DM

Blockchain no desenvolvimento de dApps

DISSERTAÇÃO DE MESTRADO

Diogo Amândio Gonçalves Gouveia
MESTRADO EM ENGENHARIA INFORMÁTICA

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

fevereiro | 2026

Blockchain no desenvolvimento de dApps

DISSERTAÇÃO DE MESTRADO

Diogo Amândio Gonçalves Gouveia
MESTRADO EM ENGENHARIA INFORMÁTICA

ORIENTAÇÃO
Leonel Domingos Telo Nóbrega

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

Faculdade de Ciências Exatas e da Engenharia

Mestrado em Engenharia Informática

Blockchain no desenvolvimento de dApps

Diogo Amândio Gonçalves Gouveia — nº 2081020

Orientado por:

Prof. Leonel Domingos Telo Nóbrega

Constituição do júri de provas públicas:

Karolina Baras (Professora Auxiliar), Presidente

Pedro Filipe Pereira Campos (Professor Catedrático), Vogal

Leonel Domingos Telo Nóbrega (Professor Auxiliar), Vogal

6 de maio de 2026

PRR Plano de Recuperação e Resiliência
REPÚBLICA PORTUGUESA
Financiado pela União Europeia NextGenerationEU

Resumo

O desenvolvimento de aplicações descentralizadas baseadas em blockchain enfrenta desafios significativos ao nível do desempenho, da escalabilidade e da experiência do utilizador, particularmente quando comparado com aplicações web tradicionais. Este trabalho analisa o impacto de diferentes estratégias de gestão de dados no contexto de uma aplicação descentralizada baseada em Hyperledger Fabric, com especial foco no papel do *caching* local, da sincronização e da resolução de conflitos. Foi concebida e implementada uma arquitetura híbrida, combinando mecanismos de cache em memória e persistente no cliente com uma camada intermédia de API que comunica com a blockchain. A avaliação experimental demonstra que estas estratégias permitem reduzir significativamente a latência percebida em operações de leitura e melhorar a previsibilidade do comportamento do sistema, embora persistam limitações ao nível da escalabilidade e das operações de escrita, inerentes à tecnologia subjacente. Os resultados obtidos evidenciam que, apesar dos compromissos necessários, é possível aproximar a experiência de utilização de uma aplicação descentralizada à de uma aplicação web convencional, mantendo simultaneamente as garantias de integridade e rastreabilidade proporcionadas pela blockchain. O trabalho contribui, assim, para a discussão sobre a adequabilidade do uso de blockchain em sistemas de gestão de dados distribuídos, identificando benefícios, limitações e direções para trabalho futuro.

Keywords: Blockchain · dApps · Hyperledger Fabric · Metamodelação

Abstract

The development of blockchain-based decentralized applications poses significant challenges in terms of performance, scalability, and user experience, particularly when compared to traditional web applications. This work investigates the impact of different data management strategies in the context of a decentralized application built on Hyperledger Fabric, with a specific focus on client-side caching, synchronization, and conflict resolution mechanisms. A hybrid architecture was designed and implemented, combining in-memory and persistent client-side caching mechanisms with an API layer acting as an intermediary between the client and the blockchain. The experimental evaluation shows that these strategies significantly reduce perceived read latency and improve system predictability, although scalability limitations and high write latencies remain, largely due to the inherent characteristics of the underlying blockchain technology. Overall, the results demonstrate that it is possible to bring the user experience of a decentralized application closer to that of a conventional web application, while preserving the integrity and traceability guarantees provided by blockchain. This work contributes to the discussion on the suitability of blockchain for distributed data management systems, highlighting both its potential and its current limitations.

Keywords: Blockchain · dApps · Hyperledger Fabric · Metamodelling

Reconhecimentos

O presente trabalho foi desenvolvido no âmbito do projeto BioComp 3.0 (Ref.ª PRR-C05-i03-I-000224), ao qual agradeço a oportunidade de participação e o contexto de investigação proporcionado. Agradeço igualmente à entidade financiadora (PRR e União Europeia através do NextGenerationEU) e à Universidade da Madeira pelo apoio concedido ao longo do desenvolvimento deste trabalho. Gostaria de expressar um especial agradecimento ao Professor Leonel Nóbrega pela orientação, confiança e disponibilidade demonstradas ao longo de todo o processo. Para além do acompanhamento científico, o seu contributo foi determinante no desenho da arquitetura do sistema e dos mecanismos implementados, promovendo a aplicação consistente de princípios clássicos da engenharia de software, como baixo acoplamento, elevada coesão, responsabilidade única e facilidade de evolução futura. A sua intervenção teve também um impacto relevante no desenho da interface, nomeadamente na definição de componentes fundamentais para a qualidade e robustez da solução final. Agradeço também aos colegas que contribuíram diretamente para o projeto. Em particular, ao Pedro Andrade, pelo desenho, implementação e gestão da infraestrutura da blockchain e da REST API, componente essencial para o funcionamento do sistema, ao Carlos Coelho, pelo contributo nas questões de desenho, modelação e gestão do metamodelo utilizado, e ao Afonso Campos e o Pedro Ferreira pelo apoio na implementação do frontend e pelas discussões e sugestões que enriqueceram a abordagem adotada. Por fim, reconheço a utilização de ferramentas de inteligência artificial como apoio à reformulação e clarificação de texto, utilizadas como instrumento auxiliar no processo de escrita, sem substituir o contributo intelectual e crítico do autor.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

CRUD *Create, Read, Update, Delete*

HLF Hyperledger Fabric

IoT *Internet of Things*

MVC Model-View-Controller

MVVM Model-View-ViewModel

ORM *Object-Relational Mapping*

PoS *Proof of Stake*

PoW *Proof of Work*

QAW *Quality Attribute Workshop*

SOA Service Oriented Architecture

UI *User Interface* (interface do utilizador)

UX *User Experience* (experiência do utilizador)

dApp Aplicação descentralizada

1 Introdução

1.1 Contexto

Esta dissertação surge no contexto do projeto BioComp 3.0[1], um projeto que se insere na agenda de investigação e inovação para a sustentabilidade da agricultura, alimentação e agroindústria, com o objetivo de promover práticas alinhadas aos princípios da Economia Circular. O foco central do projeto é promover a produção de compostos orgânicos e biológicos, aproveitando resíduos provenientes do jacinto-de-água (*Eichhornia crassipes*), uma planta aquática invasora que representa uma séria ameaça ecológica aos ecossistemas. Esta abordagem visa mitigar os impactos ambientais provocados por esta espécie, ao mesmo tempo que transforma a biomassa recolhida em produtos úteis e sustentáveis para o setor agrícola, contribuindo também para o aumento da fertilidade dos solos.

Um dos pilares deste projeto é a desmaterialização do processo de produção de composto, que inclui etapas como a monitorização, recolha e transporte da espécie, bem como a sua transformação em compostos orgânicos e, por fim, a aplicação desses compostos. Neste contexto, surgiu a oportunidade de desenvolver uma aplicação web descentralizada (dApp) baseada em blockchain, com o objetivo de assegurar a rastreabilidade, integridade e transparência de todas as fases do processo de produção.

A blockchain é uma plataforma distribuída baseada num modelo *peer-to-peer*, que atua como um livro-razão transparente e imutável, capaz de registar todas as transações efetuadas nesta rede, sem necessidade de intermediários [1, 8, 9]. Uma das suas principais características é a descentralização: em vez de depender de uma autoridade central, todos os participantes, ou *peers*, trabalham coletivamente para validar e armazenar as transações [1, 10, 11]. As informações são agrupadas em blocos que são criptograficamente ligados uns aos outros numa sequência contínua, formando uma "cadeia de blocos" [4, 9, 12]. Estas propriedades conferem à blockchain vantagens significativas, como a imutabilidade e consistência dos dados, integridade, segurança, transparência e auditabilidade [2, 3, 13]. Esta tecnologia, inicialmente popularizada pelo Bitcoin para a realização de transações financeiras [1, 14, 15], expandiu-se para diversos setores devido às suas propriedades únicas, revelando-se uma mais valia em áreas como a gestão de cadeias de abastecimento, saúde, *Internet of Things* (IoT), cidades inteligentes e na agricultura. [16–18].

[1]Projeto BioComp 3.0 – Agenda de Investigação e Inovação para a Sustentabilidade da Agricultura, Alimentação e Agroindústria (Ref.ª PRR-C05-i03-I-000224), financiado pelo PRR e NextGenerationEU.

1.2 Problema

Apesar do potencial transformador da blockchain para assegurar a rastreabilidade, integridade e transparência de processos [19, 20], a sua adoção traz um conjunto significativo de desafios técnicos e conceptuais. Uma das principais dificuldades no desenvolvimento deste tipo de aplicações deve-se à complexidade inerente da própria tecnologia, cuja infraestrutura apresenta características únicas que a diferenciam substancialmente de sistemas tradicionais, exigindo esforços não triviais por parte das equipas de desenvolvimento [16]. Além disso, a sua curva de aprendizagem exigente, combinada com a escassez de documentação clara, de suporte para as ferramentas existentes e de boas práticas amplamente estabelecidas contribuem para tornar o processo de desenvolvimento moroso e suscetível a decisões técnicas inadequadas [11, 21].

Para além destes desafios mais gerais relacionados com a complexidade da própria tecnologia e a escassez de conhecimento consolidado sobre o seu uso no desenvolvimento de dApps, esta dissertação foca-se num problema mais específico: a definição e implementação de mecanismos eficazes de gestão de dados numa dApp baseada em blockchain, a partir da perspetiva do cliente. O uso da blockchain para armazenamento de dados traz uma série de implicações arquiteturais significativas, uma vez que operações simples, como leituras ou criações de dados, estão sujeitas a restrições de latência, *throughput* e consistência. Neste cenário, torna-se necessário planear estratégias que otimizem operações fundamentais, como criação, leitura, atualização e eliminação de dados (CRUD), garantindo simultaneamente atributos de qualidade como desempenho, disponibilidade e escalabilidade, que são cruciais para o bom funcionamento de qualquer sistema.

Estas lacunas reforçam a importância de estudar abordagens arquiteturais adequadas e propor táticas orientadoras que apoiem o desenvolvimento de dApps focadas na gestão eficiente de dados e das diversas operações possíveis, contribuindo para a maturidade e adoção sustentável destas tecnologias em contextos reais. Com base nestes desafios identificados, este trabalho é orientado por um conjunto de questões de investigação que visam estruturar a análise e a avaliação das decisões arquiteturais adotadas no desenvolvimento de uma dApp baseada em blockchain, a partir da perspetiva do cliente. Em particular, procuram-se responder às seguintes questões:

- **QI1** — Qual o impacto de diferentes estratégias de gestão local de dados, incluindo mecanismos de *caching*, no desempenho e escalabilidade de uma aplicação descentralizada baseada em blockchain?
- **QI2** — De que forma podem ser geridas a sincronização e os conflitos no cliente de uma dApp, de modo a preservar a consistência dos dados e uma experiência de utilizador aceitável em cenários concorrentes?

Estas questões orientam a definição dos objetivos, critérios de aceitação e avaliação experimental apresentados nos capítulos seguintes, bem como a análise crítica das soluções propostas.

1.3 Contribuições

Esta dissertação procura contribuir para a compreensão dos desafios associados ao desenvolvimento de aplicações descentralizadas, analisando em particular de que forma a adoção de blockchain influencia decisões arquiteturais e o desenho de mecanismos de integração entre cliente e *ledger*. Através do estudo aprofundado do BioComp, utilizado como caso de estudo real, este trabalho investiga problemas que surgem na construção de uma dApp, revelando limitações, tensões arquiteturais e requisitos adicionais que não são evidentes em abordagens tradicionais.

A partir desta análise, o trabalho propõe um conjunto de reflexões, padrões e diretrizes que procuram clarificar como diferentes estratégias de integração, incluindo gestão de latência, coerência de dados, disponibilidade, resiliência e tratamento de falhas, podem ser concebidas de forma a equilibrar os benefícios da blockchain com as necessidades práticas de uma aplicação cliente moderna. Embora estas recomendações tenham emergido a partir das necessidades específicas do BioComp, elas foram formuladas de modo a serem generalizáveis e úteis para equipas que enfrentem desafios semelhantes na construção de dApps orientadas por requisitos de desempenho, escalabilidade e disponibilidade.

Importa sublinhar que o foco da dissertação recai essencialmente sobre a camada cliente e sobre o desenho de mecanismos que permitam uma integração eficaz entre a aplicação e a blockchain. Consequentemente, questões relacionadas com o desenvolvimento e *deployment* da rede blockchain, ou de APIs associadas, e o desenho e implementação de *smart contracts* ou *chaincodes* são abordadas apenas de forma indireta, servindo sobretudo para contextualizar decisões tomadas na camada de aplicação. O contributo central deste trabalho reside assim na análise arquitetural, na identificação de desafios concretos e na sistematização de práticas que apoiem o desenvolvimento de dApps mais robustas e sustentáveis.

1.4 Estrutura

Este documento encontra-se organizado da seguinte forma. O Capítulo 2 apresenta os conceitos fundamentais necessários à compreensão do trabalho, incluindo blockchain, aplicações descentralizadas e os principais desafios associados ao seu desenvolvimento. O Capítulo 3 descreve o problema em estudo, os objetivos do trabalho e os critérios de aceitação definidos. No Capítulo 4 é apresentada a evolução da arquitetura do sistema

implementado e as principais decisões de desenho adotadas, bem como os mecanismos implementados. O Capítulo 5 descreve a avaliação experimental realizada e discute os resultados obtidos, identificando também as principais limitações do trabalho. Por fim, o Capítulo 6 apresenta as conclusões do trabalho e possíveis direções para trabalho futuro.

2 Revisão de Literatura

2.1 Enquadramento e objetivos

A revisão de literatura desenvolvida neste capítulo tem como objetivo, numa primeira fase, aprofundar o conhecimento sobre blockchain, explorando o seu funcionamento, conceitos fundamentais e exemplos de aplicação prática em diferentes domínios, nomeadamente na área da agricultura. Numa segunda fase, procura-se analisar o estado atual da investigação sobre o desenvolvimento de aplicações baseadas em blockchain, com ênfase nos desafios, restrições e decisões arquiteturais que influenciam o desenho e a implementação de soluções descentralizadas. Este trabalho insere-se num contexto em que a adoção da blockchain, embora promissora, traz implicações significativas a nível da arquitetura do software e da modelação da lógica de negócio, influenciando imenso o processo de desenvolvimento de uma aplicação. Assim, torna-se essencial reunir, analisar e sintetizar estudos prévios que abordem essas questões, de forma a fundamentar as decisões e propostas apresentadas nos capítulos seguintes.

2.2 Metodologia

A revisão de literatura realizada nesta dissertação seguiu uma abordagem exploratória e estruturada, inspirada em princípios comuns das *Scoping Reviews*, mas sem ambicionar cumprir integralmente todos os requisitos metodológicos desse formato. O objetivo principal foi mapear o estado da arte sobre a tecnologia blockchain e o seu uso em aplicações distribuídas, identificar conceitos fundamentais, plataformas relevantes, padrões arquiteturais e exemplos de aplicação, e reunir evidência útil para orientar decisões de projeto no contexto do BioComp. A estratégia adotada consistiu na identificação de fontes relevantes, seleção de artigos a partir da leitura dos títulos e resumos, e análise qualitativa do conteúdo, sintetizando os seus principais contributos, limitações e implicações técnicas. Deste modo, tratou-se de uma revisão abrangente, iterativa e focada na construção de uma compreensão integrada do tema, embora não exaustiva. Estudos futuros poderão complementar ou aprofundar áreas específicas que ultrapassam o escopo desta dissertação.

A pesquisa bibliográfica foi maioritariamente conduzida entre setembro de 2024 e fevereiro de 2025, sendo atualizada pontualmente até janeiro de 2026. Esta baseou-se nas plataformas *ACM Digital Library*[2], *IEEE Xplore*[3] e *Google Scholar*[4], sendo complementada por pesquisas orientadas no *Semantic Scholar*[5], *Research*

[2] `https://dl.acm.org/`
[3] `https://ieeexplore.ieee.org`
[4] `https://scholar.google.com/`
[5] `https://www.semanticscholar.org/`

Rabbit[6] e *Consensus*[7]. As *keywords* iniciais incluíram termos como “Blockchain”, “Blockchain applications”, “Smart contracts”, “dApp development”, “Hyperledger Fabric” - esta por ser a plataforma escolhida para desenvolvimento do caso de estudo, conforme será explicado adiante, e “Blockchain Architectural Decisions”, ajustadas iterativamente conforme o avanço no trabalho desenvolvido no projeto.

O processo de seleção foi deliberadamente iterativo e pragmático: pesquisas amplas para mapeamento inicial, triagem por título/resumo para reduzir o conjunto e leitura integral dos artigos potencialmente relevantes para avaliar elegibilidade e extrair dados. Cerca de 70 artigos foram considerados relevantes e aceites para este trabalho (dos quais apenas um subconjunto é diretamente citado), e 20 foram descartados após avaliação em texto integral.

Os critérios de inclusão privilegiaram trabalhos que discutem o funcionamento da blockchain, plataformas e as suas características, quando e como utilizar blockchain para desenvolver dApps, decisões arquiteturais relevantes nesse processo de desenvolvimento, como obter determinados atributos de qualidade, nomeadamente desempenho, segurança e disponibilidade, mecanismos de sincronização *on/off-chain* e experiências práticas aplicáveis; os critérios de exclusão eliminaram artigos não técnicos (opinião sem contributo técnico), foco exclusivo em valorização de criptomoedas e casos de estudo muito específicos que não pudessem ser generalizados. Trabalhos centrados na otimização do desempenho exclusivamente ao nível da blockchain (desenho da rede ou implementação de algoritmos *on-chain*) foram analisados no âmbito do projeto mas não foram considerados para esta dissertação.

Para cada estudo incluído, foram extraídos e registados de forma padronizada: plataforma alvo, objetivo, método, principais resultados, limitações e relevância para este trabalho. O processo de síntese foi qualitativo e temático, orientado para identificar tendências, decisões arquiteturais recorrentes e lacunas que fundamentam as escolhas do projeto.

2.3 Contextualização

2.3.1 Evolução da Web e o surgimento da Web 3.0

A evolução da web é um percurso dinâmico que transformou profundamente a forma como interagimos com a informação e uns com os outros, passando de um modelo passivo de consumo para um de criação de conteúdo e, mais recentemente, para a posse e descentralização dos dados. Esta progressão é geralmente dividida em três eras principais: Web 1.0, Web 2.0 e Web 3.0.

[6] `https://researchrabbitapp.com/`
[7] `https://consensus.app/`

A Web 1.0, também chamada de web de apenas leitura, foi marcada por um modelo predominantemente informacional [19,22], no qual poucos produtores criavam páginas e a grande maioria dos utilizadores apenas consumia conteúdo [19]. Apesar de usar tecnologias fundamentais como HTML, HTTP e URL, esta geração da web tinha limitações significativas, como a comunicação unidirecional, atualização lenta de conteúdos e mecanismos de pesquisa pouco eficazes, tratando a internet mais como uma plataforma de publicação do que de participação [22].

A Web 2.0 inaugurou a era da web de leitura e escrita [22], caracterizada pelo conteúdo gerado pelo utilizador, colaboração em massa e efeito de redes sociais [3, 19, 22]. Utilizava tecnologias como AJAX, REST e *cloud computing*, e impulsionou a criação de plataformas digitais que passaram a concentrar grande parte do tráfego e dos dados, monetizando-os através de publicidade e outros modelos centralizados [22]. Apesar da maior interatividade, esta fase trouxe novos desafios de segurança, como *Cross Site Scripting* (XSS), *Cross Site Request Forging* (CSRF) e SQL Injection [22].

A Web 3.0, ou web de leitura, escrita e posse, representa a evolução rumo à descentralização [3, 19, 23]. Esta é impulsionada por tecnologias como blockchain, *smart contracts*, *edge computing* e inteligência artificial [19], e procura devolver o controlo dos dados aos utilizadores e mitigar a concentração de poder observada na Web 2.0, potencializando a descentralização [24]. Os seus elementos-chave são os tokens fungíveis e não fungíveis (NFTs), as Organizações Autónomas Descentralizadas (DAOs) e o Metaverso [3]. Os *smart contracts* constituem a base para o desenvolvimento de aplicações Web 3.0, como as dApps [3].

A integração do blockchain na Web 3.0 enfrenta desafios significativos, nomeadamente:

- Escalabilidade e desempenho: redes públicas ainda apresentam baixa taxa de transações e custos elevados [24];
- A manutenção e verificação constante dos *smart contracts* é essencial [24];
- Experiência do utilizador: a complexidade da tecnologia requer uma experiência de utilizador intuitiva e educação para os utilizadores [24];
- Privacidade e segurança: faltam padrões robustos para metadados e proteção de dados [22, 24];
- Impacto ambiental de blockchains baseadas em algoritmos de consenso *Proof of Work* [24] - estes algoritmos serão explicados mais à frente.

A transição para uma Web centrada na posse e na descentralização revela requisitos técnicos que as infraestruturas centralizadas da Web 2.0 não satisfazem: mecanismos de confiança sem intermediários, registos auditáveis imutáveis e coordenação programável entre atores independentes. Estas exigências tornaram imperativo o desenvolvimento de mecanismos de consenso e de *ledgers* distribuídos que garantam integridade, disponibilidade e resistência à censura. Nesse contexto, a blockchain emergiu como uma solução que articula criptografia, encadeamento de blocos e protocolos de consenso para suportar dApps e *smart contracts*, constituindo assim a base técnica sobre a qual as promessas da Web 3.0 podem ser materializadas, ainda que enfrentando outros desafios [3, 4, 19].

2.3.2 Surgimento da blockchain

A tecnologia blockchain foi introduzida em 2008 por uma pessoa sob o pseudónimo de Satoshi Nakamoto como parte do Bitcoin, um sistema de dinheiro eletrónico que substitui a confiança numa entidade central por mecanismos criptográficos [3, 14, 21]. O seu aparecimento coincidiu com um período de crise económica e crescente desconfiança nas instituições financeiras tradicionais, oferecendo uma alternativa descentralizada para pagamentos financeiros e interações entre entidades que não se confiam mutuamente [14, 15, 25]. Embora inicialmente criada para suportar transações financeiras, a blockchain rapidamente despertou interesse como tecnologia independente, com potencial para diversas aplicações para além das criptomoedas [1, 16].

2.3.3 Conceitos fundamentais

A blockchain consiste numa rede descentralizada, segura e distribuída, baseada numa arquitetura *peer-to-peer* [8, 26]. De maneira geral, não existem intermediários na comunicação entre os participantes da rede [11, 12, 18]. Essa descentralização, combinada com mecanismos de segurança, permite o registo seguro de transações entre utilizadores em ambientes onde a confiança entre as partes envolvidas é limitada [11, 16]. As transações, que são instruções assinadas criptograficamente pelos utilizadores da rede e que alteram o estado da blockchain [1, 9, 27], são organizadas em blocos que, por sua vez, são ligados entre si de forma sequencial [1, 12]. Este encadeamento de blocos dá origem ao termo blockchain (cadeia de blocos) [1, 21, 28]. Cada bloco contém diversas informações, incluindo um identificador único, o *hash*, que referencia o bloco anterior garantindo a imutabilidade dos dados e a integridade de toda a cadeia, pois qualquer alteração num bloco implicaria a alteração do *hash* dos blocos seguintes [9, 12, 21].

Os blocos são organizados num livro-razão distribuído (o *ledger*), replicado pelos nós participantes e responsável pelo registo imutável das transações [4, 11, 29]. Em algumas implementações empresariais,

nomeadamente o Hyperledger Fabric, efetua-se também a distinção entre o registo de transações (*append-only*) e uma base de dados do estado atual dos *assets* — o *World State Database* — que facilita consultas ao estado corrente sem percorrer todo o histórico [8]. Ao contrário do registo de transações, o *World State Database* permite alterações nos dados armazenados, refletindo assim o estado mais recente dos *assets*, que são objetos que possuem algum tipo de valor para os participantes [8].

Para que um bloco seja adicionado ao *ledger* recorre-se a um mecanismo de consenso, cujo propósito é harmonizar o estado da rede e mitigar comportamentos bizantinos [4, 25], que são adulterações dos dados por parte de nós desonestos, protegendo os dados de ataques de uma minoria ao alocar a responsabilidade de atualização dos blocos a candidatos selecionados aleatoriamente [19].

Os algoritmos de consenso mais populares são o *Proof of Work* (PoW) e o *Proof of Stake* (PoS). O *Proof of Work* foi proposto no Bitcoin [19] e é um mecanismo de consenso em que os participantes (mineradores) competem para resolver problemas criptográficos complexos, de forma a validar transações e a criar novos blocos. O primeiro minerador a resolver os problemas e a transmitir o bloco é recompensado [4, 21, 30]. É um algoritmo provado e maduro para sistemas descentralizados de longa escala [25], mas tem algumas desvantagens, como o alto consumo de energia devido aos cálculos necessários [25] e a vulnerabilidade ao ataque dos 51% [26], isto é, se uma entidade ou grupo controlar mais de 51% do poder computacional (*hash power*) de uma rede blockchain, essa entidade pode reescrever o histórico recente da blockchain, bloquear e atrasar transações de outros utilizadores e até minerar blocos mais rapidamente que o resto da rede [10]. Já o algoritmo *Proof of Stake* surgiu como alternativa mais eficiente ao PoW [9, 30] e neste os participantes da rede, denominados por validadores (em vez de mineradores), precisam de provar a sua "participação"ou "*stake*", o que envolve o bloqueio de uma certa quantia de criptomoedas ou *tokens* [4, 19, 26]. A chance de um nó ser escolhido como validador para criar o próximo bloco depende do seu *stake*, ou seja, aqueles com mais *tokens* têm uma maior probabilidade de serem selecionados [4, 19, 26]. Esta abordagem tem como vantagens a redução significativa do consumo de energia, devido à drástica redução do poder computacional necessário, tornando-a mais eficiente e escalável, tem um menor custo de entrada para a geração de blocos, quando comparado ao PoW, e é resistente a ataques dos 51% [10, 26, 30]. Contudo, também apresenta algumas vulnerabilidades, como o *nothing at stake*, isto é, os validadores não incorrem um custo financeiro ou computacional para obter recompensas, e o potencial de centralização, pois os participantes mais ricos podem dominar a rede [26, 28].

Algumas blockchains também possuem uma outra funcionalidade: os *smart contracts*, que são programas armazenados e executados na blockchain [31,32], permitindo a automatização e execução de comportamentos de negócio estabelecidos e acordados entre todos os participantes [6]. Isto garante que as transações ocorrem de forma transparente, segura e sem a necessidade de intermediários [28, 33]. Os *smart contracts* tornam a blockchain programável, o que aumenta a sua flexibilidade e funcionalidade [28]. Por exemplo, um *smart contract* pode ser utilizado para transferir automaticamente um *asset* entre dois utilizadores quando determinadas condições são atendidas. Um *smart contract* é ativado quando uma transação ocorre [31, 33]. Nalgumas plataformas, após o *deployment* de um contrato na blockchain não é possível efetuar qualquer alteração no seu código [9, 16], ou seja, este torna-se imutável. Isto tem implicações práticas importantes, pois quando são necessárias atualizações é preciso recorrer a padrões de *upgrade*, o que normalmente implica o *deployment* de outro *smart contract* [16, 34]. O modelo de execução e a política de imutabilidade variam entre plataformas e condicionam o ciclo de desenvolvimento e manutenção das dApps [16, 34].

Um componente essencial no desenvolvimento de *smart contracts* são os oráculos, pois estes atuam como ponte entre a rede e o mundo externo [9, 12, 32]. Os oráculos facilitam a coordenação de componentes dentro da rede usando um estado externo que é gerido independentemente [9]. Pode-se considerar os oráculos como serviços de dados que entregam informações de fora para dentro da blockchain. Estes são importantes no contexto dos *smart contracts*, pois os contratos não podem fazer chamadas diretamente para fora da blockchain por questões de verificabilidade [12, 32], por isso, quando é necessário aceder a informações exteriores à rede utiliza-se um oráculo, que fornece dados sobre o mundo exterior ao contrato [35].

2.3.4 Tipos de blockchain

As plataformas blockchain diferenciam-se consoante o seu modelo de permissão, existindo três tipos de blockchain: público, privado e de consórcio [25, 33]. Nas blockchains públicas (*permissionless*), qualquer utilizador pode participar livremente na rede, com acesso total aos dados, possibilidade de iniciar e validar transações, bem como executar *smart contracts* (se a plataforma suportar esta funcionalidade) [12, 33]. Estas blockchains são totalmente descentralizadas e as identidades dos participantes são anónimas [8]. Bitcoin e Ethereum são exemplos de blockchains públicas [8, 36]. Por outro lado, as blockchains privadas (*permissioned* ou permissionárias) são restritas a utilizadores autorizados [4, 36]. Este tipo de blockchain centraliza o controlo numa entidade, o que pode ter um impacto negativo na transparência. No entanto, como o número de participantes costuma ser menor comparativamente a blockchains públicas, a velocidade de processamento de transações tende a ser mais alta [33]. O Hyperledger Fabric é um exemplo de blockchain

privada [29, 36]. Por fim, alguns autores sugerem a existência de um outro tipo de blockchain, as blockchains de consórcio, que combinam as características dos modelos público e privado. A participação é restrita, como nas blockchains privadas, mas o controlo da rede é distribuído por um grupo de organizações em vez de centralizar-se numa única entidade [25, 33] e nem todos podem participar no processo de validação de transações [11]. O Hyperledger Fabric, devido às suas características, também pode ser considerado uma blockchain de consórcio [37].

2.3.5 Plataformas

A diversidade de plataformas blockchain reflete diferentes escolhas de desenho arquitetural, que por sua vez condicionam fortemente o comportamento, desempenho e os requisitos de integração das aplicações construídas sobre elas [2,4]. Redes públicas como Bitcoin, Ethereum ou Cardano privilegiam descentralização e resistência à censura, sacrificando frequentemente latência e *throughput*. Por outro lado, plataformas privadas ou de consórcio, como o Hyperledger Fabric ou o Corda, oferecem mecanismos de controlo de acesso, privacidade e maior parametrização do desempenho, características que as tornam mais apropriadas para ambientes empresariais [12, 19, 33]. A tabela 1 resume algumas das características estruturais mais relevantes das plataformas blockchain mais populares.

Tabela 1: Comparação das principais plataformas blockchain (fontes: [1, 4–7])

Plataforma	**Tipo**	**Criptomoeda**	**Consenso**	***Throughput***	**Linguagem**	**Completude Turing**
Bitcoin	Pública	Bitcoin (BTC)	PoW	Baixo	Script	Não
Ethereum	Pública	Ether (ETH)	PoS	Baixo	Solidity	Sim
Cardano	Pública	ADA	PoS	Baixo-Médio	Plutus (Haskell)	Sim
Hyperledger Fabric	Consórcio	—	Pluggable (Kafka/RAFT/PBFT/etc.)	Alto	Go/Node/Java	Sim
Corda	Consórcio	—	Unicidade/Validação	Alto	Java/Kotlin	?

As plataformas públicas mais representativas incluem o Bitcoin, que constituiu a primeira materialização da tecnologia e permanece orientado quase exclusivamente a pagamentos *peer-to-peer* [9, 18]. O seu modelo *permissionless*, baseado no algoritmo de consenso *Proof of Work* (PoW) [8,28,36], assegura elevada segurança, mas conduz a um baixo *throughput* e elevado consumo energético [28, 36]. Acresce que o seu sistema de *script* não é Turing-completo [6, 12], o que limita a capacidade de suportar transações programáveis para além de operações simples. O Ethereum expandiu estas limitações ao introduzir *smart contracts* Turing-completos, habilitando a automatização de lógica de negócio [6, 14, 38]. É uma plataforma pública e *permissionless*, mas que também pode ser configurada para operar no modo privado [30, 36], e migrou do PoW para *Proof of Stake* (PoS), reduzindo custos energéticos e melhorando alguns aspetos de escalabilidade [12, 19, 34]. Apesar disso, problemas de latência e custos de transação persistem, o que limita a sua adequação a cenários

empresariais de alta frequência [12,36,38]. O Cardano, também público e baseado em PoS, procura equilibrar segurança formal e descentralização através de uma arquitetura mais modular e de contratos inteligentes desenvolvidos com base em Haskell (Plutus), permitindo expressividade Turing-completa [5,6]. A literatura existente não fornece valores consensuais sobre o *throughput* do Cardano, que varia significativamente consoante a configuração da rede, no entanto, como a maioria das redes *permissionless*, tende a apresentar menor capacidade de processamento e maior latência do que plataformas permissionárias. Diversas evoluções protocolares têm vindo a ser propostas para mitigar estas limitações [2,5,6].

Num paradigma distinto situam-se as plataformas de consórcio. O Hyperledger Fabric destaca-se como uma solução permissionária orientada para ambientes empresariais. A sua arquitetura modular suporta diferentes algoritmos de consenso, políticas de acesso e um modelo de transação *Execute-Order-Validate* [4,29,39], que reduz a latência e aumenta o *throughput* em relação ao modelo tradicional *Order–Execute*, usado pela maioria das plataformas existentes [30,35]. Nesta plataforma, os contratos inteligentes são chamados de *chaincodes* e podem ser desenvolvidos em linguagens de propósito geral, como Go, Node.js ou Java, reduzindo a curva de aprendizagem [37,38,40]. O Fabric não possui criptomoeda nativa e introduz mecanismos como *channels*, que permitem criar sub-*ledgers* privados entre subconjuntos de organizações [30,37,41], reforçando privacidade e governação [37,41]. O Corda, também permissionário, adota um modelo ainda mais focado na privacidade, evitando a difusão global de transações e promovendo comunicação ponto-a-ponto entre entidades [1,6,35]. Os contratos são escritos em Java ou Kotlin e a plataforma não possui criptomoeda nativa embutida [4,6,12]. A literatura apresenta algumas divergências relativamente à completude de Turing da plataforma Corda: enquanto [6] classifica o Corda como não Turing-completo, [12] indica o contrário. Adicionalmente, o artigo [4] sugere implicitamente que o Corda é Turing-completo, embora sem o afirmar diretamente. Apesar destas inconsistências, o foco principal do Corda reside na interoperabilidade e na confidencialidade entre participantes [4,6].

2.3.6 Exemplos de aplicação

A tecnologia blockchain tem sido explorada em vários domínios para resolver problemas relacionados com confiança, rastreabilidade, integridade de dados e coordenação entre partes sem confiança mútua [1,10,19]. Um dos contextos mais consolidados é o das cadeias de abastecimento, onde a criação de registos imutáveis sobre lotes e eventos, como colheita, processamento ou transporte, permite garantir a rastreabilidade e autenticidade dos produtos [11,21,32], reduzindo o risco de adulteração e facilitando auditorias e respostas a incidentes de segurança alimentar [13,19,33].

Em paralelo, a convergência entre blockchain e IoT tem aberto novas possibilidades na gestão de dados agrícolas, permitindo o registo fidedigno de leituras de sensores de humidade, temperatura, condições meteorológicas ou telemetria de drones, e garantindo a integridade da informação utilizada em processos de automação e tomada de decisão [11, 19, 42]. Soluções baseadas em arquiteturas permissionárias, como o Hyperledger Fabric, possibilitam ainda a implementação de políticas de acesso, canais privados e mecanismos de *endorsement* que preservam dados sensíveis e asseguram consultas eficientes [6, 8, 29]. Estas abordagens têm sido especialmente exploradas em cenários de agricultura inteligente, cidades sustentáveis e orquestração de sistemas autónomos, incluindo redes de sensores e drones cooperativos [20].

Para além do setor agrícola, a blockchain tem tido impacto crescente em contextos financeiros e de automatização de processos, nomeadamente através da criação de livros-razão partilhados que reduzem a dependência de intermediários e aumentam a transparência e a segurança das transações [8, 11, 32]. Os *smart contracts* desempenham aqui um papel central, permitindo automatizar regras contratuais e fluxos de negócio, reduzindo custos e riscos de fraude, e simplificando a coordenação entre múltiplas organizações [26, 33, 35].

Outro domínio relevante é o da identidade digital e da certificação, onde soluções baseadas em blockchain têm sido propostas para gerir identidades, emitir certificados ou proteger dados de investigação, garantindo autenticidade e imutabilidade [19, 27, 42]. Estas mesmas propriedades de integridade e auditabilidade sustentam também aplicações em saúde e em cidades inteligentes, permitindo o armazenamento seguro de registos clínicos, a partilha controlada de informação sensível e a rastreabilidade de operações em sistemas urbanos complexos [17, 28, 43].

Finalmente, em setores ambientais e da economia circular, a tecnologia tem sido usada para associar incentivos económicos a comportamentos sustentáveis, como recolha de resíduos, reciclagem ou reutilização de biomassa, através de sistemas de *tokens* e *ledgers* transparentes [3, 19]. Neste contexto, a blockchain permite verificar práticas de transformação e acompanhar fluxos de materiais, potenciando novos modelos de mercado para subprodutos agrícolas e industriais [11, 19].

Assim, embora a adoção da tecnologia varie consoante o domínio, a sua aplicação converge na capacidade de criar confiança em ecossistemas distribuídos, reforçando a transparência, a integridade e a automatização em processos partilhados.

2.4 Discussão

Nesta secção apresenta-se um panorama crítico dos trabalhos mais relevantes para este estudo, destacando não só diretrizes e contribuições metodológicas que informam as decisões de design de dApps, mas também as lacunas e fragilidades que justificam a necessidade do presente trabalho.

2.4.1 Adequação do uso de blockchain

Um dos primeiros passos para desenvolver uma dApp baseia-se na avaliação da adequação da blockchain para o sistema em questão. Não se pretende aprofundar exaustivamente este tópico, visto que já existe literatura sobre o mesmo, mas apresenta-se antes um resumo das principais frameworks e critérios, que servem de alicerces para as recomendações práticas deste trabalho.

Nesse contexto, [44] refere que a base dessa discussão não deve ser apenas a possibilidade de usar blockchain para resolver um determinado problema, mas sim se a blockchain é a solução mais apropriada para o resolver. Além disso, os autores identificam cinco cenários para os quais a adoção da blockchain não é a melhor solução, principalmente devido às suas limitações de estrutura e desempenho:

- Substituir os sistemas monetários tradicionais pelas criptomoedas, devido à latência e estrutura fundamental da tecnologia;
- Armazenar dados confidenciais, pois estes têm de estar munidos do direito de serem esquecidos;
- Eleições digitais, pois é necessário confiar noutros elementos do sistema, o que enfraquece o valor da descentralização;
- Atender utilizadores sem acesso confiável à internet;
- Construir redes distribuídas de cálculos, uma vez que a blockchain representa uma enorme redundância de poder computacional.

Os autores deste trabalho também fizeram menção ao trabalho feito por Wüst e Gervais em [1], onde é sintetizada, num fluxograma (Figura 1), uma metodologia estruturada para determinar se a blockchain é a solução técnica apropriada para um cenário específico e, caso a sua utilização se justifique, qual o tipo de blockchain mais adequado.

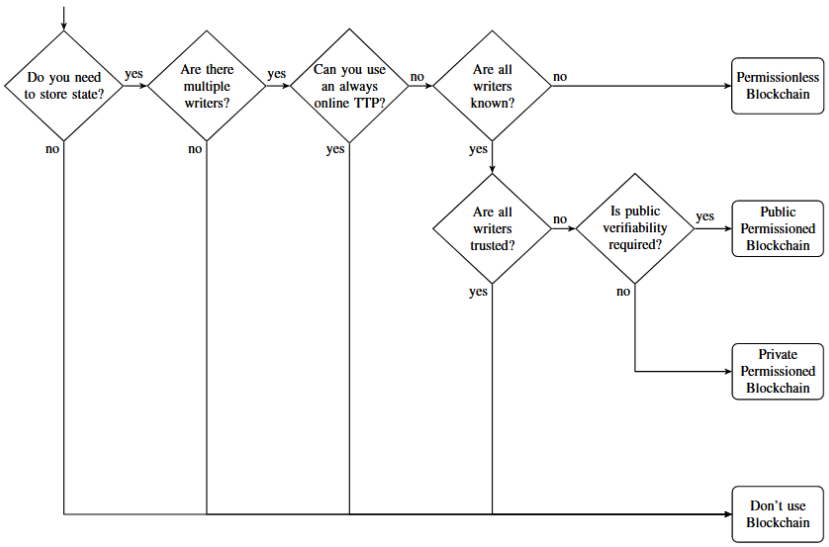

Figura 1: Fluxograma para decidir a adequação do uso de blockchain (adaptado de Wüst & Gervais [1]).

Em [2] é proposta uma framework (Figura 2) de avaliação da adequação da blockchain, que auxilia os profissionais a avaliarem a adequação da utilização da blockchain num sistema com base nas características dos seus casos de uso. Esta é útil como guia inicial para as organizações, ajudando a reduzir o desperdício de esforço em casos de uso inviáveis.

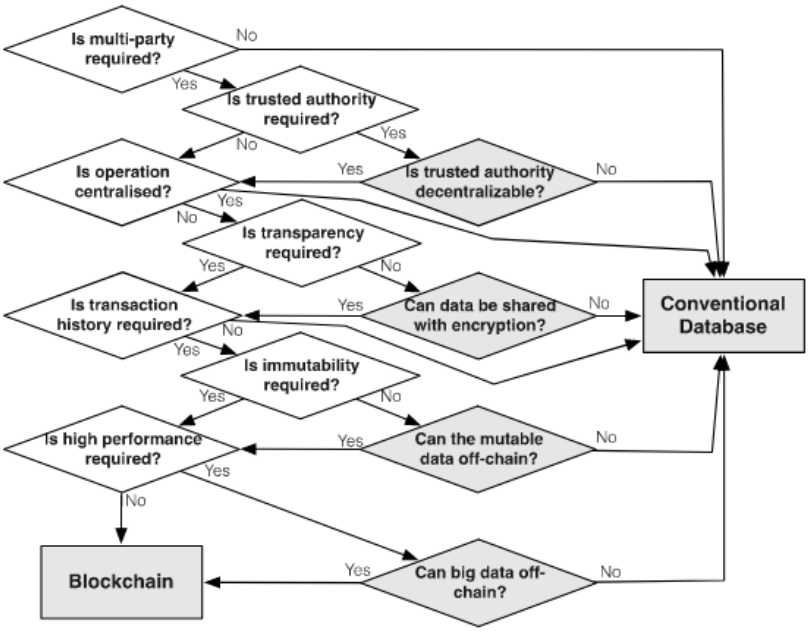

Figura 2: Framework de avaliação de adequação da blockchain para vários casos de uso (adaptado de Lo et al. [2])

2.4.2 Benefícios e dificuldades práticas

Para além de identificar cenários onde a utilização de blockchain possa ser adequada, é importante considerar os potenciais benefícios do seu uso, bem como as dificuldades práticas que surgem ao utilizar a tecnologia. Efetivamente, [21] destaca a segurança como uma característica muito bem vista pelos utilizadores, algo que é alcançado através do uso de criptografia e dos algoritmos de consenso. No entanto, este artigo também menciona a exigente curva de aprendizagem da tecnologia e os potenciais problemas de desempenho, algo que também é referido em [34], visto que as plataformas blockchain geralmente não conseguem atender aos requisitos de desempenho de muitas aplicações, o que constitui um desafio claro para o desenvolvimento de dApps baseadas em blockchain. Embora este estudo conte com uma amostra relativamente pequena (19 participantes), os resultados são suficientes para evidenciar que tanto os utilizadores quanto os *developers* enfrentam dificuldades ao utilizar blockchain, especialmente devido à falta de suporte adequado nas ferramentas disponíveis.

Além dos problemas de desempenho, [11] destaca problemas como a eficiência computacional, risco de erro em regras de negócio complexas e disponibilidade dos dados. Os autores desta revisão sistemática enfatizam também que a adoção da blockchain ainda é limitada devido à falta de evidências que comprovem a sua utilidade, mas também referem a melhoria da transparência, redução de fraudes e a rastreabilidade como principais benefícios adjacentes ao uso de blockchain no setor governamental e financeiro. Complementarmente, [14] avisa que a utilização de blockchain também pode levar a problemas de usabilidade e de experiência de utilizador (UX), caso a plataforma possua criptomoedas, pois a necessidade de ter uma carteira com criptomoedas e pagar taxas de transações para interagir com uma aplicação pode ser visto como um grande obstáculo [27, 34, 45].

Outros trabalhos enfatizam benefícios além dos anteriormente mencionados, como por exemplo em [8,39], onde é destacada a imutabilidade e resistência à adulteração, pois os dados uma vez armazenados numa transação não podem ser modificados, alterados ou excluídos. [25] classifica a imutabilidade como o benefício mais proeminente da tecnologia. Nalguns exemplos de aplicação, a descentralização também é uma grande vantagem, pois elimina a necessidade de intermediários na gestão das transações, conforme referido em [12], onde esta vantagem é destacada no contexto de transações bancárias.

2.4.3 Desafios no desenvolvimento de dApps

Para além das dificuldades inerentes à tecnologia, a literatura também aborda desafios específicos na engenharia de software para dApps. Com efeito, [27] aponta que as atividades de *Blockchain-Oriented*

Software Engineering (BOSE) ainda são um esforço desafiador, em parte porque a aceitação e adoção da tecnologia se encontram numa fase relativamente precoce, o que tem resultado numa ausência de abordagens sistemáticas e holísticas para o desenvolvimento de aplicações baseadas em blockchain. Os autores sublinham a importância de uma arquitetura bem desenhada, um tema que assume especial relevância neste trabalho, e observam que, até à data da sua revisão, faltavam orientações arquiteturais práticas e organizadas em termos de decisões de design, o que constitui uma lacuna evidente na literatura. No mesmo estudo são listadas as principais limitações das dApps, como baixo *throughput* de transações, elevados tempos de resposta, custos de transação e a imaturidade das ferramentas e práticas. Finalmente, os autores distinguem dois modelos de dApp: a *full* dApp, em que toda a lógica da aplicação corre *on-chain* e o único componente *off-chain* é o frontend, e a *hybrid* dApp, em que parte da lógica é executada *off-chain*. Essa diferenciação, que também é discutida em [16], será considerada ao longo deste trabalho.

Em [3] é proposta uma framework (Figura 3) com o propósito de identificar um conjunto de elementos a considerar na fase inicial de projeto de uma dApp, destacando a falta de diretrizes conceptuais claras neste contexto, uma lacuna que contribui para o insucesso de muitos projetos blockchain. A framework é composta por quatro elementos que devem ser documentados no início do projeto: (i) partes/*stakeholders* (utilizadores, nós, tipo de blockchain, oráculos e dispositivos IoT); (ii) requisitos (funcionais, não funcionais e restrições); (iii) governação (modelo de *tokens*, regras de participação e mecanismos de consenso); e (iv) projeto (detalhes arquiteturais, estratégia e aspetos técnicos).

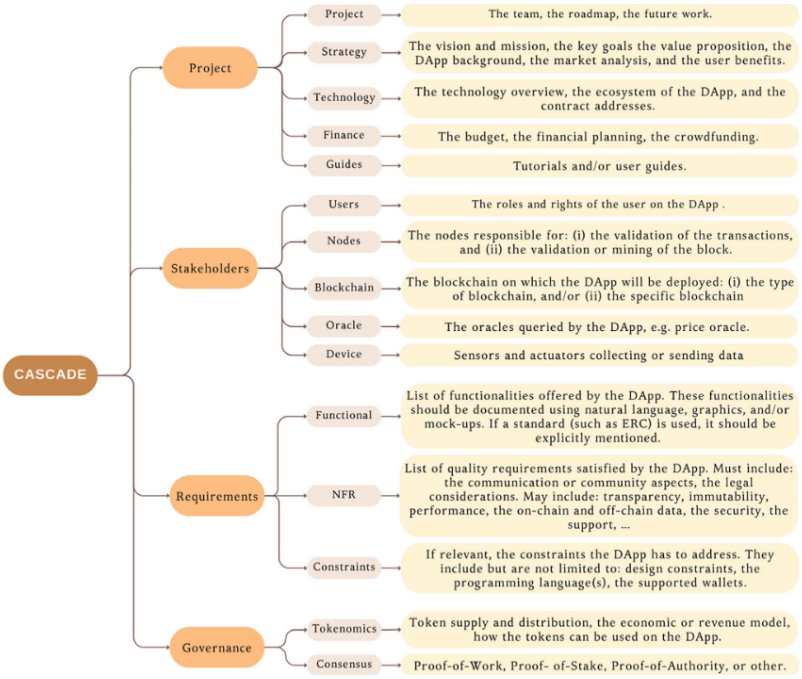

Figura 3: CASCADE — elementos a documentar no início de um projeto dApp (adaptado de Bouraga [3]).

Apesar de útil para clarificar aspetos iniciais, a framework é essencialmente conceptual e representa apenas um passo no processo de desenvolvimento de dApps. Continua a ser necessário integrar este trabalho com orientações mais concretas, como por exemplo, *checklists*, padrões arquiteturais e táticas de otimização, para suportar a implementação e manutenção real dos sistemas.

2.4.4 *Smart contracts*

Como referido em 2.3.3, as características dos *smart contracts* trazem benefícios como automatização, auditabilidade e execução "*trustless*" [6, 16, 43], mas também riscos e restrições. A sua imutabilidade torna o escrutínio prévio e os testes críticos, porque erros no código são difíceis de corrigir após implantação [40]. O modelo de custos (p.ex., o *gas* do Ethereum) penaliza operações dispendiosas e influencia o design das operações *on-chain* [23], e o determinismo obrigatório limita a utilização de funções não determinísticas ou dependentes de serviços externos [35, 40]. Em plataformas diferentes (p.ex., Ethereum vs. Hyperledger Fabric) surgem riscos distintos como vulnerabilidades de segurança típicas em contratos Solidity (*reentrancy*, *under/overflow*, dependência da ordem de transações) ou o risco de não-determinismo em *chaincodes* escritos em linguagens de propósito geral [35, 40], que exigem práticas e padrões específicos de mitigação.

Para contornar essas limitações, a literatura recomenda várias estratégias que afetam a arquitetura global da dApp: modularização e reutilização de código, separação clara entre contratos de dados e lógica de negócio, padrões de contratos atualizáveis (p.ex. *proxy*/*diamond*), e a delegação de computação pesada ou não determinística para componentes *off-chain*, com resultados verificados *on-chain*, por exemplo, via *hashes* [12, 16, 42]. Ferramentas de desenvolvimento, modelagem por máquinas de estado finito (FSM) e frameworks declarativos podem ajudar a reduzir erros e a gerir a complexidade, mas não eliminam o *trade-off* fundamental entre descentralização, custo e desempenho [13, 16, 41].

Embora o presente trabalho não se foque no desenvolvimento de *smart contracts*, reconhece-se que as suas restrições e riscos moldam decisões arquiteturais relevantes, nomeadamente a opção por arquiteturas híbridas e por padrões *client-side* que abstraiam a complexidade do backend descentralizado.

2.4.5 Aplicação na agricultura

No domínio agrícola, a literatura converge na ideia de que a combinação entre IoT e blockchain tem um potencial prático importante para colmatar falhas dos sistemas tradicionais de gestão logística e segurança alimentar. Estudos como [19, 46] destacam que os sistemas convencionais frequentemente não oferecem a transparência, rastreabilidade e auditabilidade exigidas por consumidores e reguladores. Nesse contexto, a blockchain é proposta como um mecanismo que garante imutabilidade e prova de integridade dos registos, tornando-os resistentes a uma possível adulteração. Contudo, essa promessa depende criticamente da fiabilidade na captura de dados, razão pela qual os trabalhos [46–48] sublinham a necessidade de integrar dispositivos IoT (RFID, sensores de temperatura/humidade, telemetria de drones) para recolha automática e contínua da telemetria ao longo do ciclo «*farm-to-fork*».

A literatura técnica apresenta várias propostas concretas que exemplificam padrões recorrentes e *trade-offs* práticos. O AgriBlockIoT (Caro et al. [46]) propõe uma solução totalmente descentralizada com uma arquitetura em camadas (API, controlador e camada blockchain) que permite integração com sistemas empresariais existentes e avaliou implementações em Ethereum e Hyperledger Sawtooth (uma das frameworks *open-source* do projeto Hyperledger), ilustrando desde cedo o compromisso entre imutabilidade *on-chain* e eficiência operacional [46]. De forma semelhante, Ferrández-Pastor et al. e Wang et al. [47, 49] descrevem modelos em camadas que combinam uma camada física (sensores e atuadores), uma camada de processamento e uma camada de serviços onde a blockchain fornece registo de eventos críticos e lógica contratual (*chaincodes*), soluções pensadas para minimizar o tráfego *on-chain* e reduzir latência nas decisões de controlo. Por outro lado, Soliman et al. [20] exemplificam uma solução industrial mais avançada ao orquestrar um cluster

Kubernetes com Hyperledger Fabric e introduzir um modelo *Chaincode-as-a-Service* (CaaS), para permitir atualizações dinâmicas de contratos sem interrupção da rede, uma resposta direta a problemas de manutenção e escalabilidade observados em dApps agrícolas [20].

Quanto aos benefícios reportados, os trabalhos concordam na utilidade da blockchain para garantir rastreabilidade ponta-a-ponta, autenticação de certificados, automação de verificações via *smart contracts* (*chaincodes*) e maior transparência entre atores da cadeia de abastecimento, potenciando confiança do consumidor e redução de fraude [19, 46, 49]. Em contrapartida, surgem desafios técnicos e operacionais recorrentes, designadamente a necessidade de estratégias *on/off-chain* para lidar com um grande volume de telemetria, a impossibilidade, ou dificuldade, de corrigir dados imutáveis quando erros de captura ocorrem, a vulnerabilidade do perímetro (sensores e *gateways*) que pode introduzir dados falsos no *ledger*, e o custo inicial de implantação e coordenação entre múltiplos *stakeholders* [11, 19, 48]. Estas limitações explicam a recorrência de arquiteturas híbridas, soluções hierárquicas para preservar privacidade e o uso de processamento no *edge* para reduzir o volume *on-chain* [19]. Estas observações justificam, para o caso do BioComp, a ênfase em arquiteturas híbridas e em padrões *client-side* que abstraiam a complexidade do backend descentralizado.

2.4.6 Plataformas e comparações

A literatura também aborda comparações entre plataformas blockchain com diferentes propósitos e modelos de permissão. O Ethereum destaca-se como a plataforma mais adotada para dApps e *smart contracts* [4, 6, 31]. Apesar da sua maturidade, enfrenta limitações de escalabilidade e custos elevados de transação [36], bem como vulnerabilidades de segurança nos seus *smart contracts*, que até já levaram a grandes perdas financeiras [35, 40]. Em contraste, plataformas empresariais *permissioned*, como o Hyperledger Fabric (HLF), surgem com o objetivo de fornecer maior controlo de identidade, privacidade e desempenho [35, 36, 38].

Estudos comparativos [46, 49] reforçam esta distinção: o Hyperledger Fabric e o Hyperledger Sawtooth tendem a apresentar menor latência e maior *throughput*, sendo particularmente adequados a sistemas corporativos e IoT, enquanto o Ethereum oferece um ecossistema mais maduro e suporte consolidado a *smart contracts*. A predominância inicial da investigação em torno do Bitcoin [10] evidenciou a necessidade de explorar plataformas alternativas, abrindo caminho para novas gerações de blockchains adaptadas a contextos não financeiros.

3 Análise do Problema

3.1 Objetivo desta secção

Esta secção analisa as lacunas identificadas na revisão de literatura que condicionam o desenho e a implementação de dApps. O foco é identificar os problemas concretos de engenharia, especialmente a nível arquitetural, e da comunicação entre o cliente e o *ledger*, problemas estes que a literatura raramente descreve em termos de soluções aplicadas e resultados empíricos, e traduzi-los em questões experimentais e métricas reprodutíveis. Vale a pena realçar novamente que o presente trabalho se concentra na perspetiva do cliente, pelo que os problemas relacionados com o desenvolvimento de *chaincodes* ou o *deployment* do backend não serão abordados.

3.2 Lacunas na literatura

A literatura existente destaca a necessidade de alargar a pesquisa para plataformas além do Bitcoin, bem como a necessidade de explorar ainda mais o desenvolvimento de dApps baseadas em blockchain. Embora já se note um crescimento no número de estudos sobre o desenvolvimento destas aplicações, a grande maioria destes trabalhos é ainda meramente descritiva e muito geral: apresentam a arquitetura idealizada, explicando brevemente o raciocínio por detrás do seu desenho, enquandrando as decisões tomadas com as suas regras de negócio e casos de utilização da aplicação, e apresentam resultados de desempenho isolados, raramente documentando de forma detalhada os desafios que surgiram durante a implementação, que muito frequentemente são comuns ao desenvolvimento destas aplicações (p.ex. pontos de estrangulamento de desempenho, falhas de sincronização, conflitos na escrita de dados), que padrões e tácticas arquiteturais, ou outras estratégias, aplicaram para mitigar esses e outros problemas (como a otimização de acessos à blockchain, armazenamento de dados *off-chain*, gestão de conflitos), destacando também possíveis alternativas de desenho válidas, ou as consequências empíricas das estratégias adotadas (possíveis *trade-offs* ou custos computacionais). Esta ausência de relatos práticos dificulta a transferência de conhecimento para projetos que precisem de orientações acionáveis. É precisamente esta lacuna que o presente trabalho pretende reduzir.

3.3 Problemas de engenharia

A partir da revisão de literatura e da experiência no caso BioComp, sintetizam-se os problemas de engenharia mais relevantes e recorrentes que se pretende abordar nesta dissertação. Procura-se não só identificá-los, mas também torná-los operacionais, ou seja, transformá-los em questões passíveis de experimento e medição.

1. **Registo, autenticação e gestão de utilizadores**: em dApps baseadas em blockchain, particularmente em contextos de consórcio, a gestão de utilizadores coloca desafios que vão além dos mecanismos tradicionais de autenticação utilizados em aplicações web centralizadas. A identidade de um utilizador não se resume a um login tradicional, mas envolve também o seu reconhecimento pela rede blockchain e à forma como as suas ações são registadas e validadas no *ledger*. Por isso, torna-se necessário analisar e definir o processo de adesão de novos utilizadores ao sistema, incluindo a autenticação através de *identity providers* comuns e a gestão do respetivo ciclo de vida dos membros do consórcio.

2. **Gestão de dados e desempenho**: no contexto de dApps, o acesso a dados persistidos na blockchain apresenta custos significativamente superiores aos observados em sistemas centralizados, quer em termos de latência, quer de consumo de recursos. No entanto, a literatura tende a tratar este problema de forma abstrata, não discutindo de forma sistemática como estruturar a gestão de dados no cliente para mitigar estes custos. Em particular, faltam orientações práticas sobre que mecanismos devem ser aplicados de forma a reduzir acessos à blockchain, de forma a melhorar a latência percebida pelo utilizador e o desempenho global da aplicação, sem comprometer a consistência percebida nem a flexibilidade do modelo de dados.

3. **Sincronização e resolução de conflitos**: a natureza distribuída da blockchain e a latência introduzida pelos *commits on-chain* implicam que as operações de escrita e leitura ocorram de forma inerentemente assíncrona. Como consequência, o cliente pode operar temporariamente sobre estados não confirmados ou desatualizados, especialmente em cenários com vários utilizadores. A literatura reconhece este problema, mas raramente detalha como as aplicações devem lidar com ele ao nível do cliente. É, assim, necessário definir políticas explícitas de sincronização e reconciliação que determinem como e quando o estado local é atualizado, como são tratadas confirmações tardias ou falhadas, e como resolver conflitos concorrentes na escrita de dados.

Estas lacunas e problemas podem ter consequências práticas significativas se não forem definidos mecanismos/estratégias eficientes para amenizar o seu impacto: aumentam a latência e a complexidade do sistema, prejudicam a escalabilidade e degradam a experiência de utilizador, além de multiplicarem o esforço de desenvolvimento e manutenção. Dada a escassez de relatos empíricos sobre táticas e resultados, propõe-se um método experimental baseado no caso BioComp para comparar alternativas e gerar diretrizes acionáveis.

3.4 Critérios de aceitação

Com o objetivo de transformar as lacunas e problemas identificados em questões testáveis, definiram-se critérios de aceitação pragmáticos e mensuráveis que serão usados para avaliar as soluções propostas no contexto do BioComp. O propósito é tornar explícitas as expectativas de qualidade que orientam o trabalho experimental, fornecer métricas e procedimentos de medição reprodutíveis, e permitir uma comparação objetiva entre alternativas arquiteturais e táticas de integração entre o cliente e o *ledger*. Os critérios foram pensados para cobrir tanto aspetos documentais e de justificação de decisões, que são essenciais em engenharia de software, como métricas empíricas relacionadas com desempenho, latência e coerência dos dados.

1. **Registo, autenticação e gestão de utilizadores**

   Para esta vertente, pretende-se proporcionar um processo de *onboarding* e gestão de utilizadores adequado a um ambiente de consórcio, combinando baixo atrito na autenticação com garantias de segurança e controlo de acesso.

   - **Critérios de aceitação**

     - Processo de adesão implementado, incluindo a submissão de pedidos por parte dos utilizadores e a respetiva aprovação ou rejeição pelos administradores do sistema;
     - Integração e funcionamento comprovado com pelo menos dois *providers* SSO (*Single Sign-On*), nomeadamente Google e Microsoft;
     - Gestão básica de utilizadores e permissões (*roles*) implementada e testada;

   - **Métricas de avaliação**

     - Tempo médio de autenticação ≤ 10 s;
     - Taxa de sucesso da autenticação para utilizadores válidos, em testes automatizados controlados, $\geq 99\%$;

2. **Gestão de dados e desempenho**

   O objetivo é explorar políticas de leitura e escrita, de modo a otimizar os acessos à blockchain, reduzir a latência percebida e melhorar o *throughput* do cliente, mantendo a consistência dos dados e uma UX aceitável. Pretende-se avaliar o impacto do uso de *caching* e de diferentes estratégias de leitura e atualização de dados.

   - **Critérios de aceitação**

- Implementação de mecanismos de cache local integrado no cliente.
- Operações de leitura devem poder ser servidas a partir da cache sempre que os seus dados estejam atualizados.
- Implementação de estratégias de *fetching* que otimizem o volume de dados transferidos.

– **Métricas de avaliação**

- Latência média de leitura inferior a 3 s quando os dados são servidos a partir de mecanismos de cache local;
- Capacidade de suportar cargas moderadas de leitura concorrente (até aproximadamente 50 utilizadores simultâneos) sem degradação crítica do desempenho;
- As taxas de sucesso das operações devem ser pelo menos 98% sob carga normal e 95% sob carga elevada, considerando operações locais com confirmações *on-chain*.

3. **Sincronização e resolução de conflitos**

Para resolver este problema pretende-se encontrar uma estratégia que garanta a coerência dos dados e minimize a perda de trabalho em cenários concorrentes, consoante os mecanismos de acesso e atualização da blockchain implementados.

– **Critérios de aceitação**

- Implementação de uma política de reconciliação que aproveite ao máximo as alterações dos utilizadores, procurando evitar perdas de dados;
- A interface reage de forma perceptivelmente instantânea às ações do utilizador, fornecendo *feedback* inicial (otimista ou indicação de processamento) com mediana inferior a 1 s;
- Suporte a confirmação assíncrona de operações *on-chain*, sem bloqueio da interação do utilizador;

– **Métricas de avaliação**

- Percentagem de conflitos resolvidos automaticamente em cenário de concorrência controlada $\geq$ 90%;
- Taxa de perda de operações do utilizador (operações descartadas) $\leq$ 5%;

3.5 Abordagem

O presente trabalho segue uma abordagem de caso de estudo com ciclos iterativos *design–implement–evaluate*, tendo o projeto BioComp como contexto de aplicação. Este contexto, por envolver o desenvolvimento de uma dApp, constitui um cenário prático adequado para explorar os desafios conceptuais, técnicos e arquiteturais associados à utilização de blockchain em aplicações web. Através deste caso de estudo, pretende-se testar e comparar diferentes abordagens para mitigar esses desafios, contribuindo simultaneamente para a evolução do BioComp e para a extração de diretrizes relevantes para o desenvolvimento de dApps noutros domínios.

O processo adotado baseia-se em ciclos iterativos de análise, implementação, testagem e recolha de observações, incidindo sobre subconjuntos específicos de requisitos do sistema. Esta estratégia permite que determinados requisitos sejam implementados e avaliados recorrendo a abordagens alternativas, possibilitando a sua comparação empírica e a identificação de padrões, limitações e boas práticas. No contexto do BioComp, este processo é particularmente adequado, uma vez que o projeto engloba vários subproblemas (monitorização, recolha, produção e aplicação) correspondentes às principais fases do processo de produção de composto. Embora estas fases partilhem conceitos e estruturas comuns, mantêm um grau significativo de independência, o que permite analisá-las como casos interligados, mas distintos, dentro do mesmo sistema.

Dado o foco do trabalho na perspetiva do cliente e na arquitetura de integração entre o frontend e o *ledger*, a abordagem centra-se essencialmente em decisões de desenho arquitetural, táticas de otimização e na avaliação empírica do seu impacto nos atributos de qualidade. Esta delimitação permite concentrar o esforço analítico na experiência do utilizador, na gestão e coerência dos dados e no desempenho observável a partir do cliente, aspectos centrais para a adoção prática de sistemas baseados em blockchain.

3.5.1 Tecnologias utilizadas

Para este projeto será usada a plataforma blockchain Hyperledger Fabric. Esta escolha deveu-se ao facto desta plataforma não envolver custos adicionais, por não possuir uma criptomoeda nativa, o que elimina barreiras financeiras para o seu uso. Além disso, o Hyperledger Fabric é particularmente adequado para o contexto do BioComp, que opera sob uma perspetiva de consórcio composto por várias organizações, cada uma responsável por uma determinada fase ou etapa do processo. Esta plataforma fornece uma arquitetura altamente modular e configurável, permitindo que componentes como os mecanismos de consenso, serviços de identidade e políticas de governação sejam adaptados às necessidades específicas do consórcio. Adicional-

mente, a plataforma permite que o consórcio opere a sua própria rede privada, exclusiva aos seus membros, algo que é inviável em plataformas públicas, onde o acesso ao *ledger* é aberto para todos.

Relativamente à implementação do frontend da aplicação usou-se o Angular, por oferecer um ecossistema maduro em TypeScript, forte integração com RxJS (programação reativa) e suporte a práticas modernas de desenvolvimento. Usou-se também RxDB como base de dados local no *browser*, por encaixar naturalmente com o modelo reativo do Angular e por fornecer capacidades de armazenamento persistente e *caching*, utilizadas neste trabalho sobretudo para dados estruturais e de configuração, bem como para suportar prototipagem. Usou-se também Angular Material para reutilização de componentes da interface de utilizador (UI) comuns.

Adicionalmente, e para simplificar a integração entre o cliente e o *ledger*, optou-se por mediar a comunicação através de uma REST API implementada em Node.js com Express. Esta camada de serviço atua como um *façade* sobre a blockchain, encapsulando detalhes de serialização, autenticação, *retries* e formatação das chamadas ao *ledger*, e expondo ao frontend operações simples. Por se tratar de um componente de infraestrutura com interesse secundário para o foco desta dissertação, a sua descrição técnica é mantida a um nível sumário.

4 Desenvolvimento

Neste capítulo descreve-se o processo de desenvolvimento seguido para implementar as soluções experimentais do presente trabalho, tendo como referência o caso de estudo BioComp. O objetivo não é fornecer um relatório exaustivo de todas as atividades de engenharia de software realizadas no contexto deste projeto, mas sim deixar explícitas as decisões de projeto relevantes para os problemas identificados na secção de análise (Secção 3), e justificar tecnicamente as abordagens escolhidas para os experimentos. Sempre que pertinente, explicam-se também os *trade-offs* das decisões tomadas e as alternativas consideradas.

4.1 Contexto e âmbito do desenvolvimento

4.1.1 Escopo e delimitações

O projeto BioComp já dispunha de requisitos iniciais quando a equipa começou a trabalhar. Por esse motivo, e dado o facto de que a presente dissertação não é o desenvolvimento do projeto, mas sim a descrição do seu desenvolvimento, quando relevante para os problemas que esta dissertação procura abordar, não se descrevem aqui os processos extensivos de elicitação e análise de requisitos, modelação de lógica de negócio ou atividades envolvendo os *stakeholders*. Em termos práticos, o que se segue é uma descrição do que foi implementado, porquê, e como foi avaliado, não fornecendo uma descrição completa de todas as fases de desenvolvimento da aplicação que não são relevantes para esta dissertação.

4.1.2 Procedimento inicial

O trabalho começou com um levantamento bibliográfico sobre blockchain e por testes experimentais, utilizando a rede de testes do Hyperledger Fabric [50], com o objetivo de consolidar conhecimentos práticos sobre o ciclo de desenvolvimento e *deploy* dos *chaincode*. Paralelamente, foram analisados os requisitos prioritários para os sub-casos selecionados do BioComp (monitorização, recolha e transporte, produção e aplicação) e construíram-se protótipos de baixa/média fidelidade para validar hipóteses de UX e fluxos de utilizador (Figura 4).

Figura 4: Protótipos de baixa/média fidelidade de algumas interfaces da aplicação

4.1.3 Prova de conceito

Antes de consolidar a arquitetura do cliente, realizou-se um teste de prova de conceito com C# e Entity Framework (Visual Studio 2022). O objetivo não foi ponderar a utilização de .NET na aplicação, mas avaliar se padrões de abstração de persistência, presentes em frameworks como o Entity Framework, poderiam ser úteis no contexto de uma dApp. Especificamente, pretendia-se consolidar e validar a ideia de implementar uma camada que expõe operações CRUD simples (por exemplo `getOne`, `getAll`, `create`, `update`, `delete`) e que encapsule a complexidade da comunicação com o armazenamento subjacente ou com a blockchain. Constatou-se que a presença de uma camada de abstração que expõe métodos CRUD simples ao resto da aplicação e que encapsule a complexidade das tecnologias subjacentes traz vantagens práticas: reduz o acoplamento, torna mais simples trocar ou combinar mecanismos de armazenamento e pode potencialmente facilitar a adição de sincronização com a blockchain. Para além disso, esta camada de abstração revelou-se um ponto natural de extensão para explorar estratégias distintas de obtenção de dados, inspiradas em mecanismos como *Eager Loading*, *Lazy Loading* e *Explicit Loading*, motivando a sua posterior adaptação e avaliação no contexto da aplicação descentralizada.

4.2 Modelação e implementação inicial do domínio

Após a fase de análise dos requisitos principais e de prototipagem, iniciou-se a modelação e implementação inicial do domínio, procurando traduzir as entidades (p.ex. Pilha, Local, Medição, Agendamento) e regras centrais do BioComp para estruturas de dados e componentes reutilizáveis no cliente.

Como a rede blockchain ainda estava a ser configurada/implementada, utilizou-se temporariamente uma solução de armazenamento local para prototipagem e para suportar as primeiras interações, utilizando o RxDB, por ser leve, reativo e dado o seu potencial para futuramente servir como cache. O desenvolvimento começou com classes TypeScript e interfaces estáticas que espelhavam o modelo de negócio definido nos requisitos. No entanto, rapidamente se verificaram problemas práticos com esta abordagem: a variabilidade das propriedades das entidades (novas propriedades e campos que surgiam com frequência ou mudanças de tipos dos dados armazenados) e as constantes alterações/refinamentos de requisitos obrigavam a mudanças dispersas no frontend (formulários, validações, mapeamentos para persistência), tornando a evolução e manutenção caras e propensas a erros. Além disso, o modelo estritamente tipado do TypeScript apresenta limitações em contexto de execução, uma vez que a informação de tipo é eliminada em *runtime*. Isto impede o sistema de distinguir dinamicamente entre diferentes entidades de domínio, dificultando a criação de mecanismos genéricos de renderização e manipulação de dados no cliente.

Estas limitações motivaram a procura de uma abordagem mais flexível que reduzisse o acoplamento entre modelo de domínio e o código do cliente, facilitasse alterações em tempo de execução e simplificasse o mapeamento para o backend/blockchain. A solução adotada, e que é descrita na secção seguinte, foi a transição para um modelo baseado em metamodelação, que permite representar classes, propriedades e instâncias de forma dinâmica e genérica, reduzindo o número de adaptações necessárias sempre que o domínio evolui.

4.3 Metamodelação

A metamodelação consiste em descrever explicitamente a estrutura de um modelo, ou seja, em definir um modelo de um modelo. Enquanto que um modelo representa entidades concretas de um domínio, o metamodelo define os seus conceitos e relações, ou seja, as classes, propriedades e regras que estruturam essas entidades. Esta abordagem é amplamente utilizada em engenharia de software e modelação, pois permite representar sistemas de forma genérica e extensível, em vez de codificar numerosas classes estáticas.

4.3.1 Estrutura

A abordagem de metamodelação definida para o BioComp, ilustrada na Figura 5, organiza-se em quatro conceitos principais:

- **Classifier**: descreve um tipo de entidade do domínio, funcionando de forma análoga a uma classe em programação orientada a objetos (p.ex. Pilha, Local, Agendamento).
- **Property**: descreve um atributo de um Classifier e consiste basicamente nos atributos de uma classe (nome, tipo, regras de validação, visibilidade).
- **Slot**: representa a associação entre uma Property e o seu valor efetivo numa determinada Instance. Por exemplo, para uma propriedade `name`, o seu valor real ("Pilha 1") é armazenado numa Slot.
- **Instance**: corresponde à materialização de um Classifier, isto é, um objeto concreto criado a partir da sua definição estrutural. Cada Instance mantém uma referência ao seu Classifier, para saber que tipo de entidade representa e quais propriedades possui.

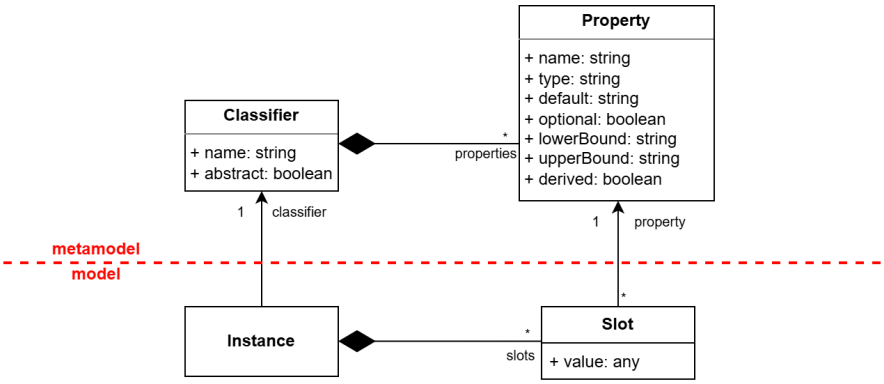

Figura 5: Metamodelo base utilizado

Os conceitos Classifier e Property constituem a camada estrutural do sistema, ou seja, o metamodelo propriamente dito. Estes elementos definem o *backbone* conceptual do domínio e são relativamente estáveis, sendo alterados apenas quando ocorrem mudanças na definição das entidades. Em contraste, os conceitos Instance e Slot pertencem à camada operacional ou de execução - o modelo em tempo de execução - onde os dados concretos são manipulados.

Em termos práticos, pode-se dizer que o metamodelo define a gramática do domínio (as regras e componentes possíveis), enquanto que o modelo corresponde às frases concretas formadas segundo essa gramática.

Assim, cada Instance é uma realização do metamodelo: é um objeto concreto que segue a estrutura descrita pelo seu Classifier e cujos atributos são armazenados nas respetivas Slots.

4.3.2 Evolução do metamodelo

Com a evolução do projeto, o metamodelo inicial foi progressivamente refinado para melhorar a sua capacidade de abstração e representar de forma mais expressiva as diferentes entidades do domínio. Este refinamento não se limitou à introdução de novos conceitos, mas procurou também tornar explícita a separação entre o nível do metamodelo e o nível do modelo em tempo de execução.

A Figura 6 apresenta a versão refinada do metamodelo, incluindo os conceitos adicionais introduzidos e a distinção clara entre a camada estrutural (metamodelo) e a camada operacional (modelo).

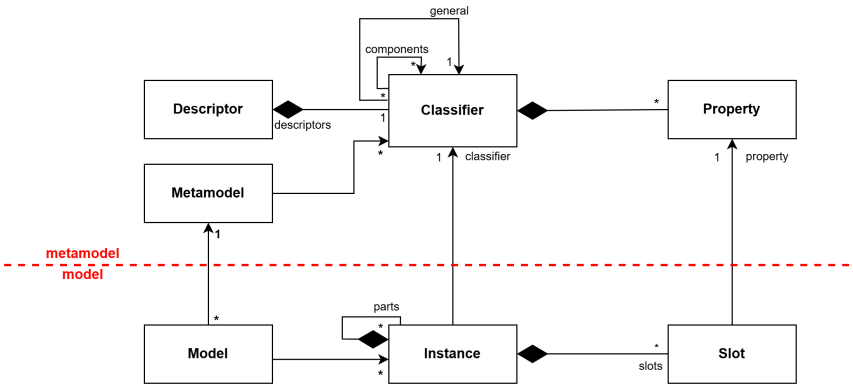

Figura 6: Metamodelo final

Uma das principais extensões foi a introdução explícita do conceito de Metamodel, responsável por agregar os *Classifiers* que definem a estrutura conceptual do domínio. De forma análoga, foi introduzido o conceito de Model, que representa o conjunto de instâncias concretas existentes num determinado momento de execução e que referencia um metamodelo específico. Esta separação torna explícita a dependência do modelo em relação ao metamodelo e permite suportar múltiplos modelos baseados numa mesma definição estrutural.

Outra alteração relevante foi a introdução do conceito de Descriptor, que descreve aspetos complementares de especificação e representação visual. Cada Descriptor agrega informações auxiliares como ícones, rótulos traduzidos e valores pré-definidos, funcionando como uma camada de metadados sobre os classificadores. Este mecanismo permite desacoplar a definição estrutural das entidades (mantida no metamodelo) da sua configuração e representação no cliente, facilitando a extensibilidade e a internacionalização da aplicação.

Foram também acrescentados dois tipos de relações entre Classifiers: a relação ***general***, que representa a generalização ou herança entre classificadores (por exemplo, um `ACTION` genérico pode especializar-se em `ACTION_SIGHTING`, correspondente ao registo de um avistamento), e a relação ***components***, utilizada para definir composições entre entidades. Esta última permite modelar Classifiers que dependem de outros. Por fim, foi acrescentada a relação ***parts*** entre instâncias, que também consiste na ideia de composição de instâncias, por exemplo uma instância de uma pilha de compostagem pode ser composta por várias instâncias de medições (temperatura, humidade, pH, etc).

4.3.3 Implicações desta abordagem

Apesar das vantagens em termos de flexibilidade e desacoplamento, a adoção de uma abordagem baseada em metamodelação introduz também algumas desvantagens e desafios técnicos. Em primeiro lugar, a manipulação e pesquisa de dados torna-se mais complexa, uma vez que as consultas já não operam sobre coleções/tabelas estáticas mas sobre instâncias de Classifiers, exigindo que cada operação especifique o identificador do tipo pretendido. Isto obriga a mecanismos de pesquisa e filtragem mais genéricos, o que pode aumentar o custo computacional e a verbosidade do código.

Em segundo lugar, a resolução de referências entre instâncias (quando o valor de uma Slot corresponde ao identificador de outra Instance) implica a implementação de mecanismos de *lookup* ou *population*, comparáveis à gestão de chaves estrangeiras em bases de dados relacionais, mas agora sob responsabilidade do cliente. Ainda assim, estes custos foram considerados aceitáveis face aos benefícios obtidos em termos de extensibilidade, manutenção e evolução do domínio.

Para além destes aspetos, a adoção do metamodelo trouxe também benefícios indiretos na camada blockchain, ao permitir que a lógica *on-chain* seja organizada de forma mais genérica e menos dependente das entidades concretas do domínio. Em vez de exigir contratos distintos para cada tipo de objeto, a estrutura flexível do metamodelo possibilita uma redução significativa da quantidade de *chaincodes* necessários, simplificando a manutenção e diminuindo o esforço associado a atualizações em redes permissionárias.

4.4 Cenários de qualidade

Para orientar as decisões arquiteturais que precisavam de ser tomadas, tomou-se como referência o *Quality Attribute Workshop* (QAW), uma metodologia amplamente utilizada para a identificação e análise de atributos de qualidade em sistemas.

Os atributos de qualidade selecionados e priorizados para este trabalho foram desempenho, disponibilidade, segurança e escalabilidade, por serem considerados os mais críticos para este tipo de sistemas e para os objetivos do BioComp. Outros atributos de qualidade, como usabilidade e modificabilidade, foram igualmente identificados durante o processo de análise e tiveram cenários de qualidade definidos. No entanto, estes atributos não são aprofundados nesta dissertação, uma vez que não estão diretamente relacionados com os problemas centrais abordados.

A seleção destes atributos é consistente com estudos recentes sobre qualidade em sistemas blockchain. Em particular, Koteska et al. [43] identificam desempenho (latência e *throughput*), segurança, usabilidade, escalabilidade e integridade de dados como alguns dos principais desafios de qualidade nas implementações atuais de blockchain, sublinhando a necessidade de abordagens arquiteturais que equilibrem estes atributos.

Os cenários que se seguem foram refinados e agrupados por atributos de qualidade considerados mais relevantes para o BioComp e para outras aplicações que interajam com um *ledger*/blockchain.

4.4.1 Desempenho

1. **QA1** — Quando um utilizador regista uma nova pilha, o sistema deve confirmar a aceitação da operação em ≤ 3 s e garantir a persistência final dos dados em ≤ 10 s.
2. **QA2** — Um utilizador altera a data de um agendamento em funcionamento normal. O sistema deverá aplicar a atualização, refletindo a alteração na interface em ≤ 3 s.
3. **QA3** — Quando um utilizador efetua login em condições normais, o sistema deverá validar as credenciais *on-chain* e confirmar a existência de uma conta associada às mesmas, concluindo a operação em ≤ 10 s.
4. **QA4** — Quando um utilizador consulta o gráfico de temperatura de uma pilha, o sistema deverá apresentar o gráfico e estatísticas (média, mínimo, máximo) em ≤ 3 s.
5. **QA5** — Quando 50 utilizadores solicitarem simultaneamente a listagem de todas as pilhas, o sistema deverá responder corretamente à totalidade dos pedidos e manter tempos de resposta aceitáveis, preferencialmente $\leq 10 - 15$ s, sem falhas ou degradação crítica do serviço.

Estes cenários estão diretamente associados aos problemas centrais abordados nesta dissertação e serão tratados através de decisões arquiteturais e estratégias específicas de gestão de dados, que serão aprofundados na Secção 4.7.

4.4.2 Disponibilidade

1. **QA6** — Quando um utilizador tenta aceder à aplicação durante a manutenção de um nó da rede blockchain, o sistema deverá manter disponíveis as funcionalidades essenciais (operações de leitura e escrita), sem indisponibilidade perceptível.
2. **QA7** — Se o sistema perder conectividade com a internet, deverá detetar e notificar o utilizador sobre essa indisponibilidade temporária, suspendendo operações que dependam da rede em ≤ 5 s.
3. **QA8** — Se a REST API ou a blockchain ficarem indisponíveis, o sistema deverá detetar e notificar os utilizadores sobre essa indisponibilidade e suspender operações que dependam da rede em ≤ 15 s.
4. **QA9** — Quando a conectividade com a internet é restabelecida após uma falha, o sistema deverá notificar os utilizadores e voltar a operar em modo normal.
5. **QA10** — Quando a REST API recupera de uma falha após ter estado temporariamente indisponível, o sistema deverá restabelecer a comunicação e voltar a operar normalmente em ≤ 1 min.

4.4.3 Segurança

1. **QA11** — Se um atacante externo tentar aceder sem credenciais válidas, o sistema deverá recusar o acesso e registar o incidente; nenhuma operação permitida.
2. **QA12** — Se um atacante tentar intercetar a comunicação cliente–REST API (MITM), a comunicação deverá estar cifrada e protegida para evitar exposição de dados sensíveis.
3. **QA13** — Quando um *token* expirado ou comprometido é reutilizado, o sistema deverá rejeitá-lo e encerrar a sessão.

4.4.4 Escalabilidade

1. **QA14** — Quando 100 utilizadores efetuam consultas de locais em < 1 minuto, o sistema deverá manter tempo médio de resposta < 10 s.
2. **QA15** — Quando 100 alterações a diferentes locais ocorrem no mesmo minuto, o sistema deverá distribuir a carga por APIs/organizações e concluir 99% das alterações em ≤ 10 s.

4.5 Arquitetura

Após o levantamento dos atributos de qualidade e da definição dos respetivos cenários de qualidade baseados nesses atributos, foi elaborado um primeiro desenho arquitetural da aplicação desenvolvida no BioComp, designada de BioGuard. Este desenho refletia o estado então alcançado pela aplicação, tendo já várias funcionalidades implementadas, embora ainda suportada apenas por armazenamento local baseado em RxDB. Este desenho incluía também alguns componentes numa fase preliminar, que, apesar de incompletos, integravam a visão arquitetural prevista para a evolução do sistema.

Este modelo inicial foi concebido já com a intenção de antecipar a futura integração da blockchain, servindo como instrumento de reflexão sobre os desafios arquiteturais envolvidos e como base para planear as adaptações necessárias. Por essa razão, representa um ponto de partida útil para compreender as decisões que orientaram as iterações subsequentes.

Antes de detalhar a organização interna da aplicação, é importante situar o BioGuard no seu ecossistema global. O sistema adota uma arquitetura distribuída de três níveis (3-*tier*), configurando uma dApp híbrida, conforme ilustrado na Figura 7. Esta arquitetura em 3 níveis difere um pouco das aplicações tradicionais visto que a camada de dados baseia-se numa tecnologia distribuída, em vez de uma base de dados tradicional.

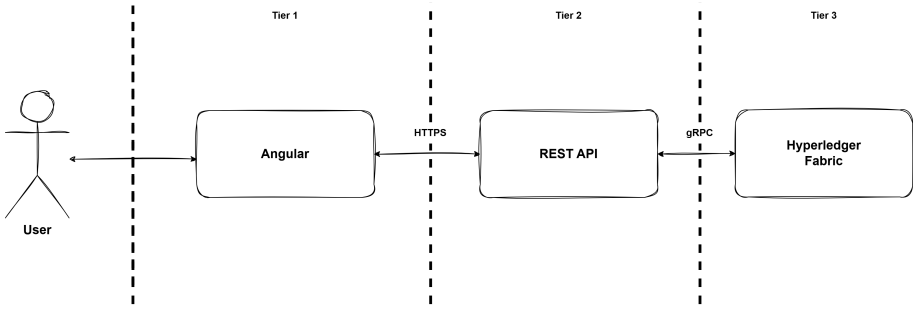

Figura 7: Arquitetura em 3 níveis do BioGuard

À luz desta organização arquitetural, as responsabilidades do sistema encontram-se distribuídas por três camadas físicas e lógicas distintas: o *Tier* 1 (Cliente), onde reside a aplicação Angular, incluindo a lógica de apresentação, domínio e persistência local; o *Tier* 2 (Intermediação), que consiste na REST API que isola o cliente das especificidades do *ledger* e permite evoluir mecanismos de otimização, segurança e gestão de estado sem impacto direto no frontend; e o *Tier* 3 (*Ledger*), correspondente à rede Hyperledger Fabric, responsável pela execução dos *chaincodes* e armazenamento persistente dos dados. A aplicação também

utiliza os serviços da Google e da Microsoft para autenticação, mas dado que são externos ao sistema não são considerados nesta descrição arquitetural.

Uma visão mais detalhada da organização interna do sistema, em particular do cliente, encontra-se ilustrada na Figura 8. Esta organização concretiza a arquitetura em camadas através de uma variação do padrão Model–View–ViewModel (MVVM), combinada com princípios de orientação a serviços (SOA), de modo a promover o desacoplamento entre a lógica de apresentação, processamento e persistência.

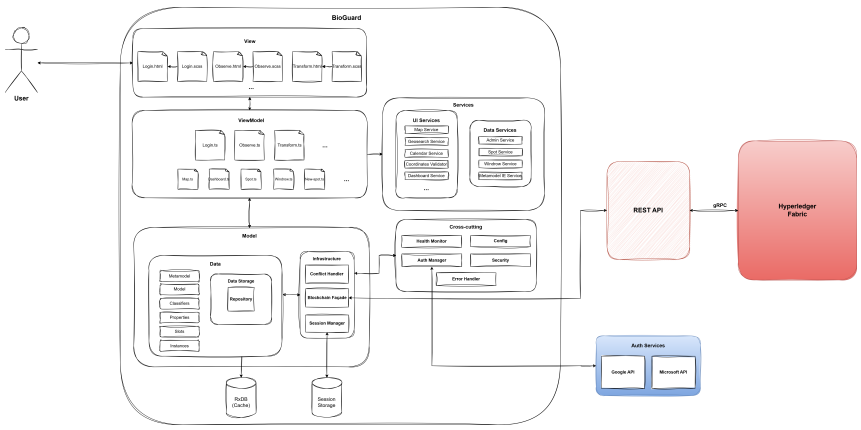

Figura 8: Arquitetura inicial do sistema

4.5.1 Camada de interface e interação (View e ViewModel)

A camada **View** corresponde à interface de utilizador, composta por *templates* HTML, estilos CSS/SCSS e componentes Angular Material, com os quais o utilizador interage diretamente.

O **ViewModel**, implementado através dos componentes Angular em TypeScript, atua como intermediário entre a View e o Model. O ViewModel reage a eventos e alterações do utilizador, processa dados e formulários, e atualiza automaticamente o estado da interface utilizando *signals*, de forma reativa. Ao contrário de um controlador típico no padrão Model-View-Controller (MVC), o ViewModel encapsula também parte da lógica de interação e coordenação entre a camada de apresentação e a de dados.

4.5.2 Camada de serviços

A camada **Services** agrega um conjunto de serviços auxiliares e é inspirada numa arquitetura orientada a serviços, no sentido de modularizar e expor funcionalidades de forma coesa e reutilizável em diferentes componentes da aplicação cliente, promovendo o desacoplamento e manutenibilidade do código. Os serviços são agrupados em dois tipos principais: **UI Services** e **Data Services**.

Os UI Services fornecem lógica auxiliar focada na camada de apresentação, atuando nalguns casos como *façades* para bibliotecas externas. São responsáveis por transformar, preparar ou gerir dados orientados à experiência do utilizador. Alguns exemplos dos serviços nesta camada são o Map Service, que permite a implementação de funcionalidades de visualização e interação geográfica, utilizando a biblioteca Leaflet, o Geosearch Service, responsável por operações de *reverse geocoding* e tratamento de coordenadas, ou o Calendar Service, responsável pela gestão e visualização de eventos e agendamentos, utilizando a biblioteca FullCalendar.

Por outro lado, os Data Services são serviços de dados especializados que oferecem operações sobre instâncias do modelo. Esta camada inclui também o serviço Metamodel I/E, responsável pela importação/exportação do metamodelo em formato JSON, assegurando a persistência e a portabilidade das definições de entidades. Estes serviços podem ser usados em diversas partes da aplicação e são altamente coesos.

4.5.3 Camada de domínio e dados (Model)

A camada **Model** constitui o núcleo lógico e estrutural do sistema, concentrando os mecanismos responsáveis pela representação, manipulação e persistência dos dados da aplicação. Ao contrário de abordagens tradicionais, esta camada não contém classes de domínio estáticas, mas sim a infraestrutura necessária para definir e instanciar dinamicamente entidades de domínio em tempo de execução, através do metamodelo.

A implementação prática do metamodelo foi baseada num conjunto de interfaces orientadas a objetos, desenhadas para materializar os conceitos definidos no metamodelo. A sua estrutura é ilustrada na Figura 9. Por motivos de simplificação, o diagrama representa apenas as interfaces usadas na implementação, sendo omitidas as classes concretas que as implementam.

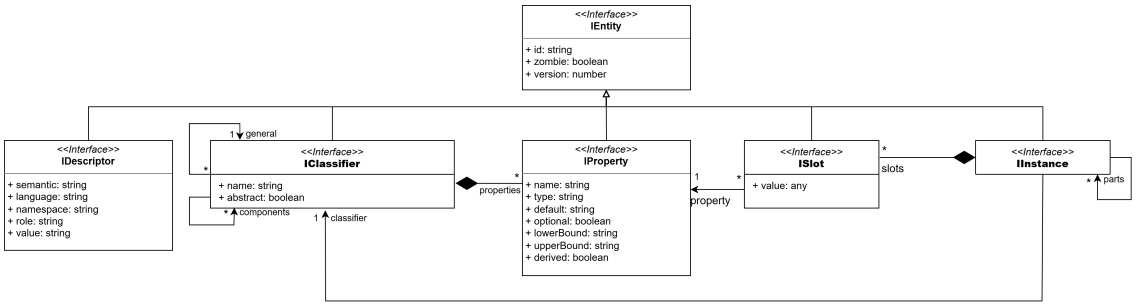

Figura 9: Diagrama de classes da implementação do metamodelo

Para acesso e persistência dos dados, foi adotado o padrão Repository, encapsulando operações CRUD genéricas e abstraindo a origem dos dados subjacentes. Esta ideia surge a partir do teste de prova de conceito apresentado na secçao 4.1.3. Os repositórios interagem com o Data Storage, que combina e gere o armazenamento local (RxDB) e a persistência *on-chain*. No interior desta camada existe ainda uma subcamada Infrastructure, que agrega o Session Manager, responsável pelo armazenamento e gestão de variáveis de sessão, o Blockchain Façade, que será um dos componentes mais importantes para este trabalho, com o objetivo de fornecer uma interface CRUD muito simples que abstraia a complexidade de comunicação com a REST API e, consequentemente, com o *ledger*, e o Conflict Handler, um componente desenhado a pensar nos casos em que as escritas na blockchain pudessem originar conflitos.

4.5.4 Camada transversal (*Cross-cutting*)

Por fim, a camada **Cross-cutting** integra serviços gerais aplicáveis a toda a aplicação. Alguns componentes desta camada foram desenhados com o objetivo explícito de suportar alguns dos cenários de qualidade, nomeadamente os de disponibilidade (QA6-QA10). O componente Health Monitor foi concebido como a concretização da tática *Monitor*, sendo responsável por observar continuamente o estado da conectividade com a internet, com a REST API e, indiretamente, com a blockchain. Este componente aplica também a tática *Ping/Echo*, permitindo detetar falhas de comunicação num intervalo de tempo limitado e notificar os restantes componentes sobre alterações no estado do sistema. Embora isoladamente não resolva todos os cenários de disponibilidade, o Health Monitor constitui a base para suportar comportamentos de adaptação e degradação controlada. Complementarmente, o Error Handler foi introduzido para suportar a tática de *Graceful Degradation*, permitindo que falhas de comunicação ou exceções em tempo de execução sejam tratadas de forma controlada, evitando a propagação de erros para a interface de utilizador e preservando a experiência de utilização. Estes mecanismos permitem que a aplicação reaja de forma previsível a falhas temporárias, suspendendo operações dependentes da rede e informando o utilizador do estado do sistema. De resto, esta camada também possui o Auth Manager para auxiliar na autenticação com os serviços externos, o Config para manter parâmetros de configuração global e o Security para garantir a integridade e autenticidade dos dados enviados e recebidos. Esta camada suporta a observabilidade e a resiliência do sistema, e é essencial para concretizar cenários de qualidade relacionados com disponibilidade e desempenho.

Esta arquitetura base representou um ponto de consolidação do sistema, oferecendo uma estrutura modular e extensível capaz de sustentar as funcionalidades já existentes e preparar a integração da blockchain. Alguns

componentes, como o Blockchain Façade e o Conflict Handler, encontravam-se ainda em estado conceptual, definidos sobretudo para antecipar futuras necessidades de comunicação, sincronização e resolução de conflitos com a blockchain.

4.5.5 Arquitetura refinada

A partir da arquitetura apresentada na secção anterior, o sistema foi progressivamente refinado para viabilizar a integração efetiva da blockchain no contexto do BioComp. Este refinamento teve como principal objetivo estabilizar a estrutura da aplicação e permitir a sua utilização funcional com persistência *on-chain*, ainda sem preocupações de otimização de desempenho, sincronização ou redução de acessos ao *ledger*, que correspondem aos problemas centrais desta dissertação. Nesta fase, a blockchain é utilizada essencialmente como um mecanismo de persistência confiável, assumindo um papel semelhante ao de uma base de dados tradicional, o que permitiu validar a correção funcional do sistema.

A arquitetura refinada, ilustrada na Figura 10, representa assim um ponto de consolidação do sistema, a partir do qual se tornam possíveis as iterações subsequentes focadas na gestão eficiente de dados e na exploração dos compromissos arquiteturais associados à utilização de blockchain.

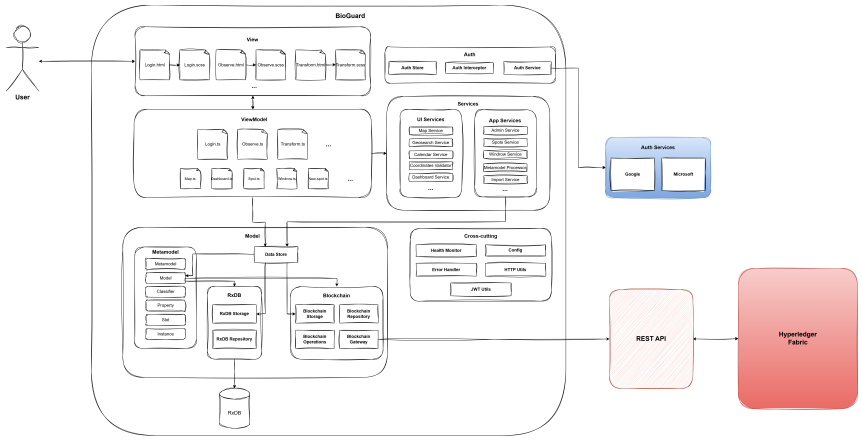

Figura 10: Arquitetura refinada

As alterações mais significativas ocorreram nas camadas de serviços e de domínio (Model), refletindo a necessidade de desacoplar a lógica de negócio e os mecanismos de persistência, bem como de acomodar a interação com a blockchain de forma transparente para a camada de apresentação.

Na camada de serviços, os antigos Data Services foram reformulados e renomeados para **App Services**. Estes serviços deixam de executar diretamente a lógica de persistência associada às operações CRUD sobre

entidades concretas do domínio, passando a assumir um papel de orquestração de alto nível. Em particular, os App Services coordenam fluxos de trabalho que envolvem validações de negócio, interações com múltiplas entidades e delegação das operações de leitura e escrita para os mecanismos apropriados da camada de dados. Esta separação permitiu reduzir o acoplamento entre os ViewModels e os detalhes de persistência, bem como preparar o sistema para futuras estratégias de otimização.

Um refinamento adicional relevante foi a introdução de uma camada dedicada à gestão de autenticação e sessão. Esta camada é responsável por centralizar e isolar as preocupações relacionadas com autenticação, autorização e gestão de sessão, sendo transversal a toda a aplicação. A sua estrutura é composta por um Auth Service, que coordena o processo de autenticação, um Auth Store para manutenção do estado de sessão no cliente, e um Auth Interceptor que assegura a autenticação transparente das requisições ao backend. Estes componentes surgiram como resposta direta ao problema que é descrito na Secção 4.6.2, permitindo encapsular toda a lógica de autenticação fora dos fluxos de negócio da aplicação.

A camada Model sofreu a redefinição mais significativa, evoluindo de um simples recipiente de entidades para o núcleo de controlo da persistência e do estado reativo do sistema. Esta evolução foi essencial para cumprir os objetivos de desacoplamento e de gestão consistente dos dados.

O principal ponto de acesso a esta camada de dados é um Signal Store designado Data Store. Este componente expõe o estado da camada de persistência de forma reativa a toda a aplicação. O metamodelo e o modelo de domínio são mantidos neste componente de forma reativa, permitindo que ViewModels e serviços do App Services acedam ao modelo através de uma interface uniforme, sem necessidade de conhecimento sobre a origem dos dados (RxDB ou blockchain).

Durante a inicialização, o Data Store consulta o Blockchain Storage para obter a versão mais recente do metamodelo e verifica se a versão local em RxDB coincide, através do RxDB Storage. Se a versão local estiver desatualizada, o metamodelo é importado do *ledger* e persistido em RxDB. Este mecanismo tem como objetivo garantir a consistência estrutural do sistema no momento de arranque e evitar carregamentos desnecessários a partir da blockchain durante a fase inicial da aplicação, servindo como *caching*. Este mecanismo é analisado e justificado na Secção 4.7.6.

Para garantir desacoplamento e robustez, o design da camada de persistência baseia-se nos padrões Repository e Null Object, conforme referido anteriormente e detalhado no diagrama de classes da Figura 11:

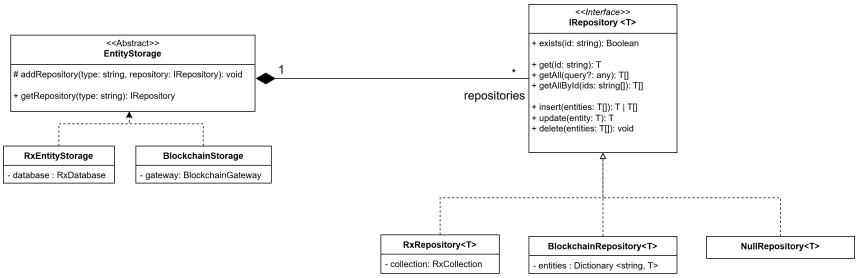

Figura 11: Diagrama de classes da camada de persistência

A interface IRepository define o contrato de persistência adotado pelo sistema. Todas as implementações concretas (RxDB Repository, Blockchain Repository), bem como a implementação de resiliência (NullRepository), respeitam este contrato, assegurando intercambialidade. O NullRepository implementa o padrão Null Object, permitindo tratar de forma segura situações em que um repositório não está disponível, sem recorrer a exceções.

O componente Model, contido no Data Store e que possui esta designação por estar fortemente relacionado com a camada operacional do metamodelo, o modelo propriamente dito, conforme apresentado na Figura 5, atua como gestor do domínio e do estado da aplicação, utilizando os Repositories para realizar operações de leitura e escrita, mantendo as camadas superiores completamente abstraídas da origem dos dados. Este componente intermedeia a comunicação entre o Data Store e os repositórios, sendo que na sua inicialização o Data Store injeta a referência do repositório concreto que o Model deve utilizar, permitindo que a lógica de negócio permaneça agnóstica à infraestrutura de armazenamento, facilitando a testabilidade e a escalabilidade do sistema.

Por fim, o antigo Blockchain Façade foi renomeado para Blockchain Gateway. Este componente é o ponto de entrada da comunicação com a rede Hyperledger Fabric. O Blockchain Gateway trabalha em conjunto com o componente Blockchain Operations, que encapsula as operações da blockchain, definindo o tipo de pedido a ser realizado (`get`, `post`, `put` ou `delete`), *endpoint* a utilizar e o formato da resposta esperado. A definição clara do Blockchain Gateway e Blockchain Operations é fundamental, pois isola a complexidade do ledger da lógica de domínio principal da aplicação.

Removeu-se do desenho o Conflict Handler, pois a ideia era introduzir um mecanismo de resolução de conflitos que não envolvesse o cliente, logo, dado que esse componente era um dos componentes em fase conceptual, optou-se por removê-lo nesta fase, dado que este subproblema será abordado mais à frente.

4.6 Registo, autenticação e gestão de utilizadores

4.6.1 Registo

Este foi um dos primeiros desafios enfrentados no desenvolvimento da aplicação e que se considerou relevante incluir nesta dissertação. Embora estes processos sejam comuns na maioria das aplicações, o contexto particular de um consórcio altera significativamente o problema. Numa blockchain pública, qualquer utilizador pode participar livremente na rede, o que elimina a necessidade de controlo de adesão. No entanto, numa blockchain de consórcio esse pressuposto deixa de ser válido: apenas entidades previamente autorizadas podem integrar a rede, o que introduz implicações técnicas e organizacionais que têm impacto direto no sistema.

Dado que o BioGuard utiliza uma blockchain de consórcio, o acesso ao sistema não pode ser totalmente aberto nem anónimo. No entanto, também não é desejável, nem escalável, exigir que cada utilizador contacte previamente um administrador para ser registado manualmente. Assim, tornou-se necessário definir um mecanismo de registo que permitisse a qualquer membro iniciar o seu próprio pedido de registo, mantendo simultaneamente o controlo administrativo sobre quem pode efetivamente utilizar o sistema. A introdução deste processo levanta questões que não surgem num sistema tradicional, sendo necessário clarificar:

- Quando é que um utilizador deve ser considerado registado na perspetiva do consórcio, momento a partir do qual deve possuir certificados e material criptográfico que lhe permitam interagir com o *ledger*;
- Como determinar a organização a que o utilizador pertence, informação essencial para o modelo de identidade do Hyperledger Fabric;
- De que forma os administradores podem consultar os pedidos de adesão e aprová-los ou rejeitá-los;

Para clarificar o processo de registo definido, elaborou-se um diagrama de atividades UML, apresentado na Figura 12. Este diagrama utiliza *swimlanes* para demarcar os fluxos de trabalho do utilizador e do administrador.

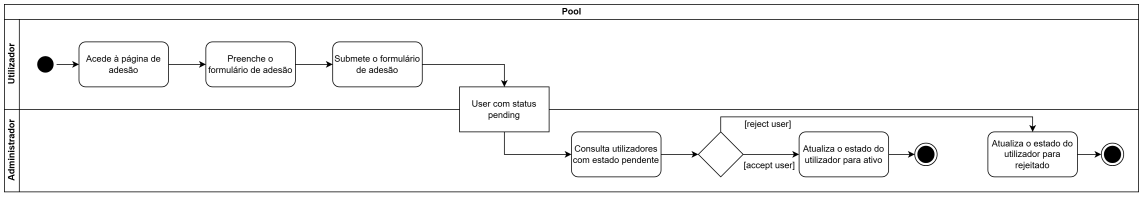

Figura 12: Diagrama de atividades do processo de adesão

O processo inicia-se quando o utilizador acede à página de adesão e preenche o formulário com o seu nome, email e a organização que pretende integrar. A escolha da organização é relevante: no Hyperledger Fabric cada utilizador pertence formalmente a uma organização específica. Por simplicidade e usabilidade decidiu-se que esta indicação seria feita pelo próprio utilizador, cabendo ao administrador corrigi-la se necessário. O email indicado deve pertencer a um dos *identity providers* suportados, dado que a autenticação é efetuada exclusivamente através destes serviços, conforme detalhado mais adiante. Após a submissão, o pedido é enviado para a REST API que cria o utilizador na blockchain mas ainda sem certificados ou material criptográfico, dado que o utilizador ainda não foi aprovado. De relembrar que o utilizador é representado como uma instância do metamodelo. Para gerir este ciclo de vida, cada utilizador possui um atributo `status` com os valores possíveis:

- `pending`: pedido de adesão submetido, mas o utilizador ainda não pode aceder ao sistema;
- `active`: utilizador aprovado, com certificados e permissões atribuídas;
- `rejected`: pedido analisado e rejeitado;
- `revoked`: utilizador anteriormente ativo cujo acesso foi posteriormente retirado, implicando a revogação dos respetivos certificados.

Criar o utilizador no momento da submissão do pedido de adesão pode parecer um potencial desperdício, dado que alguns utilizadores poderão nunca vir a ser aprovados. No entanto, esta abordagem simplifica o fluxo e garante rastreabilidade completa do processo. Além disso, o custo de armazenamento de um utilizador pendente é marginal e o facto de o utilizador já existir na blockchain facilita o processo de aprovação ou rejeição subsequente, evitando operações adicionais.

Após a submissão do pedido, cabe ao administrador aceder à página de gestão de utilizadores, onde são listados todos os utilizadores, incluindo aqueles com estado `pending`, e decidir se pretende aprovar ou rejeitar o pedido de adesão. Quando um utilizador é aprovado, o sistema atualiza o seu estado para `active` e desencadeia o processo de geração do material criptográfico necessário para interagir com a blockchain. A partir desse momento, o utilizador pode autenticar-se através da sua conta Google ou Microsoft. Caso o pedido seja rejeitado, o estado é alterado para `rejected` e o utilizador não pode aceder ao sistema. Idealmente, o utilizador seria notificado quando o seu pedido fosse aprovado ou rejeitado. Contudo, esta funcionalidade não foi considerada essencial para a aplicação ou para os objetivos da dissertação, uma vez que não tem impacto direto nas questões relacionadas com identidade, autenticação ou integração com a blockchain. Para além

disso, notificar o utilizador sobre o seu pedido de adesão iria representar um custo, uma vez que a notificação teria de ser exterior ao sistema, por exemplo por email, o que implicaria possuir um serviço de email. Por essa razão, optou-se por não a implementar.

Como o processo de aprovação depende de ação manual do administrador, trata-se naturalmente de um processo assíncrono. Uma vez que não há qualquer mecanismo de notificação sobre pedidos de adesão pendentes, o estado do utilizador é atualizado apenas quando o administrador executa a operação na interface de gestão. Esta abordagem revelou-se suficiente para o nível de consistência exigido pelo sistema.

4.6.2 Autenticação

Passando à autenticação propriamente dita, o sistema não se baseia num registo local de credenciais, de forma a evitar a criação de contas e palavras-passe adicionais. Em vez disso, o sistema recorre exclusivamente aos *identity providers* externos Google e Microsoft, que consideraram-se ser os mais comuns e prováveis no contexto dos utilizadores-alvo da aplicação.

Este mecanismo apresenta algumas particularidades face ao que é habitual nos sistemas tradicionais. Quando um utilizador se autentica com sucesso num *provider* externo, é devolvido ao cliente um token que comprova a validade das suas credenciais nesse serviço. No entanto, este passo não é suficiente no contexto de um consórcio: apesar de possuir uma conta Google ou Microsoft válida, o utilizador pode não estar autorizado a utilizar o sistema, encontrando-se, por exemplo, num estado `pending`, `rejected` ou `revoked`.

Por essa razão, após a autenticação externa, o token obtido é enviado para a camada de backend associada à blockchain. Esta camada verifica, em primeiro lugar, a validade do token junto do respetivo *provider* e, em segundo lugar, se existe um utilizador registado na blockchain com estado `active` correspondente a essa identidade. Apenas após este *handshake* o acesso é autorizado, sendo então emitido um token de sessão (JWT), utilizado pelo cliente para autenticar os pedidos subsequentes ao backend e interagir com o sistema. Este mecanismo corresponde à aplicação das táticas *Identifiy Actors*, *Authenticate Actors*, *Authorize Actors* e *Revoke Access*, dado que assegura que apenas utilizadores autenticados e com estado `active` no *ledger* podem interagir com o sistema, assumindo que possuem um token de sessão válido. Os cenários de qualidade QA11 e QA 14 definidos na Secção 4.4, relacionados com controlo de acesso e revogação de sessões, são assim suportados pelo sistema.

A coordenação entre os diversos componentes arquiteturais durante este processo é detalhada no diagrama de comunicação apresentado na Figura 13. Este diagrama evidencia a sequência de mensagens trocadas e a

dependência da validação do estado do utilizador no *ledger* (mensagem 2.1) antes da emissão do token de sessão.

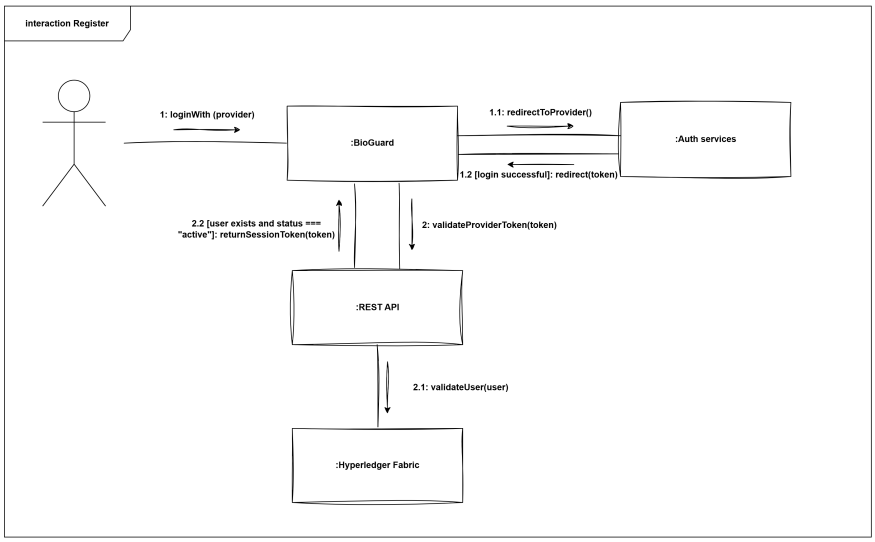

Figura 13: Diagrama de comunicação do processo de autenticação

4.6.3 Gestão de utilizadores

Para suportar o processo de adesão descrito anteriormente e assegurar a governação do consórcio, houve a necessidade de desenvolver uma página de gestão de utilizadores, acessível apenas a administradores. Esta página permite consultar todos os utilizadores, independentemente do seu estado, aprovar ou rejeitar pedidos de adesão pendentes e gerir o seu estado de acesso ao sistema.

Para controlar o acesso a determinadas funcionalidades no cliente, bem como restringir as operações permitidas no backend, implementou-se um mecanismo de papéis (*roles*), acrescentando um atributo *roles* à instância do utilizador, permitindo-lhe acumular um ou mais papéis. Este mecanismo é fundamental, pois permite alinhar a gestão de permissões da aplicação com o modelo de identidade do Hyperledger Fabric, onde os papéis são extraídos do MSP (*Membership Service Provider*).

A página de gestão inclui ainda operações que refletem as exigências dinâmicas de um consórcio. A alteração da organização associada a um utilizador é uma funcionalidade que se considerou potencialmente necessária para a evolução do projeto. Contudo, para manter a integridade e a governança do consórcio, esta operação implica a transição forçada do utilizador para o estado `pending`, exigindo nova aprovação por parte

da organização de destino. Esta abordagem garante a consistência com o princípio de que a afiliação a uma organização é uma decisão administrativa dentro do consórcio.

4.7 Gestão de dados no cliente e desempenho

A análise deste subproblema deve ser enquadrada à luz da evolução arquitetural do sistema. Numa fase inicial, conforme descrito na Secção 4.5, a aplicação funcionava exclusivamente sobre armazenamento local baseado em RxDB, servindo este como único mecanismo de persistência. Esta abordagem revelou-se adequada para implementar e validar as principais funcionalidades da aplicação, bem como para definir os fluxos de interação com os dados, num contexto controlado e de baixa latência.

Numa segunda fase, a blockchain passou a ser utilizada, de forma direta e sem preocupações de otimização, assumindo um papel semelhante ao de uma base de dados remota, conforme descrito na secção 4.5.5. As operações de leitura e escrita consistiam essencialmente em `getAlls`, `inserts` e `updates`, sendo os registos enviados e recuperados na sua totalidade. Esta estratégia refletia, em grande medida, o modelo previamente adotado quando o sistema utilizava exclusivamente RxDB como mecanismo de persistência local. Nesse contexto, as operações de consulta consistiam frequentemente na recuperação completa das coleções, seguida de filtragem e processamento em memória para obter os subconjuntos de dados pretendidos. Embora esta abordagem fosse aceitável num cenário local, a sua transposição direta para um ambiente descentralizado revelou-se inadequada, dado que a recuperação sistemática de grandes volumes de dados para posterior filtragem local introduz custos elevados de latência e processamento, comprometendo vários dos atributos de qualidade definidos anteriormente.

Tornou-se assim necessário introduzir mecanismos que permitissem restringir o volume de dados transferidos, aproximando as operações de leitura do princípio de pedir apenas o que é necessário. Contudo, ao contrário de uma base de dados tradicional, esta necessidade tem de ser conciliada com as limitações e características próprias de uma blockchain. Para enfrentar estas e outras dificuldades associadas à gestão de dados a partir da aplicação cliente, bem como fundamentar as decisões arquiteturais adotadas, foi necessário recorrer novamente à literatura, não com o objetivo de realizar uma nova revisão sistemática, mas para identificar princípios e compromissos relevantes para a gestão de dados do ponto de vista da aplicação cliente.

4.7.1 Recomendações da literatura

Embora a literatura permaneça relativamente escassa no que diz respeito à otimização do uso da blockchain do ponto de vista do cliente, alguns trabalhos fornecem reflexões relevantes para este subproblema.

Wöhrer et al. [27] discutem decisões arquiteturais que têm impacto direto no comportamento das aplicações descentralizadas. Em particular, os autores destacam que o nível de descentralização tem implicações diretas na gestão de dados, pois uma aplicação totalmente descentralizada, na qual o frontend web comunica diretamente com o backend descentralizado, aumenta a confiança, a resistência à censura e reduz o risco de um ponto único de falha. No entanto, esta abordagem tende a resultar em baixo *throughput* e elevados tempos de resposta. Por esse motivo, é frequentemente adotada uma arquitetura híbrida, que introduz componentes centralizados, à custa de uma redução parcial da confiança. Neste trabalho, como foi possível observar na secção 4.5, optou-se por uma arquitetura híbrida, utilizando uma REST API como camada intermédia. Importa salientar que esta escolha não visa aumentar o *throughput* intrínseco da blockchain, mas sim encapsular as suas especificidades e expor ao frontend uma interface de operações mais simples e controlada.

Jiang et al. [37] destacam a utilização do Fabric Gateway, um componente introduzido no Hyperledger Fabric 2.4, que permite a invocação de *chaincodes* de maneira mais simples e eficiente, o que permite o aumento do *throughput* e a redução do consumo de recursos. O acesso à blockchain neste trabalho é realizado através do Fabric Gateway, procurando beneficiar das abstrações e otimizações introduzidas por este componente, indo assim ao encontro a esta recomendação.

Diversos autores convergem também na recomendação de que a blockchain deve ser utilizada com moderação, sobretudo no que diz respeito ao armazenamento e acesso aos dados. Popchev e Radeva [51] reforçam que o armazenamento de grandes volumes de dados diretamente na blockchain é caro e lento, recomendando, para esses casos, o recurso ao armazenamento *off-chain*, utilizando sistemas de ficheiros distribuídos como IPFS [8] ou Swarm [9]. De forma complementar, Xu et al. [9] salientam que a quantidade de dados que pode ser armazenada *on-chain* é intrinsecamente limitada, tornando crucial a decisão criteriosa sobre que dados devem permanecer *on-chain* e quais devem ser mantidos *off-chain*. Kim e Park [42] observam também que aplicações cuja gestão de dados depende exclusivamente da execução de funções em *smart contracts* tendem a apresentar *throughput* reduzido e elevados requisitos de recursos computacionais, reforçando a necessidade de minimizar interações frequentes e pesadas com o *ledger*.

4.7.2 Armazenamento *off-chain*

O armazenamento *off-chain* foi fortemente considerado neste trabalho, atendendo às potenciais vantagens desta abordagem e às recomendações consistentes da literatura. De forma geral, esta estratégia consiste

[8] `https://ipfs.tech/`
[9] `https://www.ethswarm.org/`

em manter volumes significativos de dados fora do *ledger*, recorrendo a sistemas externos, que podem ser descentralizados, como o IPFS ou Swarm, ou centralizados, como bases de dados tradicionais, reforçando assim a natureza híbrida da arquitetura. Os dados armazenados externamente são referenciados *on-chain* através de identificadores únicos de conteúdo (*Content Identifiers* ou CIDs), ou de *hashes* criptográficos, permitindo verificar a integridade da informação sem armazenar diretamente os seus conteúdos na blockchain. Esta abordagem é frequentemente apontada como adequada em cenários onde o armazenamento *on-chain* se revela proibitivo, quer pelo seu custo elevado e volátil, quer pelas limitações impostas ao desempenho e à escalabilidade do sistema. À luz destas considerações, e tendo em conta o âmbito deste trabalho e os requisitos do BioComp, foi conduzido um estudo sobre as implicações da adoção de armazenamento *off-chain*, com particular foco nos impactos arquiteturais e nos benefícios potenciais para a aplicação.

No contexto específico do BioComp, esta estratégia revelou-se particularmente apelativa para o tratamento das medições associadas às pilhas de compostagem. Estas medições, que incluem valores de pH, humidade, temperatura e relação carbono–azoto, são recolhidas por sensores e importadas através de ficheiros, resultando em volumes consideráveis de dados históricos que introduzem um impacto negativo no carregamento e processamento no cliente. Adicionalmente, foi também considerada a possibilidade de, no futuro, integrar imagens captadas por drones para apoiar a monitorização do jacinto-de-água, um cenário em que o armazenamento direto de dados multimédia na blockchain seria claramente inadequado. Durante esta análise, concluiu-se que uma eventual adoção de armazenamento *off-chain* no BioComp seria muito provavelmente baseada numa solução centralizada, dado que as alternativas descentralizadas mais comuns, como o IPFS, implicam custos operacionais adicionais que não se pretendem impor ao consórcio nesta fase, enquanto que o Swarm apresenta um forte acoplamento ao ecossistema Ethereum, não alinhado com a infraestrutura baseada em Hyperledger Fabric. No entanto, a adoção de armazenamento centralizado introduziria um ponto único de falha e levantaria preocupações adicionais ao nível da privacidade e soberania dos dados, uma vez que estes teriam de ser protegidos através de mecanismos de encriptação para evitar exposição indevida. Para além disso, esta abordagem acarretaria um aumento significativo da complexidade arquitetural do sistema, tanto ao nível do cliente como da REST API, exigindo responsabilidades adicionais como a geração e validação de *hashes*, a encriptação e desencriptação de dados, e a gestão da consistência entre dados *off-chain* e referências *on-chain*. Considerando o estado do sistema nesta altura, que embora não apresentasse o desempenho de uma aplicação web tradicional, também não evidenciava limitações críticas, e atendendo ao facto de que o armazenamento *off-chain* pode atenuar algumas das propriedades fundamentais que motivam o uso de blockchain, dependendo da forma como é implementado, decidiu-se não adotar esta abordagem no âmbito

deste trabalho. Ainda assim, tendo em conta o objetivo desta dissertação de fornecer orientações arquiteturais para o desenvolvimento de aplicações descentralizadas, destaca-se o armazenamento *off-chain* como uma alternativa relevante a considerar desde as fases iniciais de conceção do sistema, sobretudo em cenários onde se antecipa a necessidade de gerir grandes volumes de dados.

4.7.3 Mango Queries para otimização de acessos

Com o objetivo de mitigar o problema da recuperação integral das instâncias nas operações de leitura e alinhar o acesso aos dados com o princípio de "pedir apenas o que é necessário", otimizando assim a utilização da blockchain, foi adotado um mecanismo de consultas baseado em Mango Queries. Esta escolha resulta da infraestrutura subjacente do Hyperledger Fabric, no qual o estado atual do *ledger* é persistido em bases de dados de estado locais a cada nó, que neste trabalho são CouchDB. Este sistema suporta nativamente consultas ricas através de Mango Queries, uma linguagem declarativa inspirada no modelo do MongoDB, que permite especificar critérios de seleção por meio de predicados estruturados e tirar partido de índices para restringir o conjunto de resultados devolvidos.

A adoção deste mecanismo revelou-se particularmente adequada por permitir definir um modelo de consultas uniforme e reutilizável, tanto no contexto do RxDB, como na interação com a blockchain, reduzindo a disparidade entre os mecanismos de persistência local e remota. Do ponto de vista da gestão de dados no cliente, esta abordagem permite deslocar parte da lógica de filtragem para o lado do sistema que fornece os dados, evitando a transferência desnecessária de grandes volumes de informação. Em particular, ao interagir com a blockchain, que conforme discutido anteriormente, deve ser utilizada com moderação, as Mango Queries possibilitam a recuperação seletiva de registos com base em atributos relevantes para o contexto da aplicação. No BioComp, esta estratégia foi aplicada, por exemplo, na recuperação de entidades associadas a um determinado contexto funcional, recorrendo a um classifier que identifica o tipo dos dados pretendidos. Um exemplo simplificado de uma consulta é apresentado na figura 14:

```
async loadRoles (model: IModel): Promise<IInstance[]> {
    const roleClassifier = this.getClassifier('Role');
    let roles = await model.getAll({
        selector: { classifier: roleClassifier.id }
    });
    return roles ?? [];
}
```

Figura 14: Exemplo de uma Mango Query para obter todas as *Roles* do sistema

Esta estrutura é construída nos componentes e transmitida ao repositório de dados, que expõe um método genérico de leitura (`getAll`). De relembrar que, conforme explicado na secção 4.5.5, a comunicação com os repositórios é intermediada pelo componente Model, que fornece uma interface semelhante à dos repositórios. O método `getAll` do repositório comunica com a blockchain através do Blockchain Gateway, passando-lhe a estrutura da *query* como objeto JSON, para que esta seja enviada para a blockchain. Embora semanticamente se trate de uma operação de leitura, a consulta é enviada através de um pedido HTTP do tipo `POST`. Esta decisão deve-se às limitações práticas do protocolo HTTP, uma vez que pedidos `GET` não suportam a inclusão de um *body* com parâmetros, como é o caso das estruturas de consulta necessárias para a execução das Mango Queries. A utilização de pedidos `POST` para operações de leitura não pretende alterar a semântica da operação, mas sim viabilizar a transmissão de critérios de consulta ricos e extensíveis, mantendo uma interface uniforme entre o cliente, a REST API e a camada de acesso à blockchain.

Apesar das vantagens introduzidas pelo uso de Mango Queries, este mecanismo apresenta uma limitação importante do ponto de vista da modelação e do acesso aos dados: a ausência de suporte nativo para operações de população de entidades, isto é, a recuperação automática de objetos associados a uma determinada entidade. Em particular, as consultas permitem selecionar e filtrar documentos com base nos seus atributos, mas não oferecem mecanismos para resolver referências entre entidades, devolvendo apenas os identificadores dos objetos relacionados. É possível observar as consequências desta limitação na Figura 15, onde uma instância de *User* não possui informação da organização a que este pertence, mas apenas o seu id.

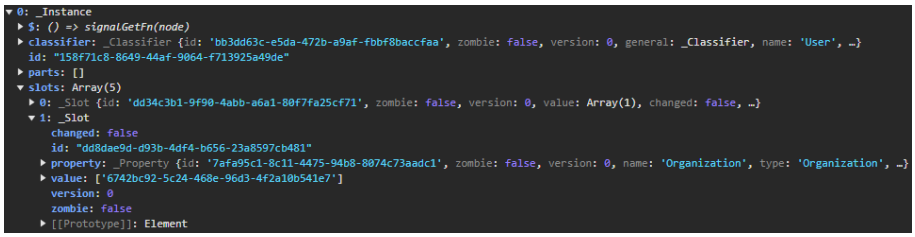

Figura 15: Exemplo de uma Mango Query que não resolve as referências das propriedades

Esta limitação revelou-se relevante no contexto do BioComp, uma vez que certas entidades do domínio apresentam relações diretas com outras entidades. Por exemplo, um utilizador está relacionado com entidades como pessoa, organização e papéis (*roles*), sendo que, na figura anterior, a organização do utilizador não foi carregada. Na ausência de um mecanismo de população, a obtenção de um objeto completo implicaria a execução de múltiplas consultas adicionais à blockchain, uma por cada entidade relacionada, conduzindo

a um padrão de acesso fragmentado e ineficiente. Tal abordagem contraria diretamente as recomendações discutidas anteriormente, que salientam a necessidade de minimizar o número de interações com o *ledger*.

Para responder a esta necessidade, foi implementado um mecanismo próprio de população de entidades, designado por *includes*. Este mecanismo permite especificar, no momento da execução de uma consulta, que propriedades de uma determinada entidade devem ser resolvidas e carregadas automaticamente, em vez de serem devolvidas apenas como identificadores. Esta abordagem inspira-se em mecanismos semelhantes presentes em *Object-Relational Mappers* (ORMs) tradicionais, mas foi adaptada ao contexto do metamodelo definido neste trabalho. Assim, ao executar uma consulta para obter instâncias de um determinado classifier, é possível especificar um conjunto de propriedades a incluir, fazendo com que os objetos associados sejam carregados e integrados na resposta final. A Figura 16 ilustra um exemplo de uma consulta usando este mecanismo de população:

```
async loadUsers(model: IModel): Promise<IInstance[]> {
    const userClassifier = this.getClassifier('User');
    let users = await model.getAll( {
        selector: { classifier: userClassifier.id },
        includes: [
            { property: userClassifier?.getPropertyByName("Person")?.id},
            { property: userClassifier?.getPropertyByName("Organization")?.id },
            { property: userClassifier?.getPropertyByName("Roles")?.id }
        ]
    } )

    return users ?? [];
}
```

Figura 16: Exemplo de uma Mango Query para obter todos os *Users* do sistema, juntamente com a sua *Person*, *Organization* e *Roles*

Importa salientar que o mecanismo de *includes* é implementado maioritariamente na camada de backend, sendo interpretado durante a execução da consulta sobre a infraestrutura de dados subjacente. Do ponto de vista da aplicação cliente, a utilização deste mecanismo resume-se à inclusão da especificação dos *includes* no objeto de consulta enviado ao backend, não sendo necessário qualquer processamento adicional para a resolução das relações entre entidades. O resultado obtido numa consulta com *includes* é apresentado na Figura 17:

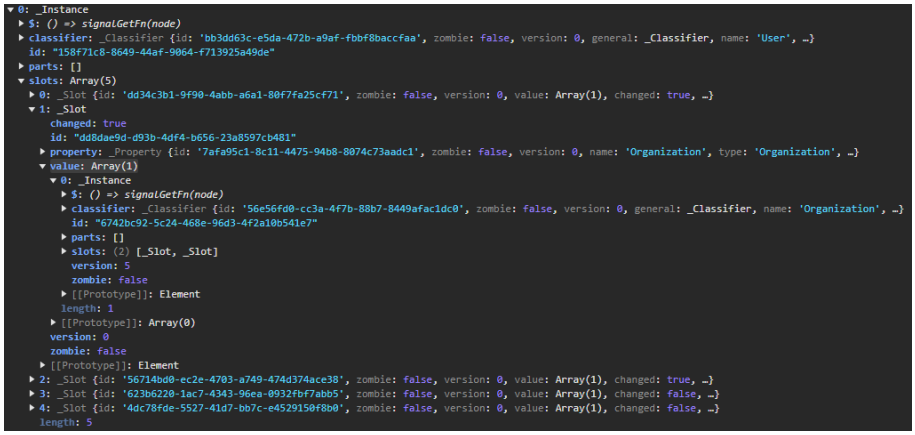

Figura 17: Resultado de uma Mango Query para obter todos os *Users* do sistema, juntamente com a sua *Person*, *Organization* e *Roles*

Desta forma, ao recuperar entidades de um determinado tipo, torna-se possível indicar que propriedades correspondentes devem também ser carregadas, evitando a necessidade de consultas adicionais para resolver essas relações. O resultado é a obtenção de objetos mais completos e semanticamente ricos, com um número reduzido de acessos à blockchain. Esta abordagem também encontra-se alinhada com boas práticas frequentemente associadas a arquiteturas REST, em particular com o objetivo de minimizar o número de interações entre cliente e servidor, reduzindo *round-trips* desnecessários e privilegiando a devolução de representações de recursos mais completas e auto-suficientes num único pedido. Do ponto de vista da gestão de dados no cliente, este mecanismo contribui significativamente para a eficiência do sistema, ao reduzir o volume total de interações com a infraestrutura descentralizada. Ao mesmo tempo, mantém-se a flexibilidade do modelo de dados, uma vez que a população de entidades é opcional e controlada explicitamente pelo cliente, permitindo ajustar o comportamento das consultas às necessidades específicas de cada contexto de uso.

A nível da aplicação cliente, o suporte a este mecanismo de consultas e população de entidades foi complementado com o aproveitamento da infraestrutura reativa disponibilizada pelo Angular. Em particular, as operações assíncronas de leitura de dados foram encapsuladas em *resources*, permitindo centralizar a gestão do estado associado às consultas, bem como refletir automaticamente na interface quaisquer alterações nos dados obtidos. Esta abordagem facilita a sincronização entre o estado da aplicação e os resultados das consultas executadas sobre a blockchain, promovendo uma experiência de utilização mais responsiva. Adicionalmente, em cenários onde foi necessário realizar operações derivadas ou transformações sobre os dados obtidos, recorreu-se a *signals*, permitindo reagir de forma declarativa às alterações dos *resources*

sem introduzir dependências explícitas ou lógica imperativa adicional. Esta combinação contribui para uma separação clara entre a obtenção dos dados, a sua transformação e a sua apresentação, reforçando os princípios de modularidade e manutenibilidade da aplicação cliente.

4.7.4 Estratégias de *fetching*

Após a implementação de um mecanismo que permitisse não só a execução de consultas simples, mas também consultas ricas sobre os dados persistidos na blockchain, tornou-se pertinente avaliar diferentes estratégias de obtenção (*fetching*) de dados. O objetivo foi identificar as vantagens e limitações de cada abordagem, bem como analisar em que medida esta estratégias iam de encontro com as recomendações presentes na literatura. Com efeito, foram consideradas as estratégias de *Eager Loading*, *Lazy Loading* e *Explicit Loading*, por serem amplamente utilizadas em ORMs como o Entity Framework, e por se terem revelado relevantes nos estudos inicias apresentados na Secção 4.1.3. Estas estratégias endereçam diretamente o problema de obtenção eficiente de dados relacionados, procurando equilibrar desempenho, consumo de recursos e complexidade de implementação. Em aplicações tradicionais baseadas em ORMs, a escolha de uma estratégia de *fetching* adequada permite reduzir o número de consultas necessárias, evitar a transferência de dados desnecessários e simplificar a lógica de acesso a dados no código da aplicação. Embora o contexto de uma aplicação descentralizada apresente características e limitações próprias, estas estratégias revelam-se conceptualmente relevantes também no acesso a dados *on-chain*, onde cada interação com o *ledger* implica custos acrescidos de latência e processamento.

A estratégia de *Eager Loading* consiste em carregar uma entidade e os seus dados relacionados numa única operação de leitura. Nesta abordagem, o conjunto de entidades associadas a carregar é definido explicitamente no momento da consulta inicial, sendo que esta é normalmente aplicada quando se sabe que os dados relacionados vão ser usados de imediato.

O *Lazy Loading* consiste em retardar o carregamento dos dados relacionados a uma entidade até ao momento em que essas propriedades são acedidas, sendo automaticamente carregadas nesse momento. É aconselhado usar esta estratégia quando os dados relacionados são acedidos esporádicamente, de forma a evitar o carregamento de dados que o utilizador pode, ou não, visualizar.

Já o *Explicit Loading* consiste em carregar os dados relacionados *on-demand*, usando métodos explícitos após obter a entidade principal. Normalmente é aconselhado usar esta estratégia quando os dados relacionados só são necessários sob certas condições de *run-time*, por exemplo, a aplicação de certos filtros.

Importa notar que estratégias como *Lazy* ou *Explicit Loading* implicam um maior número de interações com a blockchain, o que se pode traduzir em custos acrescidos de latência e processamento, e contraria também as recomendações da literatura no que toca à minimização de acessos. Por outro lado, estas estratégias podem revelar-se adequadas em cenários específicos, nomeadamente quando o carregamento antecipado de todos os dados relacionados resulta na transferência de volumes elevados de informação, daí a necessidade de testar e comparar estas diferentes abordagens.

No BioComp, a estratégia de *Eager Loading* foi implementada através do mecanismo de *includes* nas Mango Queries, conforme descrito na Secção 4.7.3. Este mecanismo permite que propriedades relacionadas sejam resolvidas e integradas na resposta de forma antecipada.

Relativamente à estratégia de *Lazy Loading*, a sua implementação direta revelou-se problemática no contexto da aplicação desenvolvida. Esta estratégia pressupõe a existência de um mecanismo capaz de detetar acessos a propriedades ainda não resolvidas e, de forma transparente, desencadear operações de carregamento assíncronas em segundo plano, notificando posteriormente a interface do utilizador. Em ORMs tradicionais, como o Entity Framework, este comportamento é tipicamente suportado através de *proxies* gerados dinamicamente, que interceptam o acesso às propriedades e gerem o seu carregamento sob procura. No entanto, no ecossistema do Angular e na linguagem TypeScript não existe uma infraestrutura nativa equivalente que permita a implementação transparente deste mecanismo ao nível das propriedades dos objetos do domínio. Embora o Angular suporte mecanismos de *Lazy Loading* noutros níveis, como no carregamento diferido de módulos de aplicação e na utilização dos *resources*, que apenas disparam operações assíncronas quando os seus valores são efetivamente necessários, estes mecanismos não se estendem à resolução automática de relações entre entidades no momento de acesso às respetivas propriedades. A tentativa de simular este comportamento através de mecanismos alternativos, como *getters* genéricos, revelou-se inadequada no contexto desta aplicação. Dada a natureza assíncrona e potencialmente lenta das interações com o *ledger*, esta abordagem resultaria na emissão de um elevado número de pedidos concorrentes ao backend, bem como na propagação de múltiplas notificações para o frontend, devido ao mecanismo de deteção de alterações do Angular. Na prática, este comportamento conduziu a um número excessivo de atualizações reativas na interface, originando uma degradação significativa de desempenho e, em cenários extremos, o bloqueio do *browser*. Por este motivo, a implementação deste mecanismo foi considerada inviável no contexto atual, tendo-se optado por descartar esta opção.

Quanto à estratégia de *Explicit Loading*, foi implementado um mecanismo de garantia e população sob procura designado de `ensures`. Este método foi concebido para assegurar que as propriedades de uma entidade que representam referências a outras entidades se encontram devidamente resolvidas, evitando erros de execução associados a referências não carregadas. Quando necessário, o método procede à resolução explícita dessas propriedades, materializando assim um comportamento de *Explicit Loading*. Esta abordagem permite que os dados relacionados sejam carregados apenas quando efetivamente requeridos, mantendo um rasto de memória reduzido. A implementação deste método é apresentada na Figura 18.

```
async ensures( instance: IInstance, property?: string ): Promise<IInstance> {
    if ( property === undefined ) {

        for ( const property of instance.classifier.getAllProperties() ) {
            if ( this.metamodel.isPropertyTypeOf( property, "Primitive"   ) ) continue;
            if ( this.metamodel.isPropertyTypeOf( property, "Enumeration" ) ) continue;
            await this.ensuresProperty( instance, property.name );
        }

    } else {
        await this.ensuresProperty( instance, property );
    }

    return instance;
}

async ensuresProperty( instance: IInstance, property: string ): Promise<IInstance> {

    let elements = instance.getValue( property );
    if ( elements === undefined ) return instance;

    if ( typeof elements === "string" ) {
        const value = await this.get( elements );
        instance.setValue( property, value );
    } else if ( Array.isArray( elements ) ) {
        if ( elements.every( v => Instance.isInstance( v ) ) ) {
            return instance;
        } else if ( elements.every( v => typeof v === "string" ) ) {
            const instances = await this.getAllById( ...elements );
            instance.setValue( property, instances );
        }
    }
    return instance;
}
```

Figura 18: Implementação do método `ensures`, crucial para o *Explicit Loading* de propriedades de uma entidade

Quando é fornecida uma propriedade específica ao `ensures`, o método `ensuresProperty` obtém o valor atualmente armazenado na respetiva Slot, que pode corresponder a um identificador único ou a um conjunto de identificadores, e verifica se a propriedade já se encontra resolvida. Caso contrário é emitido um pedido ao backend para obter as instâncias correspondentes, que é feito através do método `get` ou `getAllById`, respetivamente. O valor da Slot é então atualizado, substituindo os identificadores pelas instâncias carregadas.

Quando nenhuma propriedade é fornecida ao `ensures`, o método percorre todas as propriedades definidas no Classifier da instância, ignorando propriedades de tipo primitivo ou enumerado, e resolve explicitamente aquelas que representam referências a outras entidades.

A avaliação comparativa destas estratégias de *fetching* (*Eager Loading* e *Explicit Loading*) será apresentada na Secção 5, onde serão definidas tarefas experimentais e métricas de desempenho com o propósito de responder à QI1.

4.7.5 Operações de escrita (*create, update, delete*)

Para além das operações de leitura, as operações de escrita (criação, atualização ou remoção de entidades) e a forma como estas são realizadas também assume particular relevância no contexto das dApps baseadas em blockchain, dada a capacidade de processamento limitada da blockchain, conforme mencionado nos artigos previamente analisados. Tendo isto em conta, neste trabalho adotou-se uma estratégia orientada à minimização do volume de dados transmitidos nestas operações, procurando enviar apenas a informação estritamente necessária para a execução de cada operação. Esta decisão de design visa reduzir o tamanho das mensagens trocadas entre o cliente e a blockchain, diminuir o esforço de validação e processamento *on-chain* e, consequentemente, procurar melhorar a latência percebida e a eficiência global do sistema.

No caso das operaçoes de criação (*Create*), é necessário transmitir praticamente toda a estrutura da instância (id, classifier, propriedades e respetivas slots), juntamente com metadados adicionais, como a versão da instância e o atributo `zombie`, conforme ilustrado na Figura 19. Esta abordagem justifica-se pelo facto de a entidade estar a ser registada pela primeira vez no sistema. A versão da instância desempenha um papel central na resolução de conflitos de escrita na blockchain, conforme será detalhado na Secção 4.8.

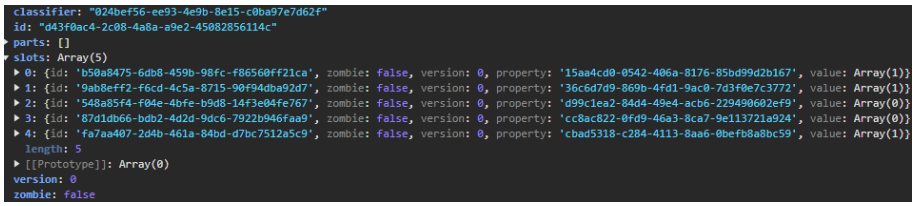

Figura 19: Informação enviada para o backend na criação de uma instância

Nas operações de atualização (*Update*) optou-se por uma abordagem baseada em *partial updates*, na qual apenas os campos efetivamente alterados são incluídos na operação de escrita. Esta estratégia aproxima-se do conceito de *delta updates*, permitindo evitar o reenvio do estado completo da entidade sempre que

ocorre uma modificação pontual. Isto é possível através de um atributo booleano associado a cada slot, denonimado `changed`, que é utilizado no frontend para sinalizar alterações na instância. Desta forma, nas operações de atualização, a informação transmitida inclui o identificador da instância, as slots modificadas, as partes associadas (no caso de instâncias compostas) e a respetiva versão. A estrutura geral desta operação é apresentada na Figura 20.

Figura 20: Informação enviada para o backend na atualização de uma instância

Por fim, nas operações de eliminação (*Delete*), a informação transmitida é reduzida ao identificador da instância, à sua versão e ao atributo `zombie` definido a *true*. Este atributo funciona como uma *flag* de *soft delete*, sinalizando que a instância foi removida logicamente, mas não eliminada de forma definitiva. Esta opção deve-se à potencial necessidade de restaurar instâncias eliminadas (*rollback*). Um exemplo da estrutura desta operação é apresentado na Figura 21

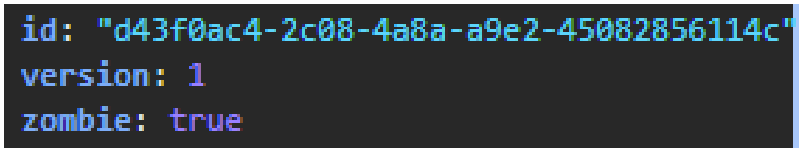

Figura 21: Informação enviada para o backend na eliminação de uma instância

Importa salientar novamente que, apesar do *ledger* ser imutável e não permitir a eliminação de registos, a base de dados que representa o estado atual dos *assets* é mutável, permitindo refletir atualizações e eliminações lógicas de forma consistente com o modelo de dados adotado.

4.7.6 Estratégias de *caching* e reutilização de dados no cliente

O *caching* foi, desde início, considerado um dos principais mecanismos para mitigar a latência inerente às interações com a blockchain e reduzir o número de acessos ao *ledger*, contribuindo para uma melhoria significativa do desempenho global da aplicação e da experiência do utilizador. No entanto, ao longo do desenvolvimento tornou-se evidente que os diferentes tipos de dados do domínio do BioComp apresentavam

características distintas em termos de frequência de alteração e padrões de acesso, justificando a adoção de diferentes abordagens de *caching* em função da natureza dos dados.

No caso dos dados estruturais do sistema, isto é, do metamodelo, optou-se por uma estratégia de *caching* persistente baseada em RxDB. Estes dados são necessários de forma transversal ao funcionamento da aplicação e constituem um pré-requisito para a correta instanciação e manipulação do modelo no cliente. Conforme descrito na Secção 4.5.5, o metamodelo é gerido centralmente pelo Data Store, que durante o arranque da aplicação consulta a blockchain através do Blockchain Storage para obter a versão mais recente do metamodelo e compará-la com a versão armazenada localmente em RxDB, se existir. O carregamento a partir do *ledger* só é desencadeado se não existir uma versão local, ou se esta não corresponder à registada na blockchain. Embora o metamodelo possa apresentar uma elevada volatilidade nas fases iniciais de desenvolvimento da aplicação, período em que são frequentemente introduzidos novos *classifiers* e ajustadas as relações do domínio, esta volatilidade tende a reduzir significativamente à medida que a lógica de negócio estabiliza. Esta característica torna o metamodelo particularmente adequado a uma estratégia de *caching* persistente. Adicionalmente, esta abordagem constitui uma solução simples e eficaz para o *caching* de uma operação de leitura volumosa, uma vez que o metamodelo é tratado como uma unidade coesa. Deste modo, não é necessária a definição de algoritmos de seleção de conteúdos para a *cache*, nem de políticas sofisticadas de invalidação ou *refresh*. A informação recebida na importação do metamodelo inclui explicitamente a sua versão, seguindo a convenção `SCHEMA_vX`, onde `X` representa a sua versão, complementada por um *timestamp* que indica o momento da última modificação, permitindo uma verificação direta e eficiente de consistência entre a *cache* local e o estado registado no *ledger*. A informação recebida numa importação do metamodelo é apresentada na Figura 22.

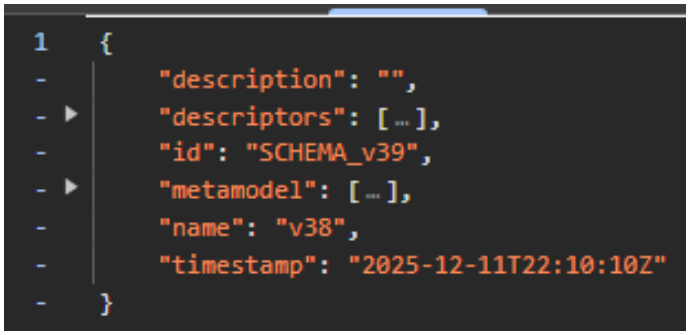

Figura 22: Informação recebida na importação do metamodelo

Depois do *caching* do metamodelo tornou-se necessário definir um mecanismo de otimização para o modelo (instâncias e respetivas *slots*). Ao contrário do metamodelo, as instâncias do modelo estão sujeitas a operações frequentes de criação, leitura, atualização e remoção, que implicam acessos repetidos à blockchain, como também a instanciação recorrente de objetos no cliente, o que introduz custos adicionais em termos de desempenho e consumo de recursos. Adicionalmente, foi identificado como um desafio crítico a possibilidade de coexistirem em memória múltiplas instâncias distintas que representassem logicamente a mesma entidade do domínio, o que poderia conduzir a inconsistências de estado no cliente, dificultando a correta propagação de alterações e a gestão reativa dos dados.

Para mitigar este problema, tornou-se essencial garantir que cada entidade do domínio fosse representada por uma única instância em memória. Com este objetivo e com a intenção de procurar otimizar o acesso às instâncias do modelo, foi implementado um mecanismo de *caching* em memória, baseado no padrão *Identity Map*, cujo objetivo é garantir que cada entidade persistente é representada por uma única instância de objeto no contexto de execução da aplicação. Isto foi feito através de um dicionário designado `entities`, onde a chave corresponde ao identificador único da instância (ID) e o valor à respetiva instância do objeto. Este dicionário, que é possível observar na Figura 23, atua como um repositório central de entidades já carregadas, permitindo controlar explicitamente quais as instâncias que se encontram presentes em memória em cada momento.

```
private entities : { [ key: string ]: IInstance } = {};
```

Figura 23: Dicionário utilizado para o *caching* em memória

Dado que um dos objetivos deste dicionário é otimizar as operações CRUD, faz sentido que este esteja no Blockchain Repository, uma vez que é aqui que essas operações estão encapsuladas. No caso dos pedidos `get`, em que é solicitada uma instância individual a partir do seu identificador, o processo inicia-se com uma verificação no dicionário para ver se esta já se encontra registada no mesmo. Nesse caso, é feita uma consulta à blockchain para a obtenção da versão atual da instância e posteriormente uma verificação para averiguar se as duas versões coincidem. Se as versões coincidirem, a instância em memória é reutilizada, evitando um novo carregamento e instanciação. Sempre que é detetada uma divergência de versões, ou se a instância ainda não está registada no dicionário, é feito o carregamento a partir do *ledger* e o dicionário é atualizado em conformidade, através do método `rehydrated`, que atualiza a versão da instância nele registada, ou insere-a

no dicionário, respetivamente. Este comportamento é conceptualmente semelhante ao mecanismo adotado para o metamodelo, embora aplicado a uma granularidade distinta.

```
async get( id: string ): Promise< IInstance | undefined > {

    if ( id in this.entities ) {
        const response = await this.gateway.model.get( id, "version" );
        if ( response.length == 0 ) return undefined;

        if ( isEntity( response[0] ) && response[0].version === this.entities[ id ].version )
            return this.entities[ id ];
    }

    const response = await this.gateway.model.get( id  );
    if ( response[0] ) {
        return this.rehydrated( response[0] );
    }
    return undefined;
}
```

Figura 24: Pedido `get` com *caching* em memória

Para pedidos do tipo `getAll` ou `getAllById`, nos quais é solicitado um conjunto potencialmente elevado de instâncias, a verificação individual de versões na blockchain revelar-se-ia dispendiosa e contra as recomendações da literatura. Nestes casos, optou-se por carregar o conjunto de dados solicitado e verificar, para cada instância retornada, se já existe uma representação correspondente no `entities`, utilizando o método `rehydrated` para garantir que essas instâncias em memória permanecem atualizadas.

```
async getAllById( ...ids: string[] ): Promise<IInstance[]> {
    return await this.getAll( { selector: { id: { $in: ids } } } );
}

async getAll( query: any = {} ): Promise<IInstance[]> {

    const response = await this.gateway.model.query( query );

    const instances: IInstance[] = [];
    for ( const instance of Object.values( response ) ) {
        instances.push( this.rehydrated( instance ) );
    }
    return instances;
}
```

Figura 25: Pedidos `getAll` e `getAllById` com *caching* em memória

Já nas operações de criação e remoção, a interação com a blockchain ocorre em primeiro lugar. Apenas após a confirmação de sucesso da operação por parte do *ledger* é que o dicionário em memória é atualizado. No caso de um *insert*, a nova instância é adicionada ao dicionário, no caso de um *delete*, a instância correspondente

é removida. Esta abordagem tira partido do facto de a blockchain fornecer uma confirmação explícita da execução da operação, evitando que o estado em memória seja alterado em caso de falha.

```
async insert( ...entities: IInstance[] ): Promise<IInstance[]> {

    const instances = entities.map( instance => Instance.jsonify( instance ) )
    const response = await this.gateway.model.insert( instances );
    return response?.map( instance => this.rehydrated( instance ) ) || [];
}

async delete( ...entities: IInstance[] ): Promise<void> {

    console.assert( Array.isArray( entities ) );
    if ( entities.length === 0 ) return;

    const instances: any[] = [];
    entities.forEach( entity => {
        instances.push( { id: entity.id, zombie: true, version: entity.version } );
    } );

    try {
        console.log(instances);
        await this.gateway.model.delete( instances ) as unknown as BlockchainResponse;
        entities.forEach( entity => delete this.entities[ entity.id ] );
    }
    catch( error ) {
    }
}
```

Figura 26: Integração do *insert* e *delete* com o dicionário *entities*

Relativamente às operações de atualização, optou-se por uma estratégia distinta. Neste caso, as alterações são aplicadas diretamente à instância em memória, seguindo a abordagem *local-first* e mantendo o dicionário automaticamente atualizado, dado que referencia a mesma instância de objeto. Apenas posteriormente é efetuada a operação de atualização na blockchain. Esta decisão simplifica o fluxo de atualização e evita a duplicação de lógica entre o estado em memória e o estado persistido, embora possa introduzir problemas caso a operação de atualização falhe na blockchain.

```
async update( entity: IInstance ): Promise<IInstance> {

    const changed = entity.slots.filter( slot => slot.changed );
    const slots = changed.map( slot => Slot.jsonify( slot ) );

    const isUser = entity.classifier.name === 'User';

    const instance = {
        id      : entity.id,
        version : entity.version,
        slots   : slots.map( s => Slot.jsonify( s ) ),
        parts   : mapToIds( entity.parts )
    } as IInstance;

    const response = isUser
    ? await this.gateway.auth.users.update(instance)
    : await this.gateway.model.update( [ instance ] );

    if ( response.length > 0 ) {
        entity.version = response[0].version;
    }
    return entity;
}
```

Figura 27: Operações de *update*

Para além dos mecanismos explícitos de *caching* persistente e em memória, o desempenho da aplicação também beneficia com o uso dos *resources* disponibilizados pelo Angular ao nível da camada de apresentação. Apesar de não preservarem o estado entre ciclos de navegação nem gerirem a identidade das instâncias do modelo, estes garantem que as consultas à blockchain são desencadeadas apenas quando os dados são efetivamente necessários por uma determinada vista, seguindo um modelo de *Lazy Loading*, conforme referido anteriormente.

```
@Injectable()
export class SpotsService {

    private store = inject(DataStore);

    $spots = resource( {
        params: () => ( { model: this.store.model() } ),
        loader: ( { params } ) => this.loadSpots()
    });

    async loadSpots(): Promise<IInstance[]> {
        const spots = await this.store.model().getAllByType( "Spot" );
        return await Promise.all( spots.map( spot => this.store.model().ensuresParts( spot ) ) );
    }
```

Figura 28: *Resource* da página *Observe*

A utilização de um Signal Store para gestão do estado da camada de dados do sistema (Data Store) combinada com os *resources* e com o mecanismo de *caching* em memória, resulta numa redução significativa

do número de acessos à blockchain. Enquanto os *resources* atuam exclusivamente como orquestradores reativos para a interface, a preservação do estado é garantida pelo Data Store, que assume a forma de um *Singleton* ao longo do ciclo de vida da aplicação. Analisando o exemplo da Figura 28, quando o *resource* desencadeia um novo pedido de consulta para a página *Observe*, o método `ensuresParts` do Model precisa de ser executado para garantir que as partes da instância são carregadas. Se detetar que as dependências e instâncias relacionadas já se encontram reidratadas no dicionário local, o sistema anula a necessidade de desencadear as várias *queries* de população, que seriam de outro modo necessárias. Desta forma, esta infraestrutura de *caching* em memória protege a blockchain de redundância, utilizando o *resource* apenas como o mecanismo para refletir esses dados de forma fluida na interface e indo de encontro às recomendações da literatura, no que toca à redução de pedidos efetuados à blockchain.

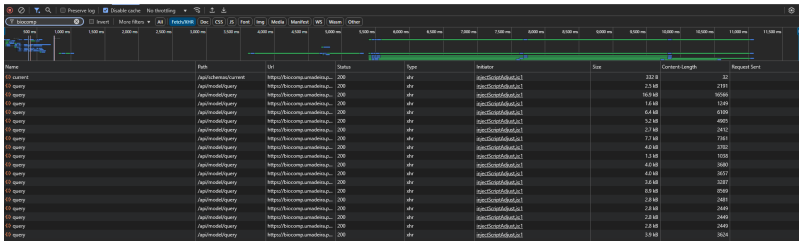

Figura 29: *Queries* executadas no primeiro acesso à página *Observe*

Na Figura 29 é possível observar a quantidade de *queries* executadas ao aceder à página *Observe* pela primeira vez, sendo necessário carregar todos os locais (*Spots*) que estão a ser monitorizados, juntamente com as entidades que os compõem, visto que o dicionário se encontra vazio nesta situação. Caso o utilizador navegue para outra secção da aplicação e posteriormente regresse a esta página, as consultas de população não são reexecutadas, pois os objetos encontram-se totalmente resolvidos no dicionário. Nesse caso, o *resource* executa apenas a *query* para obtenção dos locais, que é sempre necessária para garantir que os objetos em memória permanecem atualizados.

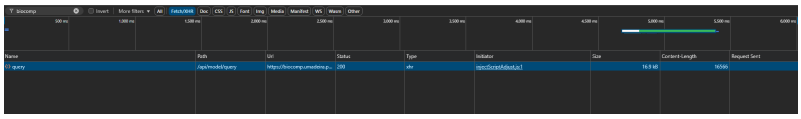

Figura 30: *Query* executada no segundo acesso à página *Observe*

Durante as fases iniciais de conceção da arquitetura, bem como durante a implementação destes mecanismos que acabaram de ser descritos, foi considerada a adoção de uma estratégia de *caching* persistente para as instâncias do modelo, semelhante à utilizada para o metamodelo, recorrendo igualmente ao RxDB. Esta abordagem visava reduzir ainda mais o número de acessos à blockchain e preparar o sistema para cenários de utilização em modo *offline*. No entanto, à medida que os mecanismos descritos anteriormente foram sendo progressivamente implementados, tornou-se evidente que o desempenho obtido era adequado para o contexto atual do sistema, não se justificando a introdução de uma camada adicional de *caching* persistente para o modelo.

A implementação de *caching* persistente para as instâncias do modelo levantaria desafios técnicos consideráveis. Ao contrário do metamodelo, tratado como uma unidade coesa e com baixa frequência de alteração, os dados do modelo apresentam elevada volatilidade e estão sujeitos a modificações frequentes por múltiplos utilizadores. Tal exigiria a definição de algoritmos de seleção de conteúdos para a cache sofisticados, bem como políticas explícitas de invalidação e *refresh*, de forma a evitar a utilização de dados rapidamente obsoletos. Considerando a dimensão do consórcio e a natureza colaborativa do sistema, o custo de conceção e manutenção destes mecanismos não se revelou proporcional aos ganhos de desempenho esperados nesta fase do projeto. A possibilidade de suporte ao funcionamento em modo *offline* foi igualmente ponderada, sobretudo tendo em conta cenários futuros associados à fase da aplicação de composto, nos quais a conectividade com a internet poderia não estar garantida. No entanto, esta funcionalidade acabou por ser descartada na presente fase do projeto. Por um lado, os subproblemas atualmente implementados (monitorização, recolha, transporte e produção) pressupõem uma utilização maioritariamente em contexto de gabinete, com conectividade disponível. Por outro lado, a adoção de um modo *offline* multiutilizador introduziria outros desafios significativos ao nível da sincronização, nomeadamente picos de carga sobre a blockchain aquando do restabelecimento da conectividade, bem como problemas relacionados com a ordenação correta das operações e a resolução de conflitos decorrentes de alterações concorrentes sobre as mesmas instâncias.

Embora estes problemas pudessem ser parcialmente mitigados através do mecanismo de replicação do RxDB, baseado num modelo de sincronização do tipo *publisher–subscriber*, tal abordagem implicaria a adoção de uma estratégia totalmente *local-first*, na qual as operações seriam inicialmente aplicadas localmente e apenas posteriormente propagadas aos restantes clientes. No contexto do sistema desenvolvido, esta opção revelou-se incompatível com os requisitos fundamentais da solução, uma vez que a blockchain constitui o mecanismo de armazenamento primário, responsável por garantir propriedades essenciais como imutabilidade,

segurança e auditabilidade. Uma abordagem *local-first* apenas se revelou interessante na perspetiva das atualizações, mas de resto aumentaria significativamente a complexidade da gestão de erros, da reversão de operações falhadas (*rollback*) e do controlo do estado de sincronização, introduzindo um elevado potencial de conflitos, especialmente nas operações de criação e remoção de entidades. Deste modo, optou-se por uma solução que privilegia a consistência, a simplicidade e a robustez do sistema, concentrando os mecanismos de *caching* persistente no metamodelo e recorrendo ao *caching* em memória, aliado à infraestrutura reativa do Angular, para otimizar o acesso e a reutilização das instâncias do modelo.

As decisões apresentadas nesta secção foram orientadas pela QI1, como também pelos cenários de qualidade de desempenho definidos anteriormente, procurando reduzir o número de interações com a blockchain, o volume de dados transferidos e a latência percebida no cliente. Embora estas estratégias não alterem o *throughput* intrínseco da blockchain, permitem mitigar os seus efeitos do ponto de vista da aplicação cliente, constituindo um compromisso pragmático entre desempenho e as propriedades fundamentais da tecnologia.

4.8 Sincronização e resolução de conflitos

Este último subproblema incide sobre os desafios associados à sincronização do estado da aplicação com a blockchain e à resolução de conflitos em cenários de utilização multiutilizador, como é o caso do BioComp. Em sistemas baseados em blockchain, a natureza distribuída, assíncrona e concorrente do *ledger* impõe decisões arquiteturais relevantes relativamente à forma como a aplicação cliente gere o estado das operações em curso, bem como à forma como lida com conflitos decorrentes de acessos e modificações concorrentes sobre os mesmos dados. A discussão nesta secção será divida em duas vertentes. Primeiro abordam-se as estratégias de sincronização do estado da aplicação com a blockchain, em particular a decisão entre abordagens síncronas e assíncronas no tratamento das transações. Depois, foca-se nos mecanismos de prevenção e resolução de conflitos, inevitáveis em aplicações colaborativas nas quais múltiplos utilizadores podem interagir concorrentemente sobre as mesmas entidades do domínio.

4.8.1 Sincronização do estado das transações e *feedback* da interface

A literatura identifica a natureza intrinsecamente assíncrona das blockchains como um dos principais desafios para o desenho de aplicações descentralizadas [27, 34]. Este caráter decorre da latência associada à propagação, validação e confirmação das transações no *ledger*, fazendo com que o estado da blockchain não possa ser considerado imediatamente consistente após a submissão de uma operação. Neste contexto, em [27]

são identificadas duas abordagens principais para a sincronização do estado das transações: a síncrona e a assíncrona. A abordagem síncrona consiste em suspender o fluxo da aplicação até que a transação atinja a sua finalidade, garantindo uma consistência estrita entre o estado da aplicação e o estado persistido na blockchain. Embora esta abordagem simplifique a gestão do estado, implica tempos de espera elevados, frequentemente incompatíveis com as expectativas de resposta dos utilizadores em aplicações interativas [27, 34]. Em alternativa, a abordagem assíncrona permite que a aplicação prossiga sem aguardar pela confirmação da transação, assumindo um modelo de consistência eventual. Esta opção melhora significativamente a experiência de utilização, mas introduz a necessidade de gerir explicitamente situações de desfasamento entre o estado local da aplicação e o estado efetivamente registado na blockchain, incluindo cenários de falha, atrasos prolongados ou a necessidade de efetuar *rollback* [27]. Assim, a escolha entre uma abordagem síncrona ou assíncrona representa um compromisso entre consistência, desempenho e experiência de utilização, devendo ser cuidadosamente avaliada em função dos requisitos e do contexto da aplicação.

No contexto do BioComp, foi adotada uma estratégia de sincronização assíncrona entre a aplicação cliente e a blockchain, onde o fluxo da aplicação e a interação do utilizador não são bloqueados enquanto as transações são processadas no *ledger*. A aplicação prossegue a sua execução normal, permitindo a realização de outras interações enquanto aguarda a confirmação das operações submetidas. Esta opção evita períodos de inatividade perceptíveis na interface. Apesar de a sincronização ser assíncrona, a forma como os efeitos das operações são refletidos na interface varia consoante o tipo de operação. Assim, em vez de aplicar uma política uniforme de atualização do estado visível, optou-se por diferenciar a estratégia de interação da interface em função do impacto semântico das operações, da sua frequência e das consequências associadas a eventuais inconsistências temporárias. Esta decisão, que atua sobre a QI2, resulta numa abordagem híbrida ao nível da interface, combinando estratégias pessimistas e otimistas conforme o contexto.

Nas operações de criação foi adotada uma estratégia de interface pessimista. Neste caso, a criação de uma nova entidade só é refletida na interface após a confirmação explícita da transação pela blockchain. Esta opção evita a apresentação temporária de entidades que nunca chegam a ser persistidas no *ledger*, cenário que poderia ocorrer caso a transação fosse rejeitada ou falhasse, dando origem a objetos inconsistentes ou "fantasma" do ponto de vista do utilizador. Uma lógica semelhante foi aplicada às operações de remoção. Dado o seu carácter destrutivo, mesmo utilizando uma abordagem de *soft delete*, e o impacto direto na perceção de fiabilidade do sistema, a eliminação de entidades apenas é refletida na interface após a confirmação da operação pela blockchain. Esta abordagem previne situações em que entidades aparentemente removidas reapareçam devido

à rejeição ou atraso da transação, o que poderia comprometer a confiança do utilizador na aplicação. Em contraste, para as operações de atualização foi adotada uma estratégia de interface otimista. Neste caso, as alterações são aplicadas imediatamente à instância em memória e refletidas de forma instantânea na interface, enquanto a atualização correspondente é submetida à blockchain em segundo plano. Esta decisão está diretamente alinhada com o mecanismo de *caching* em memória descrito na secção anterior. A principal implicação desta abordagem reside na possibilidade de existir uma divergência temporária entre o estado apresentado na interface e o estado efetivamente confirmado na blockchain. Em particular, durante o intervalo entre a submissão de uma operação de atualização e a sua confirmação no *ledger*, o utilizador pode interagir com um estado que ainda não se encontra definitivamente persistido. Em caso de falha ou conflito, o impacto limita-se à eventual perda ou *overwrite* de alterações recentes, não ocorrendo situações mais graves como o aparecimento ou desaparecimento inesperado de entidades. Adicionalmente, as operações de atualização correspondem às interações mais frequentes no sistema, tornando a melhoria da fluidez da experiência de utilização um fator determinante na decisão adotada.

Em sistemas que recorrem a interfaces otimistas, é comum complementar este modelo com mecanismos visuais que comuniquem explicitamente o estado das operações, como indicadores de progresso (*spinners*). No entanto, a implementação deste tipo de *feedback* pressupõe a existência de um modelo explícito de gestão do estado das transações no lado do cliente, capaz de representar e acompanhar o ciclo de vida das operações submetidas à blockchain. No contexto do BioComp, tal funcionalidade não foi implementada, uma vez que implicaria a introdução de uma camada adicional de complexidade arquitetural, nomeadamente a gestão explícita de transações no frontend, com mecanismos para correlação entre ações do utilizador e operações *on-chain*. Uma abordagem deste tipo aproximar-se-ia de padrões como o *Unit of Work*, cuja integração num sistema baseado em REST e blockchain levanta desafios significativos, particularmente no que diz respeito à garantia de propriedades transacionais como atomicidade e isolamento em cenários multiutilizador. Considerando os requisitos atuais do sistema e o equilíbrio entre complexidade, robustez e experiência de utilização, optou-se por uma solução mais simples, na qual a interface assume implicitamente o sucesso das operações de atualização, delegando a resolução de eventuais falhas ou conflitos para os mecanismos descritos na subsecção seguinte. Deste modo, o BioComp combina uma sincronização assíncrona com a blockchain com uma estratégia de gestão do estado visível diferenciada ao nível da interface, privilegiando abordagens pessimistas para operações estruturalmente críticas (*insert* e *delete*) e uma abordagem otimista para operações de *update*. Esta solução reflete um compromisso consciente entre consistência, simplicidade

arquitetural e experiência de utilização, adequado ao contexto colaborativo e aos requisitos funcionais do sistema.

4.8.2 Resolução de conflitos

A resolução de conflitos foi outro dos problemas que surgiu no desenvolvimento do BioComp e que se considerou relevante para este trabalho, uma vez que múltiplos clientes podem efetuar operações concorrentes sobre o mesmo conjunto de dados, de forma assíncrona, aumentando a probabilidade ocorrer um conflito. Neste trabalho, considera-se a existência de um conflito quando uma operação de atualização, ou eliminação, é submetida com base numa versão de uma instância que já não corresponde ao estado mais recente registado na blockchain, o que ocorre frequentemente quando vários utilizadores efetuam operações simultâneamente sobre a mesma instância. Os ambientes descentralizados introduzem algumas particularidades, o que causa uma maior probabilidade de ocorrerem estas situações, visto que não existe um controlo centralizado do estado e a latência inerente à comunicação com a blockchain pode levar a que diferentes utilizadores operem sobre estados divergentes do modelo. A literatura sobre sincronização de dados em sistemas distribuídos apresenta diversas estratégias para lidar com este problema. Em particular, Faiz et al. [52] destaca as seguintes abordagens como as mais comuns e recomendadas pela literatura para a resolução de conflitos:

- *Originator Wins*: utilizar os dados do originador da modificação;
- *Recipient Wins*: utilizar os dados do recetor da modificação;
- *Client Wins*: utilizar os dados do cliente;
- *Server Wins*: utilizar os dados do servidor;
- *Most Recent Wins*: utilizar os dados que foram modificados mais recentemente;
- *Duplication Apply*: criar uma cópia do item de dados para aplicar a modificação, mantendo a versão original intacta.

A literatura não aponta uma estratégia universalmente superior, sendo a escolha fortemente dependente do contexto da aplicação, dos requisitos de consistência e das limitações impostas pela arquitetura subjacente.

As estratégias *Originator Wins* e *Client Wins* foram analisadas em conjunto, uma vez que, no contexto do BioComp, o cliente corresponde diretamente ao originador da operação. Ambas privilegiam a intenção expressa pelo cliente no momento da submissão da operação, independentemente do estado local se encontrar desatualizado face ao *ledger*. Esta característica revelou-se problemática em cenários de concorrência. Por

exemplo, numa situação em que dois utilizadores operam simultaneamente sobre a mesma instância: se o utilizador A submeter uma operação de atualização, que é entretanto confirmada na blockchain, enquanto que o utilizador B, ainda com base no estado anterior, submeter uma operação de eliminação, utilizando uma estratégia do tipo *Originator/Client Wins*, a operação de eliminação iria prevalecer, removendo uma instância que já havia sido legitimamente atualizada. Este comportamento foi considerado indesejável, sobretudo pelo seu carácter destrutivo e pela dificuldade em mitigar os efeitos resultantes, uma vez que permitiria que operações baseadas em estados obsoletos anulassem alterações mais recentes já registadas no *ledger*. Por este motivo, estas estratégias foram descartadas.

De forma semelhante, as estratégias *Recipient Wins* e *Server Wins* revelaram-se inadequadas para o contexto do projeto, dado que o recetor da modificação corresponde à própria blockchain, pelo que ambas as estratégias se traduzem, na prática, na priorização sistemática do estado já registado no *ledger* em caso de conflito. Esta abordagem garante consistência forte, à custa do descarte integral das alterações submetidas pelo cliente, independentemente da sua natureza ou impacto semântico. Em cenários de atualizações concorrentes, esta estratégia pode conduzir à perda de trabalho legítimo, mesmo quando as modificações incidam sobre campos distintos ou sejam potencialmente compatíveis. Por exemplo, alterações independentes a atributos diferentes de uma mesma instância (como o nome próprio e o apelido) seriam tratadas como mutuamente exclusivas, resultando na rejeição total de uma das operações. Este comportamento foi considerado excessivamente restritivo e desalinhado com o objetivo de preservar, sempre que possível, a intenção do utilizador. Assim, estas estratégias foram igualmente descartadas.

A estratégia *Duplication Apply*, que consiste na criação de uma nova versão da entidade sempre que ocorre um conflito, foi igualmente descartada. Apesar de permitir preservar todas as alterações, esta abordagem implicaria um aumento significativo do volume de dados armazenados, com impactos negativos ao nível da escalabilidade, do desempenho e dos custos de armazenamento. Esta opção contraria diretamente as recomendações da literatura no que respeita à utilização moderada da blockchain.

Por isso, no BioComp a resolução de conflitos (QI2) baseia-se numa estratégia híbrida suportada por um mecanismo de *versioning* otimista das instâncias do modelo. Como referido anteriormente, cada instância possui um atributo de versão, gerido e incrementado pela blockchain a cada operação aceite, permitindo detetar situações em que uma modificação é submetida com base num estado desatualizado. As operações de atualização e eliminação (visto que os *deletes* são no fundo um update do atributo *zombie*) são submetidas à blockchain contendo apenas os atributos efetivamente alterados, juntamente com a versão da instância

conhecida pelo cliente no momento da modificação. Esta decisão permite uma resolução de conflitos mais fina, ao nível dos campos modificados, evitando a rejeição sistemática de operações concorrentes quando estas não são semanticamente incompatíveis. Em cenários em que dois utilizadores efetuam atualizações concorrentes sobre o mesmo atributo de uma instância, a blockchain deteta o desfasamento de versões e aplica uma estratégia do tipo *Most Recent Wins*, considerando válida a operação processada mais recentemente. Neste caso, a instância é atualizada com o novo valor e a versão é incrementada, refletindo a ordem total imposta pela blockchain na execução das transações. Em contraste, quando as atualizações concorrentes incidem sobre atributos distintos da mesma instância, apesar da existência de um conflito ao nível da versão, as alterações são consideradas compatíveis. Como apenas são submetidos os campos modificados, a blockchain consegue integrar ambas as modificações, preservando as alterações previamente aceites e aplicando as novas, resultando igualmente num incremento da versão da instância.

No entanto, as operações de eliminação (*delete*) recebem um tratamento diferenciado devido ao seu carácter destrutivo. Quando um *delete* é submetido com base numa versão desatualizada da instância, que foi entretanto modificada por outro utilizador, a operação de eliminação é descartada. Esta decisão evita que uma intenção destrutiva, baseada num estado obsoleto, prevaleça sobre alterações já confirmadas no *ledger*. De forma análoga, quando uma operação de eliminação é aceite antes de uma atualização concorrente, esta última é rejeitada, uma vez que a instância deixa de existir do ponto de vista do sistema.

Embora se considerem cenários concorrentes, assume-se que, devido aos mecanismos de validação e ordenação da blockchain, as operações são sempre processadas por ordem de chegada, ainda que com diferenças temporais mínimas, pelo que nunca ocorrerão situações em que chegam duas operações exatamente ao mesmo tempo. Esta propriedade permite resolver conflitos de forma determinística, sem necessidade de coordenação adicional entre clientes.

Desta forma, o BioComp adota uma abordagem de resolução de conflitos que combina estratégias clássicas da literatura, como *Most Recent Wins*, com uma integração seletiva de alterações compatíveis e uma priorização semântica das operações destrutivas. Esta solução reflete um compromisso entre consistência, preservação da intenção do utilizador e simplicidade arquitetural, adequado ao contexto colaborativo e descentralizado da aplicação.

4.9 Restantes cenários de qualidade

Esta secção descreve o desenvolvimento realizado para suportar cenários de qualidade que não foram tratados através da abordagem aos problemas centrais desta dissertação, como os cenários de desempenho. Em particular, são abordados os mecanismos implementados para satisfazer os cenários de qualidade definidos na Secção 4.4, que embora não constituam o foco principal do trabalho são essenciais para garantir a robustez e a usabilidade do sistema em condições reais de operação.

4.9.1 Disponibilidade

Ao contrário dos cenários associados a desempenho e escalabilidade, os cenários de disponibilidade são suportados através de componentes transversais e mecanismos de monitorização, deteção de falhas e degradação controlada. Esta secção detalha esses mecanismos e explica de que forma contribuem para satisfazer os cenários QA6–QA10.

Relativamente ao cenário QA6, associado à disponibilidade durante a manutenção ou indisponibilidade parcial de nós da rede blockchain, este é garantido essencialmente pelos próprios mecanismos de consenso e tolerância a falhas do Hyperledger Fabric. A arquitetura da blockchain permite que a rede continue operacional desde que exista um número suficiente de nós ativos para manter o consenso, assegurando a disponibilidade das operações de leitura e escrita mesmo durante a manutenção de nós individuais. Quanto aos restantes cenários de disponibilidade, conforme referido na Secção 4.5, o componente Health Monitor foi concebido para os suportar, através da aplicação das táticas *Monitor* e *Ping/Echo*. Este componente monitoriza dois níveis distintos de conectividade. Em primeiro lugar, recorre aos eventos disponibilizados pelo navegador para detetar a disponibilidade da ligação à internet. Em segundo lugar, verifica explicitamente a disponibilidade do backend através de um pedido GET periódico ao *endpoint* `/health` da REST API. Este endpoint apenas responde com sucesso quando a API se encontra operacional e consegue comunicar corretamente com a rede blockchain, funcionando assim como um mecanismo indireto de verificação da saúde do *ledger*. O pedido de verificação de estado é executado com um intervalo base de 15 segundos, valor que foi considerado adequado para o contexto do BioComp, uma vez que permite detetar falhas num intervalo de tempo reduzido sem introduzir carga excessiva sobre a REST API ou a blockchain. Este intervalo garante que os cenários QA7 e QA8 são satisfeitos dentro do limite temporal definido, ao mesmo tempo que evita verificações excessivamente agressivas.

A realização periódica de pedidos de *health check* torna-se redundante quando a aplicação está a efetuar operações regulares sobre o backend, tais como pedidos de leitura ou escrita. Nesses casos, a própria atividade normal da aplicação já constitui uma evidência implícita de que a REST API se encontra disponível. Para evitar *pings* desnecessários, foi introduzido um HTTP Interceptor que deteta a existência de atividade normal da aplicação. Este Interceptor, que consta na Figura 31, observa todas as respostas HTTP bem-sucedidas associadas a pedidos funcionais (excluindo explicitamente os pedidos de *health check*) e notifica o Health Monitor sempre que ocorre uma interação válida com o backend.

Figura 31: HTTP Interceptor que deteta atividade com o backend

Sempre que essa notificação ocorre, o Health Monitor adia a execução do próximo ciclo de verificação, garantindo que o *ping* periódico apenas é efetuado em períodos de inatividade. Esta abordagem reduz o tráfego desnecessário, melhora a eficiência do sistema e evita duplicação de mecanismos de monitorização, mantendo intacta a capacidade de deteção de falhas. Adicionalmente, o Health Monitor aplica uma estratégia de *Exponential Backoff* sempre que uma verificação de estado falha. Quando um *ping* não é concluído com sucesso, o intervalo até à próxima tentativa é duplicado relativamente ao valor base, o que evita insistir repetidamente em pedidos durante períodos de indisponibilidade prolongada, reduzindo a carga sobre o backend e prevenindo comportamentos agressivos em cenários de falha.

Quando uma falha é detetada, o Health Monitor não toma decisões diretamente sobre o comportamento da aplicação. Em vez disso, delega essa responsabilidade ao Error Handler, que centraliza o tratamento de erros e comunica o estado do sistema ao componente App Status, um componente criado para ajudar nestes cenários de qualidade.

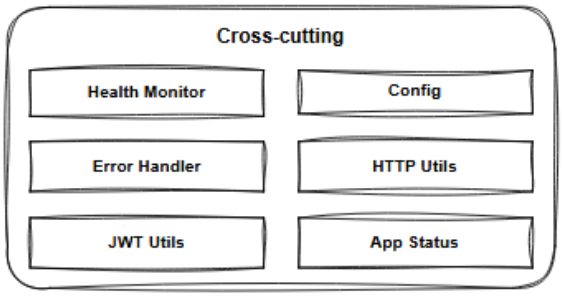

Figura 32: Camada Cross Cutting após o tratamento dos cenários de disponibilidade

O App Status atua como um serviço de estado centralizado, responsável por ativar e desativar o modo degradado da aplicação. Quando o sistema entra em modo degradado são bloqueadas operações que dependem da conectividade com o backend, evitando exceções repetidas e prevenindo erros adicionais. Simultaneamente, o utilizador é informado sobre o estado do sistema. É possível observar este estado na Figura 33.

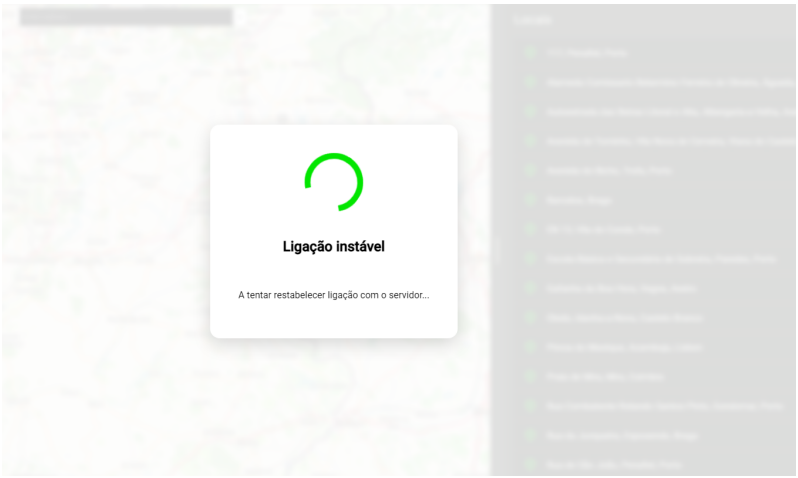

Figura 33: Modo degradado da aplicação

Esta abordagem concretiza a aplicação das táticas *Graceful Degradation* e *Exception Prevention*, permitindo que o sistema continue a operar em cenários de indisponibilidade temporária, satisfazendo os cenários QA7–QA10.

O estado do sistema é mantido de forma reativa através de *signals*, que representam, entre outros aspetos, a disponibilidade da ligação à internet e o instante da última atividade registada. Esta abordagem permite que o Health Monitor reaja automaticamente a alterações no estado do sistema, como transições entre estados *online* e *offline*, sem necessidade de lógica imperativa complexa. Desta forma, sempre que ocorre uma transição

relevante, como a perda ou recuperação da conectividade, o Health Monitor desencadeia as ações apropriadas, delegando a gestão do erro e da recuperação nos componentes responsáveis.

```
private async monitorCycle(intervalMs: number) {

    const elapsed = Date.now() - this.lastActivity();

    if (elapsed < intervalMs) { //Verifica se houve atividade entre pings
        const remaining = intervalMs - elapsed;
        setTimeout(() => this.monitorCycle(intervalMs), remaining);
        return;
    }

    try {
        await this.checkAPI();
        // Espera o cooldown normal após o sucesso
        setTimeout(() => this.monitorCycle(this.pingCooldown), this.pingCooldown);
    } catch (err) {
        // Se falhou, aplica o backoff
        setTimeout(() => this.monitorCycle(this.pingCooldown * 2), this.pingCooldown * 2);
    }
}
```

Figura 34: Implementação do Health Monitor

4.9.2 Segurança

Os cenários de segurança definidos (QA11–QA13) são maioritariamente suportados por mecanismos infraestruturais e padrões consolidados de segurança, não exigindo a introdução de componentes arquiteturais dedicados na aplicação. Alguns destes cenários não foram tratados nos problemas centrais desta dissertação, sendo no entanto assegurados através da configuração e integração adequada dos mecanismos existentes.

Os cenários QA11 e QA13, relacionados com a prevenção de acessos não autorizados, autenticação de utilizadores e gestão do ciclo de vida das sessões, são tratados de forma detalhada no subproblema Registo, autenticação e gestão de utilizadores (Secção 4.6), onde se descrevem os mecanismos de identificação, autenticação, autorização e revogação de acessos adotados pelo sistema. Relativamente ao cenário QA12, associado à proteção da comunicação contra ataques do tipo *Man-in-the-Middle* (MITM), este é garantido através da utilização sistemática do protocolo HTTPS em todas as comunicações entre o cliente e a REST API. Esta opção concretiza a aplicação da tática *Encrypt Data*, assegurando a confidencialidade e integridade dos dados transmitidos sem necessidade de implementar mecanismos de cifragem ao nível da lógica aplicacional.

Adicionalmente, a persistência dos dados na blockchain beneficia dos mecanismos de segurança inerentes à plataforma Hyperledger Fabric, incluindo controlo de acesso, imutabilidade e validação distribuída das transações. Estes mecanismos são fornecidos pela infraestrutura da blockchain e não são geridos diretamente pela aplicação, permitindo que a camada aplicacional se mantenha focada na lógica de negócio, enquanto herda propriedades de segurança robustas ao nível da persistência dos dados.

4.9.3 Escalabilidade

Os cenários de escalabilidade (QA14–QA15) representam situações de carga elevada e acesso concorrente ao sistema, aproximando-se em termos conceptuais dos cenários de desempenho, mas diferindo no volume e simultaneidade das operações. Ao contrário dos cenários de desempenho, estes cenários não são suportados por mecanismos arquiteturais dedicados no BioComp, não tendo sido alvo de decisões de desenho específicas ao nível da arquitetura, como *load balancing* ao nível da aplicação ou replicação de leituras. A escalabilidade é, assim, parcialmente delegada à infraestrutura subjacente, nomeadamente aos mecanismos próprios da rede blockchain. No caso do Hyperledger Fabric, a seleção de nós, a distribuição de pedidos entre *peers* e a tolerância a falhas são geridas ao nível da rede, permitindo um grau limitado de balanceamento de carga e continuidade do serviço sem intervenção direta da aplicação cliente. Ainda assim, estes mecanismos são transparentes para a aplicação e não eliminam os custos associados à validação, ordenação e confirmação de transações, que tendem a tornar-se significativos em cenários de elevada concorrência. Consequentemente, a escalabilidade do sistema depende sobretudo das propriedades emergentes da arquitetura proposta e dos mecanismos de otimização introduzidos para resolver os problemas centrais desta dissertação, nomeadamente ao nível da gestão de dados, sincronização e redução de tráfego desnecessário.

5 Resultados e discussão

Este capítulo apresenta os resultados da avaliação das soluções e mecanismos desenvolvidos ao longo deste trabalho, bem como a sua discussão à luz dos objetivos definidos e dos atributos de qualidade identificados. A avaliação tem como foco principal analisar o impacto das decisões arquiteturais adotadas no comportamento observável do sistema e no desempenho da aplicação em cenários representativos de utilização.

A avaliação incide sobre os seguintes aspetos fundamentais: (i) os mecanismos de obtenção de dados (*fetching*); (ii) o impacto do mecanismo de *caching* persistente do metamodelo no carregamento inicial da aplicação; (iii) o impacto do mecanismo de *caching* em memória baseado no padrão *Identity Map*; (iv) a abordagem de *partial updates* adotada nas operações de escrita, comparativamente ao envio do estado completo das entidades e (v) o mecanismo de resolução de conflitos implementado para lidar com atualizações concorrentes sobre os mesmos dados. Embora o sistema implemente diferentes estratégias de sincronização entre cliente e blockchain, incluindo abordagens de atualização pessimista ou otimista da interface, estas não foram objeto de avaliação quantitativa direta. Tal decisão deve-se ao facto de os seus impactos se refletirem sobretudo ao nível da perceção do utilizador e da experiência de utilização, aspetos que são intrinsecamente subjetivos e difíceis de medir de forma objetiva sem efetuar testes com os utilizadores-alvo do sistema. Ainda assim, estes mecanismos são indiretamente avaliados através das métricas de latência, que influenciam diretamente a *perceived performance* da aplicação.

5.1 Metodologia de avaliação

A avaliação foi conduzida através da definição de um conjunto de tarefas experimentais inspiradas em casos de uso reais do BioComp, de forma a garantir que os resultados obtidos refletem comportamentos representativos do sistema em condições de utilização plausíveis. Estas tarefas cobrem cenários de leitura intensiva, escrita pontual, acesso concorrente e situações de stress, permitindo analisar o impacto das decisões arquiteturais em diferentes contextos de carga. Sempre que aplicável, cada tarefa foi executada 15 vezes de forma independente, com o objetivo de mitigar o efeito da variabilidade da infraestrutura subjacente e das flutuações temporais do sistema. A repetição dos testes permitiu obter valores médios mais fiáveis e reduzir a influência de execuções atípicas, contribuindo para uma análise sólida dos resultados.

5.2 Tarefas experimentais

Com base nos casos de uso do sistema e nos mecanismos a avaliar, foram definidas as seguintes tarefas experimentais:

- T1 – Aceder ao painel de administração (QI1);
- T2 – Aceder à página de Transformação (*Transform*) (QI1);
- T3 – 50 utilizadores acedem simultaneamente ao painel de administração (QI1);
- T4 – 50 utilizadores acedem simultaneamente à página de Transformação (QI1);
- T5 – Atualizar o nome de uma pilha de compostagem (QI1);
- T6 – Efetuar o login no sistema (QI1);
- T7 – Dois utilizadores atualizam um local simultaneamente (QI2);

A tarefa T1 incide sobre o painel de administração, cujo acesso implica a obtenção de um conjunto razoável de utilizadores, sendo que para cada um deles também é necessário obter a organização, os papéis (*roles*) e a *Person*, uma instância que contém os seus dados pessoais. Este cenário é particularmente adequado para avaliar os mecanismos de *fetching* e população de entidades adotados, bem como o mecanismo de *caching* em memória. A tarefa T2 foca-se na página de Transformação do sistema, responsável pela visualização das pilhas de compostagem. Cada pilha está associada a um grande volume de medições históricas, o que torna este cenário relevante para avaliar o impacto das estratégias de obtenção de dados na presença de conjuntos de dados volumosos. As tarefas T3 e T4 correspondem a cenários de carga, nos quais 50 utilizadores acedem simultaneamente às funcionalidades avaliadas nas tarefas T1 e T2, respetivamente. Estes cenários foram definidos com o objetivo de analisar o comportamento do sistema sob acesso concorrente, permitindo observar o impacto das decisões arquiteturais em termos de latência e robustez em condições de maior pressão sobre a infraestrutura. A tarefa T5 foi concebida para avaliar o impacto da estratégia de *partial updates* adotada nas operações de atualização, comparando esta abordagem com o envio do estado completo da entidade em operações de atualização. A tarefa T6 corresponde ao processo de autenticação, durante o qual o metamodelo é carregado no cliente. Este cenário permite avaliar o impacto do mecanismo de *caching* do metamodelo e avaliar a sua contribuição na redução da latência associada a essa operação. Por fim, a tarefa T7 foi definida para analisar o comportamento do mecanismo de resolução de conflitos, simulando atualizações concorrentes por diferentes utilizadores sobre a mesma entidade. Ao contrário das restantes tarefas, este cenário não foi executado de forma repetida, uma vez que o objetivo principal consiste na validação do comportamento

funcional do mecanismo de resolução de conflitos, e não na obtenção de métricas estatísticas. A repetição exaustiva deste teste não traria benefícios adicionais em termos de análise quantitativa.

5.3 Procedimento experimental

Foi definido um procedimento experimental de forma a adaptar as métricas e os instrumentos de medição à natureza específica de cada mecanismo avaliado. Em vez de aplicar um conjunto fixo de métricas a todas as tarefas, foram selecionadas métricas adequadas a cada tipo de cenário, permitindo uma avaliação mais precisa e significativa dos impactos das decisões arquiteturais adotadas.

Para a recolha de métricas de latência e tempo de resposta em cenários de leitura e escrita com um único utilizador, recorreu-se às ferramentas de desenvolvimento do Google Chrome (Chrome Dev Tools), utilizando a sua aba *Network*. As métricas consideradas incluem o *Time to First Byte* (TTFB), utilizado como aproximação ao tempo de processamento na blockchain e na REST API, e o *Total Response Time* (*Total RT*), representativo da experiência final percebida pelo cliente. A avaliação do comportamento do sistema sob acesso concorrente foi realizada com recurso à ferramenta Grafana k6, que permitiu simular múltiplos utilizadores virtuais a executar as mesmas tarefas em simultâneo. Nestes cenários, foram analisadas métricas agregadas, nomeadamente o percentil $p95$ da duração das requisições e a taxa de erro, indicadores relevantes da estabilidade e robustez do sistema sob carga. Para avaliar a eficiência dos mecanismos de *caching*, foram utilizados contadores internos no Angular. Estas medições permitiram comparar cenários de *cold cache* (quando a cache está vazia) com os cenários de *warm cache* (quando a cache já possui dados), avaliando o impacto do *caching* no tempo de carregamento e processamento dos dados. Importa salientar que a avaliação dos mecanismos de *caching* não teve como objetivo validar a sua correção ou utilidade funcional, aspetos que são assumidos dado o seu comportamento esperado, mas sim quantificar o impacto efetivo dessas decisões arquiteturais no desempenho observável do sistema, nomeadamente em termos de redução de latência e estabilidade das respostas. Por fim, a avaliação do mecanismo de resolução de conflitos incidiu sobre a validação do seu comportamento funcional em cenários de atualizações concorrentes, analisando-se a taxa de sucesso das operações e a consistência do estado final do sistema.

5.4 Avaliação experimental

5.4.1 T1 – Acesso ao painel de administração

Como referido anteriormente, esta tarefa foi utilizada para testar diferentes mecanismos, começando pelos de *fetching*. Na abordagem baseada em *Eager Loading*, todas as entidades necessárias foram obtidas através

de um único pedido agregado, enquanto que na abordagem de *Explicit Loading* as relações foram resolvidas de forma incremental, resultando num total de 37 pedidos: um pedido para obter todos os utilizadores e, para cada um desses utilizadores, um pedido para obter a sua *Person*, outro para a sua organização e outro para o seu conjunto papéis. Na Tabela 2 são apresentados os valores médios obtidos para os parâmetros medidos.

Tabela 2: Resultados médios da tarefa T1 (15 execuções por estratégia)

Métrica	**Eager Loading**	**Explicit Loading**
Request sent (ms)	0,11	3,46
TTFB (s)	1,93	1,25
Content download (ms)	2,42	14,53
Total RT (s)	2,93	3,85

No caso da estratégia de *Explicit Loading*, as métricas apresentadas correspondem ao agregado dos vários pedidos efetuados em cada execução da tarefa. Para esse efeito, em cada uma das 15 execuções foi exportado o ficheiro *HAR* gerado pelo Chrome Dev Tools, a partir do qual foram extraídos e somados os valores de latência e tempo de transferência de todos os pedidos associados à operação. Esta abordagem permitiu obter métricas globais por execução, evitando uma análise manual pedido a pedido, que seria impraticável neste contexto.

Também foi analisada a distribuição dos resultados de forma a avaliar a estabilidade de cada abordagem. Observou-se que o *Total RT* no *Eager Loading* apresentou uma mediana de 2,93 s, valor que corresponde à média registada, o que reforça a consistência dos resultados. No entanto, verificou-se uma amplitude considerável nesta métrica, com valores compreendidos entre 1,15 e 4,33 s, variação que pode ser atribuída a flutuações na latência da rede e no tempo de resposta variável dos nós da blockchain durante as iterações. No caso do *Explicit Loading*, a mediana de 3,87 s corrobora a tendência de maior latência imposta pelo elevado número de pedidos sequenciais, com uma amplitude entre 3,71 e 3,97 s.

Para além da comparação entre as estratégias de *Eager Loading* e *Explicit Loading*, a tarefa T1 foi também utilizada para avaliar o impacto do mecanismo de *caching* em memória baseado no padrão *Identity Map*. Esta tarefa revelou-se adequada para esse efeito, uma vez que envolve um número significativo de entidades e relações a resolver, permitindo observar claramente os benefícios da reutilização de instâncias já carregadas em memória. De forma a isolar o impacto do *caching* em memória, o mecanismo de *fetching* foi mantido fixo, recorrendo-se exclusivamente à estratégia de *Eager Loading*. Esta decisão evita interferências entre os dois mecanismos, permitindo avaliar o efeito do *Identity Map* de forma independente. As medições tiveram

de ser realizadas ao nível da aplicação Angular, uma vez que o mecanismo de *caching* opera em memória e envolve etapas adicionais de processamento no cliente, não sendo possível captar estes tempos apenas através das ferramentas de inspeção de rede. Para este efeito, foram instrumentados contadores internos no método responsável pelo carregamento dos utilizadores, medindo o intervalo temporal desde o início da preparação da consulta até ao momento em que os dados se encontram totalmente processados e prontos a ser apresentados na interface. O cenário de *cold cache* foi obtido através da limpeza completa do estado da aplicação antes do acesso à página, enquanto que o cenário de *warm cache* corresponde a acessos subsequentes, nos quais as entidades já se encontravam disponíveis no *Identity Map*. A Tabela 3 apresenta os valores médios e medianos obtidos para ambos os cenários.

Tabela 3: Impacto do *caching* em memória na tarefa T1

Métrica	***Cold Cache***	***Warm Cache***
Média (s)	6,73	2,02
Mediana (s)	7,42	2,01

A análise da distribuição dos resultados evidencia, obviamente, uma redução substancial no tempo de carregamento quando o *Identity Map* se encontra ativo. No cenário de *cold cache*, os tempos observados variaram entre 3,79 e 8,01 s, enquanto que no cenário de *warm cache* a amplitude foi significativamente menor, situando-se entre 1,95 e 2,05 s. Esta redução da variabilidade sugere um comportamento mais estável do sistema quando os dados já se encontram disponíveis em memória.

5.4.2 T2 – Acesso à página de Transformação

Na tarefa T2, o acesso à página de Transformação implica a obtenção das pilhas de compostagem existentes e dos respetivos dados associados. Neste cenário, a estratégia de *Explicit Loading* resultou num total de quatro pedidos: um pedido inicial para obter as pilhas e, para cada uma das três pilhas, um pedido adicional para obter as ações associadas, incluindo as medições. Em contraste, a abordagem baseada em *Eager Loading* agrega estes dados num conjunto mais reduzido de pedidos, resultando num maior volume de dados transferidos por resposta. Os resultados médios obtidos para ambas as estratégias encontram-se sumarizados na Tabela 4.

Tabela 4: Resultados médios da tarefa T2

Métrica	***Eager Loading***	***Explicit Loading***
Request sent (ms)	0,13	0,63
TTFB (s)	12,16	10,16
Content download (ms)	661,56	1256,25
Payload size (KB)	3700	3820
Total RT (s)	13,47	11,68

O *Eager Loading* apresentou uma amplitude de 4,67 s (entre 11,96 e 16,63 s) e uma mediana de 12,92 s, enquanto que o *Explicit Loading* manteve os resultados num intervalo mais restrito de 2,16 s e uma mediana bastante próxima da média (11,47 s). À semelhança da tarefa T1, observa-se um compromisso entre o número de pedidos efetuados e o volume de dados transferidos, com impactos distintos no *Time to First Byte* e no tempo total de resposta.

5.4.3 T3 – 50 utilizadores acedem simultaneamente ao painel de administração

A utilização da ferramenta Grafana k6 nestes testes de carga permitiu simular utilizadores virtuais (*Virtual Users*, VUs) a executar os mesmos pedidos de forma concorrente. Os testes foram configurados com um executor do tipo `per-vu-iterations`, garantindo que cada utilizador virtual executa um número fixo de iterações. Embora o arranque dos utilizadores seja simultâneo, é expectável a existência de pequenas variações temporais entre as execuções, inerentes ao modelo de simulação e ao agendamento dos pedidos.

Nos testes realizados com a estratégia de *Eager Loading*, foi possível executar o cenário completo com os 50 utilizadores simultâneos, não se tendo observado falhas ou erros de comunicação. As métricas recolhidas incluem o *http_req_duration*, considerado neste contexto como equivalente ao *Total RT*, bem como o percentil $p95$, que indica o tempo de resposta máximo observado para 95% dos pedidos (ou seja, o tempo que 95% dos utilizadores teve de esperar). Em contraste, a execução do mesmo cenário com a estratégia de *Explicit Loading* revelou-se significativamente mais exigente do ponto de vista operacional, dado que esta abordagem implica a realização de múltiplos pedidos por utilizador, o que faz com que o número total de pedidos HTTP cresça rapidamente com o número de utilizadores simultâneos. Neste cenário com 50 utilizadores, a execução do teste tornou-se impraticável, tanto em termos de tempo de execução como de volume total de pedidos gerados. Por esta razão, a avaliação da estratégia de *Explicit Loading* foi realizada com um número reduzido de 10 utilizadores simultâneos, permitindo ainda assim comparar o comportamento relativo das duas abordagens

sob carga. Para garantir a comparabilidade dos resultados, a estratégia de *Eager Loading* foi também testada neste cenário com 10 utilizadores.

Tabela 5: Estratégias de *fetching* sob carga na tarefa T3 (10 utilizadores)

http_req_duration	***Eager Loading***	***Explicit Loading***
Média (s)	5,59	2,91
Mediana (s)	6,34	3,07
$p95$ (s)	7,70	3,81

No caso do *Eager Loading*, o *http_req_duration* apresentou valores mínimos de 1,12 e máximos de 7,78 s, refletindo uma variabilidade significativa sob carga. Já a estratégia de *Explicit Loading* revelou uma menor amplitude, com tempos mínimos de 558,77 ms e máximos de 4,11 s, indicando um comportamento mais estável no cenário avaliado com 10 utilizadores simultâneos.

5.4.4 T4 – 50 utilizadores acedem simultaneamente à página de Transformação

A execução desta tarefa revela-se particularmente exigente, uma vez que envolve a obtenção de grandes volumes de dados, nomeadamente medições históricas associadas às pilhas de compostagem. Estes testes utilizaram igualmente um executor do tipo `per-vu-iterations` no Grafana k6. Numa primeira fase, procurou-se executar o cenário com os 50 utilizadores simultâneos, de forma consistente com a definição inicial da tarefa. No entanto, verificou-se que, nesta configuração, a estratégia de *Eager Loading* apresentou um comportamento instável, com cerca de 98% dos pedidos a falharem, inviabilizando a recolha de métricas representativas. A redução do número de utilizadores para 20 permitiu a execução de uma única iteração, mas a repetição do teste ao longo de 15 iterações voltou a conduzir a falhas generalizadas. Por esta razão, a avaliação desta estratégia foi realizada com 10 utilizadores simultâneos, configuração que permitiu obter uma amostra estável de resultados. No caso da estratégia de *Explicit Loading*, o cenário com 50 utilizadores revelou-se inicialmente mais robusto, mantendo tempos de resposta inferiores a 18 segundos na primeira metade do teste, sendo que a partir desse ponto verificou-se igualmente uma degradação acentuada do desempenho, levando à sua interrupção. Por isso, e de forma a garantir a comparabilidade entre as estratégias, a avaliação do *Explicit Loading* foi igualmente realizada com 10 utilizadores simultâneos.

Tabela 6: Estratégias de *fetching* sob carga na tarefa T4 (10 utilizadores)

http_req_duration	***Eager Loading***	***Explicit Loading***
Média (s)	47,15	6,23
Mediana (s)	47,24	6,96
$p95$ (s)	57,16	8,45
Taxa de erro (%)	5,33	0,00

Na estratégia de *Eager Loading*, os tempos mínimos registados foram de 30,12 s, enquanto que os valores máximos atingiram 59,71 s, evidenciando uma elevada variabilidade e um comportamento instável sob carga. No caso do *Explicit Loading*, os tempos de resposta variaram entre 1,03 e 10,01 s, apresentando uma amplitude substancialmente inferior e uma maior previsibilidade no cenário avaliado. A taxa de erro observada no *Eager Loading*, correspondente a 8 pedidos falhados em 150, contrasta com a ausência de falhas na estratégia de *Explicit Loading*.

5.4.5 T5 – Atualizar o nome de uma pilha de compostagem

Esta foi definida com o objetivo de avaliar o impacto da abordagem de *partial updates* nas operações de escrita, comparativamente ao envio do estado completo da entidade, uma decisão arquitetural que se encontra alinhada com o princípio de utilização moderada da blockchain, procurando reduzir o volume de dados transmitidos e, potencialmente, o custo de processamento associado às operações de atualização. No contexto do domínio do sistema, as entidades disponíveis não apresentam, de forma geral, dimensões suficientemente elevadas para evidenciar diferenças significativas entre as duas abordagens. Ainda assim, de forma a maximizar o potencial impacto do teste, foi utilizada uma pilha de compostagem contendo um conjunto alargado de dados. Para este efeito, foi criada uma pilha de compostagem e importado um ficheiro contendo aproximadamente 1800 medições históricas. Esta tarefa compara o envio do estado completo da pilha, incluindo as referências a todas as medições associadas com o envio exclusivo do campo alterado (o nome da pilha). Os resultados obtidos encontram-se sumarizados na Tabela 7.

Tabela 7: Estratégias de *update* na tarefa T5

http_req_duration	***Partial Update***	***Full Update***
Média (s)	2,32	2,33
Mediana (s)	2,30	2,31
$p95$ (s)	2,45	2,42

No cenário de *full update*, observou-se um *http_req_duration* com valores mínimos e máximos de 2,29 s e 2,43 s, respetivamente. A abordagem baseada em *partial updates* apresentou um mínimo de 2,28 s e um máximo de 2,60 s.

5.4.6 T6 – Efetuar o login no sistema

Para avaliar este mecanismo de forma realista, optou-se por instrumentar contadores temporais ao nível da aplicação Angular, mais concretamente no componente Data Store, responsável por orquestrar o processo de inicialização do sistema. Esta opção permitiu medir o tempo associado aos pedidos efetuados à blockchain e à REST API, mas também o tempo adicional necessário para o processamento dos dados no cliente, incluindo a interpretação do metamodelo, a sua integração no estado da aplicação e a preparação das estruturas internas utilizadas pelo RxDB. A medição exclusiva dos pedidos de rede não captaria estes custos de forma adequada. Estas medições também foram realizadas considerando os cenários de *cold cache*, em que o metamodelo não se encontrava previamente armazenado e teve de ser integralmente carregado, e *warm cache*, em que o metamodelo já se encontrava disponível localmente. Em ambos os casos, foi medido o tempo desde o início do processo de carregamento do metamodelo até ao momento em que este se encontra totalmente carregado. A Tabela 8 apresenta os resultados obtidos para ambos os cenários.

Tabela 8: Impacto do *caching* do metamodelo no processo de login (tarefa T6)

Métrica	***Cold Cache***	***Warm Cache***
Média (s)	5,10	2,06
Mediana (s)	5,10	2,06
Mínimo (s)	4,93	1,94
Máximo (s)	5,41	2,13

5.4.7 T7 – Dois utilizadores atualizam um local simultaneamente

Esta tarefa teve como objetivo validar o comportamento funcional do mecanismo de resolução de conflitos implementado, de acordo com a política definida e descrita na Secção 4.8. Ao contrário das restantes tarefas, esta avaliação não teve como foco a recolha de métricas de desempenho, mas sim a verificação da correção e consistência do sistema em cenários de atualizações concorrentes sobre a mesma instância. Com base na política de resolução de conflitos adotada, foram definidos três cenários distintos a testar: (i) dois utilizadores a atualizar propriedades diferentes de uma instância; (ii) dois utilizadores a atualizar a mesma propriedade de uma instância; (iii) um utilizador a efetuar uma atualização enquanto outro tenta eliminar a instância com uma versão desatualizada. Embora estes cenários pudessem ser testados manualmente através de múltiplas

sessões de navegador, optou-se por utilizar novamente o Grafana k6, que permitiu definir *scripts* de teste reprodutíveis e controlar a ordem temporal das operações concorrentes. Em cada cenário um utilizador virtual executa uma operação sobre uma determinada instância e, após um pequeno intervalo de tempo, um segundo utilizador virtual executa uma operação distinta com base numa versão entretanto desatualizada dessa mesma instância. Cada cenário foi repetido cinco vezes de forma independente, apenas com o intuito de confirmar a consistência do comportamento observado, uma vez que repetições adicionais não trariam benefícios relevantes para a análise.

No primeiro cenário, dois utilizadores atualizaram propriedades diferentes da mesma instância: um utilizador alterou o nome de um local, enquanto que outro atualizou as coordenadas GPS desse local. Ambas as operações foram aceites com sucesso em todas as execuções, tendo o estado final da instância refletido corretamente ambas as modificações. Este comportamento confirma que o sistema permite atualizações concorrentes quando não existe conflito direto sobre os mesmos atributos. No segundo cenário, ambos os utilizadores atualizaram o nome da instância. O resultado observado foi consistente com a estratégia de *Most Recent Wins* implementada: a modificação mais recente foi aceite, a versão da instância foi incrementada, e a alteração anterior foi corretamente substituída. Este comportamento valida a política definida para conflitos diretos de escrita sobre o mesmo campo. Por fim, no terceiro cenário, um utilizador efetuou uma atualização válida sobre a instância, enquanto que um segundo utilizador tentou eliminá-la com base numa versão desatualizada. A operação de eliminação foi rejeitada em todas as execuções, resultando num erro consistente, enquanto que a instância permaneceu no estado atualizado. Este resultado confirma que o sistema impede operações destrutivas quando estas são baseadas em versões não atuais, garantindo a integridade e consistência do estado armazenado.

5.5 Análise e discussão dos resultados

A análise dos resultados obtidos permitiu extrair conclusões fundamentais sobre o impacto das decisões arquiteturais adotadas, sendo discutidos à luz dos objetivos definidos para o trabalho, das questões de investigação e dos atributos de qualidade identificados.

5.5.1 Estratégias de *fetching* e escalabilidade

Os resultados das tarefas T1 a T4 evidenciam um ponto de inversão claro na eficácia das estratégias de carregamento, dependente do volume de dados e do nível de concorrência. No cenário de leitura simples (T1), caracterizado por um volume moderado de dados, a estratégia de *Eager Loading* apresentou melhor

desempenho, com um *Total RT* cerca de 31,40% inferior ao observado com *Explicit Loading*. Neste contexto, a agregação das entidades num único pedido reduz a latência total, uma vez que o custo adicional de múltiplos pedidos sequenciais supera o custo de processamento do grafo de objetos no servidor. Contudo, quando o volume de dados aumenta de forma significativa, como na tarefa T2, onde as respostas atingem a ordem dos megabytes, esta tendência inverte-se. Nestes cenários, a estratégia de *Explicit Loading* revelou-se cerca de 13,29% mais eficiente, sugerindo que a decomposição do carregamento em pedidos mais granulares reduz o impacto do processamento de grandes grafos de objetos e melhora a capacidade de resposta do sistema. Este comportamento torna-se ainda mais evidente sob acesso concorrente. Nas tarefas de carga (T3 e T4), o *Explicit Loading* demonstrou maior robustez e estabilidade, mantendo valores de latência controlados e ausência de falhas mesmo com múltiplos utilizadores simultâneos. Em contraste, a estratégia de *Eager Loading* apresentou degradação significativa, culminando num *p95* de 57,16 s na tarefa T4 e numa taxa de erro de 5,33%. Estes resultados indicam que, embora o *Eager Loading* seja vantajoso em cenários simples e com baixa concorrência, o *Explicit Loading* constitui uma estratégia mais escalável, especialmente em sistemas orientados a leitura intensiva, com grandes volumes de dados e múltiplos utilizadores concorrentes.

Adicionalmente, uma reflexão relevante é a necessidade de existir um alinhamento entre as estratégias de carregamento na aplicação cliente e na blockchain. Embora este trabalho não explore diretamente a variação de estratégias de *fetching* na blockchain, é plausível especular que não faz sentido implementar *Eager Loading* no cliente se a blockchain subjacente mantém acesso explícito e granular aos dados, e vice-versa. Manter consistência entre os mecanismos de carregamento em ambos os níveis pode evitar redundâncias, reduzir latência e melhorar ainda mais a eficiência global do sistema, especialmente em cenários com grandes volumes de dados ou alta concorrência de acesso.

5.5.2 Mecanismos de *caching*

A avaliação dos mecanismos de *caching*, realizada através das tarefas T1 e T6, demonstra o impacto positivo da arquitetura orientada ao estado local na redução da latência e na estabilidade do sistema. O mecanismo de *caching* em memória baseado no padrão *Identity Map* permitiu reduzir o tempo de prontidão dos dados em cerca de 70%. Para além da redução do valor médio, o ganho mais relevante observa-se na previsibilidade do comportamento do sistema: no cenário de *warm cache*, a amplitude de resposta foi de 0,10 s, eliminando a variabilidade introduzida pela latência da rede, pelo acesso à blockchain e pelo processamento assíncrono do RxDB. De forma complementar, o mecanismo de *caching* persistente do metamodelo (T6) reduziu o tempo de inicialização da aplicação em aproximadamente 60%, passando de 5,10 s para 2,06 s. Este

resultado valida a decisão de armazenar localmente o esquema da aplicação, evitando acessos dispendiosos à blockchain durante o processo de autenticação, um momento crítico para a perceção de desempenho por parte do utilizador. Adicionalmente, e como referido anteriormente, o metamodelo caracteriza-se por uma natureza estruturalmente estável, sofrendo alterações apenas quando ocorrem mudanças significativas nos requisitos do sistema. Neste contexto, a sua persistência em RxDB revela-se particularmente eficaz, uma vez que maximiza a taxa de reutilização da informação e reduz de forma consistente a necessidade de importações subsequentes a partir da blockchain.

5.5.3 Eficiência nas operações de escrita

A tarefa T5 avaliou o impacto da abordagem de *partial updates* face ao envio do estado completo das entidades em operações de escrita. Os resultados revelam que, para o volume de dados e a infraestrutura testada, a adoção de *Partial Updates* não se traduziu numa melhoria significativa no tempo de resposta. A diferença entre as médias do *Partial Update* (2,32 s) e do *Full Update* (2,33 s) é inferior a 1%, sendo estatisticamente irrelevante. Mesmo com uma pilha de compostagem contendo 1800 medições, o tempo total de operação parece ser dominado quase exclusivamente pela latência de rede e pelo tempo de processamento/consenso da blockchain, tornando o tamanho do *payload* JSON num fator secundário. Embora a estratégia de *partial update* cumpra o objetivo de reduzir o tráfego desnecessário, os dados confirmam que em objetos JSON o custo de enviar algumas centenas de linhas adicionais é absorvido pelo *overhead* do protocolo HTTP. Ainda assim, pode-se concluir que esta estratégia justifica-se mais como uma boa prática de escalabilidade a longo prazo e economia de largura de banda do que como uma solução para reduzir a latência imediata percebida pelo utilizador.

5.5.4 Consistência e resolução de conflitos

Os testes funcionais realizados na tarefa T7 validaram a robustez do mecanismo de resolução de conflitos implementado, que se considerou o mais adequado e aceitável para o contexto do BioComp. Os resultados demonstram que o sistema é capaz de permitir edições concorrentes sobre campos distintos sem perda de informação, garantir consistência em conflitos diretos através da política de *Most Recent Wins* e impedir operações destrutivas baseadas em versões obsoletas. Estes comportamentos confirmam que o sistema assegura a integridade do estado armazenado mesmo em cenários de concorrência, alinhando-se com os requisitos de consistência definidos para o domínio da aplicação.

5.6 Validação dos cenários de qualidade

A validação dos cenários de qualidade definidos na fase de análise do problema é realizada à luz dos resultados obtidos na avaliação experimental e considerando o sistema configurado de acordo com as decisões arquiteturais que demonstraram melhor desempenho e robustez. Começando pelos cenários de desempenho, o cenário QA1 é considerado cumprido, dado que os testes de escrita indicam que a persistência das operações na blockchain ocorre consistentemente em menos de 10 s, enquanto que a confirmação da operação na interface, suportada por uma estratégia de interação pessimista, ocorre em menos de 3 s. O cenário QA2 também é cumprido: a atualização diretamente na instância em memória juntamente com a utilização de uma interface otimista permite que a alteração seja refletida de forma praticamente imediata na interface do utilizador, enquanto que a atualização *on-chain* ocorre dentro dos limites temporais definidos. Em casos excecionais de conflito, o mecanismo de resolução garante a consistência do estado, conforme validado na tarefa T7. Relativamente ao QA3, o tempo total do processo de login, incluindo a validação das credenciais e a eventual importação do metamodelo situa-se entre 6 e 7 s em condições normais, permanecendo abaixo do limite máximo de 10 s. Em cenários de *warm cache*, este tempo é reduzido para aproximadamente 2 s, o que significa que o sistema também suporta este cenário. O cumprimento do cenário QA4 depende do estado da cache. Quando os dados se encontram em memória, a apresentação dos gráficos e estatísticas ocorre dentro do limite de 3 s. Em situações onde é necessário recorrer à blockchain, o tempo de resposta pode exceder ligeiramente este valor, refletindo um compromisso aceitável entre frescura dos dados e desempenho. Por fim, o QA5 não é cumprido de forma consistente. Os testes sob carga (T4) demonstram que, embora o sistema consiga responder corretamente a 50 pedidos concorrentes numa fase inicial, a degradação do desempenho e o surgimento de falhas tornam-se evidentes à medida que a carga se prolonga. Este resultado evidencia os limites da arquitetura atual sob cenários de stress extremo e aponta para oportunidades de otimização futura.

Relativamente aos cenários de disponibilidade (QA6–QA10), a sua avaliação não foi realizada através de testes de carga ou medições temporais repetidas, mas sim por inspeção dos mecanismos arquiteturais implementados, conforme descrito na secção dedicada aos restantes cenários de qualidade (Secção 4.9), onde também se mostrou que estes são suportados pelo sistema. Esta abordagem é adequada, dado que estes cenários dependem essencialmente de capacidades de deteção, notificação e degradação controlada, e não de métricas de desempenho sensíveis a variações estatísticas.

Os cenários de segurança (QA11–QA13), também foram avaliados de forma semelhante aos cenários de disponibilidade na Secção 4.9, onde também se concluiu que o sistema os suporta.

Para finalizar, restam os cenários de escalabilidade (QA14 e QA15). Para o cenário QA14 foi realizado outro teste de carga com o *k6*, simulando a entrada progressiva de até 100 utilizadores num intervalo de tempo inferior a um minuto. Cada utilizador executa a sequência de consultas representativa do comportamento real da aplicação ao aceder à página dos locais, incluindo um pedido inicial para obtenção dos locais seguido de pedidos adicionais para carregamento explícito das entidades associadas. Os resultados obtidos demonstram que o sistema não consegue satisfazer este cenário de forma consistente. O tempo médio de resposta observado foi de aproximadamente 21,96 s, ultrapassando significativamente o limite de 10 s definido no cenário, com um valor de $p95$ de 30,26 s. Para além disso, registou-se uma taxa de falhas de 26,31%, correspondendo a 90 pedidos falhados num total de 342. Assim, o cenário QA14 é considerado como não cumprido, revelando limitações claras da arquitetura atual no que diz respeito à escalabilidade horizontal de operações de leitura. O cenário QA15, relativo à distribuição da carga por múltiplas REST APIs e organizações, não foi avaliado experimentalmente, uma vez que não se encontra implementado na versão atual do sistema. Atualmente, o BioComp utiliza uma única REST API e uma rede blockchain hospedada numa infraestrutura centralizada, o que limita a capacidade de distribuição de carga entre organizações. No entanto, este cenário foi considerado durante a definição da arquitetura. A utilização de uma rede Hyperledger Fabric com múltiplas organizações permite, em teoria, que cada entidade participante faça *host* do seu próprio nó da blockchain e uma REST API dedicada, possibilitando a distribuição da carga de leitura e escrita por diferentes infraestruturas. Esta abordagem implicaria, contudo, custos adicionais de manutenção e recursos computacionais por parte das organizações, bem como um maior esforço de coordenação operacional. Dado que o sistema se encontra ainda numa fase inicial de adoção e que os utilizadores alvo não estão neste momento distribuídos por todas as instituições do consórcio, a implementação deste cenário foi considerada fora do âmbito do presente trabalho. Assim, o QA15 é identificado como uma oportunidade clara de evolução futura, a ser explorada à medida que o sistema cresce em número de utilizadores e organizações participantes.

Importa ainda enquadrar a validação destes cenários à luz das prioridades assumidas durante o desenvolvimento do sistema. Embora todos os atributos de qualidade sejam relevantes, nem todos tiveram o mesmo peso nas decisões arquiteturais. O desempenho foi considerado o atributo mais crítico, uma vez que constitui uma das principais limitações percebidas em aplicações descentralizadas baseadas em blockchain e tem impacto direto na experiência do utilizador. Tempos de resposta elevados comprometem a usabilidade e a adoção do sistema, motivo pelo qual este atributo foi alvo de maior atenção ao longo do trabalho. A disponibilidade surge como o segundo atributo mais prioritário, dado que o sistema depende intrinsecamente de conectividade com a internet, da REST API e da rede blockchain para funcionar corretamente, não existindo um modo de

operação *offline*. Assim, garantir deteção rápida de falhas, degradação controlada e recuperação automática foi considerado essencial para manter um comportamento previsível do sistema. Por outro lado, os cenários de segurança, apesar de fundamentais, não foram o principal foco do trabalho, uma vez que se beneficia diretamente das propriedades de segurança inerentes à blockchain, do uso obrigatório de HTTPS, bem como do modelo de autenticação baseado em tokens e controlo de permissões por papéis. Estas decisões fornecem um nível de segurança considerado adequado para o contexto do sistema, permitindo priorizar outros atributos mais críticos para a experiência do utilizador. De forma semelhante, a escalabilidade não foi tratada como atributo prioritário nesta fase, uma vez que o sistema ainda se encontra numa fase inicial de adoção, sem requisitos claros quanto ao número esperado de utilizadores ou volume de operações concorrentes. Ainda assim, os resultados experimentais demonstram que a arquitetura atual apresenta limitações claras a partir de cargas moderadas, evidenciando oportunidades concretas de evolução futura.

A análise conjunta dos cenários de qualidade permite ainda identificar compromissos claros entre diferentes atributos, à semelhança do que é proposto em métodos orientados à tomada de decisão arquitetural, como o CBAM. Por exemplo, a opção arquitetural de utilizar uma única REST API e infraestrutura comum para a blockchain contribuiu para simplificar o desenvolvimento, reduzir custos operacionais e facilitar a manutenção, beneficiando o desempenho em cenários de carga baixa a moderada e a estabilidade global do sistema. No entanto, esta decisão teve impacto negativo na escalabilidade horizontal, como evidenciado nos testes de carga do cenário QA14. A distribuição da carga por múltiplas organizações e APIs, embora arquiteturalmente prevista, implicaria custos significativos adicionais ao nível de recursos computacionais, manutenção e coordenação entre entidades, o que não se justificou para o estado atual do sistema. De forma semelhante, a utilização de estratégias como *Explicit Loading* e mecanismos de *caching* em memória representa um compromisso entre desempenho e frescura dos dados. Enquanto estas abordagens permitem melhorar substancialmente os tempos de resposta em cenários com carga moderada, introduzem limitações quando o sistema é sujeito a picos intensivos de acesso concorrente ou quando os dados não se encontram em cache. Estes resultados demonstram que as decisões arquiteturais adotadas foram adequadas às prioridades e ao contexto do sistema, ao mesmo tempo que evidenciam os *trade-offs* inerentes e os pontos de evolução necessários para suportar cenários de maior escala e exigência no futuro.

5.7 Validação dos critérios e métricas de aceitação

Os resultados obtidos na avaliação experimental também permitem analisar o grau de cumprimento dos critérios de aceitação e métricas de avaliação definidos na fase de análise do problema (Secção 3).

5.7.1 Registo, autenticação e gestão de utilizadores

Os critérios de aceitação associados ao registo, autenticação e gestão de utilizadores foram validados essencialmente através da implementação funcional e de testes manuais controlados. O sistema implementa um processo completo de *onboarding*, incluindo a submissão de pedidos de adesão por parte dos utilizadores e a respetiva aprovação ou rejeição por administradores, bem como mecanismos básicos de gestão de utilizadores e permissões (*roles*), conforme descrito na Secção 4.6. Adicionalmente, foi efetuada a integração funcional com dois *providers* de *Single Sign-On* (Google e Microsoft), permitindo autenticação adequada a um ambiente de consórcio, com baixo atrito para o utilizador e controlo centralizado de acessos. Relativamente às métricas de avaliação definidas, o tempo médio de autenticação encontra-se dentro do limite de 10 s, conforme comprovado na validação do cenário QA3. Do mesmo modo, a taxa de sucesso da autenticação foi validada através de testes funcionais controlados, não se tendo observado qualquer falha na autenticação de utilizadores válidos nos cenários testados ou durante todo o processo de desenvolvimento da aplicação. Dado o caráter determinístico do fluxo de autenticação e a delegação do processo de validação de credenciais para *providers* externos consolidados, considera-se que o sistema cumpre o critério de uma taxa de sucesso superior a 99% para utilizadores válidos.

5.7.2 Gestão de dados e desempenho

Os critérios de aceitação associados à gestão de dados e desempenho foram globalmente cumpridos, conforme evidenciado pelas decisões arquiteturais adotadas e pelos resultados da avaliação experimental. O sistema implementa mecanismos de cache local integrados no cliente, recorrendo a persistência local e a estruturas de cache em memória, o que permite reduzir significativamente o número de acessos diretos à blockchain. As operações de leitura são servidas a partir da cache sempre que os dados se encontram atualizados, sendo o acesso ao backend e à blockchain realizado apenas quando necessário. Este comportamento foi validado nos cenários de leitura avaliados, nomeadamente na comparação entre *cold cache* e *warm cache*, onde se observaram reduções claras nos tempos de resposta. Adicionalmente, o sistema implementa estratégias de *fetching* orientadas à otimização do volume de dados transferidos, combinando pedidos iniciais mais leves com carregamento explícito de entidades associadas apenas quando necessário. Esta abordagem permite equilibrar a latência percebida pelo utilizador com a consistência dos dados e a eficiência dos acessos à blockchain, cumprindo o terceiro critério de aceitação definido.

Relativamente às métricas de avaliação, os resultados demonstram que a latência média de leitura quando os dados são servidos a partir da cache local se mantém inferior a 3 s, encontrando-se em linha com os valores

observados nos testes realizados. Este resultado confirma a eficácia dos mecanismos de *caching* na mitigação do impacto da latência inerente às operações *on-chain*. No que respeita à capacidade de suportar cargas moderadas de leitura concorrente, os resultados são mais heterogéneos. Alguns testes demonstram que o sistema consegue responder corretamente a um número elevado de utilizadores simultâneos durante períodos curtos, mantendo tempos de resposta razoáveis. No entanto, outros cenários de carga evidenciam degradação progressiva do desempenho e aumento da taxa de falhas à medida que a carga se prolonga, indicando que a arquitetura atual não suporta de forma robusta cenários sustentados com cerca de 50 utilizadores concorrentes. Assim, esta métrica é apenas parcialmente cumprida, revelando limitações claras ao nível da escalabilidade das operações de leitura. Por fim, relativamente às taxas de sucesso das operações, os testes indicam que, sob carga normal (até 10 utilizadores), o sistema apresenta um comportamento estável, com taxas de sucesso compatíveis com o limite de 98% definido. Em cenários de carga elevada, verifica-se um aumento do número de falhas, aproximando-se do limiar mínimo de 95% e, em alguns testes, ultrapassando-o. Estes resultados refletem os compromissos inerentes à arquitetura adotada e confirmam.

5.7.3 Sincronização e resolução de conflitos

Os critérios de aceitação associados à sincronização e resolução de conflitos também foram validados através da análise do comportamento do sistema em cenários de concorrência controlada, avaliados na tarefa T7, bem como por inspeção da política de reconciliação implementada. O sistema implementa uma política de resolução de conflitos orientada à preservação do trabalho dos utilizadores, permitindo edições concorrentes sobre campos distintos da mesma entidade sem perda de informação. Em situações de conflito direto (sobre os mesmos campos da mesma instância), é aplicada a política de *Most Recent Wins*, garantindo que a versão mais recente é aceite. Operações potencialmente destrutivas, como eliminações baseadas em versões obsoletas, são rejeitadas, evitando a perda inadvertida de alterações recentes. Este comportamento confirma o cumprimento do critério de aceitação relativo à implementação de uma política de reconciliação que aproveita ao máximo as alterações efetuadas pelos utilizadores. Relativamente à reação da interface às ações do utilizador, o sistema combina as estratégias de interação otimista e pessimista, consoante a natureza da operação. As atualizações são refletidas de forma praticamente imediata na interface, fornecendo *feedback* inicial em menos de 1 s, enquanto operações de criação e eliminação recorrem a uma abordagem pessimista, podendo introduzir uma latência superior. Esta decisão arquitetural foi tomada de forma consciente, de modo a garantir consistência e evitar estados inválidos em operações críticas, mantendo ainda assim uma experiência de utilização previsível. Adicionalmente, o sistema suporta a confirmação assíncrona das operações *on-chain*, não bloqueando a

interação do utilizador durante o processo de persistência na blockchain, cumprindo os restantes critérios de aceitação definidos.

No que respeita às métricas de avaliação, a percentagem de conflitos resolvidos automaticamente foi validada qualitativamente nos cenários testados. Nos testes realizados, todas as situações de concorrência controlada foram resolvidas sem intervenção manual, excetuando operações de eliminação realizadas sobre versões obsoletas, que foram corretamente rejeitadas. Embora não tenha sido possível estimar de forma estatisticamente rigorosa a percentagem de conflitos resolvidos automaticamente, os resultados observados indicam um comportamento consistente com o objetivo de resolução automática da maioria dos conflitos. De forma semelhante, a taxa de perda de operações do utilizador não foi quantificada numericamente, uma vez que os cenários de perda correspondem a situações específicas e pouco frequentes no domínio da aplicação, como tentativas de eliminação imediatamente após atualizações concorrentes. Nestes casos, a rejeição da operação é considerada um comportamento desejável do ponto de vista da integridade dos dados, e não uma falha do sistema. Assim, conclui-se que no contexto dos cenários avaliados a perda de operações do utilizador é residual e limitada a casos intencionalmente bloqueados pela política de consistência adotada.

5.8 Limitações do trabalho

Apesar dos resultados obtidos permitirem validar as principais decisões arquiteturais propostas, existem limitações relevantes que importa reconhecer, quer ao nível da arquitetura adotada, quer ao nível da metodologia de avaliação experimental. Em primeiro lugar, embora a utilização de mecanismos de *caching* local tenha permitido reduzir significativamente a latência percebida em operações de leitura, as operações de escrita mantêm-se fortemente dependentes do tempo de consenso da rede blockchain. Este fator traduziu-se em tempos de resposta elevados nas tarefas de escrita, como observado na tarefa T5, e reflete uma limitação inerente à tecnologia de base. O foco do trabalho esteve, assim, na mitigação do impacto dessa latência através de estratégias de experiência do utilizador (como confirmação assíncrona e interação otimista), e não na sua eliminação, uma vez que tal cai fora do controlo da aplicação cliente.

Uma segunda limitação prende-se com a escalabilidade da camada de acesso à blockchain. A arquitetura proposta recorre a uma única REST API que atua como *gateway* entre o cliente e a rede blockchain. Embora esta abordagem simplifique o desenvolvimento e centralize preocupações de segurança e validação, introduz também um único ponto de estrangulamento. Os testes de carga com 100 utilizadores concorrentes (QA14) demonstraram que, na configuração atual, esta camada não consegue sustentar picos elevados de leitura sem

degradação significativa do desempenho, evidenciada por uma taxa de falhas elevada. Desta forma, a ausência de mecanismos de balanceamento de carga ou de distribuição de pedidos por múltiplas instâncias limita a escalabilidade horizontal do sistema. Adicionalmente, a avaliação experimental baseou-se em cenários de teste controlados e *scripts* de simulação, que embora sejam representativos do comportamento esperado da aplicação, não refletem na totalidade os padrões de utilização reais dos utilizadores finais. A impossibilidade de recolher métricas durante a utilização efetiva do sistema num contexto de produção limitou a análise de indicadores como taxas reais de *cache hit* ou padrões naturais de concorrência. Uma avaliação em ambiente real permitiria obter uma visão mais abrangente e menos artificial do comportamento do sistema ao longo do tempo.

Por fim, e dado o teor deste trabalho, os testes comparativos entre estratégias de *fetching* focaram-se exclusivamente na aplicação cliente, mantendo a blockchain numa configuração estável. A não exploração de estratégias de carregamento equivalentes ao nível do backend ou da própria blockchain pode ter mitigado, ou amplificado, o impacto observado de cada abordagem. No entanto, esta decisão foi intencional, dado que o objetivo central do trabalho incidia sobre o desenho e avaliação da arquitetura da aplicação cliente, procurando isolar essa variável e reduzir o número de fatores de confusão.

6 Conclusão

O presente trabalho teve como objetivo investigar de que forma uma arquitetura baseada em blockchain pode ser integrada numa aplicação web de gestão de dados, procurando mitigar as limitações tradicionalmente associadas a este tipo de tecnologia, nomeadamente em termos de desempenho, experiência do utilizador e consistência em cenários concorrentes. Através do desenvolvimento e avaliação de um protótipo funcional no contexto do projeto BioComp, foi possível explorar decisões arquiteturais concretas e avaliar empiricamente o seu impacto nos principais atributos de qualidade do sistema.

Os resultados obtidos demonstram que, apesar da latência inerente às operações *on-chain*, é possível conceber uma aplicação com uma experiência de utilização aceitável, desde que sejam adotadas estratégias adequadas ao nível do cliente. A utilização de mecanismos de *caching* local, políticas de *fetching* ajustadas ao volume de dados e interfaces otimistas ou pessimistas consoante o tipo de operação revelaram-se fundamentais para reduzir a latência percebida e melhorar a previsibilidade do comportamento do sistema. A avaliação experimental evidenciou ganhos claros em cenários de leitura com *warm cache* e validou a robustez do mecanismo de resolução de conflitos em contextos concorrentes controlados. Por outro lado, os testes de carga e escalabilidade revelaram limitações relevantes da arquitetura atual, em particular no que respeita ao acesso concorrente intensivo. A presença de uma única REST API como ponto de entrada para a blockchain constitui um fator de estrangulamento sob cargas elevadas, limitando a capacidade do sistema para suportar picos de leitura com dezenas de utilizadores simultâneos. Estes resultados reforçam a ideia de que, embora as decisões adotadas sejam adequadas para um cenário de consórcio de pequena ou média escala, a escalabilidade horizontal exige uma evolução arquitetural adicional. À luz destes resultados, é possível responder às questões de investigação definidas no início deste trabalho. Relativamente à QI1, a avaliação experimental evidencia que estratégias de gestão local de dados no cliente, nomeadamente mecanismos de *caching* e políticas de *fetching* adequadas ao padrão de acesso, têm um impacto significativo na latência percebida e na escalabilidade da aplicação, permitindo mitigar, do ponto de vista do utilizador, as limitações intrínsecas das operações *on-chain*. Quanto à QI2, os resultados demonstram que a combinação de sincronização assíncrona com mecanismos de resolução de conflitos baseados em *versioning* otimista permite preservar a consistência dos dados e uma experiência de utilização aceitável em cenários concorrentes controlados, desde que as estratégias de interação da interface sejam ajustadas ao impacto semântico das operações. Muitas destas conclusões não eram evidentes à partida: a expectativa inicial de que a introdução de uma camada de *caching* ou de estratégias otimistas fosse suficiente para mascarar a latência da blockchain revelou-se demasiado

simplista, sendo necessária uma combinação cuidada de decisões arquiteturais, políticas de sincronização e desenho da interface. Neste sentido, os resultados obtidos evidenciam que o impacto real destas estratégias só se torna claro através de avaliação empírica, reforçando a importância de uma abordagem experimental no desenvolvimento de dApps.

As limitações identificadas não diminuem a relevância do trabalho realizado, mas antes enquadram-no de forma realista. Importa salientar que o foco do trabalho incidiu sobretudo na gestão de dados, integração com a blockchain e experiência do utilizador, e não numa otimização extrema da infraestrutura. Além disso, a avaliação experimental foi necessariamente conduzida em cenários controlados e artificiais, não tendo sido possível recolher métricas de desempenho durante a utilização do sistema em contexto real por parte dos utilizadores finais. A ausência dessa utilização prolongada limita a análise de métricas como taxas de *cache hit* ou padrões reais de concorrência, que apenas poderiam ser observados num ambiente de produção. Neste sentido, existem várias direções claras para trabalho futuro, sendo que uma das mais relevantes consiste na disponibilização do sistema aos utilizadores alvo do BioComp, permitindo avaliar se a interface e as funcionalidades implementadas correspondem efetivamente às suas necessidades e expectativas, bem como observar o comportamento do sistema em cenários de utilização realista. Adicionalmente, subproblemas ainda não explorados em profundidade, como a fase de aplicação do composto, deverão ser modelados e integrados no sistema, completando o ciclo funcional do domínio.

Do ponto de vista técnico, a evolução natural do sistema passa pela melhoria da sua escalabilidade, nomeadamente através da distribuição da carga por múltiplas REST APIs e organizações, com cada entidade do consórcio a alojar o seu próprio nó da blockchain e serviços associados. Esta abordagem, embora mais exigente em termos de manutenção e recursos computacionais, permitiria avaliar o sistema num cenário mais próximo da visão original do consórcio e explorar de forma mais aprofundada as capacidades do Hyperledger Fabric em ambientes distribuídos. Paralelamente, existe margem para continuar a estudar estratégias adicionais de otimização, com o objetivo de aproximar o sistema, tanto quanto possível, do comportamento esperado de uma aplicação web tradicional, sobretudo no que respeita à escalabilidade sob cargas elevadas.

Por fim, importa refletir sobre a adequabilidade da utilização de blockchain neste contexto. É inquestionável que uma parte significativa da funcionalidade implementada poderia ser suportada por uma base de dados tradicional, com ganhos imediatos ao nível do desempenho e simplicidade. No entanto, a adoção da blockchain introduz características relevantes, como a imutabilidade dos registos, a transparência entre entidades e a descentralização da confiança, que são particularmente adequadas a um cenário de consórcio interinstitucional

como o BioComp. Os resultados deste trabalho alinham-se com a literatura existente, que aponta para a blockchain como uma tecnologia com benefícios claros em termos de governação e confiança, mas que exige cuidados acrescidos ao nível do desenho arquitetural para garantir uma experiência de utilização aceitável. Para além do contexto específico do BioComp, as decisões arquiteturais, estratégias de otimização e lições retiradas ao longo deste trabalho possuem um grau de generalidade que permite a sua aplicação a outros contextos de desenvolvimento de dApps, particularmente em cenários onde existem requisitos de rastreabilidade, múltiplas entidades com níveis distintos de confiança e a necessidade de conciliar consistência com uma experiência de utilização aceitável. Embora os valores absolutos de desempenho e escalabilidade dependam fortemente da infraestrutura e do caso de uso, as tendências observadas e os compromissos identificados são transversais e fornecem orientações práticas para outros projetos que enfrentem desafios semelhantes.

Em síntese, este trabalho demonstra que a integração de blockchain em aplicações web é viável e tecnicamente sustentada, desde que sejam adotadas estratégias conscientes dos compromissos inerentes à tecnologia. Para além de contribuir para a evolução do BioComp, os resultados e lições aprendidas constituem um contributo relevante para a investigação e prática no desenvolvimento de dApps, reforçando a importância de continuar a explorar, de forma crítica, a aplicação de tecnologias descentralizadas em contextos reais.

Referências

[1] K. Wust and A. Gervais, "Do you Need a Blockchain?" in *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. Zug: IEEE, Jun. 2018, pp. 45–54. [Online]. Available: https://ieeexplore.ieee.org/document/8525392/

[2] S. K. Lo, X. Xu, Y. K. Chiam, and Q. Lu, "Evaluating Suitability of Applying Blockchain," in *2017 22nd International Conference on Engineering of Complex Computer Systems (ICECCS)*. Fukuoka: IEEE, Nov. 2017, pp. 158–161. [Online]. Available: http://ieeexplore.ieee.org/document/8292816/

[3] S. Bouraga, "CASCADE – Framework for the Early-Phase Development of Blockchain-Based Applications," *Business & Information Systems Engineering*, Jan. 2025. [Online]. Available: https://link.springer.com/10.1007/s12599-024-00912-4

[4] G. Di Lucca and M. Tortorella, "Comparing Blockchain Platforms for Smart Contracts: A Preliminary Framework," pp. 104–92. [Online]. Available: https://easychair.org/publications/paper/wcSq

[5] S. Barac, I. Botički, G. Perković, V. Radošević, and I. Terzić, "Cardano - What Is It and How to Start Working with It," in *2023 46th MIPRO ICT and Electronics Convention (MIPRO)*. Opatija, Croatia: IEEE, May 2023, pp. 1727–1732. [Online]. Available: https://ieeexplore.ieee.org/document/10159944/

[6] F. Wang, Y. Lu, Q. Zhang, Y. Liu, L. Liu, and Z. Zhang, "SoK: Research status and challenges of blockchain smart contracts," in *Proceedings of the 5th ACM International Symposium on Blockchain and Secure Critical Infrastructure*. Melbourne VIC Australia: ACM, Jul. 2023, pp. 145–147. [Online]. Available: https://dl.acm.org/doi/10.1145/3594556.3594620

[7] F. Ali, M. U. Khurram, A. Sensoy, and X. V. Vo, "Green cryptocurrencies and portfolio diversification in the era of greener paths," *Renewable and Sustainable Energy Reviews*, vol. 191, p. 114137, Mar. 2024. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1364032123009954

[8] D. Li, W. E. Wong, and J. Guo, "A Survey on Blockchain for Enterprise Using Hyperledger Fabric and Composer," in *2019 6th International Conference on Dependable Systems and Their Applications (DSA)*. Harbin, China: IEEE, Jan. 2020, pp. 71–80. [Online]. Available: https://ieeexplore.ieee.org/document/9045815/

[9] X. Xu, C. Pautasso, L. Zhu, V. Gramoli, A. Ponomarev, A. B. Tran, and S. Chen, "The Blockchain as a Software Connector," in *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*. Venice, Italy: IEEE, Apr. 2016, pp. 182–191. [Online]. Available: http://ieeexplore.ieee.org/document/7516828/

[10] J. Yli-Huumo, D. Ko, S. Choi, S. Park, and K. Smolander, "Where Is Current Research on Blockchain Technology?—A Systematic Review," *PLOS ONE*, vol. 11, no. 10, p. e0163477, Oct. 2016. [Online]. Available: https://dx.plos.org/10.1371/journal.pone.0163477

[11] O. Ali, A. Jaradat, A. Kulakli, and A. Abuhalimeh, "A Comparative Study: Blockchain Technology Utilization Benefits, Challenges and Functionalities," *IEEE Access*, vol. 9, pp. 12 730–12 749, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9317729/

[12] F. Daniel and L. Guida, "A Service-Oriented Perspective on Blockchain Smart Contracts," *IEEE Internet Computing*, vol. 23, no. 1, pp. 46–53, Jan. 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8598947/

[13] M. Shin, "Modeling of Blockchain and Application Concerns in Blockchain Applications," in *2023 IEEE/ACM 6th International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. Melbourne, Australia: IEEE, May 2023, pp. 37–43. [Online]. Available: https://ieeexplore.ieee.org/document/10190824/

[14] J. Saldivar, E. Martínez-Vicente, D. Rozas, M.-C. Valiente, and S. Hassan, "Blockchain (not) for Everyone: Design Challenges of Blockchain-based Applications," in *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*. Hamburg Germany: ACM, Apr. 2023, pp. 1–8. [Online]. Available: https://dl.acm.org/doi/10.1145/3544549.3585825

[15] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System." [Online]. Available: https://bitcoin.org/bitcoin.pdf

[16] L. Baresi, G. Quattrocchi, D. A. Tamburri, and L. Terracciano, "A declarative modelling framework for the deployment and management of blockchain applications," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*. Montreal Quebec Canada: ACM, Oct. 2022, pp. 311–321. [Online]. Available: https://dl.acm.org/doi/10.1145/3550355.3552417

[17] A. Abuhashim and C. C. Tan, "Smart Contract Designs on Blockchain Applications," in *2020 IEEE Symposium on Computers and Communications (ISCC)*. Rennes, France: IEEE, Jul. 2020, pp. 1–4.

[Online]. Available: https://ieeexplore.ieee.org/document/9219622/

[18] G. Maragno, L. Gastaldi, L. Tangi, and M. Benedetti, "Blockchain applications within the public sector: evidence from an international census," in *DG.O2021: The 22nd Annual International Conference on Digital Government Research*. Omaha NE USA: ACM, Jun. 2021, pp. 479–488. [Online]. Available: https://dl.acm.org/doi/10.1145/3463677.3463699

[19] W. Lin, X. Huang, H. Fang, V. Wang, Y. Hua, J. Wang, H. Yin, D. Yi, and L. Yau, "Blockchain Technology in Current Agricultural Systems: From Techniques to Applications," *IEEE Access*, vol. 8, pp. 143 920–143 937, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9159588/

[20] S. Soliman, A. Bendary, and H. Dahshan, "Enhancing Agricultural Efficiency with Blockchain-Orchestrated Drone Swarms and Kubernetes," in *2024 6th Novel Intelligent and Leading Emerging Sciences Conference (NILES)*. Giza, Egypt: IEEE, Oct. 2024, pp. 567–570. [Online]. Available: https://ieeexplore.ieee.org/document/10753154/

[21] E. F. Coutinho, M. M. Neto, A. W. Abreu, and L. O. Moreira, "An Analysis on the Use of Blockchain by Developers and Application Users," in *Proceedings of the 11th Euro American Conference on Telematics and Information Systems*. Aveiro Portugal: ACM, Jun. 2022, pp. 1–8. [Online]. Available: https://dl.acm.org/doi/10.1145/3544538.3544648

[22] K. Nath, S. Dhar, and S. Basishtha, "Web 1.0 to Web 3.0 - Evolution of the Web and its various challenges," in *2014 International Conference on Reliability Optimization and Information Technology (ICROIT)*. Faridabad, Haryana, India: IEEE, Feb. 2014, pp. 86–89. [Online]. Available: http://ieeexplore.ieee.org/document/6798297/

[23] J. Huang, L. Kong, G. Cheng, Q. Xiang, G. Chen, G. Huang, and X. Liu, "Advancing Web 3.0: Making Smart Contracts Smarter on Blockchain," in *Proceedings of the ACM Web Conference 2024*. Singapore Singapore: ACM, May 2024, pp. 1549–1560. [Online]. Available: https://dl.acm.org/doi/10.1145/3589334.3645319

[24] J. Jaenudin, A. Zahran, and D. Mahdiana, "Blockchain Utilization in Secure and Decentralized Web 3.0 Application Development," *Sinkron*, vol. 9, no. 1, pp. 594–599, Jan. 2024. [Online]. Available: https://jurnal.polgan.ac.id/index.php/sinkron/article/view/13411

[25] W. Zhao, S. Yang, X. Luo, and J. Zhou, "Dos and Don'ts in Blockchain Research and Development," in *The 2022 4th International Conference on Blockchain Technology*. Shanghai China: ACM, Mar. 2022,

pp. 37–43. [Online]. Available: https://dl.acm.org/doi/10.1145/3532640.3532646

[26] C. Lazaroiu, M. Roscia, and S. Saadatmandi, "Blockchain and Fuzzy Logic Application in EV's Charging," in *2020 9th International Conference on Renewable Energy Research and Application (ICRERA)*. Glasgow, United Kingdom: IEEE, Sep. 2020, pp. 315–320. [Online]. Available: https://ieeexplore.ieee.org/document/9242662/

[27] M. Wohrer and U. Zdun, "Architectural Design Decisions for Blockchain-Based Applications," in *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. Sydney, Australia: IEEE, May 2021, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/9461109/

[28] D. A. Snegireva, "Review of Modern Blockchain Platforms," in *2021 International Conference on Quality Management, Transport and Information Security, Information Technologies (IT&QM&IS)*. Yaroslavl, Russian Federation: IEEE, Sep. 2021, pp. 112–116. [Online]. Available: https://ieeexplore.ieee.org/document/9642822/

[29] J. J. Kim, P. Lingga, J. P. Jeong, Y. Choi, and J. Park, "A Web-Based Monitoring System of Network Security Functions in Blockchain-Based Cloud Security Systems," in *2022 International Conference on Information Networking (ICOIN)*. Jeju-si, Korea, Republic of: IEEE, Jan. 2022, pp. 454–459. [Online]. Available: https://ieeexplore.ieee.org/document/9687177/

[30] 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)*. Doha, Qatar: IEEE, Feb. 2020, pp. 608–613. [Online]. Available: https://ieeexplore.ieee.org/document/9089614/

[31] Y. Huang, B. Wang, and Y. Wang, "Research and Application of Smart Contract Based on Ethereum Blockchain," *Journal of Physics: Conference Series*, vol. 1748, no. 4, p. 042016, Jan. 2021. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/1748/4/042016

[32] X. Xu, H. Dilum Bandara, Q. Lu, I. Weber, L. Bass, and L. Zhu, "A Decision Model for Choosing Patterns in Blockchain-Based Applications," in *2021 IEEE 18th International Conference on Software Architecture (ICSA)*. Stuttgart, Germany: IEEE, Mar. 2021, pp. 47–57. [Online]. Available: https://ieeexplore.ieee.org/document/9426788/

[33] N. Legowo, N. Hawari, T. Karlina, E. Tanuwijaya, and K. Mahendra, "Design Smart Contract Based on Blockchain for Peer-to-Peer Lending Platform," in *2023 10th International Conference on ICT*

for Smart Society (ICISS). Bandung, Indonesia: IEEE, Sep. 2023, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/10291985/

[34] A. Sahu, D. P. Mishra, S. B. Mohanty, and P. P. Sahu, "Web 3.0 Decentralized Application Using Blockchain Technology," in *2023 4th International Conference on Computing and Communication Systems (I3CS)*. Shillong, India: IEEE, Mar. 2023, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/10127552/

[35] 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)*. Hangzhou, China: IEEE, Feb. 2019, pp. 1–10. [Online]. Available: https://ieeexplore.ieee.org/document/8666486/

[36] I. R. Fedorov, A. V. Pimenov, G. A. Panin, and S. V. Bezzateev, "Blockchain in 5G Networks: Perfomance Evaluation of Private Blockchain," in *2021 Wave Electronics and its Application in Information and Telecommunication Systems (WECONF)*. St. Petersburg, Russia: IEEE, May 2021, pp. 1–4. [Online]. Available: https://ieeexplore.ieee.org/document/9470519/

[37] Y. Jiang, C. Huang, H. Lu, Y. Zhao, and Y. Yu, "Consortium Blockchain-based Community Electricity Trading with Arbitration," in *2023 IEEE Sustainable Power and Energy Conference (iSPEC)*. Chongqing, China: IEEE, Nov. 2023, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/10402886/

[38] A. H. Mohammed, A. A. Abdulateef, and I. A. Abdulateef, "Hyperledger, Ethereum and Blockchain Technology: A Short Overview," in *2021 3rd International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*. Ankara, Turkey: IEEE, Jun. 2021, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/9461294/

[39] M. P. M. Sy, R. I. Marasigan, and E. D. Festijo, "Hyperledger-Operated Blockchain Integration: Writing, Deploying and Testing Custom Chaincode," in *2023 Sixth International Symposium on Computer, Consumer and Control (IS3C)*. Taichung, Taiwan: IEEE, Jun. 2023, pp. 151–154. [Online]. Available: https://ieeexplore.ieee.org/document/10219516/

[40] Y. Huang, Y. Bian, R. Li, J. L. Zhao, and P. Shi, "Smart Contract Security: A Software Lifecycle Perspective," *IEEE Access*, vol. 7, pp. 150 184–150 202, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8864988/

[41] C. G. Liu, P. Bodorik, and D. Jutla, "Automating Smart Contract Generation on Blockchains Using Multi-modal Modeling," *Journal of Advances in Information Technology*, vol. 13, no. 3, 2022. [Online]. Available: http://www.jait.us/index.php?m=content&c=index&a=show&catid=217&id=1219

[42] D. Kim and S. Park, "Blockchain-Based Caching Architecture for DApp Data Security and Delivery," *Sensors*, vol. 24, no. 14, p. 4559, Jul. 2024. [Online]. Available: https://www.mdpi.com/1424-8220/24/14/4559

[43] B. Koteska, E. Karafiloski, and A. Mishev, "Blockchain Implementation Quality Challenges: A Literature Review." [Online]. Available: https://www.researchgate.net/publication/320127088_Blockchain_Implementation_Quality_Challenges_A_Literature_Review

[44] M. G. Jaatun, P. H. Haro, and C. Froystad, "Five Things You Should Not Use Blockchain For," in *2020 IEEE Cloud Summit*. Harrisburg, PA, USA: IEEE, Oct. 2020, pp. 167–169. [Online]. Available: https://ieeexplore.ieee.org/document/9283711/

[45] F. Blum, B. Severin, M. Hettmer, P. Huckinghaus, and V. Gruhn, "Building Hybrid DApps using Blockchain Tactics -The Meta-Transaction Example," in *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. Toronto, ON, Canada: IEEE, May 2020, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/9169423/

[46] M. P. Caro, M. S. Ali, M. Vecchio, and R. Giaffreda, "Blockchain-based traceability in Agri-Food supply chain management: A practical implementation," in *2018 IoT Vertical and Topical Summit on Agriculture - Tuscany (IOT Tuscany)*. Tuscany: IEEE, May 2018, pp. 1–4. [Online]. Available: https://ieeexplore.ieee.org/document/8373021/

[47] F.-J. Ferrández-Pastor, J. Mora-Pascual, and D. Díaz-Lajara, "Agricultural traceability model based on IoT and Blockchain: Application in industrial hemp production," *Journal of Industrial Information Integration*, vol. 29, p. 100381, Sep. 2022. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2452414X22000498

[48] T. H. Pranto, A. A. Noman, A. Mahmud, and A. B. Haque, "Blockchain and smart contract for IoT enabled smart agriculture," *PeerJ Computer Science*, vol. 7, p. e407, Mar. 2021. [Online]. Available: https://peerj.com/articles/cs-407

[49] T. Wang, X. Liu, S. Guo, B. Han, and W. Yang, "Blockchain and IoT based traceability system for agricultural products," in *2022 3rd International Conference on Computer Vision,*

Image and Deep Learning & International Conference on Computer Engineering and Applications (CVIDL & ICCEA). Changchun, China: IEEE, May 2022, pp. 316–321. [Online]. Available: https://ieeexplore.ieee.org/document/9824731/

[50] "Hyperledger Fabric Test Network," https://hyperledger-fabric.readthedocs.io/en/latest/test_network.html, acedido a 15 de junho de 2024.

[51] I. Popchev and I. Radeva, "Decentralized Application (dApp) Development and Implementation," *Cybernetics and Information Technologies*, vol. 24, no. 2, pp. 122–141, Jun. 2024. [Online]. Available: https://www.sciendo.com/article/10.2478/cait-2024-0019

[52] M. Faiz and U. Shanker, "Data synchronization in distributed client-server applications," in *2016 IEEE International Conference on Engineering and Technology (ICETECH)*. Coimbatore, India: IEEE, Mar. 2016, pp. 611–616. [Online]. Available: http://ieeexplore.ieee.org/document/7569323/