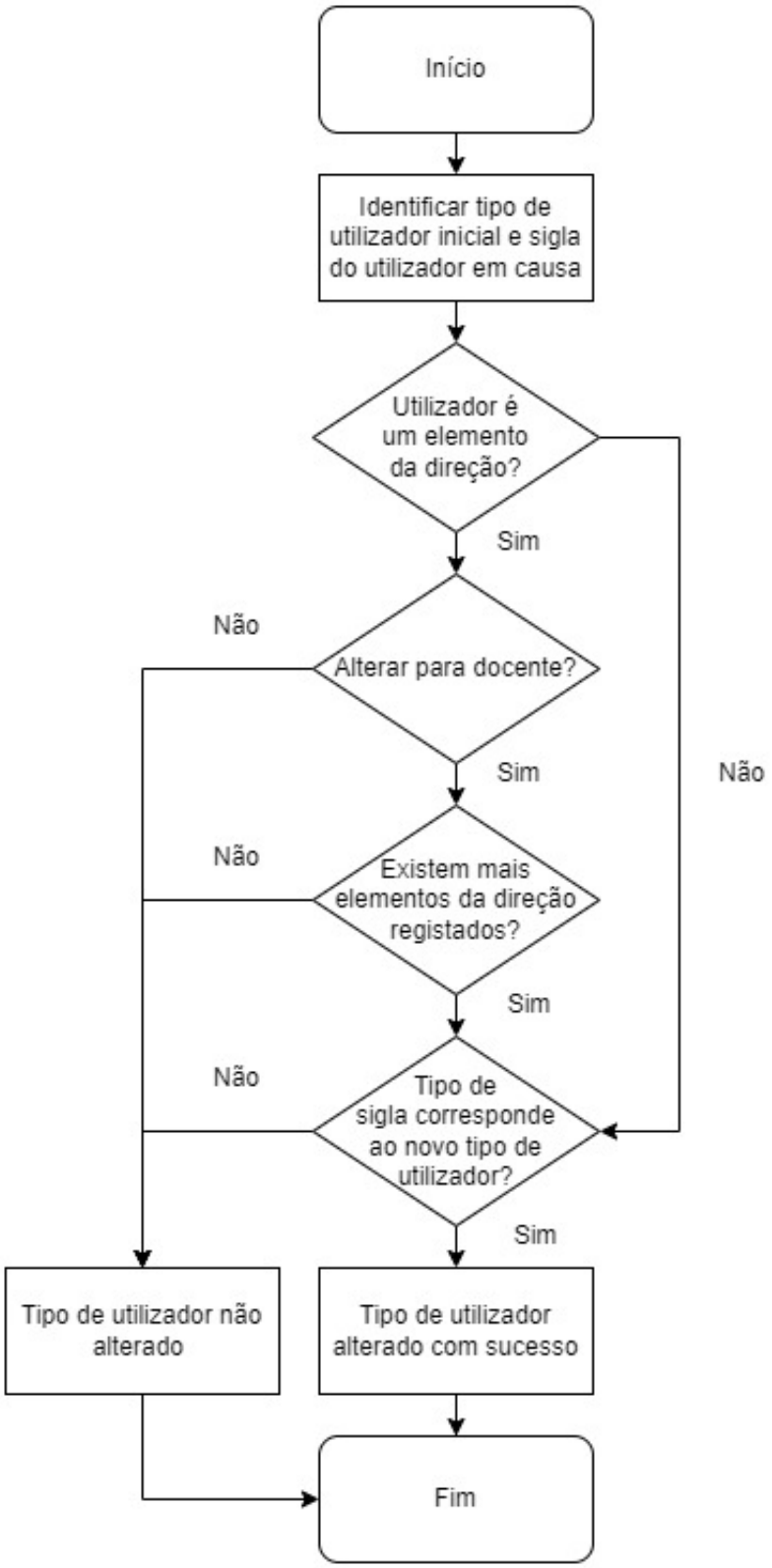


Figura 3.8: Processo de verificação de edição de um utilizador

A documentação da API foi efetuada através do *Postman*, enquanto a do MariaDB se obteve através da plataforma phpMyAdmin. Por outro lado, a base de dados Immudb, posteriormente apresentada, foi documentada manualmente.

Todas a documentação gerada pode ser consultada no Anexo A da presente dissertação.

Capítulo 4

Garantia de Não Repudiação e de Integridade dos Dados

No início da realização da presente dissertação, o SIGEDEE possuía uma arquitetura simples composta por um único servidor que permitia a inserção e edição de equipamentos, fornecedores e instalações, tal como ilustrado na Figura 4.1, cujas setas indicam o sentido em que os pedidos são iniciados, sendo que, em ambos os casos, a resposta ocorre no sentido contrário. Além disso, disponibilizava também um sistema de *backup*, fazendo uma cópia de um registo antes de o mesmo ser alterado, possibilitando que alterações indesejadas fossem revertidas. Contudo, este sistema tinha algumas lacunas. Por exemplo, os registos inseridos podiam ser facilmente alterados através de uma edição efetuada diretamente na base de dados por um utilizador com as devidas permissões, ou por um utilizador com acesso indevido, fazendo, assim, com que a inserção, ou edição, de registos fosse facilmente repudiável, pois, uma vez que não existia um mecanismo que garantisse a não repudiação, qualquer utilizador poderia alegar que tal alteração não teria sido feita por si.

De forma a combater estas falhas, exploraram-se diversas soluções que serão descritas no presente capítulo.

4.1 Assinatura digital

Tal como referido anteriormente, um dos grandes problemas que se pretendia resolver no SIGEDEE era a possibilidade de repudiar quaisquer ações efetuadas na

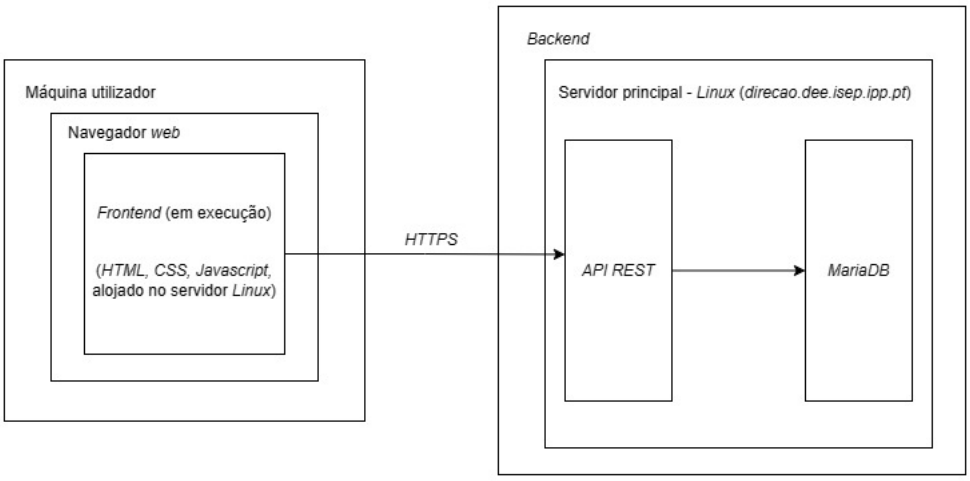


Figura 4.1: Arquitetura inicial do SIGEDEE

plataforma. Assim, de modo a resolver o problema de não repudiação presente no SIGEDEE no início da realização desta dissertação, foi introduzido um processo de assinatura digital por parte dos utilizadores que inserem novos registos na base de dados, com base num par de chaves, pública e privada, de cada utilizador. Este sistema foi implementado com a finalidade de garantir a não repudiação da inserção de registos no SIGEDEE.

4.1.1 Web Crypto API

Para implementar a assinatura digital no SIGEDEE, numa fase inicial, estudou-se a possibilidade de utilizar a biblioteca Web Crypto API, que seria responsável por gerar e armazenar os pares de chaves do utilizador e assinar os dados, com a respetiva chave privada. Estas ações seriam realizadas no *browser* do utilizador, sendo o código corrido na interface gráfica do SIGEDEE, desenvolvida em Javascript.

Apesar do par de chaves ser armazenado no *browser* do utilizador, o *frontend* do SIGEDEE é responsável por executar o código relacionado com a biblioteca Web Crypto API. Deste modo, poderiam ser levantadas algumas questões de segurança por parte dos utilizadores, podendo estes alegar que a sua chave privada nunca seria totalmente privada, uma vez que o servidor do SIGEDEE poderia ter acesso à mesma, colocando a sua segurança em causa. Assim, foi excluída a opção de utilizar esta biblioteca

De modo a contornar todas as questões relacionadas com este cenário, procedeu-se ao desenvolvimento de um sistema de assinatura digital próprio para uso exclusivo dos utilizadores do SIGEDEE.

4.1.2 Sistema de assinatura digital do SIGEDEE

Após se estudar o uso da biblioteca Web Crypto API para ser possível assinar digitalmente os dados inseridos na base de dados, decidiu-se não utilizar essa biblioteca, mas sim proceder-se ao desenvolvimento e implementação de um sistema de assinatura digital próprio do DEE, utilizando uma aplicação desenvolvida para este caso, a DEEsign. Este sistema foi desenvolvido em colaboração com o bolseiro Ângelo Pinheiro (*geloe@isep.ipp.pt*) e com os orientadores desta dissertação. Os pares de chaves gerados por esta aplicação são armazenados localmente, isto é, no dispositivo do utilizador, sendo este responsável por garantir a segurança do seu par de chaves.

De modo a ser possível efetuar qualquer inserção de um novo registo, através de inserção ou edição de dados, o utilizador deverá iniciar a aplicação DEEsign, caso contrário não conseguirá efetuar esta ação. Após tentar inserir ou editar um equipamento, fornecedor ou instalação, o cliente *web* do SIGEDEE irá enviar os dados em formato JSON para a aplicação DEEsign, que é executada localmente, através do porto 9010. Após receber os dados, a aplicação local irá abrir uma nova janela no *browser* do utilizador, questionando se este pretende usar o seu par de chaves para assinar os dados a inserir (Figura 4.2). Caso recuse, o processo é terminado. Por outro lado, caso aceite, a aplicação irá verificar se ambas as chaves já foram geradas e estão armazenadas no computador do utilizador. Se não existir um par de chaves, isto é não existir nenhuma ou apenas uma chave, será criado um novo par. Na versão atual da DEEsign, não existe uma forma segura e automática de garantir que um par de chaves pertence ao utilizador que as gerou, tendo este processo que ser realizado manualmente por cada utilizador, informando a direção via *e-mail* da criação e registo de uma nova chave pública. Após efetuar a verificação relativa ao par de chaves, a DEEsign assina o resumo dos dados com a chave privada do utilizador, utilizando o algoritmo de *hash* SHA-256 e o algoritmo de assinatura RSA, e envia para o cliente *web* do SIGEDEE um JSON com dois campos:

- *bulk* - este campo é constituído por dois campos, referentes aos dados inseridos:
  - *data* - dados originais inseridos pelo utilizador no formato JSON;
  - *signature* - resumo dos dados assinados pelo utilizador, recorrendo à sua chave privada;
- *exportedKey* - chave pública do utilizador no formato *pem*, correspondente à chave privada utilizada na assinatura dos dados.

Após receber estes dados, o *frontend* do SIGEDEE envia os dados para o *backend* de modo a serem verificados e inseridos na base de dados, caso estejam válidos. Posteriormente, utilizando os parâmetros recebidos, o *backend* irá verificar se a assinatura se encontra válida. Primeiramente é verificado se a chave pública recebida

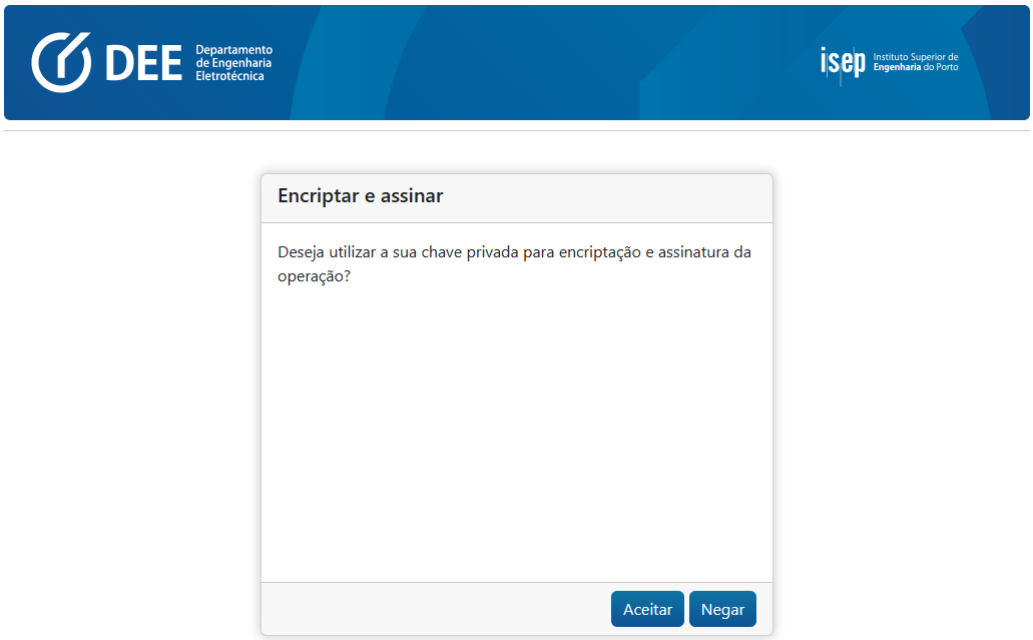


Figura 4.2: Processo de *login* com *e-mail*

já existe na base de dados. Se não existir, é inserida na base de dados, indicando o utilizador a que pertence, e se existir mas estiver registada como sendo de outro utilizador, o processo de inserção é abortado. De seguida, é efetuada a validação dos dados, recorrendo aos parâmetros recebidos do *frontend* e à função *openssl_verify()* do PHP. Caso sejam validados com sucesso, um novo registo, referente a uma inserção ou edição, será inserido na base de dados. Caso contrário, será retornada uma mensagem de erro. Esta verificação apenas é efetuada quando ocorre uma inserção ou edição de um equipamento, fornecedor ou instalação na base de dados, ou eliminação de um equipamento. Caso contrário, não será necessário realizar esta verificação, uma vez que os dados não são assinados.

A Figura 4.3 demonstra a lógica utilizada no processo de inserção de um registo digitalmente assinado de um equipamento, fornecedor ou instalação na base de dados.

Deste modo, com a introdução de um sistema de assinatura digital, será mais difícil um utilizador repudiar ações realizadas no SIGEDEE, uma vez que os dados são assinados, recorrendo a um par de chaves de cada utilizador, sendo possível identificar o autor de cada ação.

4.2 Adoção de uma rede *blockchain*

Uma vez que o SIGEDEE permitia que os seus registos fossem corrompidos, por parte de um administrador ou de um utilizador com acesso indevido à base de dados, era

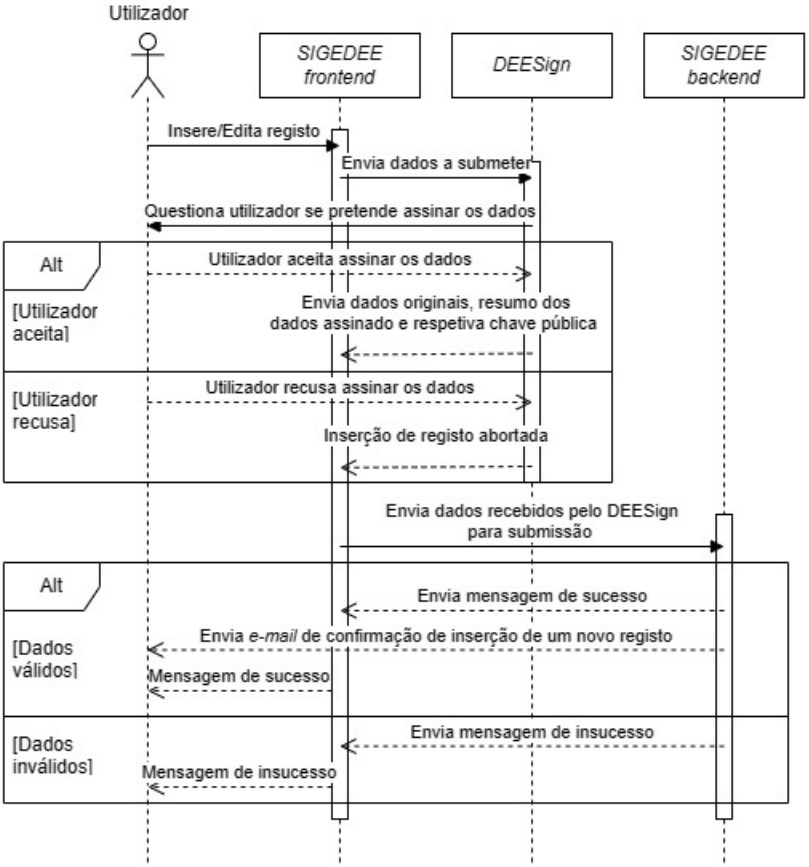


Figura 4.3: Lógica de inserção de registo digitalmente assinado de equipamento, fornecedor ou instalação nas bases de dados

necessário tentar implementar um sistema imutável no DEE. Assim, explorou-se a possibilidade de implementar uma rede *blockchain*, visto que se trata de uma rede que apenas aceita a inserção de novos registos, através de um consenso atingido por todos os nós da rede e sendo, ainda, extremamente complicado corromper quaisquer registos presentes na mesma. Além disso, este sistema introduzia o fator de não repudiação, algo também pretendido pelo DEE.

Assim, após efetuar a revisão bibliográfica apresentada na Secção 2.2.5, concluiu-se que para desenvolver uma rede *blockchain* privada, que seria o caso do DEE, a plataforma Hyperledger Fabric seria a melhor opção, uma vez que apresenta um melhor desempenho comparativamente às restantes plataformas estudadas.

Após serem realizados alguns testes para explorar a plataforma, percebeu-se que esta não seria a solução ideal para o problema em estudo, pois iria-se aplicar uma solução demasiado exigente e complexa para um sistema simples e, além disso, existia uma incerteza de quantos computadores poderiam ser utilizados para ingressar nesta rede, estimando-se que estariam disponíveis cerca de três computadores.

Deste modo, a adoção de uma rede *blockchain* como solução da falta de imutabilidade do SIGEDEE foi abortada devido aos seguintes fatores:

- Número de nós - incerteza quanto ao número de nós disponíveis para integrar a rede. Era garantida a disponibilidade de dois computadores, podendo estender-se a três, ou, no limite, quatro unidades, o que seria um número relativamente baixo para criar uma rede *blockchain*, devido ao mecanismo de consenso. Com um baixo número de unidades na rede, a falha de um nó teria um impacto muito maior do que teria numa rede com um elevado número de nós;

- Manutenção da rede a longo prazo - a implementação de um sistema *blockchain* trazia diversas questões sobre a sua manutenção a longo prazo, uma vez que, atualmente, não é uma área explorada no DEE, não havendo, assim, muitos docentes, ou técnicos, que estejam familiarizados com esta tecnologia.

4.3 Sistema de registo de transações

Após a rejeição de uma abordagem baseada numa plataforma *blockchain*, foram analisadas as restantes opções que permitissem criar um sistema de inserção de dados imutável. Após a análise das diferentes soluções de bases de dados imutáveis, decidiu-se optar pelo Immudb, uma vez que esta oferece a possibilidade de criar um sistema imutável *key-value* ou SQL. Deste modo, foi desenvolvido um sistema de inserção de registos imutável combinando as tecnologias MariaDB e Immudb. Nesta secção, será descrita a solução desenvolvida.

4.3.1 Reestruturação do MariaDB

Apesar de disponibilizar diversas tabelas, a integração de um sistema de inserção de registos imutável apenas será feita nas tabelas relativamente a equipamentos, fornecedores e instalações da base de dados do DEE. Numa fase inicial, o SIGEDEE disponibilizava um sistema rudimentar de histórico de operações para esta tabela, na altura denominado de *backup*. Tal era alcançado tendo uma tabela que continha o registo mais recente e outra tabela com todos os registos que tenham sido alterados, como, por exemplo, as tabelas *equipamentos* e *equipamentos_backup*, respetivamente. A Figura 4.4 demonstra as tabelas referidas na base de dados numa fase inicial.

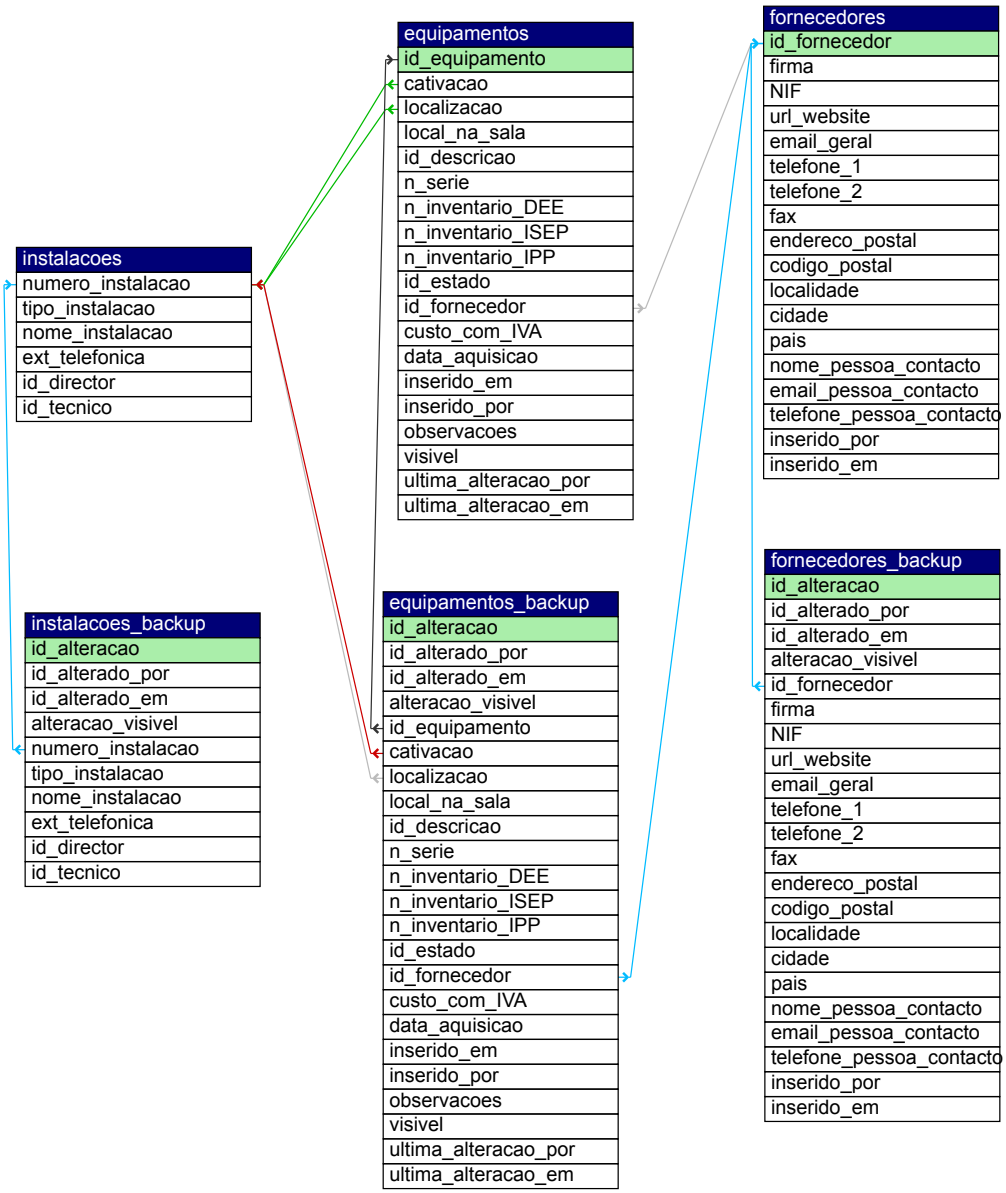


Figura 4.4: Tabelas *equipamentos*, *fornecedores* e *instalacoes* e respetivas tabelas de *backup*

O primeiro passo efetuado no desenvolvimento deste sistema, foi reformular a estrutura destas tabelas, passando agora a ter três tabelas distintas para cada um dos casos. A tabela equipamentos será utilizada como referência, sendo a mesma explicação aplicada às restantes tabelas:

- *equipamentos* - esta tabela possui apenas quatro colunas, uma referente ao ID do equipamento (*primary key* incrementada em cada inserção de um novo equipamento), outra que indica o registo mais recente referente ao equipamento e as restantes contêm informações sobre quando este foi introduzido na base de dados e por quem;

- *equipamentos_historico* - tabela de histórico dos estados de cada um dos equipamentos. Ou seja, sempre que existe uma alteração num dos atributos de um equipamento, é inserido um novo registo correspondente a esse equipamento, contendo quer os valores dos atributos alterados, quer os valores dos restantes atributos. Além de todas as colunas referentes às características de cada de equipamento, possui uma coluna a indicar o número do registo (*primary key* incrementada a cada nova inserção), outra referente ao ID do equipamento correspondente ao registo e mais duas que indicam o utilizador responsável pela operação e a respetiva marcação temporal;

- *equipamentos_transacoes* - nesta tabela, denominada tabela de transações, são armazenadas as informações recebidas diretamente pelo cliente, que serão necessárias para validar os dados sempre que necessário, recorrendo à assinatura digital. As primeiras duas colunas armazenam o ID da transação (*primary key* incrementada a cada nova inserção) e o ID do respetivo registo da tabela de histórico. As restantes colunas desta tabela armazenam todas as informações relativamente aos dados recebidos pelo cliente, necessários para a verificação da integridade dos dados através da assinatura digital, ou seja, uma coluna contém os dados recebidos em formato JSON, outra armazena o resumo dos dados originais assinado com a chave privada do utilizador e a última indica a chave pública correspondente à chave privada. As chaves públicas são armazenadas numa tabela própria indicando a que utilizador pertencem e quando foram criadas.

A Figura 4.5 ilustra a estrutura das tabelas previamente descritas na base de dados e as suas relações.

Por uma questão de redundância, foram ainda adicionados dois *triggers* a cada tabela que impedem a edição e eliminação de registos através de comandos SQL. Naturalmente, este mecanismo não impede que um administrador desative os *triggers* e efetue as alterações que pretender na base de dados. Ou seja, ainda é possível corromper os registos na base de dados, apesar de a nova estrutura facilitar a sua

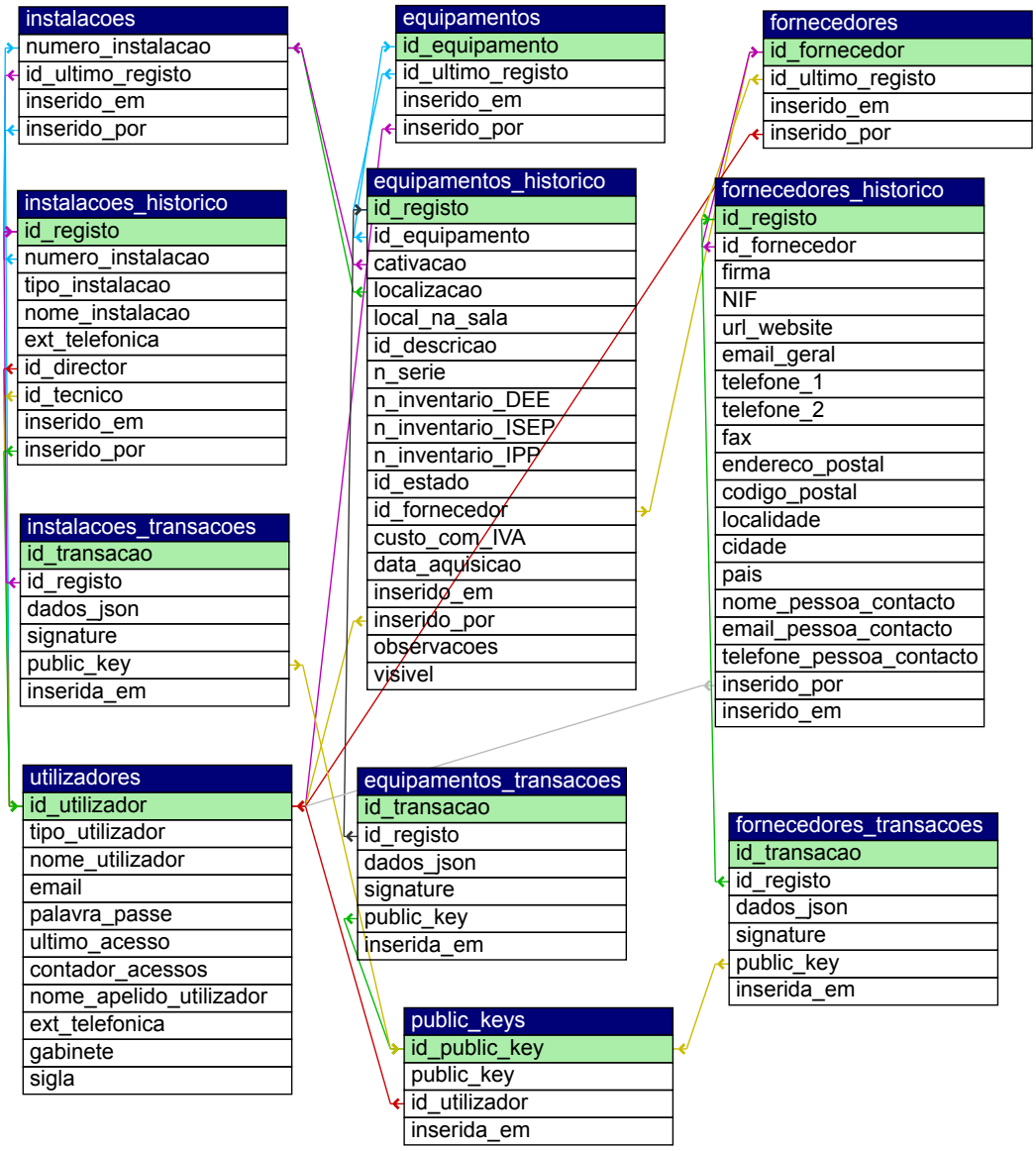


Figura 4.5: Estrutura das tabelas da base de dados no sistema de registo de transações

deteção, através da assinatura digital dos dados armazenados. Por este motivo, é necessário ter pelo menos uma cópia de segurança dos registos numa base de dados imutável, recorrendo-se assim ao Immudb.

4.3.2 Réplica Immudb

O Immudb é um sistema de gestão de base de dados imutável que oferece a possibilidade de trabalhar com bases de dados *key-value* ou SQL. Foi utilizada a versão mais recente deste *software* à data de escrita desta dissertação, a versão 1.5.0, disponibilizada a 21 de junho de 2023. A opção de criar uma base de dados SQL imutável seria a mais desejável. Contudo, a versão atual SQL do Immudb ainda não se encontra numa fase que permita implementar eficientemente todas as funcionalidades do *backend* do SIGEDEE, uma vez que, por exemplo, atualmente este sistema apenas disponibiliza quatro tipos de variáveis diferentes (*INTEGER*, *VARCHAR*, *BOOLEAN* e *BLOB*) [49], não sendo capaz de satisfazer as necessidades do SIGEDEE. Assim, o Immudb será utilizado como uma base de dados *key-value* secundária, que irá armazenar todos os registos inseridos referentes a equipamentos, instalações e fornecedores, bem como as chaves públicas dos utilizadores, de forma imutável.

Existem dois métodos possíveis para a implementação do Immudb: ou são implementadas várias réplicas ou então apenas uma réplica controlada pelo Immuaudit. No primeiro cenário, poderão ser implementadas várias réplicas do Immudb distribuídas pelos computadores dos elementos da direção do DEE, utilizando a replicação síncrona do Immudb. Para tal, deverão ser definidos os números de réplicas que devem confirmar uma transação antes desta ser efetuada, através do comando *–replication-sync-acks*. A empresa responsável pelo Immudb recomenda que o número de confirmações seja, pelo menos, igual à metade do total de réplicas presentes na rede [49]. Por exemplo, se existirem quatro réplicas, deverá ser necessário obter a confirmação de pelo menos duas máquinas antes de se proceder à inserção de um novo registo. A latência da inserção de dados no Immudb é diretamente proporcional ao número de réplicas da rede, quantos mais réplicas existirem, mais tempo demorará a efetuar uma ação de escrita e, caso o número de confirmações não seja atingida devido a, por exemplo, algumas réplicas estarem desligadas, a transação síncrona será automaticamente bloqueada, não sendo o registo inserido na base de dados [49].

A segunda opção consiste em executar uma única cópia do Immudb e monitorizar o estado da respetiva base de dados através da aplicação Immuadit, incluída no Immuclient. É aconselhável que o Immuaudit seja executado numa máquina distinta de onde está a ser executada o Immudb [49]. Assim, o Immuaudit será capaz de detetar se os registos da base de dados foram corrompidos, notificando o utilizador assim que isso aconteça. Deste modo, a integridade dos dados armazenados no

Immudb é assegurada, apesar de não ser possível reverter eventuais corrupções dos dados, uma vez que, em princípio, não haverá nenhuma cópia dos dados que não tenha sido corrompida.

Numa fase inicial, apenas será implementada a segunda opção, uma vez que, para a realização desta dissertação, apenas estavam disponíveis dois servidores, sendo arriscado implementar uma replicação síncrona, uma vez que bastava uma máquina falhar para a replicação ser cancelada. Ou seja, o Immudb estará alojado num servidor distinto do MariaDB. A Figura 4.6 ilustra nova arquitetura do SIGEDEE, contendo a sua aplicação DEEsign, responsável por assinar os dados submetidos pelo utilizador, e o novo servidor dedicado ao Immudb, que irá funcionar como réplica dos dados armazenados no MariaDB.

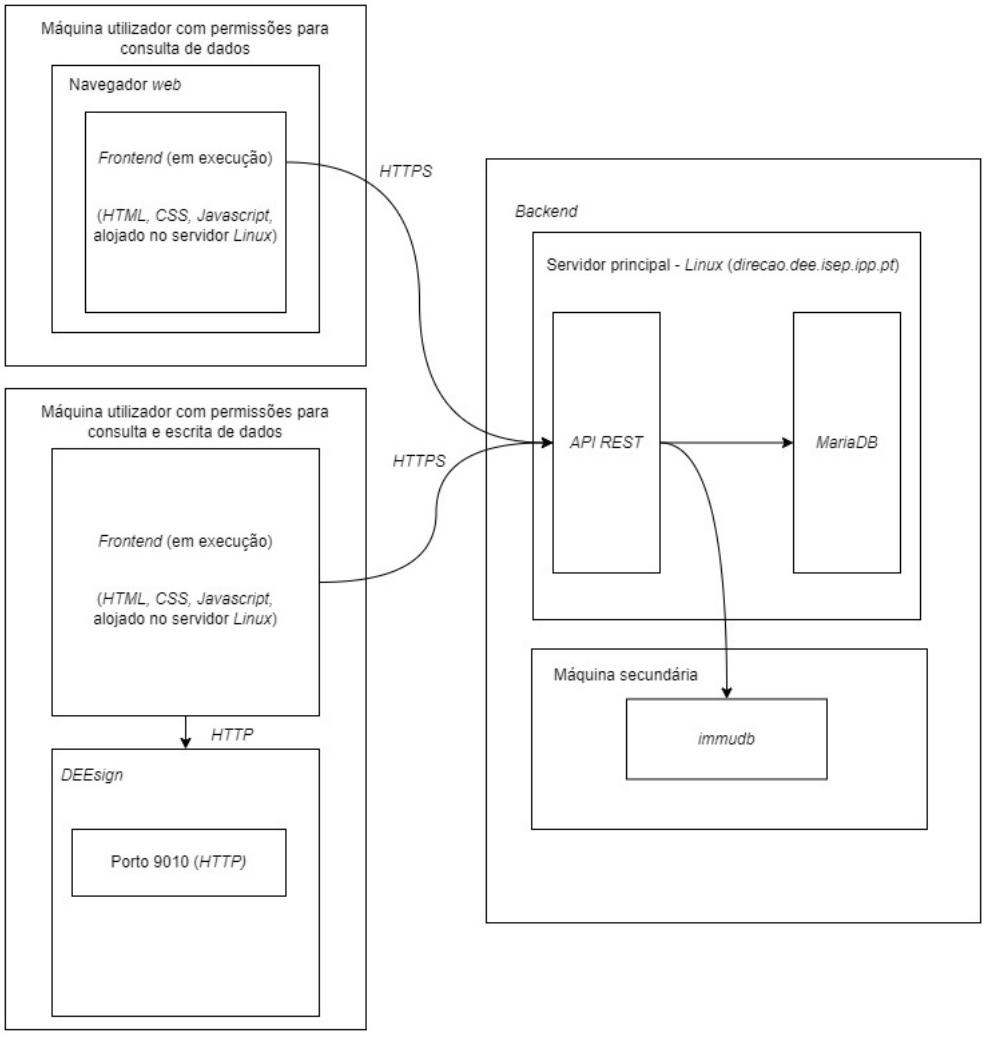


Figura 4.6: Arquitetura final do SIGEDEE

Quando utilizada como uma base de dados *key-value*, o Immudb permite consultar o histórico dos registos inseridos para cada *key*. Assim, a inserção de registos nesta base de dados seguirá uma lógica onde cada *key* é uma junção do nome da

tabela com o ID do elemento correspondente (por exemplo, *equipamentos_1*), tal como descrito na Tabela A.1 do Anexo A. No caso das chaves públicas, as chaves de cada utilizador serão armazenadas numa *key* dedicada, indicando o utilizador a quem pertence (por exemplo, *publicKey_DEE*). Este campo das chaves públicas armazena todas as chaves públicas registadas de cada utilizador, de modo a ser mais fácil consultar todas as chaves de um utilizador. Quando é inserido um equipamento, instalação ou fornecedor, são armazenados os mesmos dados que são guardados nas tabelas de transações do MariaDB, ou seja, os dados enviados pelo cliente em formato JSON e o o resumo desses mesmos dados assinado com a chave privada do utilizador. Serão também armazenados a chave pública do utilizador e a sua sigla, bem como os IDs referentes à transação e ao registo em causa, e a data em que foi inserido no Immudb e a data correspondente à inserção do primeiro registo referente ao elemento correspondente, como, por exemplo, um equipamento, no MariaDB. Estes dados serão automaticamente inseridos sempre que seja criado um novo registo no MariaDB, de modo a garantir a existência de uma cópia de segurança.

A Figura 4.7 ilustra o procedimento de inserção de um novo registo de equipamento, fornecedor ou instalação no MariaDB e no Immudb.

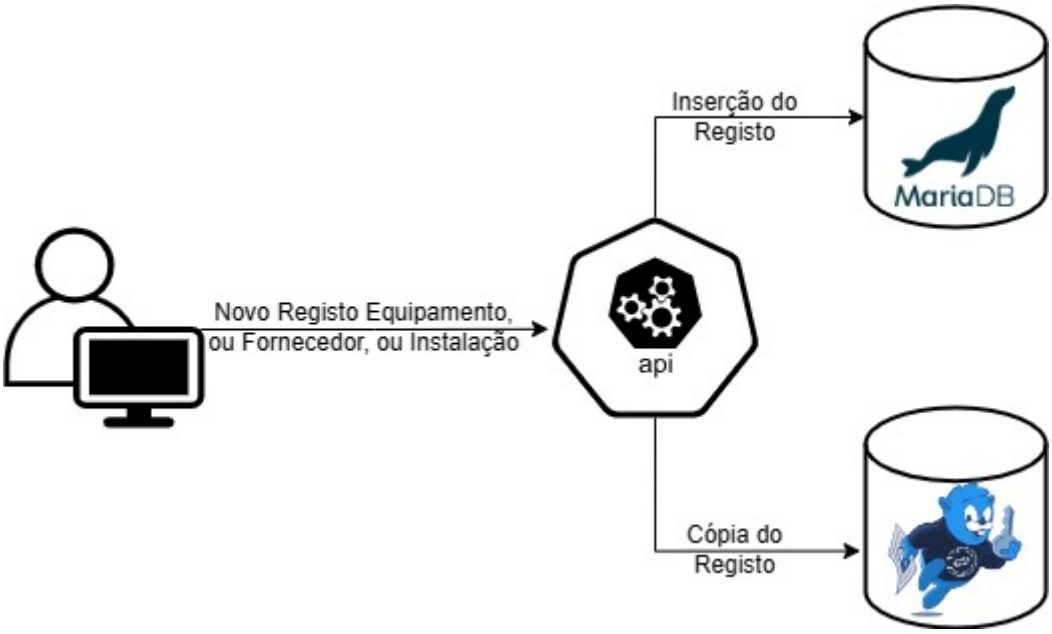


Figura 4.7: Inserção de registos no MariaDB e no Immudb

Utilizando esta arquitetura, caso os registos do SIGEDEE sejam corrompidos, tal acontecimento será facilmente detetado, uma vez que os dados armazenados no Immudb dificilmente serão alterados e, caso alterem os dados presentes no MariaDB, estes não serão idênticos aos dados armazenados no Immudb, indicando que foram corrompidos.

4.4 Sistema de envio de *e-mails*

Tal como referido diversas vezes ao longo da dissertação, diversas funcionalidades foram complementadas com um envio de *e-mail* para os utilizadores. Este sistema sistema de envio de *e-mails* para os utilizadores da plataforma foi implementado recorrendo à função *mail()* do PHP. Nesta secção, será descrito o motivo de implementação deste sistema e a sua relevância para complementar um sistema que pretende garantir a não repudiação e integridade dos dados armazenados.

Sempre que um utilizador insira um novo registo relativo à inserção ou edição de um equipamento, fornecedor ou instalação, receberá um *e-mail*, indicando que foi efetuada a inserção de um novo registo, descrevendo os detalhes do mesmo. O utilizador deverá guardar estes *e-mails*, uma vez que funcionam como um comprovativo de inserção ou edição de dados no SIGEDEE. Estes *e-mails* serão também enviados para a direção do departamento.

Além disso, este sistema desempenha um papel importante no sistema de *helpdesk*. Após ser inserido um *ticket*, serão automaticamente enviados três *e-mails* a confirmar que a avaria foi reportada: um *e-mail* para o utilizador que reportou a avaria, outro para a direção do DEE e um terceiro para o técnico responsável pela instalação onde o equipamento se encontra. Sempre que o estado do *ticket* for atualizado, a direção do DEE e o utilizador que reportou a avaria serão notificados por *e-mail*, sendo que a direção do DEE será informada sobre quem efetuou tal alteração. Por outro lado, assim que seja inserida uma observação, os intervenientes do *ticket* serão informados via *e-mail*. Uma vez que este sistema não utiliza a aplicação DEEsign para inserir, ou atualizar, *tickets*, estes *e-mails* serão importantes, uma vez que servirão como comprovativo de reparação efetuada pelo técnico responsável.

O envio de *e-mails* será também utilizado no sistema de requisição de equipamentos. Assim que seja inserido um pedido de empréstimo de um equipamento na base de dados, será enviado um *e-mail* de confirmação para o requerente e outro para a direção do DEE de modo a notificá-la sobre o mesmo. Sempre que o estado do pedido de movimento for alterado, o utilizador que efetuou o pedido será notificado por *e-mail*. Caso a decisão tomada indique que o pedido foi aceite ou que o equipamento foi emprestado ou devolvido, o técnico responsável pela instalação original do equipamento receberá também um *e-mail*. Assim, neste sistema, os *e-mails* são de elevada importância, uma vez que servem de comprovativo de levantamento e devolução de equipamento, quer para o requerente quer para o utilizador que irá emprestar o equipamento. Esta solução foi implementada, pois, apesar de a solução ideal ser através da assinatura digital, à data de escrita da dissertação, o DEEsign carece de uma aplicação para *smartphones*, não sendo prático um utilizador recorrer ao computador pessoal no ato de levantamento, ou devolução, de um equipamento para confirmar a ação realizada.

Os utilizadores do SIGEDEE deverão guardar os *e-mails* recebidos por cada ação realizada na plataforma, uma vez que estes servem de comprovativo da mesma. Caso exista uma atualização de um *ticket* do *helpdesk* ou de um registo de requisição de equipamento que não possua um único *e-mail* que comprove tal ação, tal poderá indicar que ocorreu uma corrupção dos dados presentes na base de dados. Além disso, o envio de *e-mails* acaba por providenciar um sistema de notificações, alertando um utilizador sempre que seja necessária a sua intervenção na plataforma.

Assim, estes *e-mails* têm o papel de um recibo, podendo o utilizador recorrer aos mesmos como prova de que realizou a respetiva operação. Ou seja, são uma forma de o sistema não poder repudiar a operação do utilizador. No fundo, esta é uma forma de reforçar a imutabilidade dos dados, apesar do seu baixo nível de automatização.

4.5 Verificar integridade dos registos

De modo a concluir o sistema de registo de transações do SIGEDEE, foram implementados alguns mecanismos que possibilitam verificar se algum registo presente na base de dados do DEE foi corrompido ou não. Os *scripts* apresentados nesta secção devem ser executados manualmente através da linha de comandos da mesma máquina que incorpora o *backend* do SIGEDEE.

4.5.1 Validar dados MariaDB

Tal como descrito ao longo deste capítulo, na base de dados do SIGEDEE presente no MariaDB existem três tabelas para cada caso (equipamentos, fornecedores ou insatalações) que estão relacionadas entre si, como por exemplo, as tabelas *equipamentos*, *equipamentos_historico* e *equipamentos_transacoes*. Assim, foi desenvolvido um *script* PHP que irá verificar se os dados presentes nestas tabelas não foram corrompidos, recorrendo ao sistema de assinatura digital.

Primeiramente, o utilizador escolherá que tabela pretende analisar: *equipamentos*, *fornecedores*, *instalacoes* ou todas. De seguida, o *script* irá efetuar um pedido à base de dados obtendo as colunas *id_transacao*, *id_registo*, *dados_json*, *signature* e *public_key* da tabela transações. Além disso, obtém também os valores de todas as colunas da tabela histórico referentes aos registos encontrados na tabela de transações. Serão obtidos todos os valores presentes na base de dados.

De seguida, os registos serão analisados um a um. Primeiramente, verifica-se se a assinatura dos dados ainda se encontra válida, recorrendo aos valores retirados das colunas *dados_json*, *signature* e *public_key* da tabela de transações, isto é, dos dados enviados pelo cliente quando é efetuada a inserção ou edição de um registo (dados originais em formato JSON, resumo dos dados assinados com a chave privada do utilizador e sua chave pública). Caso a assinatura não esteja válida, a transação analisada é contabilizada como inválida e passa-se para a transação seguinte. Caso

contrário, os dados armazenados em formato JSON na tabela de transações são comparados com os dados presentes na tabela de histórico com o ID do registo correspondente. Se estes dados forem diferentes, considera-se que os dados foram corrompidos e passa-se para o registo seguinte.

Finalmente, quando este processo terminar, o utilizador receberá uma mensagem indicando o número de transações analisadas e quantas foram corrompidas. Caso não seja possível analisar todas as transações da base de dados, será emitido um erro.

A Figura 4.8 demonstra o funcionamento do *script* de verificação dos dados da base de dados MariaDB.

4.5.2 Validar dados Immudb

Além de se ter desenvolvido um *script* para analisar os registos do MariaDB, procedeu-se ao desenvolvimento de dois *scripts* de modo a validar os dados armazenados no Immudb, um em PHP, de modo a aproveitar o código de consulta de dados da base de dados MariaDB, e outro em Python, para ser possível consultar os dados armazenados no Immudb.

Tal como na fase anterior, o utilizador deverá selecionar que tabelas pretende analisar. De seguida, as tabelas serão analisadas e, numa fase inicial, será obtido o ID mais elevado da *primary key* da tabela de transações e da tabela original, por exemplo, nas tabelas *equipamentos_transacoes* e *equipamentos*, respetivamente. De seguida, serão analisadas todas as transações, de 1 até ao ID de transação mais elevado presente na base de dados. Deste modo, é efetuada uma consulta à base de dados de forma a obter os resultados correspondentes ao ID em estudo. Caso esse ID exista na base de dados, os dados (*id_transacao*, *id_registo*, *dados_json*, *public_key*, *signature*, *sigla*, *inserido_em* e *$pk*) são enviados como parâmetro de um comando executado na linha de comandos para o *script* Python, de modo a comparar os valores presentes no MariaDB com aqueles armazenados no Immudb. Assim, será verificado se existe algum registo no Immudb que corresponda ao registo do MariaDB e, caso exista, os dados serão comparados. Caso os dados coincidam, será analisado o próximo registo. Caso contrário, o registo em causa será registado como corrompido, antes de se analisar o registo seguinte.

Por outro lado, caso não seja encontrado nenhum registo no MariaDB com o respetivo ID de transação, serão analisados os registos de todos os equipamentos um a um presentes no Immudb. Caso exista um registo de um equipamento com o ID de transação em falta no Immudb, significa que esse mesmo registo foi eliminado da base de dados MariaDB. Por exemplo, se existirem quatro registos armazenados no MariaDB e os seus IDs forem 1, 2, 3 e 5, tal poderá significar que o registo número 4 foi eliminado. Caso esse registo esteja presente no Immudb, significa que foi eliminado do MariaDB.

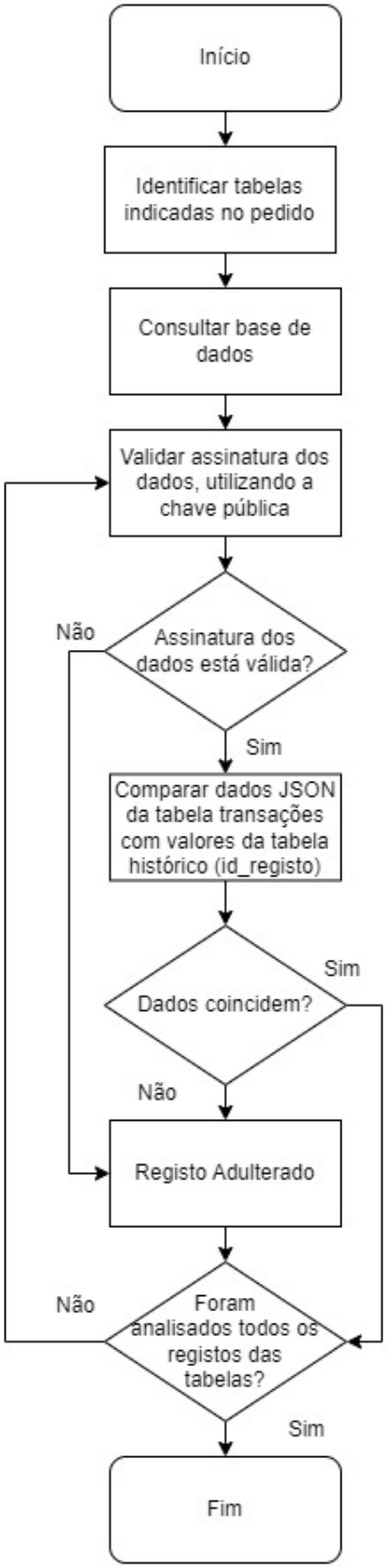


Figura 4.8: Verificação dos registos da base de dados MariaDB

No final deste processo, os utilizadores receberão uma mensagem indicando o número de registos analisados e quantos registos foram corrompidos ou eliminados, por cada tabela.

A Figura 4.8 ilustra o funcionamento do *script* de verificação dos dados da base de dados Immudb.

Além disso, poderá também ser utilizada a ferramenta do Immudb, o Immuaudit, para averiguar se os registos da base de dados foram ou não corrompidos.

4.5.3 Sincronizar dados entre MariaDB e Immudb

De modo a evitar discrepâncias entre os registos das duas bases de dados, foram ainda desenvolvidos dois scripts, em PHP e Python, com o intuito de as sincronizar, ou seja, inserir registos no Immudb, caso estes existam apenas no MariaDB.

Tal como nos cenários anteriores, o utilizador também tem a hipótese de selecionar quais tabelas pretende sincronizar. Após selecionar a tabela pretendida, são obtidos todos os registos presentes no MariaDB, sendo analisados individualmente. Posteriormente, são enviados, como parâmetros de um comando executado na linha de comandos, para o *script* Python os dados necessários para a inserção de um registo na base de dados Immudb (já descritos anteriormente). De seguida, é verificado se existe algum registo armazenado no Immudb referente ao ID referido, por exemplo *equipamentos_1*, e, caso exista, se existe algum registo que contenha os mesmos valores de *id_transacao* e *id_registo*. Caso não seja encontrado nenhum resultado, será automaticamente inserido o registo em falta na base de dados Immudb.

Assim que o processo terminar, o utilizador será informado do número de registos analisados, bem como do número de registos inseridos no Immudb.

A Figura 4.10 demonstra o funcionamento do *script* de sincronização dos dados entre as bases de dados MariaDB e Immudb.

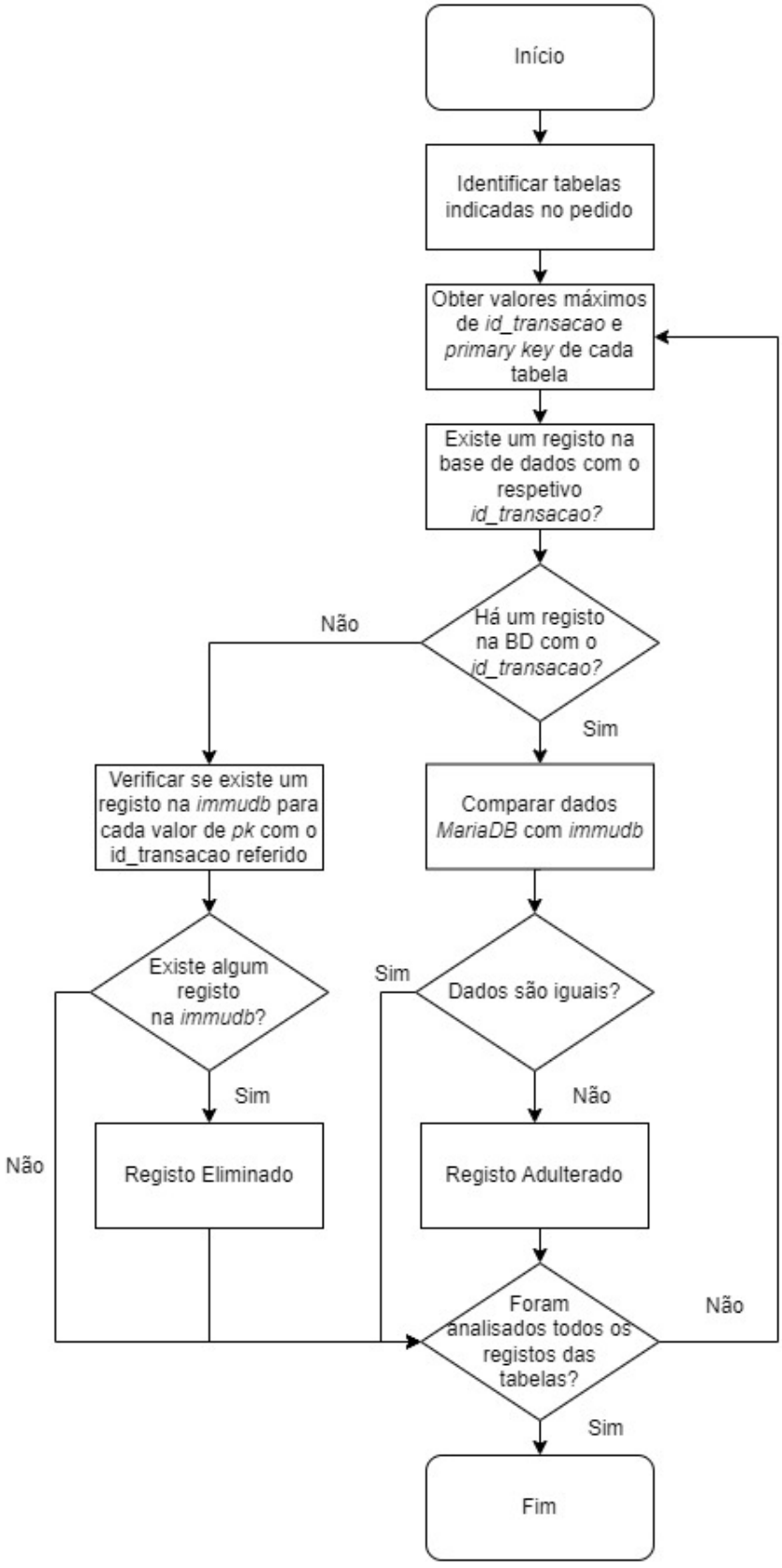


Figura 4.9: Verificação dos registos da base de dados Immudb

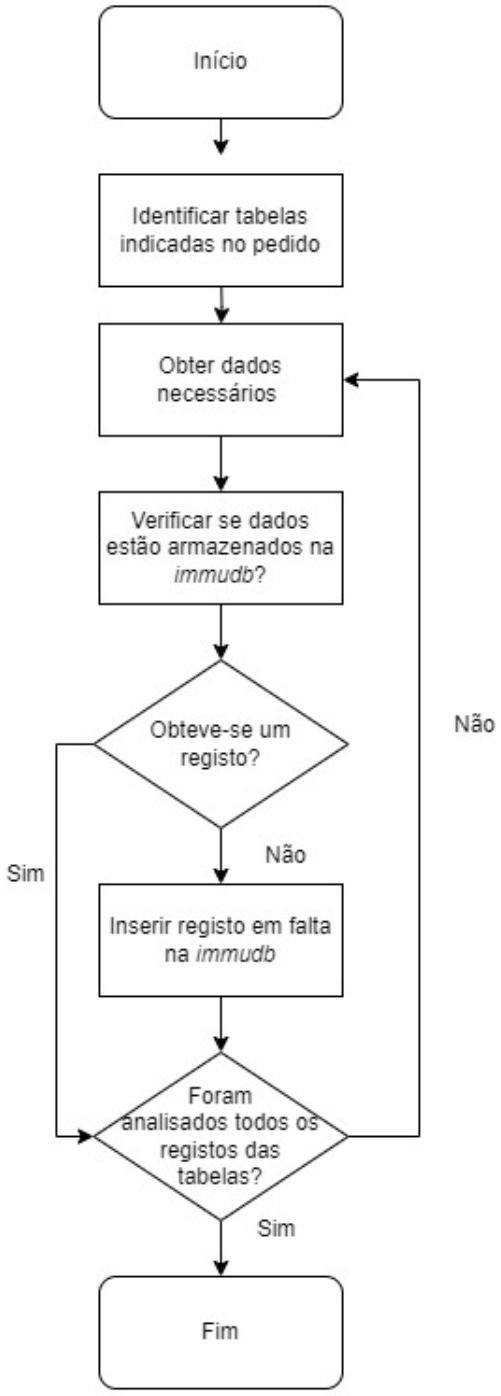


Figura 4.10: Sincronização dos dados entre as bases de dados MariaDB e Immudb

Capítulo 5

Avaliação do Desempenho da API

À medida que foram sendo introduzidas novas alterações na API do SIGEDEE, foram sendo efetuados testes de desempenho recorrendo ao Postman. Neste capítulo, serão apresentados os resultados desses mesmos testes e será avaliado o impacto que cada alteração teve no desempenho da API. Cada pedido foi repetido 100 vezes, e foram efetuados 3 testes por cada pedido. Ou seja, foram efetuados no total 300 testes para cada pedido. Aqui, será apresentada apenas a média desses resultados.

Nos pedidos *POST* e *PUT* apenas foram inseridos um registo de cada vez. Contudo, no caso dos pedidos *GET*, foram obtidos vários registos de uma vez. No entanto, o valor de registos obtidos varia consoante a tabela analisada, tal como demonstra a Tabela 5.1

Tabela 5.1: Número de registos de cada tabela no pedido *GET*

Tabela	**Número de Registos**
Equipamentos	2000
Fornecedores	100
Instalações	100
Helpdesk	300
Movimentos	300

5.1 Estado Original da API

De modo a ter um ponto de referência, foram medidos os tempos de resposta da API no seu estado inicial, obtendo os resultados apresentados na Tabela 5.2.

Tabela 5.2: Tempos de resposta do estado inicial da API, em milissegundos

Tabela	***GET***	***POST***	***PUT***
Equipamentos	178	35	38
Fornecedores	31	33	37
Instalações	29	33	36

5.2 Sistema de Registo de Transações

Após se analisar o impacto da introdução do novo sistema de registo de transações, percebeu-se que o mesmo teve um impacto praticamente nulo, uma vez que os resultados obtidos foram muito próximos dos resultados iniciais. A Tabela 5.3 demonstra os resultados obtidos.

Tabela 5.3: Tempos de resposta da API após introdução do sistema de registo de transações, em milissegundos

Tabela	***GET***	***POST***	***PUT***
Equipamentos	207	37	41
Fornecedores	32	38	38
Instalações	28	40	36
Helpdesk	36	31	34
Movimentos	33	31	42

Contudo, a introdução da assinatura digital causou um ligeiro aumento no tempo de inserção de registos, principalmente dos equipamentos, tal como demonstra a Tabela 5.4.

Tabela 5.4: Tempos de resposta daAPI após introdução da assinatura digital, em milissegundos

Tabela	***POST***	***PUT***
Equipamentos	60	39
Fornecedores	42	39
Instalações	54	36
Helpdesk	31	34
Movimentos	31	42

5.3 Envio de *e-mail*

Tal como referido anteriormente ao longo desta dissertação, foi introduzido um sistema de envio de *e-mails* para os utilizadores, de forma a confirmar a inserção de um novo registo. Contudo, a introdução deste mecanismo teve um impacto enorme no desempenho da API, fazendo com que os tempos de resposta atingissem valores superiores a 2500 milissegundos (Tabela 5.5).

Tabela 5.5: Tempos de resposta daAPI após introdução do envio de *e-mails*, em milissegundos

Tabela	***POST***	***PUT***
Equipamentos	2631	2522
Fornecedores	2628	2522
Instalações	2578	2508
Helpdesk	3401	2718
Movimentos	3427	2846

De modo a anular este aumento de tempo, introduziu-se uma função própria do PHP (*fastcgi_finish_request*), antes de os *e-mails* serem enviados. Esta função permite que o *backend* retorne imediatamente a resposta de sucesso para o cliente e continue a execução do *script* em *background*, diminuindo os tempos de resposta da API significativamente, voltando a apresentar valores minimamente aceitáveis, tal como demonstra a Tabela 5.6

Tabela 5.6: Tempos de resposta daAPI após introdução do envio de *e-mails* com *fastcgi_finish_request*, em milissegundos

Tabela	***POST***	***PUT***
Equipamentos	63	52
Fornecedores	69	60
Instalações	71	53
Helpdesk	31	63
Movimentos	32	68

5.4 Introdução do Immudb

Por último, analisou-se o impacto de ter uma cópia de dados no Immudb num servidor à parte. Mais uma vez, esta alteração teve um impacto significativo, sendo os resultados obtidos superiores a 550 milissegundos (Tabela 5.7). Contudo, adotando a mesma estratégia do caso anterior, os tempos de resposta da API diminuíram, tal como demonstram os resultados apresentados na Tabela 5.8

A Figura 5.1 ilustra os diferentes tempos de resposta da API do SIGEDEE para os vários pedidos de equipamentos, em diferentes fases de desenvolvimento.

Tabela 5.7: Tempos de resposta daAPI após introdução do Immudb, em milissegundos

Tabela	***POST***	***PUT***
Equipamentos	568	608
Fornecedores	647	588
Instalações	681	586

Tabela 5.8: Tempos de resposta daAPI após introdução do Immudb com *fastcgi_finish_request*, em milissegundos

Tabela	***POST***	***PUT***
Equipamentos	98	100
Fornecedores	95	105
Instalações	105	108

Assim, é possível verificar que os tempos de resposta da API foram aumentando com a introdução dos novos mecanismos, sendo a introdução do Immudb aquela que teve um maior impacto final, mesmo este tendo sido reduzido com recurso à função *fastcgi_finish_request*. O aumento do tempo de resposta é naturalmente uma desvantagem, contudo este mantém-se dentro de valores que podem ser considerados aceitáveis, tendo sido o compromisso possível para providenciar o nível de confiabilidade desejado ao sistema.

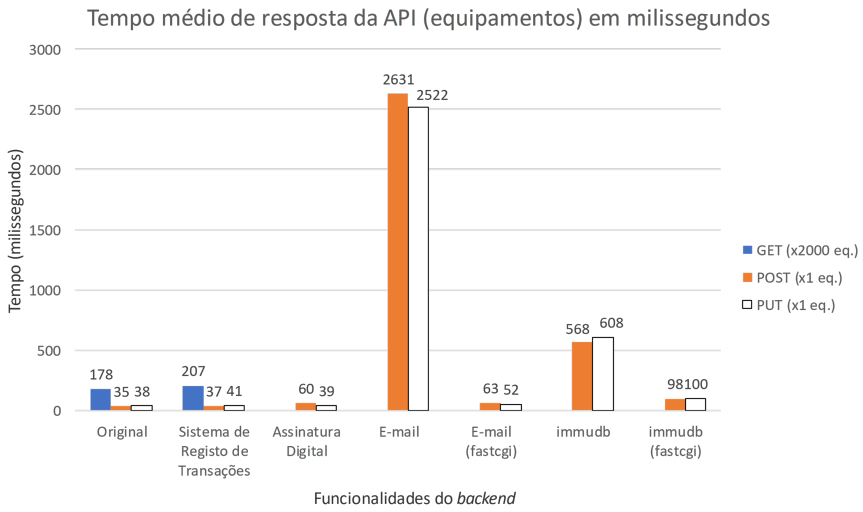


Figura 5.1: Tempos de resposta da API do SIGEDEE para os vários pedidos de equipamentos, em diferentes fases de desenvolvimento

Capítulo 6

Conclusões

Inicialmente, o SIGEDEE era um sistema muito simples que permitia inserir e editar equipamentos, fornecedores e instalações do DEE. Após a realização desta dissertação, o DEE possui agora um sistema mais completo que, além das funções iniciais, permite efetuar *login* de duas maneiras diferentes, e possui agora três novos sistemas incorporados: um dedicado à gestão dos utilizadores, outro para reportar avarias de equipamentos e acompanhar a resolução das mesmas, e por último um mecanismo que permite pedir e controlar os movimentos de equipamentos realizados entre utilizadores do DEE.

Foi ainda implementado com sucesso um sistema de registo de transações, recorrendo ao Immudb, com o objetivo de detetar a eventual inserção de registos a posteriori, assim como a eliminação de registos anteriormente inseridos. Caso algum registo seja corrompido na base de dados MariaDB, este deverá ser facilmente detetado uma vez que não irá coincidir com aquele armazenado no Immudb. Para tal, foram desenvolvidos três *scripts* que permitem verificar a integridade dos dados armazenados no MariaDB e no Immudb, bem como, sincronizar os dados presentes nas respetivas bases de dados. Foi ainda implementado um sistema de assinatura digital, utilizando uma aplicação exclusiva do DEE (DEEsign), para garantir a não repudiação dos dados e, também, contribuir no processo de validação dos mesmos.

Apesar de, após estas alterações, os tempos de resposta da API terem aumentado, é possível afirmar que se cumpriram todos os requisitos para o bom funcionamento do SIGEDEE.

6.1 Trabalho Futuro

Futuramente, é aconselhado que esta API seja publicada numa plataforma que permita efetuar o controlo de versões de forma mais simples, como, por exemplo, o GitHub. Além disso, o projeto Immudb deverá ser monitorizado, uma vez que é algo relativamente recente, e poderão ser introduzidas algumas alterações importantes na sua vertente SQL que permitam o SIGEDEE migrar por completo para este sistema. Por outro lado, sendo um projeto recente, poderá também ser descontinuado, pelo que deverão ser analisadas outras alternativas, no caso de tal acontecer. Entretanto, poderá ser analisada a hipótese de possuir diversos servidores a servir de réplica do Immudb, criando assim um sistema de replicação síncrona. Esta replicação também poderá ser efetuada no MariaDB, introduzindo, também, um balanceador de carga, de modo a distribuir os pedidos dos clientes pelos diferentes servidores disponíveis.

A aplicação DEEsign deverá ser atualizada, de modo a que seja possível garantir de forma segura e automática que um par de chaves pertence ao utilizador que as gerou. Além disso, deverá ser também estudado o desenvolvimento de uma versão dedicada a *smartphones*, facilitando o processo de assinatura digital em qualquer lugar.

Por último, os mecanismos responsáveis por verificar a integridade dos registos, apresentados na Secção 4.5, deverão ser otimizados, de modo a poderem ser executados em qualquer máquina e deverão ser automatizados. Ou seja, assim que seja iniciada a máquina responsável pelo Immudb, deverá ser executado o *script* de sincronização entre o MariaDB e o Immudb, bloqueando quaisquer inserções no Immudb, enquanto a sincronização não terminar.

Referências

[1] M. Vigil, J. Buchmann, D. Cabarcas, C. Weinert, and A. Wiesmaier, "Integrity, authenticity, non-repudiation, and proof of existence for long-term archiving: A survey," *Computers & Security*, vol. 50, pp. 16–32, may 2015. [Citado nas páginas 5 e 6]

[2] R. Zhang, R. Xue, and L. Liu, "Security and privacy on blockchain," *ACM Computing Surveys*, vol. 52, pp. 1–34, jul 2019. [Citado nas páginas xi, 6, 8, 12, 13, 14, 15, 16 e 19]

[3] R. Kaur and A. Kaur, "Digital signature," in *2012 International Conference on Computing Sciences*, IEEE, sep 2012. [Citado nas páginas xi, 6 e 7]

[4] "Web cryptography api." https://w3c.github.io/webcrypto/. [Citado na página 7]

[5] "W3c." https://www.w3.org/. [Citado na página 7]

[6] H. Halpin, "The w3c web cryptography API," in *Proceedings of the 23rd International Conference on World Wide Web*, ACM, apr 2014. [Citado na página 8]

[7] "Indexed database api 3.0." https://www.w3.org/TR/IndexedDB/. [Citado na página 8]

[8] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Citado na página 8]

[9] "Bitcoin." https://bitcoin.org/en/. [Citado na página 8]

[10] M. A. Ferrag, M. Derdour, M. Mukherjee, A. Derhab, L. Maglaras, and H. Janicke, "Blockchain technologies for the internet of things: Research issues and challenges," *IEEE Internet of Things Journal*, vol. 6, pp. 2188–2204, apr 2019. [Citado nas páginas ix, 8, 11, 12 e 13]

[11] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang, "Untangling blockchain: A data processing view of blockchain systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, pp. 1366–1385, jul 2018. [Citado nas páginas ix, 8, 12, 20, 27, 28, 29, 30 e 31]

[12] H.-N. Dai, Z. Zheng, and Y. Zhang, "Blockchain for internet of things: A survey," *IEEE Internet of Things Journal*, vol. 6, pp. 8076–8094, oct 2019. [Citado nas páginas ix, xi, 8, 11, 13, 15, 16 e 21]

[13] F. Casino, T. K. Dasaklis, and C. Patsakis, "A systematic literature review of blockchain-based applications: Current status, classification and open issues," *Telematics and Informatics*, vol. 36, pp. 55–81, mar 2019. [Citado nas páginas ix, 9, 10 e 15]

[14] K. Košťál, P. Helebrandt, M. Belluš, M. Ries, and I. Kotuliak, "Management and monitoring of IoT devices using blockchain," *Sensors*, vol. 19, p. 856, feb 2019. [Citado nas páginas 9 e 15]

[15] R. P. Pinto, B. M. C. Silva, and P. R. M. Inacio, "A system for the promotion of traceability and ownership of health data using blockchain," *IEEE Access*, vol. 10, pp. 92760–92773, 2022. [Citado na página 9]

[16] "Web of science." https://www.webofscience.com/wos. [Citado na página 11]

[17] M. Raikwar, D. Gligoroski, and G. Velinov, "Trends in development of databases and blockchain," in *2020 Seventh International Conference on Software Defined Systems (SDS)*, IEEE, apr 2020. [Citado nas páginas 11 e 14]

[18] S. S. Panda, B. K. Mohanta, U. Satapathy, D. Jena, D. Gountia, and T. K. Patra, "Study of blockchain based decentralized consensus algorithms," in *TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON)*, IEEE, oct 2019. [Citado nas páginas ix, xi, 15, 17, 18, 19, 21, 22, 24, 25, 26 e 27]

[19] W. Viriyasitavat and D. Hoonsopon, "Blockchain characteristics and consensus in modern business processes," *Journal of Industrial Information Integration*, vol. 13, pp. 32–39, mar 2019. [Citado nas páginas 17 e 18]

[20] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *Advances in Cryptology — CRYPTO' 92*, pp. 139–147, Springer Berlin Heidelberg, 2001. [Citado na página 18]

[21] L. Lamport, "The part-time parliament," *ACM Transactions on Computer Systems*, vol. 16, pp. 133–169, may 1998. [Citado na página 21]

[22] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, pp. 382–401, jul 1982. [Citado nas páginas ix, 22 e 23]

[23] M. Castro and B. Liskov, "Practical byzantine and fault tolerance," 1999. [Citado na página 24]

[24] "Ethereum." https://ethereum.org/en/. [Citado na página 26]

[25] "Hyperledger fabric." https://www.hyperledger.org/use/fabric. [Citado na página 27]

[26] "Corda." https://corda.net/. [Citado na página 27]

[27] "Consensys quorum." https://consensys.net/quorum/. [Citado na página 27]

[28] "Parity." https://www.parity.io/. [Citado na página 27]

[29] A. Afif, M. Olov, S. Karl, Andersson, P. Student, P. Associate, and Professor, "Performance evaluation of permissioned blockchain platforms," in *2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*, no. being, IEEE, dec 2020. [Citado nas páginas ix, 29, 31 e 32]

[30] "Linux foundation." https://www.linuxfoundation.org/. [Citado na página 29]

[31] "Hyperledger." https://www.hyperledger.org/. [Citado na página 29]

[32] Z. Zheng, S. Xie, H.-N. Dai, W. Chen, X. Chen, J. Weng, and M. Imran, "An overview on smart contracts: Challenges, advances and platforms," *Future Generation Computer Systems*, vol. 105, pp. 475–491, apr 2020. [Citado nas páginas 29 e 34]

[33] Q. Nasir, I. A. Qasse, M. A. Talib, and A. B. Nassif, "Performance analysis of hyperledger fabric platforms," *Security and Communication Networks*, vol. 2018, pp. 1–14, sep 2018. [Citado nas páginas 32 e 33]

[34] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. D. Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, ACM, apr 2018. [Citado nas páginas ix, x, 32, 33 e 34]

[35] "grpc." https://grpc.io/. [Citado na página 32]

[36] S. Shalaby, A. A. Abdellatif, A. Al-Ali, A. Mohamed, A. Erbad, and M. Guizani, "Performance evaluation of hyperledger fabric," in *2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT)*, IEEE, feb 2020. [Citado nas páginas ix, x, 33 e 35]

[37] K. Yamashita, Y. Nomura, E. Zhou, B. Pi, and S. Jun, "Potential risks of hyperledger fabric smart contracts," in *2019 IEEE International Workshop on*

Blockchain Oriented Software Engineering (IWBOSE), IEEE, feb 2019. [Citado na página 34]

[38] N. Andola, Raghav, M. Gogoi, S. Venkatesan, and S. Verma, "Vulnerabilities on hyperledger fabric," *Pervasive and Mobile Computing*, vol. 59, p. 101050, oct 2019. [Citado nas páginas x, 35 e 36]

[39] B. Duncan and M. Whittington, "Creating and configuring an immutable database for secure cloud audit trail and system logging," *International Journal On Advances in Security*, vol. 10, pp. 155–166, 12 2017. [Citado na página 37]

[40] "Amazon qldb." https://aws.amazon.com/pt/qldb/. [Citado na página 37]

[41] "Dolthub." https://www.dolthub.com/. [Citado na página 37]

[42] "Terminusdb." https://terminusdb.com/. [Citado na página 37]

[43] "Trillian." https://transparency.dev/#trillian. [Citado na página 37]

[44] "Bigchaindb." https://www.bigchaindb.com/. [Citado na página 37]

[45] "Tendermint." https://tendermint.com/. [Citado na página 37]

[46] "Mongodb." https://www.mongodb.com/. [Citado na página 37]

[47] B. Gmbh, "Bigchaindb 2.0 the blockchain database," 2016. [Citado nas páginas x e 38]

[48] "Codenotary." https://codenotary.com/. [Citado na página 38]

[49] "immudb." https://immudb.io/. [Citado nas páginas 38 e 68]

[50] "Go." https://go.dev/learn/. [Citado na página 38]

[51] I. Codenotary, M. Paik, D. Zimmer, M. Meloni, and V. Padurean, "immudb: A lightweight, performant immutable database," 2020. [Citado nas páginas x, 39 e 40]

[52] "Badgerdb." https://dgraph.io/docs/badger/. [Citado na página 39]

[53] "The fastest-growing open-source startups in q4 2021." https://runacap.com/ross-index/q4-2021/. [Citado na página 40]

[54] S. R. Leonard Richardson, *RESTful Web Services*. O'Reilly Media, 2007. [Citado na página 41]

[55] "Restful api modeling language (raml)." https://raml.org/. [Citado na página 41]

[56] "Apache avro™ 1.8.2 idl." https://avro.apache.org/docs/1.8.2/idl.html. [Citado na página 41]

[57] "Web application description language (wadl)." https://www.w3.org/Submission/wadl/. [Citado na página 41]

[58] "Postman." https://www.postman.com/. [Citado na página 41]

[59] IJSCE, "API features individualizing of web services: REST and SOAP," *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, pp. 664–671, aug 2019. [Citado na página 41]

[60] "Swagger." https://swagger.io/. [Citado na página 41]

[61] "Openapi specification." https://www.openapis.org/. [Citado na página 41]

[62] V. B. Surwase, "Rest api modeling languages - a developer's perspective," *International Journal For Science Technology And Engineering*, vol. 2, pp. 634–637, 2016. [Citado na página 41]

Anexo A

Documentação

A.1 Documentação da API

SIGEDEE

Descrição da API do Sistema de Gestão de Equipamentos do DEE (SIGEDEE).

Existem três tipos de utilizadores a aceder a este serviço:

- 0 - elementos da direção do DEE
- 10 - técnicos do DEE
- 20 - docentes do DEE

Algumas funcionalidades apenas podem ser acedidas por determinados tipos de utilizador.

AUTHORIZATION Bearer Token

Token	<token>

Login

Descrição dos diferentes métodos de autenticação.

Acessível por utilizadores de todos os tipos.

GET Login

url/token

Efetuar Login com Username e respetiva Password

AUTHORIZATION Basic Auth

Username	<username>
Password	<password>

HEADERS

Authorization

PARAMS

GET **Login Mail**

url/token?requester=url_client&sendto=user_mail

Efetuar Login com E-mail do Utilizador

HEADERS

Authorization

PARAMS

requester	url_client
sendto	user_mail

Raw

Obter dados específicos. Úteis para obter dados de seleção para inserir ou editar registo.

Acessível por utilizadores de todos os tipos.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

GET **Informações Equipamento**

/raw/equipamentos/id_equipamento

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

GET **Descrição de Equipamento**

url/raw/descricoes_de_equipamento/id_descricao_equipamento

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Nomes

Obter dados específicos. Úteis para obter dados de seleção para inserir ou editar registo.

Acessível por utilizadores de todos os tipos.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

GET Equipamentos de Uma Instalação

url/equipamentos/nomes/numero_instalacao

Obter os equipamentos de uma determinada instalação.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

GET Estados do Equipamento

url/estados_do_equipamento/nomes/

Obter os diferentes estados que um equipamento pode assumir.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

GET Fornecedores

url/fornecedores/nomes/

Obter o id e o nome da firma de todos os fornecedores da base de dados.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

GET Helpdesk

url/helpdesk/nomes/id_ticket

Obter informações sobre tickets.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

GET Instalações

url/instalacoes/nomes/

Obter os números das instalações do DEE.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

GET Instalações do Utilizador

url/instalacoes_own/nomes/

Obter as instalações da responsabilidade do utilizador.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

GET Movimentos - Emprestimo Aceite (1) ou Não Aceite (0)

url/movimentos/nomes/id_movimento

Verificar se o empréstimo de um equipamento se encontra aceite.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

GET Prioridade Ticket

url/prioridade/nomes/

Obter as diferentes prioridades que um ticket pode ter.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

GET Tipos de Equipamento

url/tipos_de_equipamento/nomes/

Obter os diversos tipos de equipamento registados.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

GET Tipos de Instalação

url/tipos_de_instalacoes/nomes/

Obter os diferentes tipos de instalação do DEE.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

GET Tipos de Utilizador

url/tipo_utilizador/nomes

Obter os diferentes tipos de utilizador (0 - direção; 10 - técnico, 20 - docente)

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

GET Utilizadores

url/utilizadores/nomes/tipo_utilizador

Obter id e sigla de directores ou técnicos de instalações.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Equipamentos

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

Backup

Acessível por utilizadores do tipo 0 e 10.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

GET **Visualizar Backups Equipamentos Disponíveis**

url/equipamentos/alterados

Obter todos os backups de equipamentos.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

GET **Visualizar Backups Equipamentos Disponíveis dos Últimos x Meses**

url/equipamentos/alterados?meses=meses

Obter todos os backups de equipamentos dos útimos x meses.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

PARAMS

meses	meses

GET **Visualizar Detalhes de Backup Equipamento**

url/equipamentos/alterados/id_registo

Obter detalhes de um backup de equipamento

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

PUT **Recuperar Backup Equipamento**

url/equipamentos/alterados/id_registo

Apenas pode ser recuperado um backup de cada vez.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Body raw (json)

json

```
{
  "bulk": [
    {
      "data": '{
        "id_registo":{id_registo},
        "id_equipamento":{id_equipamento},
        "cativacao":"{cativacao}",
        "localizacao":"{localizacao}",
        "tipo_equipamento":"{tipo_equipamento}",
        "caracteristicas":"{caracteristicas}",
        "marca":"{marca}",
        "modelo":"{modelo}",
        "local_na_sala":"{local_na_sala}",
        "n_serie":"{n_serie}",
        "n_inventario_DEE":"{n_inventario_DEE}",
        "n_inventario_ISEP":"{n_inventario_ISEP}",
        "n_inventario_IPP":"{n_inventario_IPP}",
        "custo_com_IVA":"{custo_com_IVA}",
        "data_aquisicao":"{data_aquisicao}",
        "observacoes":"{observacoes}",
        "estado":"{estado_equipamento}",
        "firma":"{firma}",
        "inserido_em":"{inserido_em}",
        "inserido_por":"{sigla_utilizador}",
        "ultima_alteracao_em":"{ultima_alteracao_em}",
        "ultima_alteracao_por":"{sigla_utilizador}"
        }',
      "signature": [Array_Mensagem_Assinada_SHA256]
    }
  ],
  "exportedKey": "-----BEGIN PUBLIC KEY-----\n" +
                 "USER_PUBLIC_KEY\n" +
                 "-----END PUBLIC KEY-----\n"
}
```

GET

Acessível por utilizadores de todos os tipos.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

GET **Equipamentos**

url/equipamentos

Obter todos os equipamentos da base de dados.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

GET Equipamentos com Limite de Resultados

url/equipamentos?limite=numero_resultados

Obter x equipamentos da base de dados.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

PARAMS

limite	numero_resultados

GET Detalhes Equipamento

url/equipamentos/id_equipamento/id_descricao_equipamento

Consultar detalhes de um equipamento.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

PARAMS

POST

Acessível por utilizadores do tipo 0 e 10.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

POST Inserir Equipamento

url/equipamentos

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Body raw (json)

json

```
{
  "bulk": [
    {
      "data": '{
                "id_tipo_equipamento":"{id_tipo_equipament
                "marca":"{marca}",
                "modelo":"{modelo}",
                "n_serie":"{n_serie}",
                "caracteristicas":"{caracteristicas}",
                "localizacao":"{localizacao}",
                "cativacao":"{cativacao}",
                "local_na_sala":"{local_na_sala}",
                "id_fornecedor":"{id_fornecedor}",
                "id_estado":"{id_estado_equipamento}",
                "custo_com_IVA":"{custo_com_IVA}",
                "data_aquisicao":"{data_aquisicao}",
                "n_inventario_DEE":"{n_inventario_DEE}",
                "n_inventario_ISEP":"{n_inventario_ISEP}",
                "n_inventario_IPP":"{n_inventario_IPP}",
                "observacoes":"{observacoes}"
                }',
      "signature": [Array_Mensagem_Assinada_SHA256]
    }
  ],
  "exportedKey": "-----BEGIN PUBLIC KEY-----\n" +
                 "USER_PUBLIC_KEY\n" +
                 "-----END PUBLIC KEY-----\n"
}
```

POST Inserir Equipamento Com Novo Tipo de Equipamento

url/equipamentos

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Body raw (json)

json

```
{
  "bulk": [
```

```
    {
      "data": '{
                "n_tipo_equipamento":"{novo_tipo_equipamen
                "marca":"{marca}",
                "modelo":"{modelo}",
                "n_serie":"{n_serie}",
                "caracteristicas":"{caracteristicas}",
                "localizacao":"{localizacao}",
                "cativacao":"{cativacao}",
                "local_na_sala":"{local_na_sala}",
                "id_fornecedor":"{id_fornecedor}",
                "id_estado":"{id_estado_equipamento}",
                "custo_com_IVA":"{custo_com_IVA}",
                "data_aquisicao":"{data_aquisicao}",
                "n_inventario_DEE":"{n_inventario_DEE}",
                "n_inventario_ISEP":"{n_inventario_ISEP}",
                "n_inventario_IPP":"{n_inventario_IPP}",
                "observacoes":"{observacoes}"
                }',
      "signature": [Array_Mensagem_Assinada_SHA256]
    }
  ],
  "exportedKey": "-----BEGIN PUBLIC KEY-----\n" +
                 "USER_PUBLIC_KEY\n" +
                 "-----END PUBLIC KEY-----\n"
}
```

POST **Inserir Múltiplos Equipamentos (Excel)**

url/equipamentos/bulk

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Body raw (json)

json

```
{
  "bulk": [
    {
      "data": '{
                "sala":"{numero_instalacao}",
                "local_na_sala":"{local_na_sala}",
                "tipo_equipamento":"{tipo_equipamento}",
                "caracteristicas_principais":"{caracterist
                "marca":"{marca}",
                "modelo":"{modelo}",
                "n_serie":"{n_serie}",
                "n_inventario_dee":"{n_inventario_dee}",
                "n_inventario_isep":"{n_inventario_isep}",
                "n_inventario_ipp":"{n_inventario_ipp}",
                "estado (1- ok, 2-utilização limitada, 3-a
                "obs":"{observacoes}"
                }',
      "signature": [Array_Mensagem_Assinada_SHA256]
    },
    {
```

```
      "data": '{
                "sala":"{numero_instalacao}",
                "local_na_sala":"{local_na_sala}",
                "tipo_equipamento":"{tipo_equipamento}",
                "caracteristicas_principais":"{caracterist
                "marca":"{marca}",
                "modelo":"{modelo}",
                "n_serie":"{n_serie}",
                "n_inventario_dee":"{n_inventario_dee}",
                "n_inventario_isep":"{n_inventario_isep}",
                "n_inventario_ipp":"{n_inventario_ipp}",
                "estado (1- ok, 2-utilização limitada, 3-a
                "obs":"{observacoes}"
                }',
      "signature": [Array_Mensagem_Assinada_SHA256]
    },
    .
    .
    .
  ],
  "exportedKey": "-----BEGIN PUBLIC KEY-----\n" +
                 "USER_PUBLIC_KEY\n" +
                 "-----END PUBLIC KEY-----\n"
}
```


PUT

Acessível por utilizadores do tipo 0 e 10.

Utilizadores do tipo 20 podem editar um equipamento desde que este esteja numa instalação da sua responsabilidade.

Utilizadores do tipo 10 e 20 apenas podem mudar um equipamento de uma instalação sua para outra da sua responsabilidade.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

PUT Editar Equipamento

url/equipamentos/id_equipamento

Apenas pode ser editado um equipamento de cada vez.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Body raw (json)

json

```
{"bulk":
[
    {"data":
        {
        "id_tipo_equipamento":"{id_tipo_equipamento}",
        "marca":"{marca}",
        "modelo":"{modelo}",
        "n_serie":"{n_serie}",
        "caracteristicas":"{caracteristicas}",
        "localizacao":"{localizacao}",
        "cativacao":"{cativacao}",
        "local_na_sala":"{local_na_sala}",
        "id_fornecedor":"{id_fornecedor}",
        "id_estado":"{id_estado_equipamento}",
        "custo_com_IVA":"{custo_com_IVA}",
        "data_aquisicao":"{data_aquisicao}",
        "n_inventario_DEE":"{n_inventario_DEE}",
        "n_inventario_ISEP":"{n_inventario_ISEP}",
        "n_inventario_IPP":"{n_inventario_IPP}",
        "observacoes":"{observacoes}"
        },
    "signature":[Array_Mensagem_Assinada_SHA256]
    }
    ],
  "exportedKey": "-----BEGIN PUBLIC KEY-----\n" +
                 "USER_PUBLIC_KEY\n" +
                 "-----END PUBLIC KEY-----\n"
}
```

PUT Mudar Sala de Equipamento(s)

url/equipamentos/id_equipamento

É possível alterar a sala de vários equipamentos em simultâneo.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Body raw (json)

json

```
{
  "bulk": [
    {
      "data": '{
                "localizacao":"{localizacao}"
               }',
      "signature": [Array_Mensagem_Assinada_SHA256]
    }
  ],
  "exportedKey": "-----BEGIN PUBLIC KEY-----\n" +
                 "USER_PUBLIC_KEY\n" +
                 "-----END PUBLIC KEY-----\n"
}
```

DELETE

É possível eliminar vários equipamentos em simultâneo.

Acessível por utilizadores do tipo 0.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

DELETE Eliminar Equipamento

url/equipamentos/id_equipamento

StartFragment

É possível eliminar vários equipamentos em simultâneo.

EndFragment

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Body raw (json)

json

```
{
  "bulk": [
    {
      "data": '{
                "id_equipamento":{id_equipamento},
                "cativacao":"{cativacao}",
                "localizacao":"{localizacao}",
                "id_descricao_equipamento":{id_descricao_e
                "tipo_equipamento":"{tipo_equipamento}",
                "marca":"{marca}",
                "modelo":"{modelo}",
                "caracteristicas":"{caracteristicas}",
                "local_na_sala":"{local_na_sala}",
                "n_serie":"{n_serie}",
                "n_inventario_DEE":"{n_inventario_DEE}",
                "n_inventario_ISEP":"{n_inventario_IPP}",
                "n_inventario_IPP":"{n_inventario_ISEP}",
                "custo_com_IVA":"{custo_com_IVA}",
                "data_aquisicao":"{data_aquisicao}",
                "observacoes":"{observacoes}",
                "estado":"{estado_equipamento}",
                "firma":"{firma}",
                "inserido_em":"{inserido_em}",
                "inserido_por":"{id_utilizador}",
                "ultima_alteracao_em":"{ultima_alteracao_e
                "ultima_alteracao_por":"{id_utilizador}",
                "state":true,
                "visivel":0
                }',
```

```
      "signature": [Array_Mensagem_Assinada_SHA256]
    }
  ],
  "exportedKey": "-----BEGIN PUBLIC KEY-----\n" +
                 "USER_PUBLIC_KEY\n" +
                 "-----END PUBLIC KEY-----\n"
}
```

Fornecedores

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

Backup

Acessível por utilizadores do tipo 0 e 10.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

GET Visualizar Backups Fornecedores Disponíveis

url/fornecedores/alterados

Obter todos os backups de fornecedores.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

GET Visualizar Backups Fornecedores dos Últimos x Meses

url/fornecedores/alterados?meses=meses

Obter todos os backups de fornecedores dos útimos x meses.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

PARAMS

meses	meses

GET Visualizar Detalhes de Backup Fornecedor

url/fornecedores/alterados/id_registo

Obter detalhes de um backup de fornecedor.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

PUT Recuperar Backup Fornecedor

url/fornecedores/alterados/id_registo

Apenas pode ser recuperado um backup de cada vez.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Body raw (json)

json

```
{
  "bulk": [
    {
      "data": '{
                "id_registo":{id_registo},
                "id_fornecedor":{id_fornecedor},
                "firma":"{firma}",
                "NIF":"{NIF}",
                "url_website":"{url_website}",
                "email_geral":"{email_geral}",
                "telefone_1":"{telefone_1}",
                "telefone_2":"{telefone_2}",
                "fax":"{fax}",
                "endereco_postal":"{endereco_postal}",
                "codigo_postal":"{codigo_postal}",
                "localidade":"{localidade}",
                "cidade":"{cidade}",
                "pais":"{pais}",
                "nome_pessoa_contacto":"{nome_pessoa_conta
                "email_pessoa_contacto":"{email_pessoa_con
                "telefone_pessoa_contacto":"{telefone_pess
                "alterado_por":"{sigla_utilizador}"
      }',
      "signature": [Array_Mensagem_Assinada_SHA256]
    }
  ],
  "exportedKey": "-----BEGIN PUBLIC KEY-----\n" +
                 "USER_PUBLIC_KEY\n" +
                 "-----END PUBLIC KEY-----\n"
}
```

GET

Acessível por utilizadores de todos os tipos.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

GET Fornecedores

url/fornecedores

Obter todos os fornecedores da base de dados.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

GET Fornecedores com Limite de Resultados

url/fornecedores?limite=numero_resultados

Obter x fornecedores da base de dados.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

PARAMS

limite	numero_resultados

GET Detalhes Fornecedor

url/fornecedores/id_fornecedor

Consultar detalhes de um fornecedor.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

POST

Acessível por utilizadores do tipo 0.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

POST Inserir Fornecedor

url/fornecedores

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Body raw (json)

json

```
{
  "bulk": [
    {
      "data": '{
                "firma":"{firma}",
                "NIF":"{NIF}",
                "url_website":"{url_website}",
                "email_geral":"{email_geral}",
                "telefone_1":"{telefone_1}",
                "telefone_2":"{telefone_2}",
                "fax":"{fax}",
                "endereco_postal":"{endereco_postal}",
                "codigo_postal":"{codigo_postal}",
                "localidade":"{localidade}",
                "cidade":"{cidade}",
                "pais":"{pais}",
                "nome_pessoa_contacto":"{nome_pessoa_conta
                "email_pessoa_contacto":"{email_pessoa_con
                "telefone_pessoa_contacto":"{telefone_pess
                }',
      "signature": [Array_Mensagem_Assinada_SHA256]
    }
  ],
  "exportedKey": "-----BEGIN PUBLIC KEY-----\n" +
                 "USER_PUBLIC_KEY\n" +
                 "-----END PUBLIC KEY-----\n"
}
```

PUT

Acessível por utilizadores do tipo 0.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

PUT Editar Fornecedor

url/fornecedores/id_fornecedor

Apenas pode ser editado um fornecedor de cada vez.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Body raw (json)

json

```
{
  "bulk": [
    {
      "data": '{
                "firma":"{firma}",
                "NIF":"{NIF}",
                "url_website":"{url_website}",
                "email_geral":"{email_geral}",
                "telefone_1":"{telefone_1}",
                "telefone_2":"{telefone_2}",
                "fax":"{fax}",
                "endereco_postal":"{endereco_postal}",
                "codigo_postal":"{codigo_postal}",
                "localidade":"{localidade}",
                "cidade":"{cidade}",
                "pais":"{pais}",
                "nome_pessoa_contacto":"{nome_pessoa_conta
                "email_pessoa_contacto":"{email_pessoa_con
                "telefone_pessoa_contacto":"{telefone_pess
                }',
      "signature": [Array_Mensagem_Assinada_SHA256]
    }
  ],
  "exportedKey": "-----BEGIN PUBLIC KEY-----\n" +
                 "USER_PUBLIC_KEY\n" +
                 "-----END PUBLIC KEY-----\n"
}
```

Instalações

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

Backup

Acessível por utilizadores do tipo 0.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

GET Visualizar Backups Instalações Disponíveis

url/instalacoes/alterados

Obter todos os backups de instalações.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

GET Visualizar Backups Fornecedores dos Últimos x Meses

url/instalacoes/alterados?meses=meses

Obter todos os backups de instalações dos útimos x meses.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

PARAMS

meses	meses

GET Visualizar Detalhes de Backup Instalação

url/instalacoes/alterados/id_registo

Obter detalhes de um backup de instalação.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

PUT Recuperar Backup Instalação

url/instalacoes/alterados/id_registo

Apenas pode ser recuperado um backup de cada vez.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Body raw (json)

```
json

{
  "bulk": [
    {
      "data": '{
                "id_registo":{id_registo},
                "numero_instalacao":"{numero_instalacao}",
                "tipo_instalacao":"{tipo_instalacao}",
                "nome_instalacao":"{nome_instalacao}",
                "ext_telefonica":"{ext_telefonica}",
                "alterado_por":"{sigla_utilizador}",
                "id_director":"{sigla_director}",
                "id_tecnico":"{sigla_tecnico}"
              }',
      "signature": [Array_Mensagem_Assinada_SHA256]
    }
  ],
  "exportedKey": "-----BEGIN PUBLIC KEY-----\n" +
                 "USER_PUBLIC_KEY\n" +
                 "-----END PUBLIC KEY-----\n"
}
```

GET

Acessível por utilizadores de todos os tipos.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

GET Instalações

url/instalacoes

Obter todas as instalações da base de dados.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

GET Instalações com Limite de Resultados

url/instalacoes?limite=numero_resultados

Obter x instalações da base de dados.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

PARAMS

limite numero_resultados

GET Detalhes Instalação

url/instalacoes/numero_instalacao

Consultar detalhes de uma instalação.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

POST

Acessível por utilizadores do tipo 0.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

POST Inserir Instalação

url/instalacoes

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Body raw (json)

json

```
{
  "bulk": [
    {
      "data": '{
                "numero_instalacao":"{numero_instalacao}",
                "tipo_instalacao":"{tipo_instalacao}",
                "nome_instalacao":"{nome_instalacao}",
                "ext_telefonica":"{ext_telefonica}",
                "id_director":"{id_director}}",
                "id_tecnico":"{id_tecnico}"
                }',
      "signature": [Array_Mensagem_Assinada_SHA256]
    }
  ],
  "exportedKey": "-----BEGIN PUBLIC KEY-----\n" +
                 "USER_PUBLIC_KEY\n" +
```

```
                "-----END PUBLIC KEY-----\n"
}
```

PUT

Acessível por utilizadores do tipo 0.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

PUT Editar Instalação

url/instalacoes/numero_instalacao

Apenas pode ser editado um fornecedor de cada vez.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Body raw (json)

json

```
{
  "bulk": [
    {
      "data": '{
                "numero_instalacao":"{numero_instalacao}",
                "tipo_instalacao":"{tipo_instalacao}",
                "nome_instalacao":"{nome_instalacao}",
                "ext_telefonica":"{ext_telefonica}",
                "id_director":"{id_director}",
                "id_tecnico":"{id_tecnico}"
                }',
      "signature": [Array_Mensagem_Assinada_SHA256]
    }
  ],
  "exportedKey": "-----BEGIN PUBLIC KEY-----\n" +
                 "USER_PUBLIC_KEY\n" +
                 "-----END PUBLIC KEY-----\n"
}
```

Movimentos

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

GET

Acessível por utilizadores de todos os tipos.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

GET Movimentos

url/movimentos?filtro_tipo=pedido_por&filtro_pedido=estado_pedido

Consultar todos os pedidos de movimento de equipamento.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

PARAMS

filtro_tipo	pedido_por
filtro_pedido	estado_pedido

GET Movimentos com Limite de Resultados

url/movimentos?filtro_tipo=pedido_por&filtro_pedido=estado_pedido&limite=numero_resultados

Consultar x pedidos de movimento de equipamento.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

PARAMS

filtro_tipo	pedido_por
filtro_pedido	estado_pedido
limite	numero_resultados

GET Detalhes Movimento

url/movimentos/id_movimento

Consultar detalhes de um pedido de movimento de equipamento.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

POST

Acessível por utilizadores de todos os tipos.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

POST Inserir Pedido de Movimento

url/movimentos

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Body raw (json)

json

```
{
    "id_equipamento":"id_equipamento",
    "nova_localizacao":"{{nova_localizacao}}"
}
```

PUT

Apenas pode ser editado um movimento de cada vez.

Acessível por utilizadores de tipo 0 e 10.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

PUT Aceitar Pedido de Movimento

url/movimentos/id_movimento/accept

Acessível por utilizadores de tipo 0.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

PUT Rejeitar Pedido de Movimento

url/movimentos/id_movimento/decline

Acessível por utilizadores de tipo 0.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

PUT Emprestar Equipamento

url/movimentos/id_movimento/loan

Acessível por utilizadores de tipo 0 e 10 (se este for o técnico responsável pela instalação onde se encontra o equipamento).

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Body raw (json)

```
json

{
    "codigo_movimento":"codigo_movimento"
}
```

PUT Devolver Equipamento

url/movimentos/id_movimento/return

Acessível por utilizadores de tipo 0 e 10 (se este for o técnico responsável pela instalação onde se encontra o equipamento).

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Body raw (json)

```
json
```

```
{
    "id_estado_final":"{{id_estado_final}}",
    "codigo_movimento":"codigo_movimento"
}
```

Helpdesk

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

GET

Acessível por utilizadores de todos os tipos.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

GET Tickets Helpdesk

url/helpdesk?filtro_tipo=pedido_por&filtro_pedido=estado_pedido

Consultar todos os tickets.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

PARAMS

filtro_tipo	pedido_por
filtro_pedido	estado_pedido

GET Tickets Helpdesk com Limite de Resultados

url/helpdesk?filtro_tipo=estado_pedido&filtro_pedido=pedido_por&limite=numero_resultados

Consultar x tickets.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

PARAMS

filtro_tipo	estado_pedido
filtro_pedido	pedido_por
limite	numero_resultados

GET Detalhes Ticket

url/helpdesk/id_ticket

Consultar detalhes de um ticket.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

PARAMS

POST

Acessível por utilizadores de todos os tipos.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

POST Inserir Ticket no Helpdesk

url/helpdesk

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Body raw (json)

json

```
{
    "id_equipamento":"id_equipamento",
    "titulo_problema":"{{titulo_problema}}",
    "descricao_problema":"{{descricao_problema}}"
}
```

POST Inserir Observações

url/helpdesk/id_ticket/obs

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Body raw (json)

json

```
{
    "mensagem":"observacoes"
}
```

PUT

Apenas pode ser editado um ticket de cada vez.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

PUT Alterar Prioridade de um Ticket

url/helpdesk/id_ticket/priority

Acessível por utilizadores de tipo 0.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Body raw (json)

json

```
{
    "id_prioridade":"id_prioridade"
}
```

PUT Ticket Cancelado

url/helpdesk/id_ticket/cancelled

Acessível por utilizadores de todos os tipos. Apenas pode ser cancelado por quem o inseriu.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

PUT Ticket Em Processamento

url/helpdesk/id_ticket/processing

Acessível por utilizadores de tipo 0 e 10.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

PUT Ticket Resolvido

url/helpdesk/id_ticket/fixed

Acessível por utilizadores de tipo 0 e 10.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Descrição de Equipamento

Obter as descrições de equipamento na base de dados.

Acessível por utilizadores de todos os tipos.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

GET Marca

url/variantes?param={"id_tipo_equipamento":"id_tipo_equipamento"}

Obter as marcas correspondentes ao tipo de equipamento referido.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

PARAMS

param	{"id_tipo_equipamento":"id_tipo_equipamento"}

GET Modelo

url/variantes?param={"id_tipo_equipamento":"id_tipo_equipamento", "marca":"marca"}

Obter os modelos correspondentes ao tipo de equipamento e marca referidos.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

PARAMS

param	{"id_tipo_equipamento":"id_tipo_equipamento", "marca":"marca"}

GET Características

url/variantes?param={"id_tipo_equipamento":"id_tipo_equipamento", "marca":"marca", "modelo":"modelo"}

Obter as características correspondentes ao tipo de equipamento, marca e modelo referidos.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

PARAMS

param	{"id_tipo_equipamento":"id_tipo_equipamento", "marca":"marca", "modelo":"modelo"}

Documentos

Acessível por utilizadores de tipo 0 e 10.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

POST Inserir Documento Equipamento

url/documentos/id_descricao_equipamento

Inserir documento sobre a descrição de equipamento referida.

Acessível por utilizadores de tipo 0 e 10.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Body formdata

pdf

DELETE Eliminar Documento Equipamento

url/documentos/id_descricao_equipamento/nome_ficheiro

Eliminar documento sobre a descrição de equipamento referida e com o nome indicado.

Acessível por utilizadores de tipo 0.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Fotos

Acessível por utilizadores de tipo 0 e 10.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

POST Inserir Foto Equipamento

url/fotos/id_descricao_equipamento

Inserir imagem de equipamento com a descrição referida.

Acessível por utilizadores de tipo 0 e 10.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Body formdata

foto

DELETE **Eliminar Foto Equipamento**

url/fotos/id_descricao_equipamento/nome_ficheiro

Eliminar imagem de equipamento com a descrição referida e nome indicado.

Acessível por utilizadores de tipo 0 e 10.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Utilizadores

Acessível por utilizadores de tipo 0.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

GET

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

GET **Consultar Utilizadores**

url/utilizadores

Consultar todos os utilizadores registados.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

GET **Consultar Utilizadores Com Limite de Resultadores**

url/utilizadores?limite=numero_resultados

Consultar x utilizadores registados.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

PARAMS

limite	numero_resultados

GET Consultar Dados de um Utilizador

url/utilizadores/id_utilizador

Consultar dados de um utilizadores de forma detalhada.

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

POST

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

POST Inserir Novo Utilizador

url/utilizadores

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

Body raw (json)

json

```
{
    "nome_apelido_utilizador":"{nome_apelido_utilizador}",
    "sigla":"{sigla}",
    "id_tipo_utilizador":"{id_tipo_utilizador}",
    "gabinete":"{gabinete}",
    "ext_telefonica":"{ext_telefonica}"
}
```

PUT

Apenas pode ser editado um utilizador de cada vez.

AUTHORIZATION Bearer Token

This folder is using Bearer Token from collection SIGEDEE

PUT Alterar Tipo de Utilizador

url/utilizadores/id_utilizador/tipo

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

json

```
{
    "novo_tipo_utilizador":"{id_tipo_utilizador}"
}
```

PUT Desativar Utilizador

url/utilizadores/id_utilizador/desativar

AUTHORIZATION Bearer Token

This request is using Bearer Token from collection SIGEDEE

A.2 Documentação MariaDB

1 datasheet_equipamento

Criação: 31-Jul-2023 às 13:23

Coluna	Tipo	Atributos	Nulo	Predefinido	Extra	Ligações para	Comentários	MIME
id_datasheet	int(11)		Não		auto_increment			
filename	text		Não					
id_descricao_equipamento	int(11)		Não			-> descricao_equipamento.id_descricao_equipamento ON UPDATE RESTRICT ON DELETE RESTRICT		
inserido_em	datetime		Não					
inserido_por	int(11)		Não			-> utilizadores.id_utilizador ON UPDATE RESTRICT ON DELETE RESTRICT		
visivel	int(11)		Não					

2 descricao_equipamento

Criação: 31-Jul-2023 às 13:23

Coluna	Tipo	Atributos	Nulo	Predefinido	Extra	Ligações para	Comentários	MIME
id_descricao_equipamento	int(11)		Não		auto_increment			
id_tipo_equipamento	int(11)		Não			-> tipo_equipamento.id_tipo_equipamento ON UPDATE RESTRICT ON DELETE RESTRICT		
marca	text		Não					
modelo	text		Não					
caracteristicas	text		Não					

3 equipamentos

Criação: 31-Jul-2023 às 13:23

Coluna	Tipo	Atributos	Nulo	Predefinido	Extra	Ligações para	Comentários	MIME
id_equipamento	int(11)		Não		auto_increment			
id_ultimo_registo	int(11)		Sim	NULL		-> equipamentos_historico.id_registo ON UPDATE RESTRICT ON DELETE RESTRICT		
inserido_em	datetime		Não				para saber quando deu entrada na BD	
inserido_por	int(11)		Não			-> utilizadores.id_utilizador ON UPDATE RESTRICT ON DELETE RESTRICT		

4 equipamentos_historico

Criação: 31-Jul-2023 às 13:23

Coluna	Tipo	Atributos	Nulo	Predefinido	Extra	Ligações para	Comentários	MIME
id_registo	int(11)		Não		auto_increment			
id_equipamento	int(11)		Não			-> equipamentos.id_equipamento ON UPDATE RESTRICT ON DELETE RESTRICT		
cativacao	varchar(255)		Não			-> instalacoes.numero_instalacao ON UPDATE RESTRICT ON DELETE RESTRICT		
localizacao	varchar(255)		Não			-> instalacoes.numero_instalacao ON UPDATE RESTRICT ON DELETE RESTRICT		
local_na_sala	text		Sim	NULL				
id_descricao	int(11)		Não			-> descricao_equipamento.id_descricao_equipamento ON UPDATE RESTRICT ON DELETE RESTRICT		
n_serie	text		Sim	NULL				
n_inventario_DEE	text		Sim	NULL				
n_inventario_ISEP	text		Sim	NULL				
n_inventario_IPP	text		Sim	NULL				
id_estado	int(11)		Não			-> estado_equipamento.id_estado_equipamento ON UPDATE RESTRICT ON DELETE RESTRICT		
id_fornecedor	int(11)		Não			-> fornecedores.id_fornecedor ON UPDATE RESTRICT ON DELETE RESTRICT		
custo_com_IVA	double		Sim	NULL				
data_aquisicao	date		Sim	NULL				

inserido_em	datetime		Não				para saber quando foi efetuado registo (insercao ou edicao)	
inserido_por	int(11)		Não			-> utilizadores.id_utilizador ON UPDATE RESTRICT ON DELETE RESTRICT		
observacoes	text		Sim	NULL				
visivel	int(11)		Não					

5 equipamentos_transacoes

Criação: 31-Jul-2023 às 13:23

Coluna	Tipo	Atributos	Nulo	Predefinido	Extra	Ligações para	Comentários	MIME
id_transacao	int(11)		Não		auto_increment			
id_registo	int(11)		Não			-> equipamentos_historico.id_registo ON UPDATE RESTRICT ON DELETE RESTRICT		
dados_json	longtext		Não					
signature	varbinary(1024)		Não					
public_key	int(11)		Não			-> public_keys.id_public_key ON UPDATE RESTRICT ON DELETE RESTRICT		
inserida_em	datetime		Não					

6 estado_equipamento

Criação: 31-Jul-2023 às 13:23

Coluna	Tipo	Atributos	Nulo	Predefinido	Extra	Ligações para	Comentários	MIME
id_estado_equipamento	int(11)		Não		auto_increment			
estado	text		Não					
estado_ordem	int(11)		Não					

7 estado_pedido

Criação: 31-Jul-2023 às 13:23

Coluna	Tipo	Atributos	Nulo	Predefinido	Extra	Ligações para	Comentários	MIME
id_estado_pedido	int(11)		Não		auto_increment			
estado_pedido	text		Não					
estado_pedido_ordem	int(11)		Não					

8 estado_ticket

Criação: 31-Jul-2023 às 13:23

Coluna	Tipo	Atributos	Nulo	Predefinido	Extra	Ligações para	Comentários	MIME
id_estado_ticket	int(11)		Não		auto_increment			
estado_ticket	text		Não					
estado_ticket_ordem	int(11)		Não					

9 fornecedores

Criação: 31-Jul-2023 às 13:23

Coluna	Tipo	Atributos	Nulo	Predefinido	Extra	Ligações para	Comentários	MIME
id_fornecedor	int(11)		Não		auto_increment			
id_ultimo_registo	int(11)		Sim	NULL		-> fornecedores_historico.id_registo ON UPDATE RESTRICT ON DELETE RESTRICT		
inserido_em	datetime		Não					
inserido_por	int(11)		Não			-> utilizadores.id_utilizador ON UPDATE RESTRICT ON DELETE RESTRICT		

10 fornecedores_historico

Criação: 31-Jul-2023 às 13:23

Coluna	Tipo	Atributos	Nulo	Predefinido	Extra	Ligações para	Comentários	MIME
id_registo	int(11)		Não		auto_increment			
id_fornecedor	int(11)		Não			-> fornecedores.id_fornecedor ON UPDATE RESTRICT ON DELETE RESTRICT		
firma	text		Não					
NIF	text		Sim	NULL				
url_website	mediumtext		Sim	NULL				
email_geral	mediumtext		Sim	NULL				
telefone_1	mediumtext		Sim	NULL				
telefone_2	mediumtext		Sim	NULL				
fax	mediumtext		Sim	NULL				
endereco_postal	mediumtext		Sim	NULL				
codigo_postal	mediumtext		Sim	NULL				
localidade	mediumtext		Sim	NULL				
cidade	mediumtext		Sim	NULL				
pais	mediumtext		Sim	NULL				
nome_pessoa_contacto	mediumtext		Sim	NULL				
email_pessoa_contacto	mediumtext		Sim	NULL				
telefone_pessoa_contacto	mediumtext		Sim	NULL				
inserido_por	int(11)		Não			-> utilizadores.id_utilizador ON UPDATE RESTRICT ON DELETE RESTRICT		
inserido_em	datetime		Não					

11 fornecedores_transacoes

Criação: 31-Jul-2023 às 13:23

Coluna	Tipo	Atributos	Nulo	Predefinido	Extra	Ligações para	Comentários	MIME
id_transacao	int(11)		Não		auto_increment			
id_registo	int(11)		Não			-> fornecedores_historico.id_registo ON UPDATE RESTRICT ON DELETE RESTRICT		
dados_json	longtext		Não					
signature	varbinary(1024)		Não					
public_key	int(11)		Não			-> public_keys.id_public_key ON UPDATE RESTRICT ON DELETE RESTRICT		
inserida_em	datetime		Não					

12 foto_equipamento

Criação: 31-Jul-2023 às 13:23

Coluna	Tipo	Atributos	Nulo	Predefinido	Extra	Ligações para	Comentários	MIME
id_foto	int(11)		Não		auto_increment			
filename	text		Não					
id_descricao_equipamento	int(11)		Não			-> descricao_equipamento.id_descricao_equipamento ON UPDATE RESTRICT ON DELETE RESTRICT		
inserido_em	datetime		Não					
inserido_por	int(11)		Não			-> utilizadores.id_utilizador ON UPDATE RESTRICT ON DELETE RESTRICT		
visivel	int(11)		Não					

13 helpdesk

Criação: 31-Jul-2023 às 13:23

Coluna	Tipo	Atributos	Nulo	Predefinido	Extra	Ligações para	Comentários	MIME
id_ticket	int(11)		Não		auto_increment			
id_equipamento	int(11)		Não			-> equipamentos.id_equipamento ON UPDATE RESTRICT ON DELETE RESTRICT		
titulo_problema	text		Não					
descricao_problema	text		Não					
reportado_em	datetime		Não					
reportado_por	int(11)		Não			-> utilizadores.id_utilizador ON UPDATE RESTRICT ON DELETE RESTRICT		
prioridade_ticket	int(11)		Não			-> prioridade.id_prioridade ON UPDATE RESTRICT ON DELETE RESTRICT		
estado_ticket	int(11)		Não			-> estado_ticket.id_estado_ticket ON UPDATE RESTRICT ON DELETE RESTRICT		
cancelado_em	datetime		Sim	NULL				
cancelado_por	int(11)		Sim	NULL		-> utilizadores.id_utilizador ON UPDATE RESTRICT ON DELETE RESTRICT		
processamento_em	datetime		Sim	NULL				
processamento_por	int(11)		Sim	NULL		-> utilizadores.id_utilizador ON UPDATE RESTRICT ON DELETE RESTRICT		
resolvido_em	datetime		Sim	NULL				

resolvido_por	int(11)		Sim	NULL		-> utilizadores.id_utilizador ON UPDATE RESTRICT ON DELETE RESTRICT		

14 historico_conversacao

Criação: 31-Jul-2023 às 13:23

Coluna	Tipo	Atributos	Nulo	Predefinido	Extra	Ligações para	Comentários	MIME
id_mensagem	int(11)		Não		auto_increment			
id_ticket	int(11)		Não			-> helpdesk.id_ticket ON UPDATE RESTRICT ON DELETE RESTRICT		
mensagem	text		Não					
enviada_por	int(11)		Não			-> utilizadores.id_utilizador ON UPDATE RESTRICT ON DELETE RESTRICT		
enviada_em	datetime		Não					

15 instalacoes

Criação: 31-Jul-2023 às 13:23

Coluna	Tipo	Atributos	Nulo	Predefinido	Extra	Ligações para	Comentários	MIME
numero_instalacao	varchar(255)		Não					
id_ultimo_registo	int(11)		Sim	NULL		-> instalacoes_historico.id_registo ON UPDATE RESTRICT ON DELETE RESTRICT		
inserido_em	datetime		Não					
inserido_por	int(11)		Não			-> utilizadores.id_utilizador ON UPDATE RESTRICT ON DELETE RESTRICT		

16 instalacoes_historico

Criação: 31-Jul-2023 às 13:23

Coluna	Tipo	Atributos	Nulo	Predefinido	Extra	Ligações para	Comentários	MIME
id_registo	int(11)		Não		auto_increment			
numero_instalacao	varchar(255)		Não			-> instalacoes.numero_instalacao ON UPDATE RESTRICT ON DELETE RESTRICT		
tipo_instalacao	text		Não					
nome_instalacao	text		Não					
ext_telefonica	int(11)		Não					
id_director	int(11)		Não			-> utilizadores.id_utilizador ON UPDATE RESTRICT ON DELETE RESTRICT		
id_tecnico	int(11)		Não			-> utilizadores.id_utilizador ON UPDATE RESTRICT ON DELETE RESTRICT		
inserido_em	datetime		Não					
inserido_por	int(11)		Não			-> utilizadores.id_utilizador ON UPDATE RESTRICT ON DELETE RESTRICT		

17 instalacoes_transacoes

Criação: 31-Jul-2023 às 13:23

Coluna	Tipo	Atributos	Nulo	Predefinido	Extra	Ligações para	Comentários	MIME
id_transacao	int(11)		Não		auto_increment			
id_registo	int(11)		Não			-> instalacoes_historico.id_registo ON UPDATE RESTRICT ON DELETE RESTRICT		
dados_json	longtext		Não					
signature	varbinary(1024)		Não					
public_key	int(11)		Não			-> public_keys.id_public_key ON UPDATE RESTRICT ON DELETE RESTRICT		
inserida_em	datetime		Não					

18 movimentos

Criação: 04-Ago-2023 às 11:44

Coluna	Tipo	Atributos	Nulo	Predefinido	Extra	Ligações para	Comentários	MIME
id_movimento	int(11)		Não		auto_increment			
codigo_levantamento	text		Sim	NULL				
codigo_devolucao	text		Sim	NULL				
id_equipamento	int(11)		Não			-> equipamentos.id_equipamento ON UPDATE RESTRICT ON DELETE RESTRICT		
localizacao_original	varchar(255)		Não			-> instalacoes.numero_instalacao ON UPDATE RESTRICT ON DELETE RESTRICT		
nova_localizacao	varchar(255)		Não			-> instalacoes.numero_instalacao ON UPDATE RESTRICT ON DELETE RESTRICT		
pedido_por	int(11)		Não			-> utilizadores.id_utilizador ON UPDATE RESTRICT ON DELETE RESTRICT		
data_requisicao	datetime		Não					
estado_inicial	int(11)		Não			-> estado_equipamento.id_estado_equipamento ON UPDATE RESTRICT ON DELETE RESTRICT		
estado_pedido	int(11)		Não			-> estado_pedido.id_estado_pedido ON UPDATE RESTRICT ON DELETE RESTRICT		
autorizado_por	int(11)		Sim	NULL		-> utilizadores.id_utilizador ON UPDATE RESTRICT ON DELETE RESTRICT		
autorizado_em	datetime		Sim	NULL				

emprestado_por	int(11)		Sim	NULL		-> utilizadores.id_utilizador ON UPDATE RESTRICT ON DELETE RESTRICT		
emprestado_em	datetime		Sim	NULL				
devolvido_por	int(11)		Sim	NULL		-> utilizadores.id_utilizador ON UPDATE RESTRICT ON DELETE RESTRICT		
devolvido_em	datetime		Sim	NULL				
estado_final	int(11)		Sim	NULL		-> estado_equipamento.id_estado_equipamento ON UPDATE RESTRICT ON DELETE RESTRICT		

19 prioridade

Criação: 31-Jul-2023 às 13:23

Coluna	Tipo	Atributos	Nulo	Predefinido	Extra	Ligações para	Comentários	MIME
id_prioridade	int(11)		Não		auto_increment			
prioridade	text		Não					
prioridade_ordem	int(11)		Não					

20 public_keys

Criação: 31-Jul-2023 às 13:23

Coluna	Tipo	Atributos	Nulo	Predefinido	Extra	Ligações para	Comentários	MIME
id_public_key	int(11)		Não		auto_increment			
public_key	text		Não					
id_utilizador	int(11)		Não			-> utilizadores.id_utilizador ON UPDATE RESTRICT ON DELETE RESTRICT		
inserida_em	datetime		Não					

21 tipo_equipamento

Criação: 31-Jul-2023 às 13:23

Coluna	Tipo	Atributos	Nulo	Predefinido	Extra	Ligações para	Comentários	MIME
id_tipo_equipamento	int(11)		Não		auto_increment			
tipo_equipamento	text		Não					

22 tipo_utilizador

Criação: 31-Jul-2023 às 13:23

Coluna	Tipo	Atributos	Nulo	Predefinido	Extra	Ligações para	Comentários	MIME
id_tipo_utilizador	int(11)		Não					
tipo_utilizador	text		Não					

23 utilizadores

Criação: 31-Jul-2023 às 13:23
Última actualização: 09-Set-2023 às 12:35

Coluna	Tipo	Atributos	Nulo	Predefinido	Extra	Ligações para	Comentários	MIME
id_utilizador	int(11)		Não		auto_increment			
tipo_utilizador	int(11)		Não			-> tipo_utilizador.id_tipo_utilizador ON UPDATE RESTRICT ON DELETE RESTRICT		
nome_utilizador	text		Não					
email	text		Sim	NULL				
palavra_passe	text		Sim	NULL				
ultimo_acesso	datetime		Não					
contador_acessos	int(11)		Não	0				
nome_apelido_utilizador	text		Não					
ext_telefonica	int(11)		Sim	NULL				
gabinete	text		Sim	NULL				
sigla	text		Não					

A.3 Documentação da Base de Dados Immudb

Tabela A.1: Documentação Immudb

Grupo	***Key***	***Value***
"public_keys"	"publicKey_{sigla}"	"{"public_key":"{public_key}, "utilizador":"{sigla}, "inserida_em":"{data de inserção}"}"
"equipamentos"	"equipamentos_{id_equipamento}"	{"id_transacao", id_transacao, "inserida_por":"sigla do utilizador", "inserida_em":"data de inserção da transação", "id_registo":id_registo, "dados":"dados em formato JSON", "signature":"dados assinados", "public_key":"chave pública do utilizador", "data_primeira_insercao":"data do primeiro registo do id na MariaDB"}
"fornecedores"	"fornecedores_{id_fornecedor}"	
"instalacoes"	"instalacoes_{numero_instalacao}"	