UNIVERSIDADE D
COIMBRA

Bernardo Luís Tojal Barbosa

COLECIONÁVEIS EM BLOCKCHAIN

Dissertação no âmbito do Mestrado em Engenharia Informática, especialização em Engenharia de Software, orientada pelo Professor Doutor Bruno Sousa e pelos Engenheiros Raul Fonseca e Miguel Santos e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Julho de 2022

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA
DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Bernardo Luís Tojal Barbosa

Colecionáveis em Blockchain

Dissertação no âmbito do Mestrado em Engenharia Informática, especialização em Engenharia de Software, orientada pelo Professor Doutor Bruno Sousa e pelos Engenheiros Raul Fonseca e Miguel Santos e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Julho de 2022

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Bernardo Luís Tojal Barbosa

Blockchain Collectibles

Dissertation in the context of the Master in Informatics Engineering, specialization in Software Engineering, advised by Professor Bruno Sousa and by Engineers Raul Fonseca and Miguel Santos and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July 2022

Agradecimentos

Esta página é destinada a todos os que me acompanharam e ajudaram ao longo deste percurso.

Em primeiro lugar, quero agradecer aos meus pais por todo o apoio e pela ajuda que sempre me disponibilizaram para que conseguisse realizar este percurso académico da maneira mais confortável possível.

À WIT Software, agradeço pela confiança, oportunidade, e pelas condições de trabalho que me proporcionaram para a realização deste estágio. Ao orientador Raul Fonseca, ao tutor Miguel Santos e à analista de negócio Ana Fernandes, deixo aqui o meu sincero obrigado por toda a ajuda, transmissão de conhecimento e por todo o acompanhamento ao longo desta etapa.

Ao Professor Bruno Sousa, quero agradecer pela sua disponibilidade, pelos seus conselhos e sugestões, e pela orientação prestada ao longo do estágio.

A todos os amigos que me acompanharam durante este percurso, deixo também o meu enorme obrigado.

Obrigado!

Resumo

Qualquer produto ou serviço que endereça diretamente o mercado de consumo, enfrenta alguns desafios nas áreas de aquisição de novos clientes, e na retenção e satisfação dos mesmos. Deste modo, as operadoras de telecomunicações móveis começaram a explorar medidas que possam ajudar a resolver estes problemas, nomeadamente através do uso da gamificação.

Por conseguinte, associado ao impacto positivo das técnicas de gamificação na indústria das telecomunicações, a empresa WIT Software desenvolveu um produto de gamificação, mais tarde adotado pela Vodafone Portugal e ao qual foi atribuído o nome Shake It. Esta aplicação móvel funciona à base de um sistema de recompensas onde são atribuídos prémios sempre que é completado um objetivo, através da coleção de cartas colecionáveis, em prol do aumento do *user engagement*. No âmbito deste estágio, o objetivo passa por mitigar um dos problemas deste jogo: a percepção de falta de transparência no sistema de atribuição de cartas, por parte dos utilizadores.

Neste sentido, o estagiário ficou encarregue de potencializar o Shake It com capacidades de *Blockchain*, nomeadamente através do uso de *Non-Fungible Tokens*. Dada a descentralização derivada desta tecnologia e devido ao facto de cada transação ser registada de forma pública, unívoca e imutável na *Blockchain*, o sistema em torno do Shake It possuirá um mecanismo mais seguro e confiável, oferecendo mais transparência aos clientes e destacando-se dos seus competidores.

No presente relatório de estágio, será exposto todo o processo que o aluno e a empresa seguiram no decorrer do ano curricular 2021/2022.

Palavras-Chave

Blockchain, *Smart Contract*, *Non-fungible Token*, NFTs, Ethereum, Polygon, Gamificação, *User engagement*, *InterPlanetary File System*, IPFS

Abstract

Any product or service that directly addresses the consumer market faces some challenges in the areas of acquiring new customers, and retaining and satisfying them. Thus, mobile telecommunications operators began to explore measures that could help solve these problems, namely through the use of gamification.

Therefore, associated with the positive impact of gamification techniques in the telecommunications industry, the company WIT Software developed a gamification product, later adopted by Vodafone Portugal and given the name Shake It. This mobile application is the basis of a rewards system where prizes are awarded whenever a goal is completed, through the collection of collectible cards, in order to increase user engagement. With this internship, the objective is to mitigate one of the problems with this game: the perception, by users, of lack of transparency in the card distribution system.

In this regard, the intern is in charge of enhancing Shake It with Blockchain capabilities, namely through the use of Non-Fungible Tokens. Given the decentralization derived from this technology and due to the fact that each transaction is registered in a public, univocal and immutable way on the Blockchain, Shake It will possess a safer and more reliable mechanism around it, offering more transparency to customers and standing out from its competitors.

In this report, the entire process surrounding the research and development made during the course of this internship is exposed.

Keywords

Blockchain, Smart Contract, Non-fungible Token, NFTs, Ethereum, Polygon, Gamification, User engagement, InterPlanetary File System, IPFS

Índice

Acrónimos

NFT Non-Fungible Token.

DLT Distributed Ledger Technology.

URL Uniform Resource Locator.

HTTP Hypertext Transfer Protocol.

IPFS InterPlanetary File System.

CID Content Identifier.

TPS Transações Por Segundo.

PoS Proof of Stake.

PoW Proof of Work.

PoH Proof of History.

API Application Programming Interface.

REST Representational State Transfer.

Lista de Figuras

Lista de Tabelas

Capítulo 1
Introdução

O presente relatório foi desenvolvido no âmbito da disciplina de estágio curricular inserida no Mestrado em Engenharia Informática com especialização em Engenharia de Software, no Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

O estágio foi realizado durante o ano letivo de 2021/2022 sob acompanhamento do Professor Bruno Sousa do Departamento de Engenharia Informática da Universidade de Coimbra, assim como dos Engenheiros Raul Fonseca e Miguel Santos, da WIT Software, a empresa responsável pelo presente estágio.

1.1 WIT Software

Fundada em Março de 2001 como *spin-off* da Universidade de Coimbra, a WIT Software [1] é uma empresa que cria soluções avançadas e produtos *white-label* para operadores de telecomunicações móveis.

A WIT tem mais de 20 anos de experiência na área das telecomunicações e é uma empresa francamente exportadora. Prova disso é o facto do *software* que desenvolve estar presente em mais de 46 países [1]. Todos estes projectos foram desenvolvidos com uma metodologia muito forte de *software*, sempre com requisitos de alta-disponibilidade, segurança, *performance*, escalabilidade e especial cuidado com a *user-experience*. A empresa conta com os seguintes clientes: Grupo Vodafone, Deutsche Telekom, NTT DoCoMo, KDDI, Softbank, AT&T, Verizon, Safaricom, Vodacom, Unitel, entre outros [1].

Desde a sua criação que mantém uma parceria tecnológica com a Vodafone Portugal, para a qual desenvolve vários serviços móveis na área de *mobile messaging* e soluções à medida. Um dos produtos procedentes desta colaboração é a aplicação móvel Shake It, sobre a qual é incidido o trabalho realizado ao longo deste estágio.

1.2 Enquadramento e Motivação

As técnicas de gamificação têm sido usadas, desde há alguns anos, para influenciar comportamentos dos consumidores em vários sectores: nas empresas, para aumentar a produtividade, no *retail*/*e-commerce* para aumentar as vendas, ou na indústria de entretenimento para promover o envolvimento.

A WIT Software já comprovou o impacto deste conceito na indústria das telecomunicações, com o fornecimento de soluções de gamificação como o Shake It. Este

jogo alterou a forma como os clientes se relacionam com a operadora móvel e ajudou a promover o *user engagement*. O jogo funciona à base de um sistema de recompensas, onde são atribuídos prémios sempre que é completado um objetivo. Existe uma caderneta com várias categorias, a qual se vai preenchendo à medida que são colecionadas cartas digitais. Estas cartas obtêm-se em função do envolvimento de cada jogador com a aplicação, inclusive através de funcionalidades como a troca de cartas com amigos. Depois, quando uma categoria da caderneta é preenchida na totalidade, o jogador recebe prémios.

No âmbito deste estágio, pretende-se avaliar a evolução de conceitos de gamificação usando tecnologias de *Blockchain*. Desta forma, o objetivo principal é enriquecer o Shake It com capacidades de *Blockchain*, tirando assim partido das vantagens desta tecnologia em prol do aumento do *user engagement* e da redução da taxa de rotatividade dos consumidores.

Para além da necessidade pela inovação tecnológica, é importante possuir um sistema confiável e seguro. Atualmente, dado que o Shake It é centralizado, muitas vezes surgem dúvidas sobre o mecanismo de jogo e se o mesmo é autêntico ou não. Em fóruns *online*, existem alguns comentários de desconfiança em torno do sistema de atribuição de cartas, com receio que o mesmo seja manipulado pela operadora e que seja impossível receber prémios. Estes problemas seriam mitigados com a integração do Shake It em *Blockchain*, visto que uma das características desta tecnologia é a **transparência**, uma vez que é guardado um registo de cada transação de forma pública, unívoca e imutável. Além disso, permite que os clientes troquem cartas entre si de forma autónoma e descentralizada.

1.3 Objetivos

Esta secção descreve os objetivos para este estágio e, como tal, encontra-se dividida em duas subsecções. A primeira retrata os objetivos pessoais do autor e a segunda explica os objetivos do projeto em si.

1.3.1 Objetivos pessoais

A nível pessoal, o autor considera que o principal objetivo do estágio é aperfeiçoar e consolidar todo o conhecimento adquirido ao longo dos últimos 5 anos, tanto na Licenciatura como no Mestrado. Para além disso, na qualidade de estagiário, espera começar a desenvolver uma ética de trabalho sólida e consistente.

A nível profissional, o autor espera desenvolver um projeto que traga valor para a WIT Software e que o permita finalizar o Mestrado com sucesso, aprimorando em simultâneo as suas *soft* e *hard skills* e aprendendo processos e metodologias envolvidas num projeto empresarial.

1.3.2 Objetivos do projeto

O objetivo do estágio consiste no desenvolvimento de um protótipo com base no Shake It, onde algumas das funcionalidades do mesmo serão alteradas e adaptadas consoante as necessidades do projeto, através da integração com a *Blockchain*.

A aplicação deverá permitir que uma operadora de telecomunicações lance e gira coleções de cartas digitais colecionáveis - *Non-fungible Tokens* ou NFTs (Secção 2.2.3) - sobre uma *Blockchain* com suporte para *Smart Contracts* (Secção 2.2.2).

Os clientes dessa operadora poderão receber as cartas colecionáveis em função do seu nível de envolvimento com a marca. Para além disso, esta aplicação deverá permitir que os clientes possuidores de cartas as possam transacionar entre eles de forma autónoma e descentralizada.

Este trabalho deverá ser possível de atingir através do desenvolvimento dos seguintes componentes aplicacionais:

- *Smart Contract* para criar (*mint*) e disponibilizar NFT colecionável a clientes elegíveis;
- Integração com NFT *storage* para alojar o bem digital;
- Servidor aplicacional para permitir que marcas possam facilmente criar novas coleções, e para permitir que clientes do serviço reclamem as cartas que lhe são atribuídas pela marca e para acompanharem o progresso da sua coleção;
- Integração com carteiras digitais para permitir que utilizadores possam transacionar os NFTs.

No final do estágio, deverá existir um protótipo funcional, que seja facilmente demonstrável e que mostre bem os conceitos explorados durante o estágio. Porventura, o mesmo poderá ser posteriormente apresentado ao cliente para possível adoção.

1.4 Estrutura do documento

O presente documento está estruturado em 6 capítulos e o objetivo de cada um é apresentar detalhadamente cada uma das etapas de trabalho vividas pelo estagiário.

- No **capítulo 1** é dada uma introdução do projeto, apresentando a empresa onde o estágio tomou lugar, o contexto e motivações para este projeto, e os objetivos que são pretendidos alcançar até ao fim do estágio;
- No **capítulo 2** é feito o levantamento do Estado da Arte, onde o objetivo passa por apresentar as noções de base fundamentais à compreensão do trabalho realizado neste estágio;
- O **capítulo 3** descreve a metodologia usada durante o estágio curricular, o planeamento que foi delineado para o mesmo e a análise dos riscos;
- No **capítulo 4** são apresentados todos os requisitos funcionais e não funcionais, os respetivos diagramas de casos de uso, e os critérios de sucesso;
- No **capítulo 5**, por sua vez, é apresentada a arquitetura de *software* do projeto, as ferramentas e tecnologias escolhidas e a análise de custos do projeto;

- No **capítulo 6** é apresentada toda a fase de desenvolvimento e os requisitos desenvolvidos;
- O **capítulo 7** tem como objetivo apresentar a fase de testes e validações que foram realizados após o desenvolvimento da aplicação;
- Por fim, no **capítulo 8** é feita uma conclusão do relatório, indicando o trabalho realizado, os desafios encontrados e sugestões para uma futura iteração deste trabalho.

Capítulo 2
Estado da Arte

O presente capítulo explora alguns dos tópicos associados à *Blockchain* e à Gamificação, para assim criar uma base de conhecimentos que possibilitaram compreender todo o contexto em redor do tema do projeto. O capítulo está dividido em quatro secções, são estas:

Na **primeira secção** é explorado o conceito de Gamificação, explicando o que é, em que áreas se pode aplicar e são apresentados alguns exemplos que ilustram este tema.

A **segunda secção** explora o conceito de *Blockchain* explicando o que é e de que tipo pode ser. São também abordados os conceitos de *Smart Contract* e *Non-fungible Token* (NFT), dizendo o que são e de que maneira se relacionam. Depois, é descrito o armazenamento de NFTs, e é apresentada uma solução denominada *InterPlanetary File System*. Por fim, são analisadas e comparadas oito *blockchains*.

Na **terceira secção** é feita uma análise de como estes dois temas se complementam, indicando de que forma a *Blockchain* é vantajosa para a Gamificação. Depois, são analisadas 4 aplicações que incorporam estes conceitos com sucesso.

2.1 Gamificação

Como referido no capítulo 1, as técnicas de gamificação têm sido usadas, desde há alguns anos, para influenciar comportamentos dos consumidores em diferentes setores.

Deste modo, a presente secção tem como objetivo apresentar a definição de gamificação, e onde e como pode ser aplicada. Em adição, são apresentados alguns exemplos de aplicações que utilizam o conceito de gamificação de forma bem sucedida.

2.1.1 Descrição

A gamificação recorre ao uso do *design* e da mecânica de jogos para enriquecer diversos contextos normalmente não relacionados a jogos - como um *website*, uma comunidade *online*, um sistema de gestão de aprendizagem ou um negócio -, com o objetivo de instruir, influenciar o comportamento e incentivar resultados práticos.

Esta técnica tem sido popularizada nos últimos anos, sendo uma das maneiras mais eficazes de melhorar o *user engagement*. Atualmente, temos exemplos como o ensino, onde a gamificação é usada na educação através de jogos lúdicos e atrativos, com o intuito de estimular o interesse dos alunos, tornando a sua aprendizagem mais dinâmica, rápida e fácil.

Nas redes sociais, em aplicações de telemóvel, em plataformas de *e-learning* e em muitos outros serviços, as técnicas de gamificação são cada vez mais usadas, recompensando os utilizadores por completarem determinadas ações ou tarefas.

Deste modo, é evidente que a gamificação tem o potencial de tornar qualquer tecnologia mais convidativa, em virtude da maximização do envolvimento dos utilizadores.

2.1.2 Áreas onde é aplicada a gamificação

Aqui são apresentadas algumas das áreas onde são aplicadas técnicas de gamificação fora da indústria dos jogos, tanto em atividades pessoais como em sociais [57].

- **Educação**: Neste caso, a gamificação é usada para garantir o envolvimento do utilizador, auxiliando-o na conclusão de tarefas habitualmente aborrecidas, e incentiva os mesmos a obterem melhores resultados.
- **Trabalho e Produtividade**: São usadas ferramentas para medir e comparar o desempenho de alguns trabalhadores, encorajando uma maior produtividade e competição.
- **Cibersegurança**: Neste contexto, a gamificação é usada para que *hackers* e especialistas em cibersegurança possam competir entre si para identificar falhas de segurança num dado sistema de *software* ou *hardware*, em troca de reconhecimento e remuneração económica.
- **Saúde**: Alguns parâmetros fisiológicos são medidos e mudanças positivas são avaliadas e reconhecidas, também através de desafios.
- ***Marketing***: Do lado do consumidor, as técnicas de gamificação são usadas para aumentar a lealdade do cliente, e do lado do vendedor, são usadas para incentivar a realização de certos objetivos.

2.1.3 Exemplos de aplicações que aplicam ténicas de gamificação

De seguida, e de forma a ilustrar como o uso destas técnicas podem transformar qualquer área em algo maior e mais poderoso, são apresentados alguns exemplos de aplicações que adotaram técnicas de gamificação.

Starbucks Rewards

Como vimos, a gamificação pode ser usada para influenciar o comportamento humano, e até mesmo incentivar a compra de algo. A empresa Starbucks capitalizou-se nesse aspeto e criou a aplicação Starbucks Rewards [2], a qual recompensa os clientes sempre que compram algum produto, sendo estes prémios cada vez maiores e melhores à medida que se vão fazendo compras.

Em Outubro de 2020, o Starbucks Rewards já contava com cerca de 20 milhões de utilizadores e estima-se que 50% da receita da empresa se devia a este programa de recompensas [3].

Outro fator a ter em conta é a conveniência, sendo que com a aplicação os pagamentos e os pedidos *online* são feitos de forma mais fácil e eficiente. Desta forma, oferecem aos seus utilizadores uma experiência convidativa e inovadora, garantido a sua lealdade.

Nike+ Run Club

A Nike é uma das principais marcas na área do desporto, especialmente ao nível dos equipamentos, mas tem também algumas *apps* interessantes. Uma dessas *apps* é a Nike+ Run Club [4], que funciona como um parceiro de corrida, oferecendo um conjunto vasto de funcionalidades.

Permite ao utilizador monitorizar as suas corridas e seguir um treino personalizado adaptado ao seu nível e objetivos. Também permite a partilha de estatísticas de corrida nas redes sociais, e a participação em desafios para receber distintivos e troféus.

Todas estas características constituem um mecanismo competitivo, o qual é bastante eficiente na implementação da gamificação. O objetivo desta aplicação, para além de reter utilizadores, é melhorar e divulgar a imagem da Nike à comunidade desportiva.

Duolingo

Outro exemplo a ter em conta é o Duolingo [5], na área do ensino. Esta aplicação muda a forma como uma pessoa aprende uma língua nova, tornando este processo o mais simples e divertido possível.

Aprender um novo idioma pode ser uma tarefa desafiadora, e por conseguinte o Duolingo trouxe técnicas de gamificação para dentro da plataforma para envolver melhor os utilizadores, ajudando-os a reter o máximo de informação possível. Ao gamificar toda a experiência de aprendizagem com pontos, distintivos e recompensas, ajuda a criar o ímpeto necessário para motivar os utilizadores no seu estudo.

FoldIt

Em 2009, a sida já havia matado cerca de 30 milhões de pessoas [6] e, em 2020, foram reportados cerca de 37.6 milhões [7] casos de pessoas que contraíram o vírus VIH. Durante 15 anos, muitos dos melhores cientistas do mundo tentaram decifrar uma estrutura para um dos vírus causadores da sida, chamado *Mason-Pfizer Monkey Virus* (M-PMV), mas não conseguiram resolvê-lo [8].

Felizmente, o departamento de *Game Science* da Universidade de Washington em colaboração com o departamento de Bioquímica criou uma aplicação denominada Foldit [9]. Esta plataforma consiste num jogo de *Puzzle online* sobre enovelamento de proteínas. O FoldIt utiliza uma interface gamificada que permite pessoas de todo o mundo competirem e tentarem descobrir estruturas de proteínas que encaixem nos critérios de um investigador, como um *puzzle*.

O jogo acabou por ter uma boa adesão, com cerca de 240 000 jogadores registados que competiram entre eles viciosamente, e, em apenas 10 dias, uma solução para a estrutura do vírus M-PMV foi encontrada [8].

Finalizando, conseguimos ver que as técnicas de gamificação podem ser usadas não só para proveito próprio ou de uma empresa, como também para o avanço tecnológico e da investigação noutras áreas.

Shake It

Ainda na vertente do *marketing*, com o intuito de criar um jogo cativante e que gerasse interesse, foi desenvolvido o Shake It [10], sendo que foi em torno deste que se focou o trabalho realizado no decorrer do presente estágio. Esta aplicação móvel foi criada pela WIT Software e adotada pela empresa de telecomunicações Vodafone Portugal, e o seu principal objetivo é angariar clientes e aumentar a sua fidelização. Isto é alcançado através da adoção de técnicas de gamificação, como por exemplo o uso de recompensas como: telemóveis, saldo e dados móveis extra, acessórios, entre outros.

O mesmo está disponível através da aplicação MyVodafone (a área de cliente da Vodafone) e para jogar são precisos "Shakes", os quais se podem obter de diversas formas, como: ao abrir uma "Mystery Box" ou um "Prémio Joker"; ao partilhar o código de referência com outras pessoas; ao trocar cartas repetidas; ao fazer carregamentos de saldo e ao fazer o pagamento semanal do tarifário.

Para abrir um "Shake" o utilizador abana o telemóvel, recebendo assim uma carta. Depois, à medida que vai colecionando novas cartas, o objetivo passa por tentar completar várias categorias de uma caderneta. Sempre que é completada uma nova categoria, são atribuídos prémios ao jogador.

Dentro das funcionalidades com mais relevância para o contexto do estágio, destacam-se as seguintes:

- **Abrir um Shake** – por cada Shake, o utilizador recebe uma carta que poderá colar na sua caderneta, crescendo assim a sua coleção. Sempre que se recebe uma nova carta de coleção, o jogador tem de tentar adivinhar o nome da mesma respondendo a uma pergunta de escolha múltipla. Dependendo do tempo de resposta e se respondeu corretamente ou não, serão creditados pontos na conta do utilizador;
- **Consultar a caderneta** – esta funcionalidade permite que o utilizador visite a sua coleção e consiga ver quantas cartas tem colecionadas, quantas cartas lhe faltam para completar uma determinada categoria e também onde pode consultar detalhes sobre cada uma;
- **Troca de cartas repetidas** – neste ponto, o utilizador pode decidir trocar quatro cartas repetidas por um Shake, ou trocar qualquer carta repetida por mais pontos;
- **Troca de cartas com amigos** – a troca de cartas com amigos pode ser efetuada quando certas condições são atendidas, nomeadamente: as cartas transacionadas têm de ser repetidas; a carta enviada pelo amigo não pode existir na coleção do utilizador, e vice-versa;
- **Consultar o perfil** – aqui, o utilizador poderá consultar várias informações acerca da sua conta, como por exemplo a quantidade de Shakes que tem disponíveis ou o número de pontos que possui.

Este tipo de aplicações não só melhora o envolvimento do utilizador, como também garante a lealdade dos clientes, uma vez que os mesmos se sentem atraídos a toda a panóplia de recompensas que facilmente têm à sua disposição, divertindo-se em simultâneo.

2.1.4 Sumário

Por fim, dos exemplos mencionados anteriormente e do que foi explicado ao longo desta secção, conseguimos retirar alguns conceitos que nos permitem entender melhor as estratégias de gamificação usadas pelas empresas. Dentro dessas técnicas e estratégias de gamificação, estão incluídas as seguintes [57]:

- A **criação de sistemas de pontuação**, pois garantem que as pessoas se mantêm envolvidas, gerando um estímulo contínuo e competitivo. Quanto mais perto um jogador estiver de atingir o seu objetivo no jogo, maior o número de pontos;
- O **registo da evolução** de um utilizador é essencial uma vez que permite que as conquistas do mesmo sejam registadas e facilmente acessíveis;
- **Estabelecer níveis** é outra estratégia de gamificação bastante usada, dado que não faz sentido evoluir e permanecer no mesmo nível, isto é, os jogadores devem ser motivados a alcançar novos patamares dentro do jogo;
- **Elaborar metas secundárias e um objetivo principal** torna-se noutra técnica importante, pois mantêm o utilizador ativo e permitem que novas etapas sejam alcançadas gradualmente e de forma consistente;
- **Estimular a competição de forma saudável** é um outro incentivo para os participantes, como vimos no caso do Nike+ Run Club;
- A criação ou acumulação de **bens colecionáveis** é outra estratégia amplamente usada, e possui valor dada a unicidade do recurso em causa. Para o presente estágio, esta será uma das técnicas de gamificação adotadas, como iremos ver de seguida;
- Finalmente, uma das estratégias mais comuns é o incentivo através de **recompensas**, e estas poderão vir em diversos formatos, como vimos pelos exemplos mencionados anteriormente, seja através de medalhas, pontos, distintivos ou mesmo através de prémios físicos, como no caso do Shake It.

2.2 Blockchain

O conceito de *Blockchain* é relativamente recente e a introdução desta tecnologia deu-se em 2008, através de um artigo denominado de "Bitcoin: A Peer-to-Peer Electronic Cash System" publicado por Satoshi Nakamoto [11]. A Bitcoin é uma moeda digital (criptomoeda) cujas características e propriedades são alcançadas uma vez que a tecnologia que está na sua base é a *Blockchain*.

2.2.1 Descrição

A *Blockchain*, ou *Distributed Ledger Technology* (DLT), é uma tecnologia que permite criar registos que não podem ser adulterados e é a solução usada para, por exemplo, gerir criptoativos, garantir a validade de *Smart Contracts* ou impedir a falsificação de documentos.

Uma diferença importante entre uma base de dados típica e a *Blockchain* é a maneira como os dados são estruturados. Uma *Blockchain* recolhe informação em grupos, isto é, em blocos que contêm conjuntos de informações. Estes blocos, por sua vez, têm certas capacidades de armazenamento e, quando preenchidos, são fechados e ligados ao bloco previamente preenchido, formando assim uma cadeia de blocos [12] (Fig. 2.1). Esta ligação é assegurada através da *hash* de cada bloco, que é um identificador único criado usando criptografia, permitindo que um bloco recém-criado se possa ligar sequencialmente à *Blockchain*, guardando o *hash* do bloco anterior.

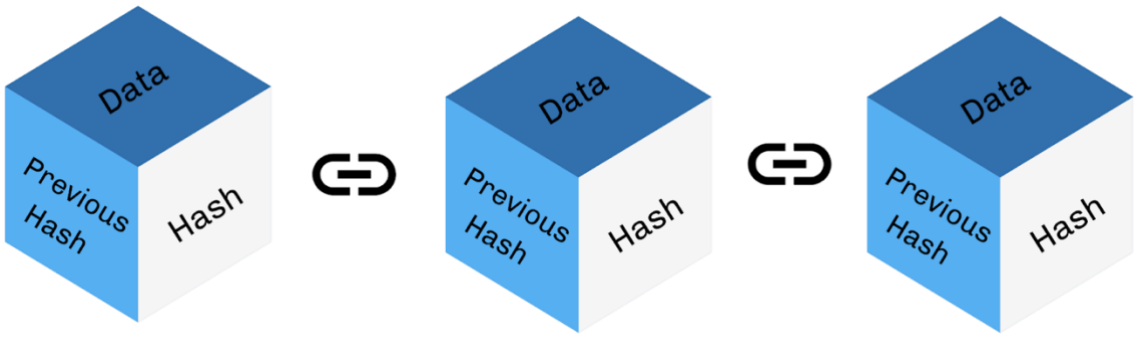


Figura 2.1 - Cadeia de blocos [13]

Desta forma, a *Blockchain* funciona como uma base de dados distribuída em que toda a informação é armazenada em blocos numa rede ponto a ponto (Fig. 2.2), permitindo assim a distribuição e descentralização da mesma, impedindo que uma entidade seja sua detentora e, por conseguinte, garantindo mais confiança, eficiência e segurança.

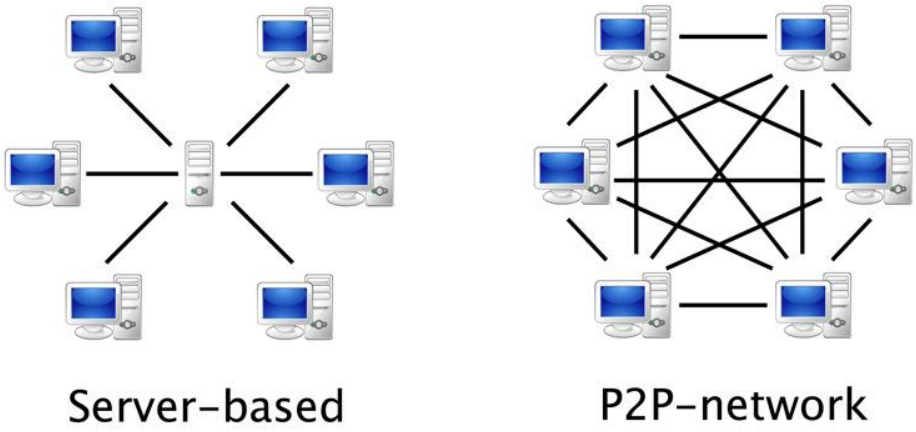


Figura 2.2 - Rede cliente-servidor (Server-based) versus rede ponto a ponto (P2P-network) [14]

A fim de compreender melhor o funcionamento da *Blockchain*, na Figura 2.3 são descritos todos os passos desde um pedido de transação até à sua conclusão.

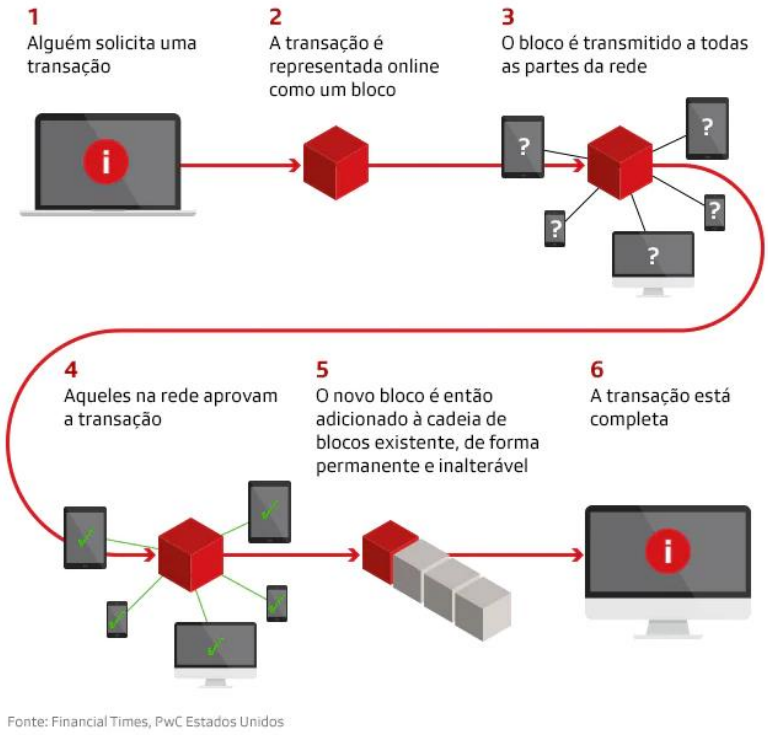


Figura 2.3 - Processamento de uma transação na *Blockchain* [15]

Algoritmos de consenso

Relativamente aos passos 4 e 5 da Figura 2.3, é importante mencionar a forma como é feita a validação de um bloco, para que o mesmo seja adicionado à *Blockchain*, de forma permanente e inalterável. Para este efeito, cada *Blockchain*, na sua criação, adota um mecanismo de consenso sob o qual irão ser coordenadas as diferentes validações na rede.

Estes algoritmos permitem que todos os membros (os nós da rede ponto a ponto) de uma determinada *Blockchain* possam concordar com uma única fonte de verdade, isto é, garantindo que chegam a um consenso.

Adiante, encontram-se brevemente descritos alguns dos modelos de consenso mais usados atualmente, para que possamos ter uma visão mais detalhada sobre o seu funcionamento:

- ***Proof of Work*** (PoW) – Este é o modelo mais popular, tendo sido adotado pela *Blockchain* da Bitcoin em 2008. O seu funcionamento resume-se ao seguinte: um utilizador (nó da rede) publica o próximo bloco se for o primeiro a resolver uma série de problemas computacionais. Depois, todos os outros nós na rede podem facilmente validar a solução encontrada para resolver esses problemas, autorizando a criação do próximo bloco e rejeitando blocos que não satisfizerem os problemas mencionados anteriormente. Após validar e publicar um novo bloco, o nó da rede que o publicou recebe uma recompensa (normalmente através de criptomoedas) de modo a compensar o trabalho que realizou. Por fim, é importante referir que este algoritmo, apesar de ser completamente descentralizado e seguro, exige grande poder computacional, pelo que tem um impacto negativo no ambiente devido à energia que requer. [78]

- ***Proof of Stake*** (PoS) – Em contrapartida, o PoS é um algoritmo de consenso que visa solucionar os problemas de escalabilidade e ambientais do PoW. Este mecanismo baseia-se na ideia de que quanto maior for o valor que um nó tem investido na rede, maior será o seu interesse em que a rede seja bem-sucedida. Ao contrário do PoW, o PoS não requer um recurso externo como eletricidade ou *hardware*, mas sim um interno, na forma de criptomoedas. Depois, as regras variam de acordo com o protocolo, mas geralmente o nó precisa de bloquear uma certa quantia de fundos por um determinado período de tempo, e este processo é chamado de "*Staking*". Através disto, o nó está apto para receber recompensas na rede e para publicar novos blocos na mesma, sendo que quanto maior for o valor investido, maior será a recompensa. [78]
- ***Proof of History*** (PoH) – O PoH assegura que uma *Blockchain* seja rápida e que ao mesmo tempo mantenha a segurança e a descentralização, e foi criado pela Solana [19], sobre a qual iremos falar na secção 2.2.4. Este algoritmo de consenso usa um método que incorpora o tempo em si na *Blockchain*, com o objetivo de reduzir a carga a que os nós da rede estão sujeitos, quando processam blocos. Numa *Blockchain* tradicional, obter consenso sobre o momento em que um bloco foi publicado é tão importante quanto obter consenso sobre as transações nesses blocos, dado que esse registo da data e hora informa a rede de que as transações ocorreram numa ordem específica. Através do algoritmo PoH, esta limitação causada pelo tempo é superada, tornando a *Blockchain* mais leve e rápida, reduzindo o peso do processamento [32]. No entanto, uma das desvantagens deste algoritmo é que para participar na rede como validador de blocos, é necessário que o nosso *hardware* atenda certos requisitos obrigatórios. Se não cumprirmos com essas especificações, somos excluídos do consenso. Isto limita a descentralização da Solana consideravelmente, visto que por exemplo no PoS, qualquer equipamento permite que um nó participe no consenso, tornando-se assim muito mais descentralizado [79].

Atualmente, existem milhares de *Blockchains* [12]. Este número irá continuar a aumentar à medida que esta tecnologia for adotada e adaptada a novos casos de uso. Existem também vários tipos de *Blockchains*, sendo estas classificadas como: públicas, privadas, híbridas, ou de consórcio (Fig. 2.4).

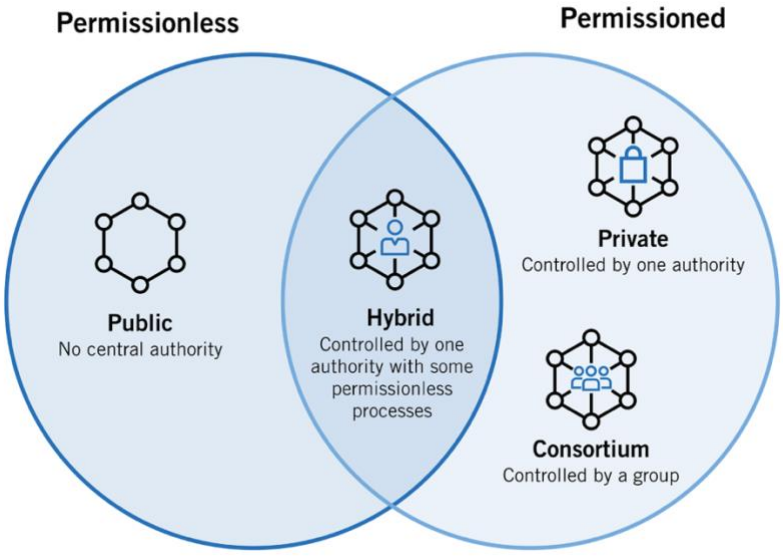


Figura 2.4 - Tipos de *Blockchains* [16]

Uma *Blockchain* pública, por exemplo, é independente uma vez que não tem nenhuma autoridade central. Uma *Blockchain* privada, por sua vez, é controlada por uma autoridade, e, portanto, não é descentralizada. Uma *Blockchain* híbrida combina alguns elementos de ambos os tipos mencionados anteriormente e, por fim, uma *Blockchain* de consórcio é semelhante a uma privada, mas em vez de ser controlada por uma entidade, é controlada por um grupo de organizações.

Na figura 2.5 são mencionadas algumas vantagens e desvantagens para cada um destes tipos de *Blockchains*, assim como alguns casos de uso.

	Public (permissionless)	**Private** (permissioned)	**Hybrid**	**Consortium**
ADVANTAGES	+ Independence + Transparency + Trust	+ Access control + Performance	+ Access control + Performance + Scalability	+ Access control + Scalability + Security
DISADVANTAGES	– Performance – Scalability – Security	– Trust – Auditability	– Transparency – Upgrading	– Transparency
USE CASES	■ Cryptocurrency ■ Document validation	■ Supply chain ■ Asset ownership	■ Medical records ■ Real estate	■ Banking ■ Research ■ Supply chain

Figura 2.5 - Vantagens, desvantagens e casos de uso dos 4 tipos de *Blockchains* [17]

Além disto, vários projetos públicos como o Ethereum [18], Solana [19] e Cardano [20], por exemplo, correm na sua própria *Blockchain*, como veremos na secção 2.1.3. No entanto isto nem sempre acontece, e existem projetos que são construídos noutras *Blockchains* que não a sua, normalmente em plataformas que suportem *Smart Contracts*. Isto inclui projetos como o Chainlink [21], DAI [22], USDC [23], OpenSea [24], Decentraland [25], entre outros [26], que correm na *Blockchain* pública do Ethereum, por exemplo.

A vantagem de correr noutra *Blockchain* que não a sua é a facilidade de configuração e o facto de conseguirem todos os benefícios associados à cadeia principal, como a segurança e escalabilidade.

2.2.2 Smart Contracts

Um *Smart Contract* [27] é um contrato digital composto por um conjunto de programas escritos em código, armazenados numa *Blockchain*. Os computadores ligados à rede ponto a ponto dessa *Blockchain* executam esse contrato quando condições predeterminadas são verificadas e atendidas. Através desta automatização, os *Smart Contracts* tornam-se a solução ideal quando se pretende garantir que os acordos são cumpridos, dado que não há intermediação por parte de uma empresa, governo ou entidade. Assim, todos os participantes podem ter a certeza imediata do resultado, sem envolvimento de terceiros ou perda de tempo.

Dado que todas as informações são registadas na *Blockchain* de forma pública e unívoca, para além de seguro, um *Smart Contract* é imutável, não podendo ser alterado após a sua criação. Ademais, o algoritmo alusivo ao contrato pode ser consultado por qualquer pessoa que tenha o endereço do mesmo, podendo esta garantir a fidedignidade do que foi acordado.

2.2.3 Non-Fungible Tokens

Um *Non-Fungible Token* (NFT) é um tipo de bem digital que, tal como o nome indica, não é fungível, isto é, não pode ser recriado, duplicado ou alterado após a sua criação. Estes atributos tornam-se possíveis através da *Blockchain*, onde é guardado um registo de quem possui um determinado NFT. O mesmo registo não pode ser falsificado uma vez que é assegurado por uma rede ponto a ponto composta por milhares de computadores em todo o mundo.

É também de notar que os *Smart Contracts* são fundamentais para o funcionamento de um NFT, pois garantem que a informação armazenada relativamente ao mesmo é imutável. Adicionalmente, é através dos *Smart Contracts* que conseguimos verificar quem é o proprietário de um determinado ativo, garantindo ainda que o mesmo não possa ser replicado. Assim, os *Non-Fungible Tokens* apenas podem ser criados em *Blockchains* que suportem o desenvolvimento de *Smart Contracts*.

Praticamente qualquer coisa pode ser representada digitalmente na *Blockchain* (Fig. 2.6), e o que dá valor a um NFT é a sua escassez e raridade, sendo que o mesmo pode ter uma finalidade intrínseca ou pode apenas servir como um bem colecionável.

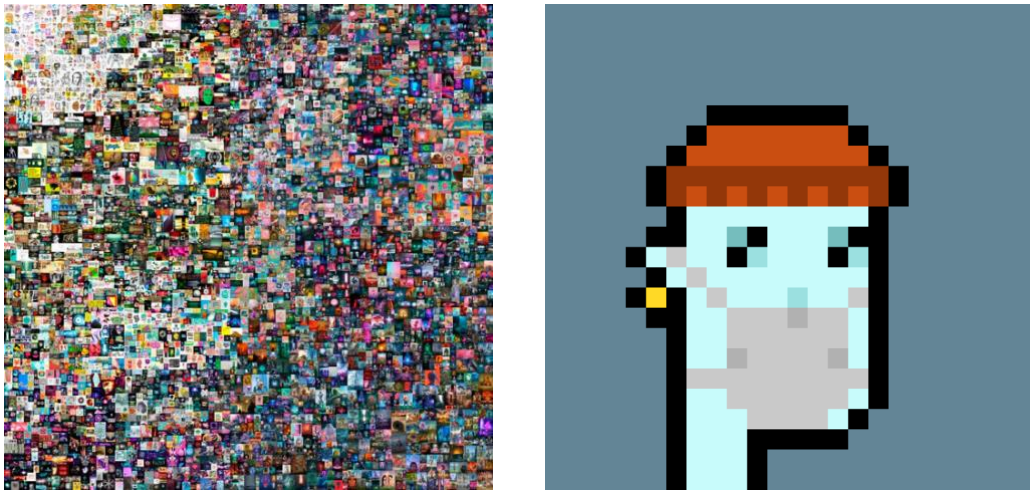


Figura 2.6 - Exemplos de NFT. Imagem da esquerda - Beeple's "Everydays: The First 5000 Days" [28]; Imagem da direita - CryptoPunk #7523 [29]

2.2.4 Armazenamento de NFTs

Quanto à maneira como os NFTs são guardados na *Blockchain*, existem alguns pontos que se têm de ter em conta, nomeadamente o tamanho dos dados que podem ser armazenados num bloco da *Blockchain*. Este é relativamente baixo, tornando difícil e dispendioso o armazenamento de uma grande quantidade de dados como uma imagem associada a um NFT, por exemplo. Indicativamente, em julho de 2020, quando as taxas na *Blockchain* do Ethereum eram significativamente mais baixas que em 2021, o preço para armazenar 1 MegaByte de uma imagem no Ethereum custava mais de 11,000 euros [58].

Assim, apesar *do Smart Contract* referente a cada NFT estar guardado, imutavelmente, na *Blockchain*, a imagem e toda a *metadata* (como o nome ou a descrição da imagem) associada ao mesmo está, tipicamente, armazenada noutro lugar, e na *Blockchain* apenas é guardado um identificador ou um *link* para esse conteúdo.

Por exemplo, um projeto poderá guardar a imagem de cada NFT num URL HTTP. Isto significa que se por algum motivo alguém alterar o conteúdo desse URL, a imagem associada àquele NFT será perdida [76].

Deste modo, surgiram soluções como o *InterPlanetary File System* [59] (IPFS) que permitem contornar este problema de segurança.

InterPlanetary File System (IPFS)

Este é um sistema de arquivos completamente distribuído, espalhado por uma rede ponto a ponto que conecta todos os dispositivos de computação ao mesmo sistema de arquivos. Desta forma, garante a disponibilidade e permanência dos mesmos, uma vez que permite a existência de várias cópias nos diferentes nós que suportam a rede (Figura 2.7).

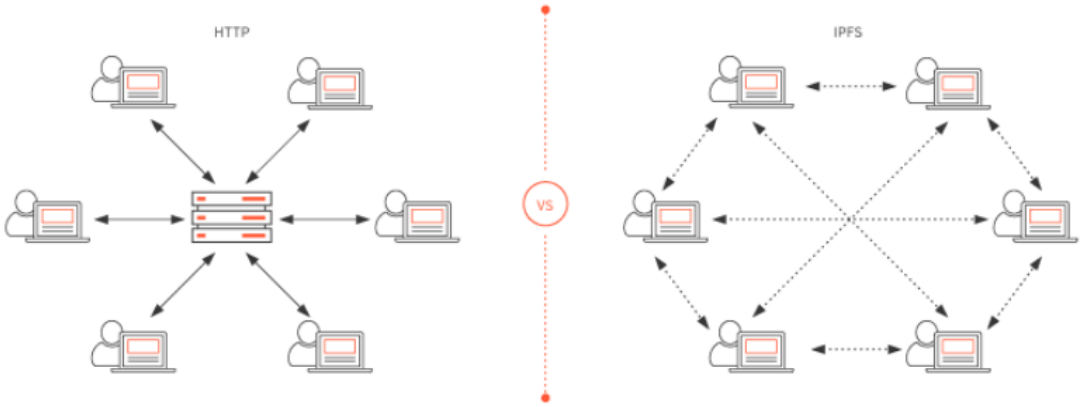


Figura 2.7 - HTTP versus IPFS [77]

Por exemplo, se os servidores da Wikipédia ficassem offline seria impossível procurar por um ficheiro. Com o IPFS, como este é totalmente descentralizado, toda a informação que lá reside é acessível a qualquer momento.

No caso dos NFTs em que a arte digital associada aos mesmos esteja armazenada num servidor centralizado, um mecanismo de armazenamento como o IPFS seria a solução ideal para a descentralização total destes.

Ao guardar a metadata de um NFT no IPFS é atribuído um CID (*content identifier*), o qual não pode ser modificado (ao contrário de um URL).

Por exemplo para o NFT criado por Beeple (Figura 2.6), a metadata associada ao mesmo está guardada no seguinte link:

https://ipfs.io/ipfs/QmPAg1mjxcEQPPtqsLoEcauVedaeMH81WXDPvPx3VC5zUz

Neste caso, o CID (isto é, o identificador único) é a *string* "QmPAg1mjxc EQPPtqsLoEcauVedaeMH81WXDPvPx3VC5zUz".

A partir deste, podemos visualizar a imagem original do NFT, que está sob o atributo "imageUrl", com o seguinte link:

https://ipfsgateway.makersplace.com/ipfs/QmZ15eQX8FPjfrtdX3QYbrhZxJpbLpvDpsgb2p3VEH8Bqq

Por fim, recapitulando, um NFT (e o respetivo *Smart Contract*) é como uma assinatura única ou a prova de propriedade do recurso que representa. Portanto, apesar de conseguirmos aceder

à imagem acima e guardá-la no nosso computador, o proprietário legítimo da mesma é quem está especificado no *Smart Contract* do NFT como "*owner*".

Carteiras Digitais

Quanto ao armazenamento do *token* em si após a aquisição de um NFT, existem carteiras digitais que servem para esse propósito. Quando se cria uma carteira nova, é gerada uma chave pública e uma chave privada. Esta última deve ser guardada e não partilhada com entidades desconhecidas. A chave pública, por sua vez, é para ser partilhada com pessoas com quem desejamos efetuar uma transação.

Estas carteiras podem ser de 2 tipos:

- ***Custodial Wallet***: é definida como uma carteira na qual as chaves privadas são mantidas por terceiros.
- ***Non-custodial Wallet***: é um tipo de carteira que permite que os utilizadores tenham controlo total sobre os seus fundos e a chave privada associada. Se o titular dessa carteira perder a chave privada, perde o acesso aos seus fundos.

2.2.5 Plataformas de *Blockchain*

Como foi referido anteriormente, atualmente já existem milhares de *Blockchains* [12], cada uma com propriedades e casos de uso diferentes. As maiores diferenças encontram-se predominantemente na descentralização, escalabilidade e segurança.

Adiante são apresentadas algumas das *Blockchains* estudadas, dentro das que suportam *Smart Contracts* e *Non-Fungible Tokens*, e são exploradas certas características inerentes a cada uma delas, assim como pontos fortes e fracos.

Ethereum

O Ethereum [18] foi lançado em 2015, e é uma *Blockchain* pública que possibilita o desenvolvimento de aplicações descentralizadas, *Smart Contracts*, e transações com a criptomoeda Ether (ETH), sem o controlo ou interferência de terceiros. Atualmente, a Solidity é a principal linguagem de programação usada no Ethereum, e a mesma é usada maioritariamente para escrever *Smart Contracts* (Figura 2.8).

HelloWorld.sol

```
1  // My First Smart Contract
2
3  pragma solidity >=0.5.0 <0.7.0;
4
5  contract HelloWorld {
6    function get()public pure returns (string memory) {
7      return 'Hello World';
8    }
9  }
```

Figura 2.8 - Exemplo de um *Smart Contract* programado em Solidity

É também de notar que o Ethereum possui características que o diferenciam de outras plataformas de *Smart Contracts*. A primeira, e como foi mencionado anteriormente, a descentralização é uma das grandes vantagens desta *Blockchain*, não havendo uma entidade que seja sua detentora. Além disso, o Ethereum tem a maior comunidade de programadores do mundo (dentro do ecossistema das *Blockchains*) [30], dando-lhe uma maior vantagem sobre outras plataformas. A interoperabilidade, por sua vez, é uma outra característica desta plataforma, sendo que sempre que é desenvolvida uma aplicação no Ethereum, é possível conectá-la instantaneamente a centenas de outros protocolos já existentes, tornando o seu desenvolvimento mais fácil, rápido e eficiente, ao invés de começar do zero.

Em contrapartida, as velocidades são muito lentas em comparação à enorme procura, o que causa um congestionamento da rede e os preços das taxas aumentam de forma desmedida, tornando-se pouco económico para as pessoas que continuam a usar esta *Blockchain*.

Solana

A Solana [19], por sua vez, é uma *Blockchain* pública descentralizada que, tal como o Ethereum, consegue interagir com *Smart Contracts*. Todavia, ao contrário do que acontece com este, a Solana é bastante superior no que toca à escalabilidade, sendo este um dos fatores que mais se destaca dentro deste sistema. Esta é uma das suas qualidades principais e é atualmente uma das melhores soluções para o problema da escalabilidade na *Blockchain*. Isto é conseguido através da combinação do algoritmo de consenso *Proof of History* (PoH) [32] mencionado na secção 2.2.1 com o mecanismo Tower BFT (*Byzantine Fault Tolerant consensus*) [31], que é um sistema de segurança que permite que os utilizadores dêem *Stake* dos seus *tokens* para poderem votar na validade de uma *hash* do PoH.

O custo associado a cada transação e a velocidade de processamento são também significativamente mais baixos comparativamente ao Ethereum, por exemplo. A rede da Solana consegue processar um máximo de 60,000 transações por segundo (TPS) [33] com um custo médio por transação de 0.00022€ [34], enquanto que o Ethereum apenas atinge cerca de 15 TPS [35], com um custo médio por transação de 23.33€ [36].

Cardano

A Cardano [20] é um projeto de *Blockchain* que é *open-source* e descentralizado, e possibilita o desenvolvimento de aplicações descentralizadas e integração com *Smart Contracts*.

O seu foco encontra-se no desenvolvimento de uma rede descentralizada eficiente, escalável e segura, através de uma abordagem sistemática sobre a pesquisa e desenvolvimento da *Blockchain*. Além disso, a Cardano utiliza um algoritmo de consenso de *Proof of Stake* (PoS), o qual necessita de menos poder computacional que o algoritmo usado atualmente pelo Ethereum, por exemplo, que é o *Proof of Work* (PoW). Desta forma, para além da poupança na energia, o algoritmo PoS propicia uma *Blockchain* com mais potencial no que toca à escalabilidade. No entanto, é menos seguro que o PoW, que é totalmente descentralizado [37].

Em termos de velocidade das transações, a Cardano ainda tem muito espaço para melhorias. Atualmente, a sua rede consegue realizar cerca de 257 TPS [38], mas o seu objetivo é, no futuro, atingir 1 milhão de transações por segundo [39].

Quanto à integração desta *Blockchain* com *Smart Contracts*, isto é algo recente, tendo sido lançado a 12 de Setembro de 2021 [40]. A linguagem de programação usada para escrever estes contratos é o Haskell, a qual não é adotada por muitas *Blockchains* como linguagem dos *Smart*

Contracts. E ainda, dado que atualmente existem poucas aplicações a serem construídas na Cardano, há poucas referências para o desenvolvimento de *Smart Contracts* em Haskell [41].

Cosmos

A Cosmos [42] é um projeto que visa construir um sistema de interoperabilidade de *Blockchains* descentralizado, rápido, seguro e barato, para expandir as capacidades desta tecnologia. Assim, o seu foco passa por fornecer uma infraestrutura capaz de unir diversas *blockchains* independentes e torná-las interoperáveis, com a intenção de permitir que o valor e as aplicações contidas nessas redes possam comunicar entre si de forma descentralizada e segura.

Isto torna-se possível através do protocolo *Inter-Blockchain Communication* (IBC) [43]. O IBC permite a comunicação entre *blockchains* díspares, conectando-as através da *Cosmos Hub* e, consequentemente, tornando-as interoperáveis. A *Cosmos Hub* foi a primeira *blockchain* a ser lançada na rede Cosmos, e foi construída com o intuito de servir de intermediário entre todas as *blockchains* independentes criadas na rede Cosmos. Atualmente, já existem mais de 20 *Blockchains* [44] construídas no Cosmos SDK [45] - uma *framework* usada para construir *Blockchains* - com o IBC, como por exemplo a Osmosis [46], Cronos [47] e Terra [48].

Polygon

Uma das plataformas que explora a interoperabilidade do Ethereum é a Polygon [49], que é uma *Sidechain* da rede do Ethereum que visa resolver as limitações desta *Blockchain*, sem sacrificar a segurança e a descentralização. Estas limitações incluem a baixa escalabilidade e os custos elevados da *Blockchain* do Ethereum.

Uma *Sidechain* é uma *Blockchain* separada, no entanto não é uma plataforma independente, pois está conectada à cadeia principal (Fig. 2.9), que neste caso é o Ethereum. A *Blockchain* principal e a *Sidechain* são interoperáveis, o que significa que os ativos podem fluir livremente de uma para a outra.

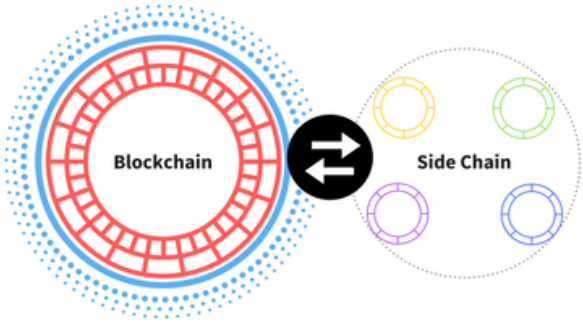


Figura 2.9 - *Sidechain* [50]

Sendo uma *Sidechain*, a Polygon permite que inúmeras *Blockchains* e aplicações descentralizadas aproveitem os recursos do Ethereum sem sofrer com as velocidades de transação lentas e taxas altas do mesmo, sendo que a rede da Polygon consegue processar até 65,000 TPS [51]. Além disso, a linguagem de programação usada para escrever os *Smart Contracts* é a mesma do Ethereum. Isto é positivo uma vez que a Solidity foi uma das primeiras linguagens de programação a ser usada no desenvolvimento de *Smart Contracts* [41], contando assim com várias referências e com uma documentação bem estabelecida.

Tezos, Stellar e Polkadot

As Blockchains Tezos [52], Stellar [53] e Polkadot [54] são outros exemplos de redes que suportam o desenvolvimento de *Smart Contracts*. No entanto, cada uma conta ainda com um conjunto de características que as diferencia das restantes.
A rede Tezos, por sua vez, por ser uma *Blockchain* descentralizada, não só oferece todos os benefícios convencionais como também traz uma série de recursos exclusivos, entre eles a sua capacidade de auto-correção. Esta propriedade permite que a rede Tezos atualize automaticamente o seu protocolo sem passar por um *hard fork*, isto é, sem dividir a *Blockchain* principal em duas *Blockchains* separadas, que correm paralelamente uma com a outra. Isto é crucial para as *Blockchains*, uma vez que os *hard forks* podem afastar a comunidade, fragmentar os recursos, alterar as recompensas das partes interessadas e diluir todos os benefícios acumulados ao longo do tempo. Assim, a auto-correção permite a coordenação e implementação contínuas de mudanças e atualizações futuras. Em termos de escalabilidade, a rede Tezos é capaz de executar cerca de 40 TPS [55].
Quanto à *Blockchain* da Stellar, esta funciona como uma infraestrutura de pagamentos, onde duas entidades são conectadas para realizarem as suas transações financeiras de forma rápida, segura e barata. Uma grande vantagem desta plataforma é a velocidade das transações, e quando um indivíduo efetua uma transferência, internacional ou não, ela demora apenas entre 2 e 5 segundos para ser completada [56].
Por fim, em relação à Polkadot, esta é uma *Blockchain* que une múltiplas outras *Blockchains*, cada uma projetada com um propósito específico, juntando-as numa rede unificada. Possibilita assim um ambiente em que as inúmeras *Blockchains* possam comunicar entre si, com segurança e confiança. Assim, a interoperabilidade e escalabilidade tornam-se o foco deste sistema, sendo neste aspeto que a Polkadot se destaca das restantes plataformas.
Além disso, de forma semelhante ao Tezos, a Polkadot também possui a capacidade de atualizar a sua rede, sem ter que recorrer a um *hard fork*.
Com o passar do tempo, à medida que vão sendo integradas novas *Blockchains* nesta rede e à medida que as mesmas vão sendo otimizadas, espera-se que daqui a uns anos as transações nesta plataforma cheguem a velocidades de 1 milhão de TPS.

2.2.6 Sumário

Em síntese, na Tabela 2.1 são apresentadas algumas das características das plataformas de *Blockchain* mencionadas anteriormente. Os valores apresentados [112] verificam-se até à data de escrita do presente relatório (Julho de 2022).

Tabela 2.1 - Comparação entre as *Blockchains* estudadas

Blockchain	Smart Contracts	Principais linguagens de programação	Transações por segundo	Taxa média por transação (€)
Ethereum	✓	Solidity	15	23.33
Solana	✓	Rust	60,000	0.00022
Cardano	✓	Haskell	257	0.3
Cosmos	✓	Ethermint	10,000	0.0088
Polygon	✓	Solidity	65,000	0.00012
Tezos	✓	Michelson	40	0.0088
Stellar	✓	JavaScript, Java, Go	1,000	0.000033
Polkadot	✓	Rust	1,000	-

Seleção da rede Blockchain a ser usada no projeto

Dentro das *Blockchains* estudadas, a que parece mais promissora face aos requisitos do presente estágio é a **Polygon**.

A equipa optou por selecionar esta plataforma uma vez que a mesma oferece limites de escalabilidade relativamente altos quando comparado às outras *Blockchains*, mantendo a segurança e a descentralização da rede. Em adição, as taxas associadas a cada transação são significativamente baixas quando comparadas às do Ethereum, por exemplo.

Para além disto, dado que a Polygon é uma *sidechain* do Ethereum, a documentação quanto ao desenvolvimento de aplicações neste ambiente é relativamente extensa. Este foi um fator que pesou bastante na decisão final, dado que o aluno não tinha conhecimento prévio sobre o desenvolvimento com tecnologias *Blockchain* ou com a linguagem de programação Solidity.

2.3 O papel da *Blockchain* na gamificação

Depois de analisado o conceito de gamificação e as estratégias mais comummente usadas, assim como a *Blockchain* e os tópicos inerentes à mesma, é pertinente assumir que esta é a tecnologia ideal para usar em conjunto com a gamificação.

Nesta secção é feita uma descrição de como a *Blockchain* é vantajosa para a gamificação e são apresentados alguns exemplos de aplicações movidas através destes 2 conceitos.

2.3.1 Vantagens que a Blockchain traz para a gamificação

A *Blockchain* oferece inúmeras vantagens em contextos de gamificação e graças às suas características de imutabilidade, transparência e interoperabilidade, entre outras, abre caminho para novas e mais eficazes estratégias de gamificação. Dentro dessas vantagens, estão incluídas as seguintes [57]:

- A **imutabilidade**, por sua vez, garante que nada pode ser adulterado por nenhuma das partes envolvidas. É uma característica importante uma vez que oferece mais segurança

aos utilizadores, pois os mesmos ficam assegurados de que não perderão os recursos ou reconhecimento adquiridos.

- A **transparência** é outra propriedade fundamental, uma vez que ao tornar os mecanismos de gamificação claros, transparentes ou até mesmo codificados por meio de *Smart Contracts*, os utilizadores tendem a confiar mais e, consequentemente, a investir mais recursos em termos monetários e tempo.
- Ao trazer a gamificação para a *Blockchain*, a **interoperabilidade** torna-se outro recurso essencial, por exemplo em situações onde os ativos adquiridos num determinado contexto têm valor quando transportados para outros contextos semelhantes.
- A **autoridade** tem também um papel importante, uma vez que a informação reportada, se devidamente assinada por organizações e entidades reconhecidas, é indiscutível. Por outras palavras, é importante assegurar às partes que não se conhecem que as informações declaradas na *Blockchain* são confiáveis.

Em síntese, a tecnologia *Blockchain* torna-se vantajosa em contextos de gamificação quando maximiza as características listadas acima, entre outras propriedades inerentes ao funcionamento da mesma.

2.3.2 Exemplos de plataformas de gamificação movidas pela tecnologia *Blockchain*

Atualmente, já existem vários serviços que exploram o potencial da gamificação e da Blockchain, introduzindo-os lado a lado para criar produtos inovadores, capitalizando as características convidativas de um e a confiabilidade e segurança do outro.

Aqui são apresentadas algumas dessas plataformas que contribuem para o desenvolvimento de soluções gamificadas usando a tecnologia *Blockchain*, relembrando que as estratégias de gamificação usadas servem, principalmente, para:

- aumentar o envolvimento dos utilizadores e, consequentemente, a sua lealdade e reconhecimento da marca;
- incentivar os utilizadores a atingir metas e a comprar e a gastar mais;
- construir uma comunidade com os seus utilizadores.

Socios.com

A aplicação Socios.com [60] é uma plataforma que alavanca a tecnologia *Blockchain* para propiciar às maiores organizações desportivas a nível mundial as ferramentas necessárias para interagir com as suas comunidades de fãs. A mesma é alimentada pela criptomoeda Chiliz (CHZ), um *token* na *Blockchain* do Ethereum, o qual é usado para comprar *Fan Tokens*. Estes, por sua vez, são ativos digitais associados a uma equipa ou organização desportiva, como por exemplo os clubes de futebol: Juventus FC, Paris Saint-Germain FC, Club Atlético de Madrid, seleção portuguesa de futebol, entre outros.

Ao comprar um *Fan Token*, os fãs têm a possibilidade de participar em votações oficiais do seu clube preferido - onde a quantidade de *tokens* pesa no voto - ou de aceder a outras

funcionalidades como competir para receber experiências VIP, mercadoria exclusiva, bilhetes, etc. Os *Fan Tokens* são limitados em número e são fungíveis, e são eles que representam o direito que um adepto tem de se envolver com a sua equipa favorita. Estes *tokens* podem ser negociados e o seu preço é determinado pelo mercado.

Assim, através da junção da *Blockchain* com a gamificação, a aplicação Socios.com proporciona uma nova forma de os adeptos terem um maior envolvimento com os clubes que decidirem apoiar.

CryptoKitties

Existem ainda projetos como o CryptoKitties [61], um jogo digital criado em 2017 pelo estúdio canadense Dapper Labs, que consiste na compra, venda e criação de diferentes tipos de gatos virtuais - semelhante ao Tamagotchi [62], um brinquedo lançado na década de 1990 que consistia em cuidar de um animal virtual.

O jogo utiliza a *Blockchain* do Ethereum, uma vez que o mesmo está assente em *Smart Contracts* - para provar e rastrear o proprietário de um determinado ativo -, e em *Non-Fungible Tokens* (NFT) - para representar cada ativo, neste caso, um gato virtual -, e todas as negociações e transações são feitas através da criptomoeda Ether.

Depois de criar uma conta, para começar a jogar é preciso ter uma carteira digital, a qual é identificável através de um endereço público e único. Depois, é enviada para essa carteira a quantia de Ether necessária para comprar o CryptoKitty que se deseja. Para efetuar transações de compra e venda, existe um mercado dedicado a este propósito, onde se pode procurar por milhares de gatos virtuais, cada um com características diferentes. Uma vez que se tratam de bens colecionáveis, o preço de cada gato varia consoante o que o vendedor estiver disposto a negociar. No entanto, há gatos mais procurados que outros, devido à sua raridade, utilidade e aparência, sendo estes alguns dos aspetos que têm influência no preço.

Outra maneira de obter um CryptoKitty é através da reprodução entre dois gatos, sendo que quantos mais se criarem, maior a desvalorização dos gatos parentes. Depois de cada "nascimento" existe um tempo de espera até que os gatos se possam reproduzir outra vez, e o mesmo vai aumentando de gravidez para gravidez. Quanto às possíveis CryptoKitties que podem existir, há milhões de combinações genéticas possíveis, nunca havendo dois gatos iguais.

Em 2017, em apenas 15 dias após o lançamento do jogo, o CryptoKitties já contava com cerca de 150 mil utilizadores registados, 15 milhões de dólares em transações e mais de 260 mil gatos em existência [63]. Ademais, em apenas 8 dias após o lançamento, foi também responsável por 25% do tráfego na rede Ethereum, o que causou um congestionamento na mesma, devido ao problema da escalabilidade desta *Blockchain*.

Atualmente, os dados dos últimos 30 dias até 6 de Outubro de 2021 indicam que foram transacionados aproximadamente 9.5 milhões de dólares, em cerca de 7000 vendas [64].

Gods Unchained

Lançado em setembro de 2020, o Gods Unchained [65] é um jogo de cartas gratuito que combina componentes de NFTs com outras características de jogos tradicionais de troca de cartas, como o Yu-Gi-Oh! [66] ou o Pokémon Trading Card Game [67].

O jogo é bastante competitivo e os jogadores devem pensar estrategicamente de maneira a derrotar o oponente, construindo um baralho com cartas capazes de combater uma ampla variedade de táticas e habilidades. Ao jogar o jogo vão-se recebendo vários pacotes de cartas digitais, as quais não têm valor monetário, no entanto, dependendo do modo de jogo, um jogador pode receber cartas colecionáveis únicas - um NFT -, as quais poderá posteriormente vender no mercado da plataforma. Como o jogo foi construído na *Blockchain* do Ethereum, estas transações de compra e venda são feitas através da criptomoeda Ether. Dado que estas cartas não são fungíveis, o proprietário de cada carta tem a liberdade de trocar, comprar ou vender a mesma da forma que entender, tal como aconteceria na vida real, com cartas tangíveis.

Zed Run

O Zed Run [68] é uma aplicação distribuída que está assente na rede Ethereum e *usa Smart Contracts* para permitir que os utilizadores comprem, vendam, criem e compitam com cavalos digitais estatisticamente únicos. Baseia-se num jogo de corridas de cavalos onde cada cavalo é representado por um NFT.

Para jogar, os utilizadores criam o seu próprio estábulo de cavalos ao comprá-los no mercado da plataforma. Ao ganhar uma corrida, os jogadores podem ganhar lucros substanciais e podem inclusive vender os seus NFT em troca de Ether.

Cada cavalo tem características distintas, em que cada uma influencia o seu comportamento e *performance* na pista de corrida, sendo este um dos fatores determinantes no preço de um desses NFT. Até à data, desde o lançamento do jogo, houve um número total de transações de 145 870 e as vendas totais rondam os 115 milhões de dólares [69]. O cavalo mais caro foi vendido por 43 ETH (Ether), que atualmente equivale a 193 500 dólares [70].

Capítulo 3
Metodologia e Planeamento

Neste capítulo é abordada a metodologia que foi adotada durante a realização deste projeto assim como todo o planeamento e a análise de riscos do mesmo.

Deste modo, encontra-se dividido em três subsecções. A primeira apresenta a metodologia usada, descrevendo as suas características e outros conceitos inerentes à mesma.

A segunda, por sua vez, expõe o planeamento definido pela equipa do projeto, para ambos os semestres. Adicionalmente, é apresentada uma análise para demonstrar se o que estava previsto inicialmente foi cumprido na totalidade ou se foi preciso fazer alguma adaptação relativamente ao que estava projetado.

Por fim, na terceira secção é apresentada a gestão de riscos para este projeto.

3.1 Metodologia

A gestão de um projeto é essencial uma vez que estabelece as metas, objetivos e princípios a adotar durante a fase de desenvolvimento do mesmo, até obter o produto final. Como tal, definir previamente uma metodologia que melhor se adeque às necessidades do projeto é imprescindível para que o mesmo seja bem-sucedido.

Dado que na empresa WIT Software a metodologia de trabalho mais usada é o *Scrum*, o aluno foi informado a priori que seria essa a metodologia a seguir no decorrer do estágio curricular. Como tal, a próxima subsecção tem como objetivo explicar o funcionamento desta técnica e a forma como foi adaptada ao presente estágio, dada a dimensão da equipa e a carga horária no primeiro semestre.

3.1.1 *Scrum*

O *Scrum* é um tipo de metodologia ágil [71] utilizado no desenvolvimento de *software*, e é baseado num processo iterativo e incremental. Foi desenvolvido na década de 1990 [72] e é atualmente uma das *frameworks* ágeis mais populares (Figura 3.1).

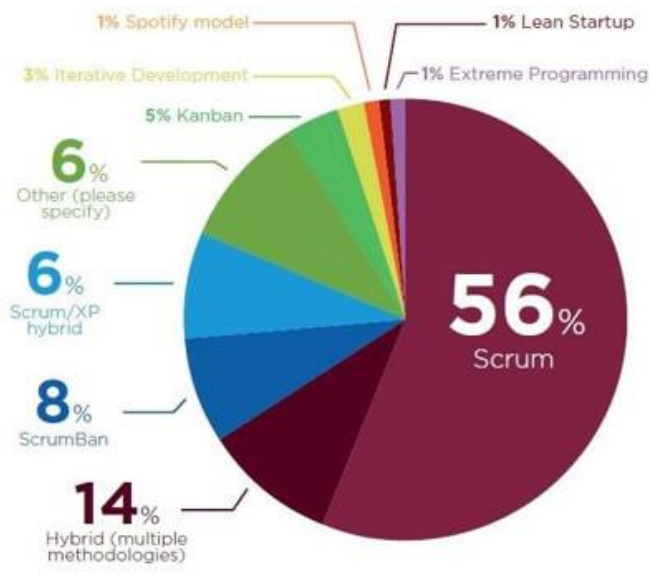


Figura 3.1 - As metodologias ágeis mais comummente usadas. [73]

Dadas as suas características de adaptabilidade, flexibilidade e eficácia, o *Scrum* é projetado para oferecer valor ao cliente durante todo o desenvolvimento do projeto. Isto é, o seu maior objetivo é satisfazer as necessidades do cliente através de um ambiente de transparência na comunicação, responsabilidade e progressos contínuos. O desenvolvimento parte de uma ideia geral do que precisa de ser construído, elaborando uma lista de requisitos ordenados por prioridade (*backlog*) que o proprietário do produto deseja obter. Assim, a metodologia *Scrum* está assente em 3 fundamentos importantes, são estes:

- **Transparência**: toda a informação do produto deve ser facilmente acessível por todos os elementos da equipa, e deve ser usada uma linguagem leiga, sendo a comunicação um dos elementos chave desta metodologia.
- **Inspeção**: os artefactos e produtos do *Scrum* devem ser regularmente e diligentemente inspecionados, de forma a garantir o máximo de qualidade possível.
- **Adaptação**: sempre que uma inspeção mostra algo que se desvie do padrão de qualidade esperado, a equipa deve fazer os ajustes e correções necessárias.

Papéis no *Scrum*

De forma a respeitar os valores acima, a metodologia *Scrum* requer que sejam definidos certos papéis e responsabilidades, incluindo os seguintes:

- ***Product Owner***: é o elemento que possui a autoridade máxima sobre o produto final, sempre com os interesses do cliente em mente. É também a pessoa responsável por garantir que os requisitos, funcionalidades e prioridades são entendidas e alcançadas, e mantém o *backlog* atualizado.
- ***Scrum Master***: atua como líder da equipa, sendo responsável por organizar as reuniões diárias, melhorar as interações entre os elementos da equipa e maximizar a produtividade.

- **Equipa de Desenvolvimento**: esta equipa inclui todas as pessoas necessárias para desenhar, produzir, testar e lançar o produto final, de acordo com as funcionalidades descritas no *backlog*.

Eventos e Artefactos do *Scrum*

Por fim, o *Scrum* dispõe de uma terminologia única, usada para descrever eventos e artefactos durante o desenvolvimento do projeto. Os elementos mais relevantes no contexto do presente estágio são: o ***product backlog***, que é uma lista com todas as tarefas e requisitos que têm de ser incluídos no produto final; o ***sprint***, que é um período de tempo - tipicamente, 2 semanas, mas pode variar consoante as necessidades do projeto e da equipa - em que se tem que completar um certo número de tarefas do *backlog*; e, por fim, o ***daily scrum***, em que a equipa do projeto se reúne todos os dias com o intuito de discutir sobre qualquer progresso conseguido no dia anterior, o progresso que é esperado obter no dia atual e qualquer problema que tenha sido encontrado.

Planeamento do *Scrum*

Para garantir o sucesso de cada *sprint*, é essencial fazer um planeamento coeso e estruturado. Assim, no início de cada *sprint*, a equipa deve reunir-se para delinear as tarefas que vão ser implementadas durante o período do mesmo, tendo em conta as prioridades definidas pelo *product owner*.

Para o presente estágio, cada *sprint* teve a duração de 4 semanas, aproximadamente. Durante este tempo, e dado que no primeiro semestre o aluno trabalhou com a empresa em tempo parcial, as *daily scrums* foram feitas 2 vezes por semana. Depois, a partir do segundo semestre, foram realizadas 5 vezes por semana.

No fim de cada *sprint* o autor realizou uma apresentação na empresa, onde o objetivo era descrever todos os avanços conseguidos durante esse sprint, e também os problemas encontrados. O *feedback* recebido nessas reuniões mostrou ser valioso, e permitiu que o aluno melhorasse certos aspetos do projeto.

3.1.2 Comunicação

Para que fosse feito um acompanhamento rigoroso e para possibilitar a troca de ideias e documentos entre os elementos da equipa, foram utilizadas as seguintes ferramentas:

- **Zoom**: plataforma usada para reunir remotamente.
- **Rocket.Chat** e **Skype**: aplicações para enviar e receber mensagens entre os elementos da equipa e da empresa.
- **Webmail**: para troca de documentos ou marcação de reuniões.

3.2 Planeamento

Nesta secção é apresentado o planeamento delineado para ambos os semestres, e encontra-se dividida em 4 subsecções.

Nas duas primeiras subsecções, são listadas as tarefas propostas para cada semestre, definidas pela equipa do projeto.

Na terceira subsecção, é dada uma visão geral do planeamento esperado para ambos os semestres, recorrendo a um diagrama de *Gantt*.

Por fim, na última subsecção, é apresentado o cronograma obtido no final do estágio e é feita uma análise do mesmo, comparativamente ao que estava previsto.

3.2.1 Primeiro semestre

No decorrer do primeiro semestre do presente estágio, as seguintes etapas eram esperadas:

- **Contextualização** – O primeiro passo para este estágio era definir o contexto, as motivações e os objetivos do projeto, para assim perceber o desafio proposto e poder idealizar maneiras de o solucionar. Esta parte está disponível no Capítulo 1;
- **Investigação** – Para esta etapa era esperado o levantamento do Estado da Arte, tarefa a qual foi fundamental para o projeto. Conceitos como a Gamificação, *Blockchain*, *Smart Contracts* e *Non-Fungible Tokens*, são alguns dos temas que devem ser dominados durante a realização desta tarefa, para poder compreender e desenhar a solução do projeto. Esta pesquisa está documentada no Capítulo 2;
- **Metodologia e planeamento** – Esta tarefa consistiu na elaboração de um plano de trabalho viável a seguir em ambos os semestres, e inclui a gestão de riscos e a metodologia adotada pela equipa. Toda esta preparação pode ser consultada no presente capítulo;
- **Requisitos** – Para esta tarefa era esperada a definição dos requisitos funcionais e os respetivos casos de uso, e isto pode ser visto no Capítulo 4;
- **Arquitetura** – Quanto à arquitetura de *software*, era também esperado no primeiro semestre o desenho da arquitetura proposta e a seleção da *Blockchain* a ser usada na implementação do protótipo. Isto pode ser consultado no Capítulo 5;
- **Relatório intermédio** – Por fim, e desenvolvido em paralelo com as etapas anteriores, esperava-se para o primeiro semestre a escrita da documentação intermédia da tese.

3.2.2 Segundo semestre

Para o segundo semestre, o planeamento previamente delineado contava com as seguintes etapas:

- **Requisitos** – Para esta tarefa, pretendia-se que fossem definidos os requisitos não funcionais e os critérios de sucesso, e foi acrescentado ao Capítulo 4;
- **Arquitetura** – Para esta tarefa era necessário definir as tecnologias e ferramentas a serem usadas na implementação do projeto, e encontra-se no Capítulo 5;
- **Desenvolvimento** – Esta é uma das fases mais importantes e extensas do semestre e inclui: o tempo de aprendizagem e de adaptação às tecnologias a usar; o *setup* do projeto; a elaboração do *Smart contract*; a implementação do *Backoffice* e do *Web Client*; e a implementação do servidor API responsável por comunicar com a *Blockchain*. Toda a documentação referente a esta etapa pode ser consultada no Capítulo 6;
- **Testes e Validações** – Depois de concluída toda a implementação, passaremos à fase de testes, onde será avaliada a robustez do produto obtido através de testes funcionais, testes unitários e da avaliação dos requisitos não funcionais. Qualquer resultado inesperado durante esta etapa deve ser resolvido. Isto pode ser consultado no Capítulo 7;
- **Relatório final** – Por fim, a documentação final da presente tese deve ser desenvolvida em paralelo com as etapas previamente mencionadas, e deve abordar todas as decisões que foram tomadas, desafios enfrentados, soluções e trabalho futuro. Em adição, o relatório deve ser revisto tanto pelos orientadores da empresa como pelo orientador da Universidade.

3.2.3 Visão geral do planeamento

Na Figura 3.2, presente no Anexo A.1, recorrendo a um diagrama de *Gantt* e seguindo as etapas das subsecções 3.2.1 e 3.2.2, é apresentado o planeamento esperado durante o período do estágio.

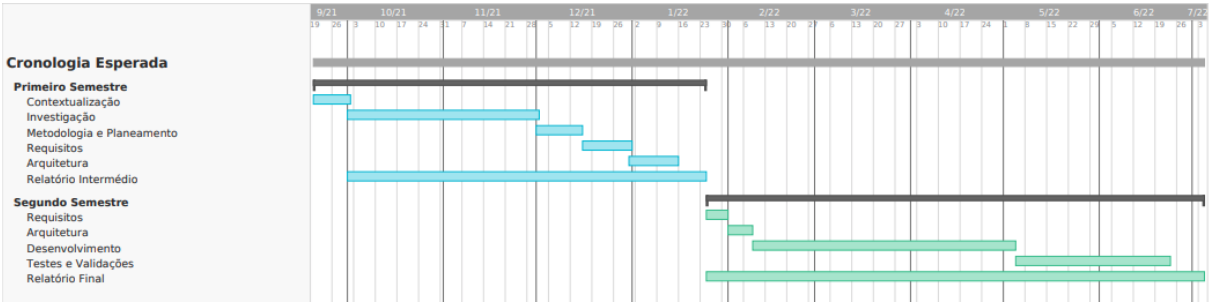


Figura 3.2 - Diagrama de Gantt com a cronologia esperada para ambos os semestres.

3.2.4 Análise do planeamento

Nesta subsecção, é apresentado na Figura 3.3 outro diagrama de *Gantt*, e o mesmo mostra a duração real das tarefas realizadas ao longo do estágio. A figura mencionada está também presente no Anexo A.1.

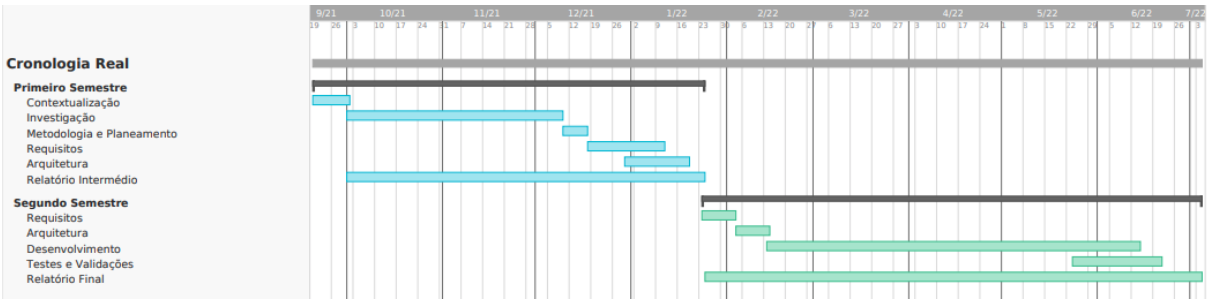


Figura 3.3 - Cronologia obtida, para ambos os semestres

Ao comparar o diagrama da cronologia esperada (Fig. 3.2) com o diagrama da cronologia obtida (Fig. 3.3) as diferenças são evidentes, particularmente no período de desenvolvimento no segundo semestre.

Em relação ao planeamento esperado (Fig. 3.2), é discutível que as estimativas previstas inicialmente foram ingénuas e otimistas, especialmente porque haviam muitas ferramentas e tecnologias com as quais o estagiário não estava familiarizado, o que também levou a um tempo de aprendizagem mais longo, prolongando assim o desenvolvimento.

Depois, os desvios com maior impacto na duração real das tarefas do desenvolvimento foram: o *setup* do Shake It na máquina do estagiário, o qual deu alguns problemas inicialmente devido a certas configurações e dependências; e, por último, a delegação das transações (requisito funcional RF10) que, sendo este ainda um tópico não muito explorado na área da *Blockchain*, tornou-se um requisito cuja implementação demorou mais que o expectável dado que envolveu mais tempo gasto na pesquisa de soluções e na reestruturação de certas decisões arquiteturais.

Em retrospetiva, e apesar destes imprevistos terem retirado tempo de desenvolvimento doutras tarefas, todos os objetivos e requisitos do presente estágio foram cumpridos dentro do tempo delimitado.

3.3 Gestão de Riscos

Esta secção explica como foram definidos e analisados os riscos para este estágio, antes do desenvolvimento do projeto.

A primeira secção descreve a métrica utilizada para fazer a gestão dos ricos, e a importância da mesma. Na segunda secção são expostos os riscos identificados pelo aluno e na terceira secção a matriz de riscos obtida. Por fim, na última secção é feita uma análise de riscos.

3.3.1 Métrica usada

A identificação prévia dos riscos é essencial num projeto de *software*, uma vez que torna possível prever e tentar evitar que os mesmos ocorram ou que, pelo menos, o seu impacto sobre o resultado final seja minimizado.

Como tal, é importante seguir uma metodologia ou um mecanismo que nos permita fazê-lo prudentemente. Assim, a gestão de riscos envolve as seguintes atividades [74]:

- **Identificação de Riscos**: consiste na identificação de fatores que possam comprometer o sucesso do projeto.
- **Análise de Riscos**: quando um risco é identificado, o mesmo deve ser analisado consoante o seu impacto e a sua probabilidade de ocorrência.

  O impacto de um risco pode ser representado da seguinte forma:

  - **Alto**: quando tem um impacto grande sobre o cronograma e realização do projeto. Os critérios de sucesso podem não ser atingidos.

  - **Médio**: tem um impacto significativo sobre o cronograma e realização do projeto. Os critérios de sucesso podem ser atingidos.

  - **Baixo**: poderá ter um pequeno impacto sobre o cronograma e realização do projeto. Os critérios de sucesso podem ser facilmente atingidos.

  A probabilidade de ocorrência de um risco é dada pelos seguintes valores:

  - **Alta**: a probabilidade de acontecer é maior que 80%.

  - **Média**: a probabilidade de acontecer está entre 30% e 80%.

  - **Baixa**: a probabilidade de acontecer é menor que 30%.
- **Planeamento**: nesta fase, o objetivo é mitigar ou eliminar a ocorrência de riscos, e, para isto, é necessário estabelecer um plano de mitigação.
- **Monitoração de riscos**: nesta fase, os riscos são monitorados ou atualizados durante o desenvolvimento do projeto.

Após a identificação e avaliação dos riscos, é possível apresentar os resultados na forma de uma matriz de riscos. Esta é uma ferramenta que oferece uma representação visual sobre a análise de riscos feita na fase de planeamento, e categoriza os mesmos com base na sua probabilidade e impacto.

3.3.2 Descrição dos riscos

Nesta secção são descritos os riscos para este projeto, identificados durante o primeiro semestre.

Risco 1 - Mudanças nos Requisitos: Todos os requisitos funcionais foram escritos durante o primeiro semestre, pelo que devemos estar preparados caso seja necessário fazer alguma alteração aos mesmos.

Probabilidade: Média.

Impacto: Médio.

Plano de mitigação: Organizar mais reuniões com o orientador, de forma a controlar e a prioritizar os requisitos.

Risco 2 - Tecnologias: Dado que algumas das tecnologias que vão ser usadas no desenvolvimento são desconhecidas ao estagiário, poderão existir dificuldades que causem atrasos no desenvolvimento das tarefas.

Probabilidade: Média.

Impacto: Alto.

Plano de mitigação: Treinar, atempadamente, ao resolver tutoriais *online* sobre essas tecnologias. Além disso, comunicar com os orientadores, os quais têm conhecimentos sobre as mesmas e poderão auxiliar em qualquer ponto do desenvolvimento.

Risco 3 - Estimativas: O estagiário não tem muita experiência profissional, o que o poderá levar a definir estimativas erradas durante o planeamento do projeto.

Probabilidade: Alta.

Impacto: Alto.

Plano de mitigação: Pedir ajuda aos orientadores durante a fase de planeamento e classificar cada requisito usando a metodologia *MoSCoW* [75], para definir a prioridade de cada um.

Risco 4 - Pandemia: Com a evolução da pandemia, torna-se difícil o trabalho presencial na empresa. Isto poderá ter consequências no que toca ao suporte que o aluno teria em condições normais.

Probabilidade: Alta.

Impacto: Baixo.

Plano de mitigação: Em adição às *daily scrums* é essencial ter disponibilidade para qualquer contacto adicional, remotamente.

Risco 5 – Qualidade do protótipo: O protótipo apresentado nas demonstrações internas poderá não corresponder aos requisitos e expectativas iniciais.

Probabilidade: Média.

Impacto: Alto.

Plano de mitigação: Perceber quais os requisitos ou expectativas que não foram verificados e organizar uma reunião com a equipa para decidir que ações tomar.

3.3.3 Matriz de riscos

Como foi dito na secção 3.3.1, após identificar e avaliar os riscos obtivemos a seguinte matriz de riscos, de forma a facilitar a visualização dos mesmos.

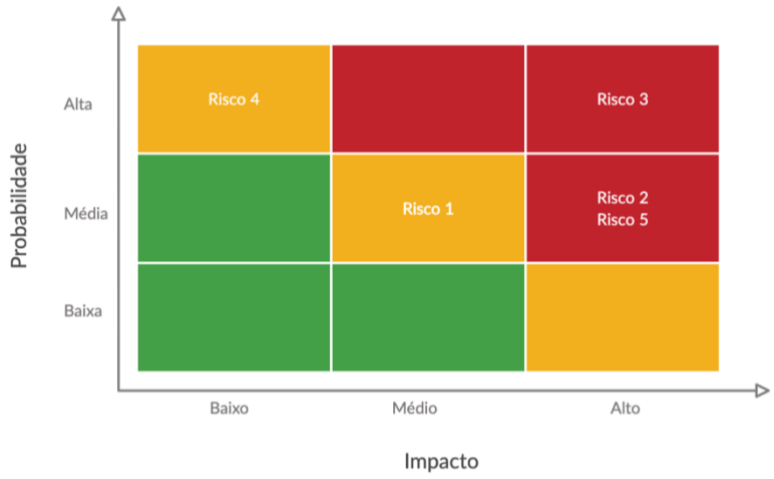


Figura 3.4 – Matriz de Riscos

Ao analisar a matriz de riscos da Figura 3.4, podemos observar que todos os riscos se enquadram nas categorias média e alta, o que significa que os planos de mitigação precisam de ser cuidadosamente seguidos pelo estagiário, pois se os riscos ocorrerem, o seu impacto pode condicionar o sucesso do projeto.

3.3.4 Análise de riscos

Como mencionado na secção 3.2.4, durante a fase de desenvolvimento do segundo semestre, dois dos riscos previamente identificados no primeiro semestre concretizaram-se. Foram estes o Risco 2 (Tecnologias) e o Risco 3 (Estimativas). No entanto, dado que estes já eram conhecidos pelo estagiário desde o primeiro semestre, os respetivos planos de mitigação foram postos em prática e o seu impacto foi diminuído. Apesar do atraso causado, todos os objetivos e requisitos do estágio foram devidamente cumpridos, concluindo assim que uma boa gestão de riscos é fundamental para o sucesso de um projeto de *software*.

Capítulo 4
Análise de Requisitos

Um dos primeiros passos num projeto de *software*, e também um dos mais importantes, é a análise de requisitos. Este processo ajuda a equipa a entender o problema, permitindo que haja uma ideia clara sobre aquilo que vai ser implementado, e também contribui para a mitigação dos riscos.

Por esses motivos, neste capítulo é feito o levantamento e especificação de todos os requisitos a serem implementados no decorrer do 2º semestre.

Na primeira secção, é dado um pouco de contexto sobre o processo até chegar aos requisitos.

Na segunda secção, são listados todos os requisitos funcionais, classificados por prioridade, e os respetivos diagramas de casos de uso são apresentados na terceira secção.

Por fim, nas últimas duas secções, são descritos os requisitos não funcionais e os critérios de sucesso.

4.1 Contexto

Antes de identificar os requisitos, é necessário delinear o problema que pretendemos resolver.

Neste caso, e como mencionado no Capítulo 1, o propósito do presente estágio é pegar na solução e nas funcionalidades atuais do Shake It, e integrá-las com *Blockchain*, recorrendo a *Non-Fungible Tokens*. Além disto, e tirando proveito das características desta tecnologia, outro objetivo é oferecer mais transparência aos seus utilizadores.

É de notar que, apesar do Shake It ter sido adotado pela Vodafone Portugal, a solução desenvolvida para o presente estágio pertence à WIT Software e, portanto, a equipa é que decide o *roadmap* de funcionalidades e requisitos, sejam eles funcionais ou não funcionais. Não obstante, esta é uma atividade exploratória que pode ser interessante para este cliente (Vodafone) ou para outros potenciais clientes.

Com isto em mente, o estagiário idealizou diversas soluções, apresentando as mesmas ao resto da equipa, descrevendo o cenário e as respetivas vantagens e desvantagens de cada um. Depois de reunidas e analisadas todas estas abordagens, a equipa optou por seguir um modelo em que todas as cartas do jogo passassem a ser *Non-Fungible Tokens* (NFTs). Deste modo, os requisitos funcionais identificados na próxima secção foram desenvolvidos com base neste cenário, com foco sobre as funcionalidades mais relevantes do Shake It.

Este processo e alguns dos cenários que foram considerados encontram-se documentados no Anexo B.1.

4.2 Requisitos funcionais

Os requisitos funcionais representam todos os problemas e necessidades que devem ser atendidos e resolvidos pelo *software*, por meio de funções ou serviços. Quanto menos ambíguos e mais objetivos forem estes requisitos, maior será a qualidade do *software* desenvolvido.

Em adição, é importante mencionar que para os requisitos funcionais apresentados na Tabela 4.1, foi feita uma avaliação da prioridade dos mesmos seguindo o método de *MoSCoW* [75].

Assim, uma das seguintes categorias foi atribuída a cada requisito:

- ***Must Have* (M)**: representa um requisito que é crítico para o sucesso do projeto, e corresponde à prioridade máxima.
- ***Should Have* (S)**: representa um requisito que é importante mas não obrigatório para o sucesso do projeto, e corresponde a uma prioridade média.
- ***Could Have* (C)**: representa um requisito que é desejável mas não é obrigatório para o sucesso do projeto, e apenas será implementado caso haja tempo para tal. Corresponde a um nível de prioridade baixo.
- ***Won't Have this time* (W)**: representa um requisito que não vai ser implementado nesta versão do projeto, mas poderia ser implementado no futuro. Esta categoria é a que tem o nível de prioridade mais baixo.

Tabela 4.1 - Requisitos Funcionais.

ID	Requisito Funcional	Prioridade
Backoffice		
RF1	Lançar coleção de NFTs	**(M)**
RF2	Ver coleção de NFTs da operadora	**(M)**
RF3	Ver coleção de NFTs de um utilizador	**(M)**
Web Client		
RF4	Receber NFT ao abrir um Shake	**(M)**
RF5	Ver detalhes sobre os NFTs colecionados	**(M)**
RF6	Trocar NFTs por Shakes	**(M)**
RF7	Trocar NFTs por pontos	**(M)**
RF8	Trocar NFTs com amigos	**(M)**
Carteiras digitais		
RF9	Criar uma *custodial wallet* para cada utilizador	**(M)**
RF10	Delegar uma transação	**(M)**
RF11	Associar uma *non-custodial wallet*	**(C)**
Prémios		
RF12	Receber NFT como prémio	**(W)**

4.3 Diagramas de casos de uso

De forma a clarificar e organizar os requisitos mencionados na secção anterior, foram elaborados diferentes casos de uso. Foi feito um diagrama para cada um dos temas enunciados na Tabela 4.1.

A Figura 4.1 corresponde às funcionalidades do *Backoffice*, às quais apenas o administrador (a operadora) tem acesso. Permitem-lhe lançar uma coleção de NFTs colecionáveis, ver os NFTs que estão na sua própria carteira digital e, por fim, ver a coleção de um utilizador, pesquisa a qual faz através do número de telemóvel do mesmo.

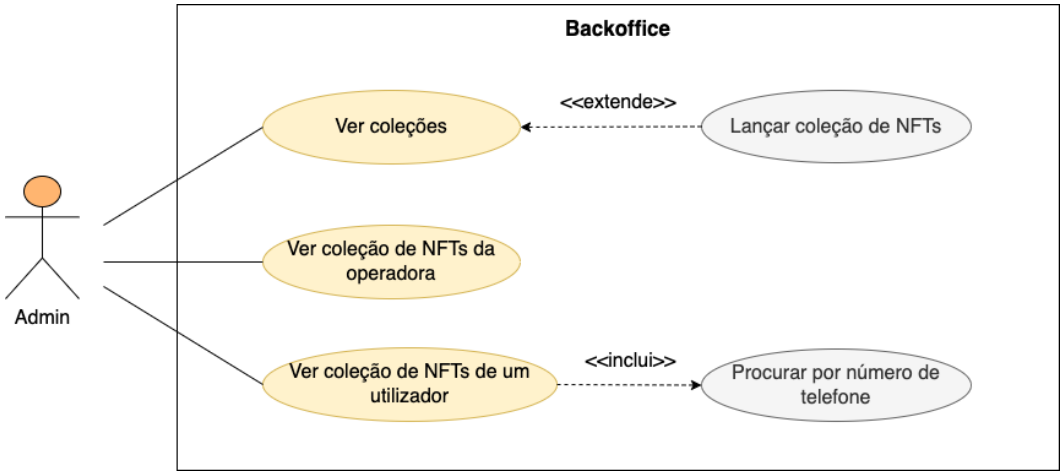


Figura 4.1 - Caso de uso - *Backoffice*.

Na Figura 4.2 são apresentadas as funcionalidades do *Web Client*, que permitem um utilizador colecionar NFTs e transacionar os mesmos, de forma autónoma.

Em primeiro lugar, ao abrir um Shake, o utilizador deverá receber o respetivo NFT.

Depois, para consultar os detalhes de um NFT específico da sua coleção, o utilizador navega para a secção da caderneta e clica no NFT cujas informações deseja visualizar.

Por fim, quanto à troca de cartas repetidas, basta navegar para os respetivos menus e efetuar as transações pretendidas.

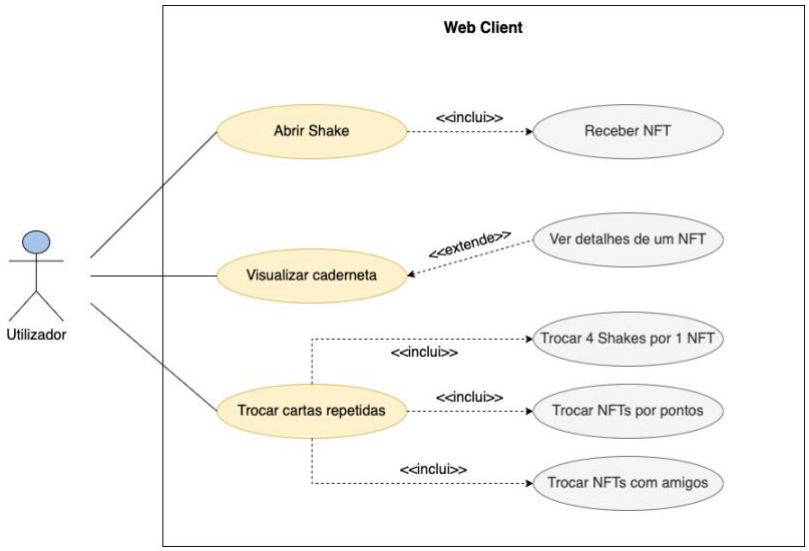


Figura 4.2 - Caso de uso – *Web Client*

Passando para o caso de uso da Figura 4.3, inicialmente, todos os utilizadores terão uma *custodial wallet*, tendo mais tarde a opção de associar a sua própria carteira digital, se assim entenderem. Esta funcionalidade é indicada apenas para utilizadores avançados, com conhecimento prévio sobre as tecnologias usadas. Para além disso, para a parte da delegação das transações, esta é uma ação realizada após a criação da carteira digital de cada utilizador.

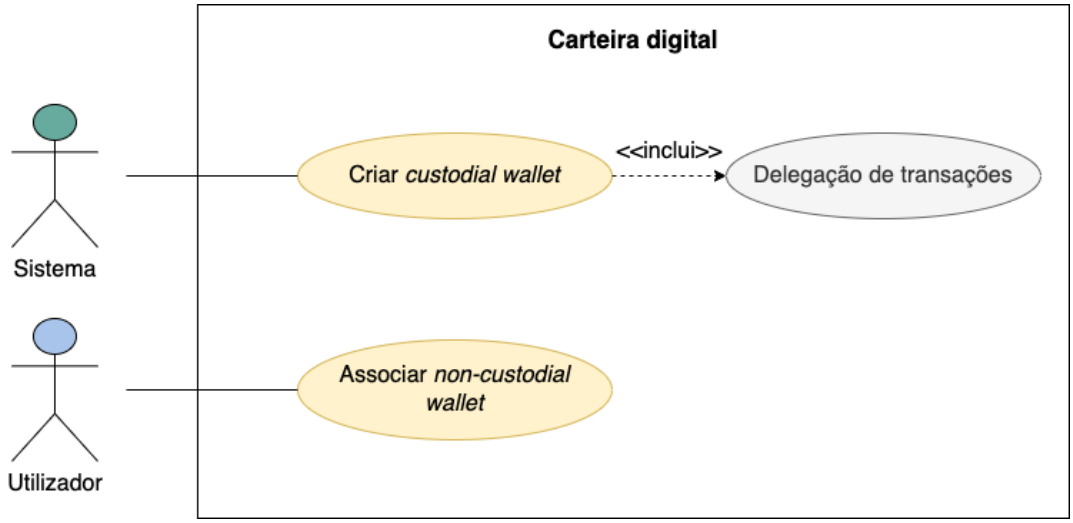


Figura 4.3 - Caso de uso – Carteiras digitais

Por fim, na Figura 4.4 é apresentado o caso de uso que mostra como um utilizador resgata um NFT caso o receba como prémio por completar a respetiva categoria da caderneta. Este requisito não irá ser implementado durante o presente estágio, dada a prioridade que lhe foi atribuída na Tabela 4.1.

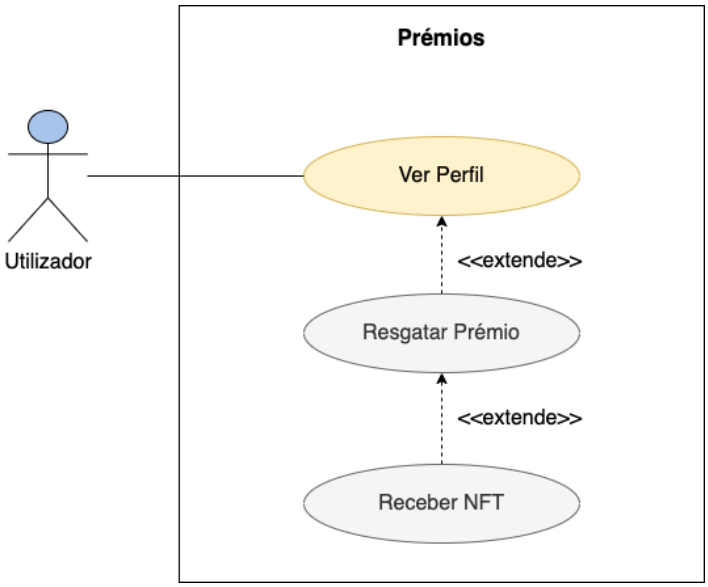


Figura 4.4 - Caso de uso – Prémios

4.4 Requisitos não funcionais

Os requisitos não funcionais representam as características que o sistema desenvolvido deve suportar, assegurando a qualidade e as limitações do mesmo, descrevendo ainda a maneira como o sistema se deve comportar. Apesar de não estarem diretamente ligados às funcionalidades, são igualmente importantes.

Nesta secção, são abordados os atributos de qualidade que a aplicação final deverá obedecer para que a mesma seja considerada válida.

No Capítulo 7, encontram-se descritos os testes efetuados para a validação de cada um dos requisitos não funcionais definidos na Tabela 4.2.

Tabela 4.2 – Requisitos não funcionais.

ID	Requisito não funcional	Prioridade
RNF1	Modularidade	**(M)**
RNF2	Desempenho	**(S)**
RNF3	Escalabilidade	**(C)**

Relativamente à priorização dos atributos de qualidade mencionados na Tabela 4.2, e de forma análoga aos requisitos funcionais mencionados na secção 4.2, foi utilizada a escala de *MoSCoW* [75].

4.4.1 RNF1 – Modularidade

Em engenharia de *software*, a modularidade refere-se à capacidade de um determinado produto de *software* ser dividido em diferentes componentes, de forma a que uma mudança num dos componentes tenha um impacto mínimo nos restantes. [80]

No contexto deste projeto, pretende-se que este processo de modularização seja usado, uma vez que permite separar as funcionalidades de um programa em vários módulos independentes, de maneira a que cada módulo contenha tudo o que é necessário para executar apenas parte da funcionalidade desejada.

4.4.2 RNF2 – Desempenho

O desempenho está ligado às necessidades que garantem que um sistema funcione de maneira rápida, sem ocorrências que possam impactar a usabilidade do mesmo.

Através do presente atributo de qualidade, pretende-se que o sistema seja capaz de processar e responder a todos os pedidos num curto período de tempo. Para tal, foram definidos 2 cenários, apresentados nas Tabelas 4.3 e 4.4.

Esta divisão foi feita para que se pudesse fazer uma análise mais rigorosa dos tempos de resposta obtidos para cada pedido. Dado que no Cenário 1 o tempo de resposta para cada pedido está dependente da *Blockchain*, os valores esperados são significativamente maiores.

Tabela 4.3 - Cenário 1: Desempenho (RNF2)

Cenário 1 – *Endpoints* + *Blockchain*	
Atributo de qualidade	Desempenho
Fonte do estímulo	Utilizador
Estímulo	É enviado um pedido que envolve comunicação com a *Blockchain*
Artefacto	*Blockchain Gateway*
Ambiente	Condições normais
Resposta	O servidor devolve uma resposta
Métrica de resposta	O servidor deve responder no máximo em 2 minutos

Tabela 4.4 - Cenário 2: Desempenho (RNF2)

Cenário 2 - *Endpoints*	
Atributo de qualidade	Desempenho
Fonte do estímulo	Utilizador
Estímulo	É enviado um pedido que não envolve comunicação com a *Blockchain*
Artefacto	*Blockchain Gateway*
Ambiente	Condições normais
Resposta	O servidor devolve uma resposta
Métrica de resposta	O servidor deve responder no máximo em 2 segundos

O artefacto *Blockchain Gateway*, mencionado nos cenários anteriores, encontra-se descrito no Capítulo 5.

4.4.3 RNF3 – Escalabilidade

Escalabilidade é a capacidade de o sistema ser capaz de se adaptar e manter um bom desempenho quando o número de pedidos feitos ou o número de conexões em simultâneo aumentam.

Relativamente a este requisito, pretende-se que a plataforma, num cenário de grande *stress*, se mantenha disponível e responda a qualquer pedido num mínimo de tempo que esteja em conformidade com os valores testados para o desempenho (RNF2).

Os dois cenários criados são apresentados nas Tabelas 4.5 e 4.6.

Tabela 4.5 - Cenário 3: Escalabilidade (RNF3)

Cenário 3 – *Endpoints* + *Blockchain*	
Atributo de qualidade	Escalabilidade
Fonte do estímulo	Utilizadores
Estímulo	São enviados vários pedidos em simultâneo (que envolvem comunicação com a *Blockchain*)
Artefacto	*Blockchain Gateway*
Ambiente	Carga intensa
Resposta	O servidor devolve uma resposta
Métrica de resposta	O tempo de resposta não se deverá desviar mais de 50% acima dos valores obtidos nos testes de Desempenho (RNF2)

Tabela 4.6 - Cenário 4: Escalabilidade (RNF3)

Cenário 4 - *Endpoints*	
Atributo de qualidade	Escalabilidade
Fonte do estímulo	Utilizadores
Estímulo	São enviados vários pedidos em simultâneo
Artefacto	*Blockchain Gateway*
Ambiente	Carga intensa
Resposta	O servidor devolve uma resposta
Métrica de resposta	O servidor deve responder no máximo em 2 segundos

4.5 Critérios de sucesso

Para que um projeto seja considerado bem-sucedido, deve ser estipulado um conjunto de condições mínimas que devem ser atendidas até à conclusão do mesmo. Assim, para o presente estágio, foi definido o seguinte grupo de critérios:

- A arquitetura e todos os requisitos deverão ser revistos e aprovados pelo *product owner*;
- Todos os requisitos funcionais e não funcionais, classificados com "*Must Have*" na escala de *MoSCoW*, deverão ser devidamente cumpridos até ao final do projeto;
- O estágio e todos os objetivos descritos anteriormente deverão ser cumpridos dentro do tempo planeado.

Capítulo 5
Arquitetura de *Software*

No presente capítulo é feita a análise da arquitetura de *software* para este projeto, seguindo uma abordagem de alto nível. Para tal, foi construído um diagrama no qual são apresentados os componentes principais do sistema a ser desenvolvido na fase de implementação do presente estágio. Depois, são apresentadas as ferramentas e as tecnologias que serão adotadas durante esta fase.

Para terminar o capítulo, é apresentada uma análise dos custos estimados que se poderão expectar, associados às transações com a *Blockchain*.

5.1 Arquitetura

Como referido anteriormente, no diagrama da Figura 5.1, também presente no anexo C, é dada uma visão geral sobre a arquitetura a ser adotada para este projeto. Isto permite-nos compreender de que forma os diferentes componentes se relacionam entre si, para mais tarde facilitar o processo de implementação.

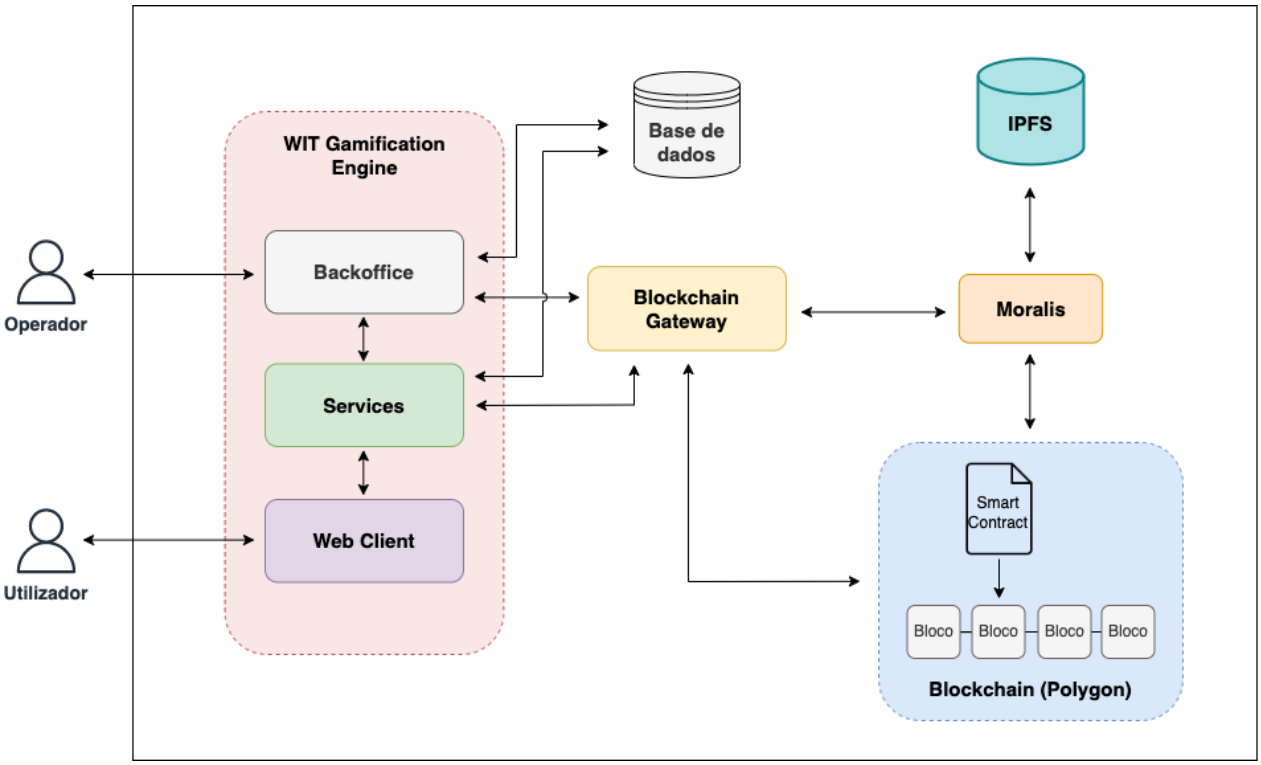


Figura 5.1 - Arquitetura

De forma a perceber a função de cada componente e a sua importância para o desenvolvimento de todo o sistema, nas seguintes subsecções são caracterizadas cada uma das partes mencionadas na Figura 5.1.

5.1.1 *WIT Gamification Engine*

Este componente representa o local onde se encontram desenvolvidos o *Backoffice*, os *Services* (*backend*) e o *Web Client* (*frontend*), os quais são a base arquitetural da solução já existente do Shake It, e sobre o qual uma parte do desenvolvimento deste estágio irá incidir. Deste modo, os componentes mencionados sofreram alterações por parte do estagiário.

5.1.2 *Backoffice*

Como referido anteriormente, o *Backoffice* é uma das aplicações já existentes no domínio do Shake It, e funciona como um painel de controlo para o mesmo. Naturalmente, apenas um administrador tem acesso a esta aplicação.

No contexto do estágio, o mesmo foi alterado para acomodar as mudanças necessárias ao desenvolvimento dos requisitos funcionais do *Backoffice*, listados no Capítulo 4.

5.1.3 *Services*

De igual forma, os *Services* são o conjunto de microsserviços que já se encontravam desenvolvidos, e representam o *backend* de toda a aplicação. Este componente esteve sujeito a alterações por parte do estagiário, nomeadamente para implementar a ligação ao *Blockchain Gateway* e para implementar os requisitos funcionais definidos no Capítulo 4.

5.1.4 *Web Client*

Quanto ao *Web Client* (*frontend*), este representa toda a parte visual da aplicação, com a qual o utilizador interage, e o mesmo pertence também à solução existente do Shake It.

No decorrer do estágio, o mesmo sofreu algumas alterações, são elas: a adição de um botão no perfil do utilizador, que quando pressionado redireciona-o para uma página com o histórico de transações da sua carteira digital; e, a adição dos detalhes de cada NFT que o utilizador tem colecionados.

5.1.5 *Blockchain Gateway*

O servidor API *Blockchain Gateway* foi criado pelo estagiário e é a aplicação onde se encontram desenvolvidos os métodos necessários para cumprir com os requisitos do projeto.

É também o elo de ligação principal entre o *Backoffice* e *Services* com a *Blockchain*, possibilitando a comunicação com o *Smart Contract*.

5.1.6 InterPlanetary File System (IPFS)

O IPFS, detalhado no Capítulo 2, é onde é armazenada de forma descentralizada toda a *metadata* associada aos NFTs, incluindo a imagem dos mesmos, também ela armazenada no IPFS.

5.1.7 Moralis

O Moralis [81] é uma plataforma para o desenvolvimento de aplicações Web3 [82], e atua maioritariamente como um *Node Provider*. Este, é um serviço que oferece uma maneira de comunicarmos com uma *Blockchain* sem termos que executar o nosso próprio *node*, isto é, sem termos que nos conectar à rede ponto a ponto dessa *Blockchain*. Sem esse *node*, a nossa aplicação não conseguiria interagir com a *Blockchain* da nossa escolha.

Para o presente estágio, o Moralis foi usado para obter um *node* RPC (*Remote Procedure Call*) sobre o qual pudéssemos operar e, também, para fazer o *upload* da imagem e da metadata de um NFT para o IPFS. Em adição, serviu também para obter os NFTs de um dado endereço público.

Desta forma, este componente possui ligações com a *Blockchain*, com o IPFS, e com o serviço *Blockchain Gateway*.

5.1.8 *Blockchain (Polygon)*

Por fim, este componente representa a *Blockchain* que foi usada no desenvolvimento para dar *deploy* do *Smart Contract* e com a qual são realizadas diversas interações, e é a Polygon. Esta seleção e os motivos por trás da mesma encontram-se detalhados no Capítulo 2 (secção 2.2.6).

5.2 Ferramentas e tecnologias escolhidas

Nesta secção são apresentadas as ferramentas e as tecnologias escolhidas para o desenvolvimento do projeto, para cada um dos componentes descritos anteriormente.

Primeiramente, é de salientar que as tecnologias utilizadas na solução já existente do Shake It foram tidas em conta, de forma a ser fornecido o maior apoio possível ao aluno e aproximar o sistema a ser construído do já existente, facilitando uma futura integração.

WIT Gamification Engine

Em sequência desta consideração, começando pelo *WIT Gamification Engine* e pelas aplicações desenvolvidas no seu domínio, a tecnologia utilizada na construção do *backend* foi a *framework* Spring Boot [83], baseada em Java [84]. Para o *frontend* foi utilizado HTML, CSS [85], e a *framework* AngularJS [86], baseada em JavaScript [87]. Por fim, o servidor usado para montar as aplicações foi o Apache Tomcat [88].

Base de dados

Relativamente à base de dados, a escolha recaiu sobre a Oracle Database [89], por ser esta a tecnologia usada no sistema já existente.

Blockchain Gateway

Para a construção do *Blockchain Gateway*, sendo este um serviço fora do domínio do *WIT Gamification Engine*, o estagiário teve uma maior liberdade na escolha das tecnologias.

Deste modo, as principais tecnologias usadas neste módulo foram NodeJS e ExpressJS [90][91], para a construção do servidor. Adicionalmente, foi utilizada a linguagem de programação JavaScript e a biblioteca Web3JS [92], sendo através desta que são realizadas as interações com a *Blockchain*.

Por fim, é também utilizada a API do Moralis [81], para efetuar as operações mencionadas na secção 5.1.7 do presente capítulo.

Blockchain

Quanto à *Blockchain*, foi usada a rede de testes da Polygon, intitulada de "Mumbai" [106]. Ao usar esta rede, que é uma réplica da rede principal, temos acesso a um número infinito de *tokens* (neste caso, a criptoemoeda MATIC, da Polygon) usados para pagar as taxas associadas às transações na *Blockchain*. Isto permite-nos testar configurações e explorar várias implementações, sem custos associados.

Smart Contract

Finalmente, para a escrita do *Smart Contract* foi usada a linguagem de programação Solidity [93], sendo esta a linguagem adotada pela *Blockchain* selecionada.

5.3 Análise de custos

Cada transação na *Blockchain* tem uma taxa associada, que serve essencialmente para recompensar os *nodes* da rede ponto a ponto que validaram essa transação. Estas taxas são habitualmente pagas através da criptomoeda nativa dessa *Blockchain* que, no presente estágio, é a MATIC, a moeda digital da Polygon.

Deste modo, dado que parte das funcionalidades que serão implementadas neste projeto estão fortemente assentes sobre a *Blockchain*, e tendo em conta que todas as transações serão pagas pela Operadora e não pelos utilizadores, é importante estimar os custos que estarão associados à utilização normal do produto desenvolvido.

Como tal, a Tabela 5.1 apresenta os gastos estimados que se poderão esperar, consoante o número de utilizadores e o número de transações que cada um poderá realizar.

Os valores obtidos são apresentados em MATIC e o número estimado de transações por utilizador é para um período de 6 meses, que corresponde à duração de uma coleção do Shake It. Por fim, o preço médio por transação que foi usado para realizar os cálculos foi 0.0002 MATIC, conforme a Tabela 2.1 do Capítulo 2.2.6.

Tabela 5.1 - Custos estimados para as transações feitas por utilizadores (preços em MATIC)

Número de utilizadores	**Pior caso (120 transações/utilizador)**	**Caso mais provável (80 transações/utilizador)**	**Melhor caso (40 transações/utilizador)**
200 000	4 800	3 200	1 600
500 000	12 000	8 000	4 000
750 000	18 000	12 000	6 000
1 000 000	24 000	16 000	8 000
2 000 000	48 000	32 000	16 000

Capítulo 6
Desenvolvimento

O presente capítulo é dedicado à fase de desenvolvimento, sendo apresentado o processo adotado pelo aluno no decorrer do segundo semestre até obter o produto final, assim como as funcionalidades desenvolvidas face aos requisitos funcionais levantados no primeiro semestre, cuja especificação se encontra disponível no Capítulo 4.

6.1 Processo

A fase de desenvolvimento deste projeto foi decomposta em 6 etapas distintas, e seguiu a ordem de implementação à frente descrita.

Em primeiro lugar, foi feito o ***setup*** da *WIT Gamification Engine* na máquina do estagiário, adicionando todas as dependências e configurações necessárias para este efeito, incluindo a Base de dados. Esta fase foi realizada em simultâneo com a etapa seguinte.

Quanto à **prototipagem do Smart Contract** responsável pelas operações principais do jogo, também este foi desenvolvido numa fase inicial da implementação.

Depois, integrada na **fase de aprendizagem e de adaptação às tecnologias** a serem usadas, foi criada pelo estagiário uma aplicação *Web*, fora do contexto do Shake It. Esta aplicação foi realizada com o intuito de dar ao aluno a oportunidade para dominar alguns dos conceitos em volta da comunicação com a *Blockchain* e, mais especificamente, com o *Smart Contract* previamente descrito. No entanto, e reiterando, esta foi uma atividade meramente exploratória e de aprendizagem e não está diretamente relacionada com o Shake It.

Depois de auferidos os conhecimentos necessários através das etapas anteriormente descritas, começou a fase de implementação dentro do contexto do Shake It. Nomeadamente, foi desenvolvido pelo estagiário o serviço ***Blockchain Gateway***, responsável pela ligação entre a *WIT Gamification Engine* e a *Blockchain*, assim como com os métodos criados no *Smart Contract* referido anteriormente.

Por fim, quanto às duas últimas etapas do desenvolvimento, foram realizadas alterações ao ***Backoffice*** e ao ***Web Client***, de modo a cumprir com os requisitos delineados.

As seguintes secções descrevem o trabalho realizado pelo aluno em cada uma das etapas mencionadas.

6.2 Ambiente de desenvolvimento

As seguintes subsecções pretendem descrever o ambiente de desenvolvimento adotado neste projeto, fornecendo informações sobre as decisões tomadas e as ferramentas utilizadas.

6.2.1 Local

No início do projeto, as ferramentas e tecnologias necessárias foram instaladas e configuradas localmente, na máquina do estagiário.

Para a preparação deste ambiente, foram utilizados o IntelliJ IDEA [94] e o GitLab [95]. O primeiro, por ser um IDE (*Integrated Development Environment*) intuitivo, eficiente e com suporte para todas as tecnologias a serem usadas no desenvolvimento. Em adição, dado que este IDE também é usado por membros da equipa do estagiário, os mesmos poderiam assisti-lo em caso de dúvidas. Por sua vez, o GitLab foi usado por ser a ferramenta onde se encontra o repositório com todo o código e documentação referente ao Shake It. Além disso, o GitLab também foi usado para gerir o controlo de versões e guardar as alterações realizadas pelo estagiário.

Depois de instaladas as dependências exigidas, e através do servidor Apache Tomcat, foram executadas as aplicações e os microsserviços do *WIT Gamification Engine*, podendo assim começar a usar o Shake It.

6.2.2 Virtualização

Numa fase final do desenvolvimento, foi montado um ambiente de desenvolvimento *online*, recorrendo ao Docker [96], ao Jenkins [97] e ao GitLab. Esta virtualização permite que o produto desenvolvido possa ser facilmente partilhado entre os membros da equipa, e também para realizar pequenas demonstrações.

O Docker é uma ferramenta que facilita a criação, o *deployment* e a execução de aplicações através do uso de *containers*, que são unidades de *software* que empacotam não só o código como também todas as suas dependências. Deste modo, o seu propósito é tornar o processo de configuração de uma aplicação num novo ambiente o mais fácil e rápido possível.

A imagem de um *container* corresponde a um pacote de *software* leve e executável, que inclui tudo o que é necessário para correr uma aplicação. O Docker consegue construir estas imagens automaticamente através das instruções e dos comandos descritos no Dockerfile. [98]

O Dockerfile do presente projeto encontra-se na Figura 6.1.

```
FROM tomcat:9.0.62-jre8-openjdk-slim

RUN apt-get update && apt-get install -y --no-install-recommends nano less curl apache2 libapache2-mod-jk

# tomcat
COPY docker-services/tomcat/web.xml /usr/local/tomcat/conf/web.xml
# tomcat webapps
COPY yornminibar/yornminibar-bo/target/yornminibar-bo-*.war /usr/local/tomcat/webapps/yornminibar-bo.war

COPY yornminibar/yornminibar-prizes-service/target/yornminibar-prizes-service-*.war /usr/local/tomcat/webapps/yornminibar-prizes-service.war
COPY yornminibar/yornminibar-users-service/target/yornminibar-users-service-*.war /usr/local/tomcat/webapps/yornminibar-users-service.war
COPY yornminibar/yornminibar-shakes-service/target/yornminibar-shakes-service-*.war /usr/local/tomcat/webapps/yornminibar-shakes-service.war
COPY yornminibar/yornminibar-cards-service/target/yornminibar-cards-service-*.war /usr/local/tomcat/webapps/yornminibar-cards-service.war
COPY yornminibar/yornminibar-messages-service/target/yornminibar-messages-service-*.war /usr/local/tomcat/webapps/yornminibar-messages-service.war

COPY yornminibar/yornminibar-service/target/yornminibar-service /usr/local/tomcat/webapps/yornminibar-service
COPY yornminibar/yornminibar-webclient/target/dist/ /usr/local/tomcat/webapps/yornminibar-service/

COPY docker-services/httpd/httpd-proxy.conf /etc/apache2/sites-enabled/
RUN rm -f /etc/apache2/sites-enabled/000-default.conf

COPY docker-services/httpd/apache2.conf /etc/apache2/apache2.conf
COPY docker-services/start.sh /start.sh
RUN ["chmod", "+x", "/start.sh"]
RUN a2enmod proxy proxy_ajp proxy_http rewrite deflate headers proxy_balancer proxy_connect proxy_html lbmethod_byrequests
# RUN a2enmod proxy proxy_ajp proxy_http rewrite

ENV CATALINA_OPTS="-Dlog4j2.formatMsgNoLookups=true"
# VOLUME ["/var/log/apache2"]
EXPOSE 80
EXPOSE 8080

# Node
RUN apt-get install -y --no-install-recommends nodejs npm
WORKDIR /usr/src/shakeit-blockchain-api

RUN curl -fsSL https://deb.nodesource.com/setup_17.x | bash - && apt-get install -y nodejs

COPY shakeit-blockchain-api/* /usr/src/shakeit-blockchain-api/

RUN npm install && npm install express

CMD ["/start.sh"]
```

Figura 6.1 - Ficheiro Dockerfile

```
#!/bin/sh

# apachectl -k start -DFOREGROUND
service apache2 restart
catalina.sh run

cd /usr/src/shakeit-blockchain-api
nohup node server.js > app.log 2>&1 &
```

Figura 6.2 - Ficheiro start.sh

Na Figura 6.2 é descrito o ficheiro “start.sh”, referenciado no Dockerfile, e inclui o comando necessário para executar o *Blockchain Gateway* e para criar o ficheiro de *logs* deste serviço, intitulado “app.log”.

Para dar *deploy* do Dockerfile recorremos à ferramenta Docker Compose [99], a qual consiste num ficheiro YAML que descreve as configurações dos serviços do sistema. Este encontra-se disponível na Figura 6.3.

```yaml
version: "3.3"
services:
  shakeit-blockchain-services-docker:
    container_name: shakeit-blockchain-services-docker
    image: shakeit-blockchain-services-docker:latest
    ports:
      - 80:80
      - 8081:8080
    restart: unless-stopped
    environment:
      - SERVER_URL=https://lciqmsf6eh6k.usemoralis.com:2053/server
      - APP_ID=bvSAPVcwkrSURIE5CiWyEG7ri9Gn2E2uhN4BFU2C
      - MASTER_KEY=Yj2kVrzCQqebGG566w642bzHZ6yI7pEhuJ0szwjB
      - ADMIN_ADDRESS=0xCA42F06E9f3Db574a9925e1da9FA041A3Bf34DF4
      - ADMIN_KEY=0x4d0244e9a3c6713d268b40fcbc46c791c911198789b63b85f45fcc8708e2ebc5
      - CONTRACT_ADDRESS=0x62b25055937f846f610413bceeb019ce63145e21
      - MORALIS_RPC_NODE=https://speedy-nodes-nyc.moralis.io/d35d78304830e96985c4a678/polygon/mumbai
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - /etc/localtime:/etc/localtime:ro
```

Figura 6.3 - Ficheiro docker-compose.yml

Neste ficheiro também são declaradas as variáveis de ambiente, neste caso relacionadas com o serviço *Blockchain Gateway*, assim como os portos que serão usados.

Por fim, quanto ao Jenkins, este é um servidor de automação *open source* que fornece uma grande variedade de *plugins* para dar suporte à criação, *deployment* e automação de um projeto de *software*. [97] Neste projeto, foi usado para montar o ambiente de desenvolvimento descrito na presente subsecção.

Após a criação dos ficheiros do Docker mencionados anteriormente, no Jenkins é feita a configuração responsável por aceder à *branch* do GitLab sobre a qual queremos montar o projeto. Desta forma, é feita a construção da aplicação usando o *Maven* [100], que compila todos os ficheiros, e, depois, é executado o ficheiro "docker-compose.yml" (Fig. 6.3), o qual irá criar os *containers* com as imagens necessárias.

6.3 Prototipagem do *Smart Contract*

Nesta secção é apresentado o *Smart Contract* desenvolvido para o presente estágio, explicando as suas funcionalidades e todo o processo desde a escrita até ao *deployment* do mesmo na Polygon, a *Blockchain* a ser usada neste projeto.

O contrato foi desenvolvido no Remix [101], um IDE *open source* para escrever e compilar código em Solidity, a linguagem de programação usada para construir *Smart Contracts* na Polygon. Além disso, este IDE permite realizar o *deploy* do contrato diretamente para a *Blockchain*, como irá ser descrito na subsecção 6.3.3.

6.3.1 *Token Standard* escolhido

Cada *token* numa *Blockchain*, seja ele fungível ou não fungível (NFT), segue um *standard* específico. Este *standard* corresponde ao tipo de *Smart Contract* e às características que os *tokens* emitidos por ele irão possuir.

Desta forma, antes de iniciar a implementação do contrato e das respetivas funcionalidades, é importante definir que *standard* queremos adotar, neste caso para os *Non-Fungible Tokens* (NFTs).

Para o presente estágio, esta escolha recaiu sobre o *standard* **ERC-1155** [102], uma vez que o mesmo permite que um *smart contract* consiga gerar um número infinito de NFTs, enquanto que num *standard* como o ERC-721 [103], por exemplo, este necessita de um novo *smart contract* para cada tipo de *token*.

Além disso, dado que no Shake It cada carta pode ser enviada para milhares de utilizadores diferentes, tem de ser desenvolvido um contrato capaz de, para uma carta, criar múltiplas cópias. A imagem da Figura 6.4 tem como objetivo evidenciar essa diferença, sob a qual foi tomada a decisão referida anteriormente.

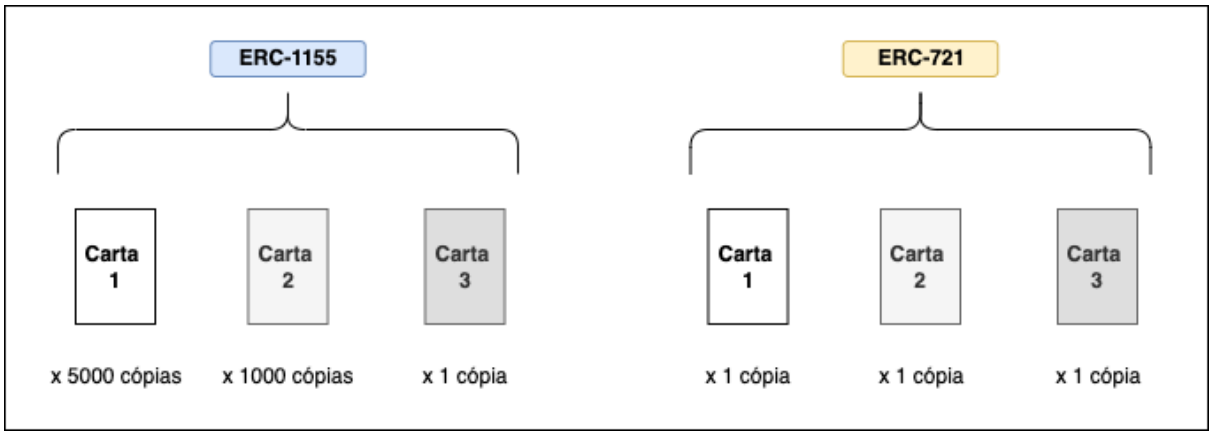


Figura 6.4 - ERC-1155 vs ERC-721

6.3.2 Implementação

Na presente subsecção, é apresentada a última iteração realizada sobre o *Smart Contract* desenvolvido no decorrer do segundo semestre. O mesmo pode ser consultado na Figura 6.5, e também se encontra disponível no Anexo D.

Ao longo da secção são também enumerados e detalhados os métodos que foram utilizados, assim como algumas das alterações realizadas ao longo da implementação.

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.4;
3
4  import "@openzeppelin/contracts/token/ERC1155/ERC1155.sol";
5  import "@openzeppelin/contracts/access/Ownable.sol";
6
7  contract ShakeItNFT is ERC1155, Ownable {
8
9      mapping (uint256 => string) private _uris;
10
11     constructor() ERC1155("") {}
12
13     function setURI(string memory newuri) public onlyOwner {
14         _setURI(newuri);
15     }
16
17     function mint(address account, uint256 id, uint256 amount, string memory tokenURI)
18         public
19         onlyOwner
20     {
21         _mint(account, id, amount, "");
22         setTokenURI(id, tokenURI);
23     }
24
25     function uri(uint256 tokenId) override public view returns (string memory){
26         return(_uris[tokenId]);
27     }
28
29     function setTokenURI(uint256 tokenId, string memory newuri) public onlyOwner {
30         _uris[tokenId] = newuri;
31     }
32 }
```

Figura 6.5 - *Smart Contract*

Primeiramente, é importante referir que parte dos métodos que irão ser referidos são herdados do *standard* ERC-1155, e a Figura 6.5 apenas descreve os métodos aos quais foi necessário dar *override* para fazer as alterações necessárias, e também os métodos criados pelo estagiário.

Criação dos NFTs

A função **mint**, como designada na Figura 6.5, é responsável por criar cada NFT. Deste modo, a mesma recebe os seguintes parâmetros:

- **account** – corresponde ao endereço público da carteira para onde será enviado o NFT;
- **id** – número identificador (mencionado adiante como *token_id*) que queremos associar a esse NFT;
- **amount** – quantidade de *tokens* que queremos criar para determinado *token_id*;
- **tokenURI** – corresponde à metadata que irá estar associada a cada *token_id*.

De modo a que para cada NFT fosse possível associar a respetiva metadata, o estagiário alterou a função **mint**. Para esta finalidade, foi utilizado o método **setTokenURI** e o *mapping* **_uris**.

É ainda de salientar que apenas o criador do *smart contract*, isto é, o endereço que fez o *deploy* do mesmo, é que tem permissões para invocar a função **mint**. Isto é assegurado através da condição **onlyOwner**, a qual é herdada da classe **Ownable** e atribuída ao método referido.

Transferências

Quanto à transferência de NFTs de um endereço para outro, são utilizadas duas funções, ambas herdadas do ERC-1155: **safeTransferFrom**, para realizar transferências de um único *token* entre dois endereços; e, a função **safeBatchTransferFrom**, para realizar transferências em lote, ou seja, para numa só transação poderem ser enviados múltiplos *tokens*.

Estas funções podem ser invocadas por qualquer pessoa, desde que a mesma possua o *token* que pretende transferir. Os parâmetros recebidos pelos métodos referidos são:

- **from** – endereço público da carteira de onde será transferido o *token* (este endereço tem de ser igual ao endereço que chama o método);
- **to** – endereço público para onde será enviado o *token*;
- **id** – *token_id* do *token* que pretendemos transferir. No caso de uma transferência em lote, o **id** será um *array* com os *token_id* dos *tokens* que queremos enviar;
- **amount** – corresponde à quantidade de *tokens* que pretendemos transferir. No caso de uma transferência em lote, o **amount** será um *array* com as quantidades de *tokens* que queremos enviar para cada *token_id*.

Delegação

Por fim, o último método que irá ser utlizado é o **setApprovalForAll**, herdado do ERC-1155. Esta função permite que a pessoa que a invoca conceda ou revogue permissões a um determinado endereço público (operador) para transferir os *tokens* na sua carteira. Este método recebe os seguintes parâmetros:

- **operator** – endereço público ao qual irá ser concedida ou revogada a aprovação;
- **approved** – este valor será *True* se quisermos aprovar o **operator** e *False* se quisermos revogar a aprovação.

No Shake It, e como irá ser descrito nas próximas secções, esta função foi utilizada para que pudéssemos transferir os *tokens* na carteira de um utilizador para outro. Desta forma, os utilizadores não têm de pagar os custos associados a cada transação, sendo esse valor acolhido pelo operador.

6.3.3 *Deployment*

Depois de escrever o *smart contract*, resta apenas compilá-lo e fazer o respetivo *deploy* na Polygon. Como referido anteriormente, tudo isto foi realizado no IDE Remix.

Ao compilar o contrato, é produzido um ABI (*Application Binary Interface*), que é um JSON que descreve o contrato e os métodos nele inseridos. Este ABI é necessário para mais tarde a nossa aplicação conseguir perceber o *smart contract* e poder interagir com ele.

Por fim, para dar *deploy*, é necessária uma carteira digital que tenha fundos suficientes para poder pagar os custos desta transação. Deste modo, a carteira é conectada ao Remix, a rede *Blockchain* onde iremos publicar o contrato é selecionada e, finalmente, confirmamos a transação, sendo-nos devolvido o endereço na *Blockchain* onde o contrato está alojado.

Para consultar informações sobre este ou qualquer outro endereço na Polygon, foi usado o **Polygon Scan** [104], que é o *blockchain explorer* [105] para esta rede.

6.4 Aplicação de teste

Como mencionado na secção 6.1, no início do segundo semestre foi criada pelo estagiário a aplicação *Web* descrita na presente secção. A mesma foi desenvolvida durante o período de aprendizagem com o intuito de criar bases sólidas de conhecimento para que mais tarde, dentro do contexto do Shake It, pudessem ser implementados os requisitos necessários.

Sendo este o primeiro contacto que o aluno teve com tecnologias como a *Blockchain*, o foco da aplicação, desenvolvida em JavaScript, Web3JS e Bootstrap [113], dividiu-se entre os tópicos seguintes:

- **Familiarização com o Moralis**, que incluiu a configuração do servidor, do *node* RPC, e da base de dados onde os utilizadores e as respetivas carteiras digitais são guardados. Além disso, foram estudados os serviços que o Moralis oferece através da sua API, como por exemplo a funcionalidade de ver os NFTs numa determinada carteira;
- **Comunicação com o *Smart Contract***, e incluiu operações como a criação de uma coleção de NFTs, transferência de NFTs e a delegação de transações. Além disso, foram implementados os métodos que permitem carregar para o IPFS a metadata e a imagem de cada NFT;

6.4.1 Implementação

Ao longo da presente subsecção, são descritas as funcionalidades implementadas na aplicação *Web*, a qual se encontra dividida em três ecrãs:

- Página de *login* e registo;
- Painel de administração;
- Página do utilizador.

Página de *login* e registo

Nesta página, apresentada na Figura 6.6, é onde são realizados o registo e a autenticação quer dos utilizadores, quer do administrador.

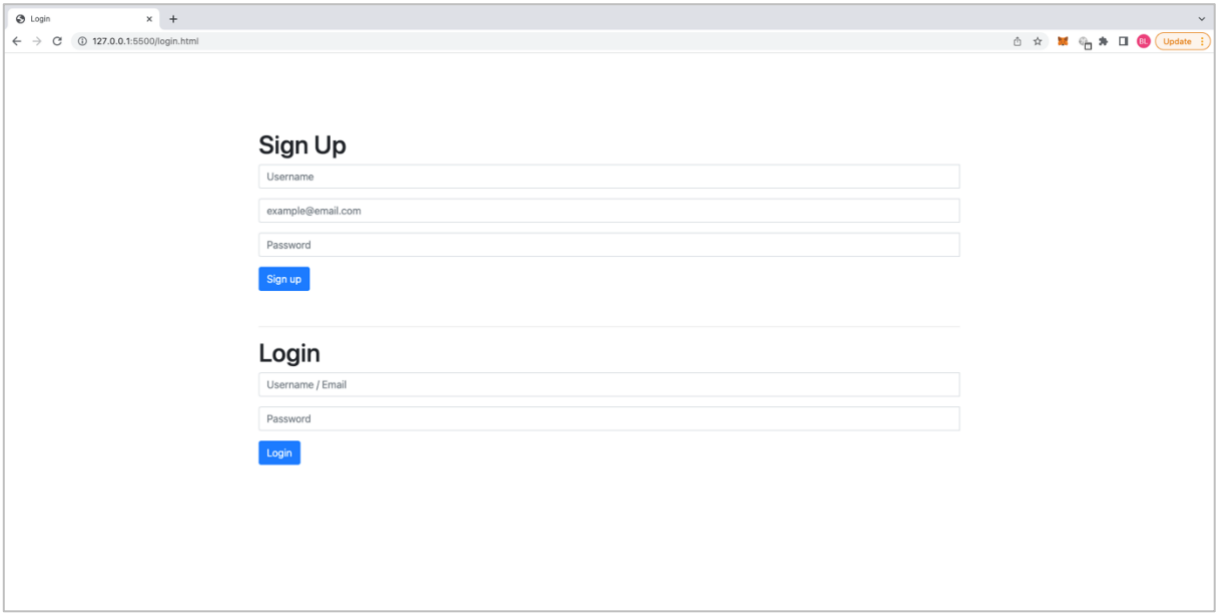


Figura 6.6 - Página de *login* e registo

Durante o registo de um novo utilizador, é-lhe gerada uma carteira digital, isto é, um endereço público e a respetiva chave privada. Estas informações são armazenadas na base de dados do servidor do Moralis.

Painel de administração

Quanto à página do administrador, esta dispõe de três secções diferentes, representadas nas Figuras 6.7, 6.8 e 6.9.

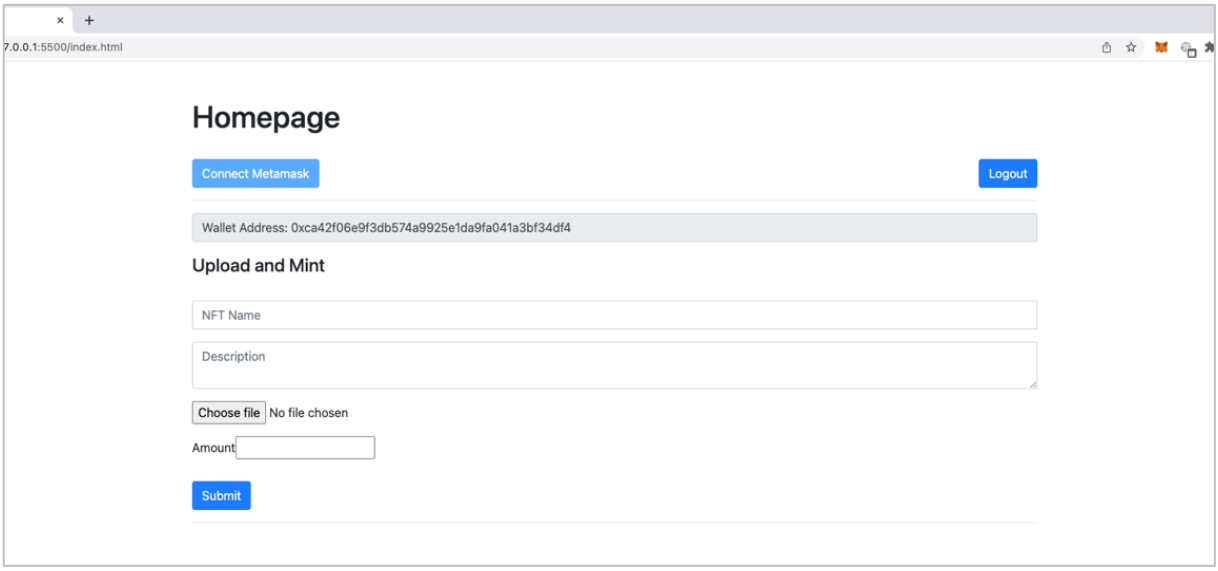


Figura 6.7 - *Upload* e *Mint* de NFTs

Na primeira (Fig. 6.7), é onde é criado um novo NFT, recebendo como *input* o nome, descrição, imagem e a quantidade. Quando estes parâmetros são submetidos, em primeiro lugar é feito o *upload* da imagem para o IPFS. Depois de obter o *link* do IPFS para essa imagem, é criado um ficheiro JSON com toda a metadata - o nome, a descrição e o *link* da imagem -, e o mesmo é também carregado para o IPFS, obtendo assim um *link* com a metadata que ficará associada ao nosso NFT.

Depois, é construída a transação que será enviada, na qual invocamos a função **mint** do nosso *smart contract*. Por fim, através da API do Web3JS, a transação é assinada com a chave privada do administrador e, posteriormente, é enviada para a *Blockchain*. Esta comunicação é assegurada através do *node* RPC a ser usado, fornecido pelo Moralis.

A secção seguinte corresponde à transferência de NFTs, e é representada na Figura 6.8.

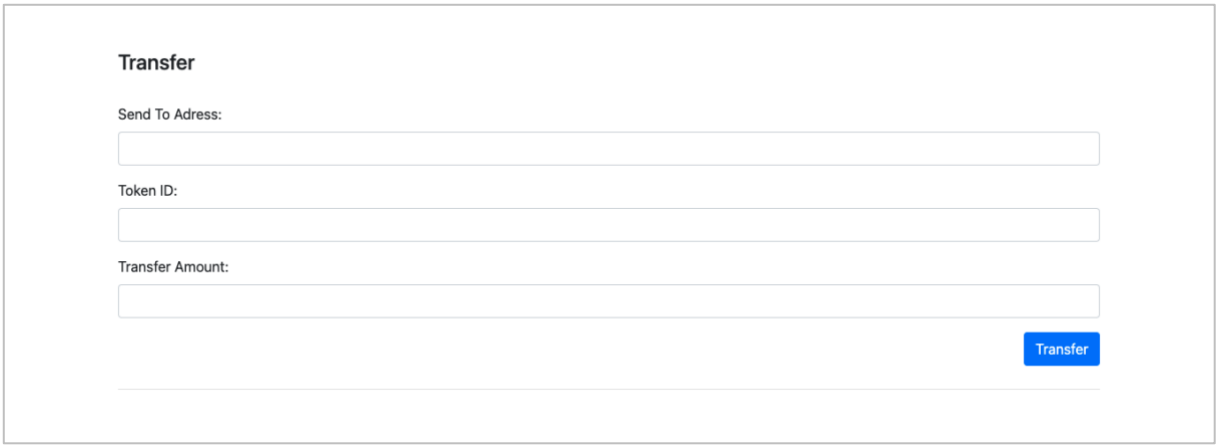


Figura 6.8 - Secção de transferência de NFTs

Aqui, os parâmetros recebidos como *input* são o endereço público da carteira para a qual será transferido o NFT, o número de identificação do NFT que pretendemos enviar e, por fim, a quantidade de *tokens* que queremos enviar. Com estes parâmetros é construída a transação, na qual é invocada a função **safeTransferFrom** do nosso *smart contract*. Posteriormente, a transação é assinada com a chave privada do administrador e enviada para a *Blockchain*.

Por último, na última secção da página do administrador (Figura 6.9), o mesmo pode consultar os NFTs que se encontram na sua carteira. Esta lista é obtida através da API do Moralis.

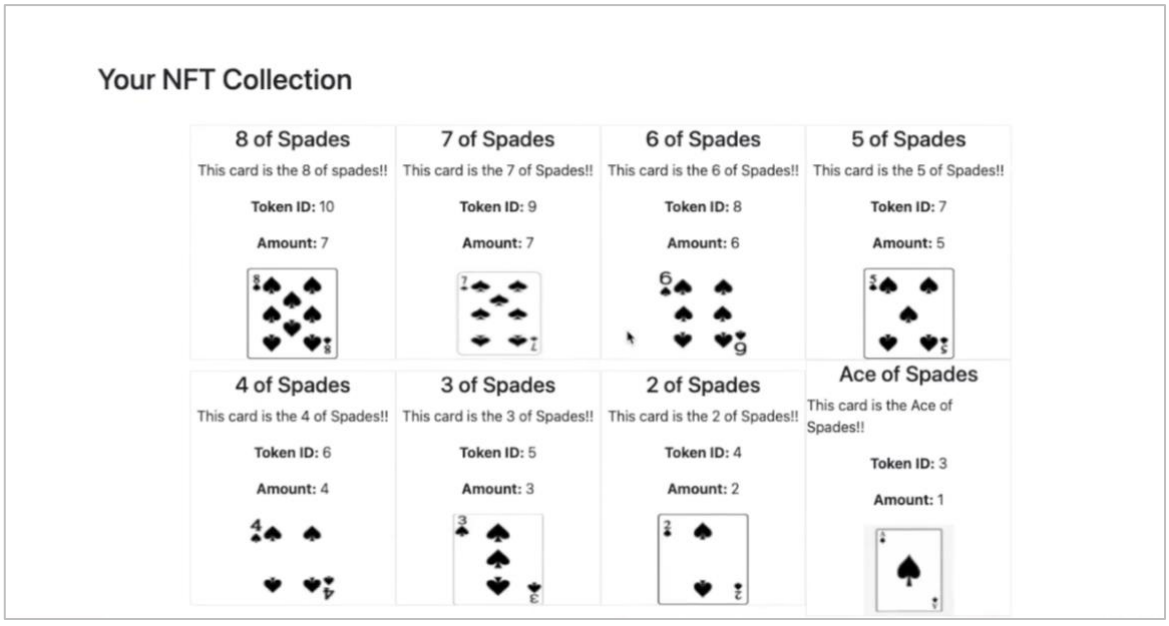


Figura 6.9 - Coleção de NFTs do administrador

Página do utilizador

O último ecrã desenvolvido corresponde à página do utilizador (Figura 6.10), onde o mesmo pode transferir NFTs e também ver os NFTs que tem colecionados. Estas operações funcionam da mesma maneira que as funcionalidades descritas anteriormente para o administrador, à exceção de quem envia as transações, que, neste caso, são delegadas para o administrador. Desta forma, o utilizador não paga qualquer taxa quando transfere um NFT.

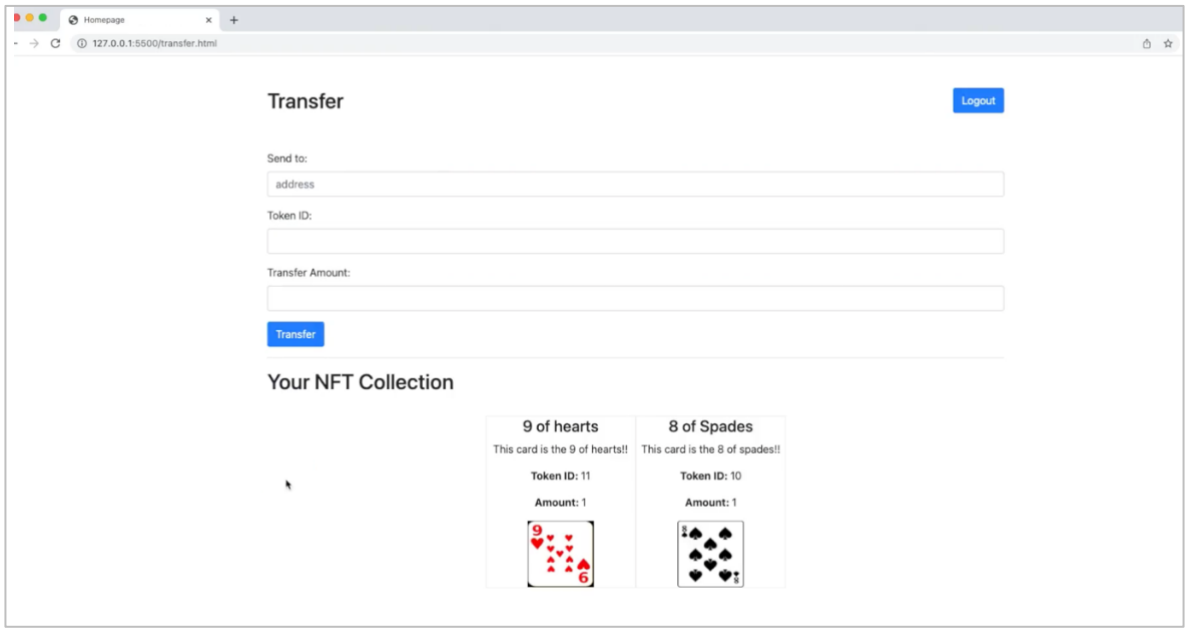


Figura 6.10 - Página do utilizador

6.4.2 Sumário

Dentro das funcionalidades desenvolvidas na aplicação *Web* descrita nas subsecções anteriores, estão incluídas as seguintes:

- A criação de uma *custodial wallet* para cada utilizador;
- O *upload* de ficheiros para o IPFS;
- Criação de NFTs;
- Transferência de NFTs entre dois endereços distintos;
- Delegação das transações dos utilizadores para o administrador.

Relacionando as funcionalidades anteriores com os requisitos funcionais levantados no Capítulo 4, constatamos que ambos se interligam.

Concluindo, com a realização desta aplicação, o estagiário construiu as bases necessárias para poder começar a fase seguinte do desenvolvimento, dentro do domínio do Shake It. Inclusive, o código desenvolvido nesta aplicação pôde ser reutilizado durante essa fase.

6.5 Blockchain Gateway

Como explicado no Capítulo 5, o *Blockchain Gateway* é o serviço responsável por satisfazer os requisitos do projeto, e é a ponte de ligação entre a *WIT Gamification Engine* e a *Blockchain*. Consiste numa API REST construída em NodeJS e ExpressJS, e na presente secção são apresentados os *endpoints* desenvolvidos, explicando o propósito e o comportamento de cada um.

6.5.1 GET /api/getContractAddress

Para os pedidos realizados a este *endpoint*, é retornado o endereço público do *Smart Contract* a ser usado pela nossa aplicação. A resposta devolvida deve incluir o respetivo endereço e deve enviar um código *HTTP 200 (OK)*. Em caso de erro, é retornada uma mensagem de erro e o código *HTTP 400 (Bad Request)*.

6.5.2 GET /api/getAccount

Este *endpoint* é responsável por criar uma nova carteira digital e, portanto, devolve um endereço público e a respetiva chave privada, geradas através da API do Web3JS. A resposta deve incluir a carteira, em formato JSON, e deve enviar um código *HTTP 200 (OK)*. Em caso de erro, é retornada uma mensagem de erro e o código *HTTP 400 (Bad Request)*.

6.5.3 POST /api/register

Para este projeto, um dos objetivos era manter a experiência do utilizador o mais genérica possível, e de certa forma equivalente à usabilidade presente no Shake It. Com isto em mente, para o presente estágio não podíamos esperar que todos os utilizadores possuíssem o *know-how* necessário para interagir com carteiras digitais, *Non-Fungible Tokens* ou com a *Blockchain* no geral. Por este motivo, uma das primeiras decisões que foi feita foi a custódia das carteiras digitais. No entanto, a esta decisão acresce um desafio, que é a delegação das transações dos utilizadores, para que os custos associados às mesmas não sejam pagos por eles.

Deste modo, o *endpoint* descrito nesta subsecção é o mecanismo central responsável pela delegação das transações (requisito funcional RF10).

A solução criada para contornar este desafio envolve duas etapas essenciais, realizadas durante o primeiro registo de um utilizador na aplicação. Estas etapas são representadas na Figura 6.11.

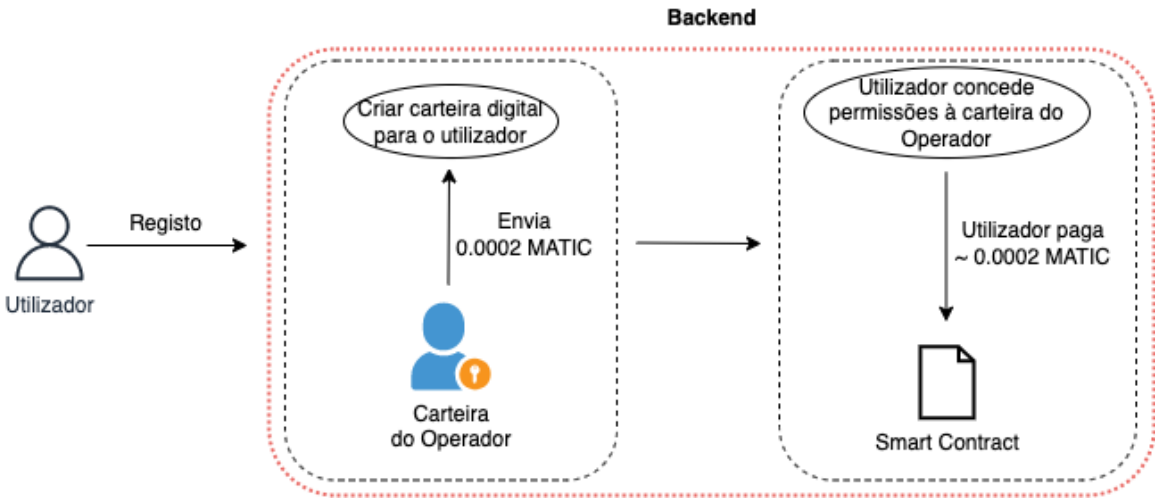


Figura 6.11 - Solução criada para a delegação de transações

Ao analisar a imagem da Figura 6.11, é possível distinguir as duas etapas previamente mencionadas:

- Depois de criada a carteira digital de um utilizador, é feito o **envio de MATIC** (criptomoeda da Polygon) para a carteira do mesmo, quantia essa que seja suficiente para efetuar apenas uma transação. No momento da escrita do presente relatório, este valor corresponde a aproximadamente 0.0002 MATIC (Capítulo 2, Tabela 2.1). A lógica usada para realizar esta etapa encontra-se no excerto de código da Figura 6.12;

```
async sendMatic(userAddress){
    const amountToSend = this.web3.utils.toWei( val: '0.0002', unit: 'ether');

    this.web3.eth.accounts.wallet.add(this.adminKey);

    const _nonce = await this.web3.eth.getTransactionCount(this.adminAddress, defaultBlock: 'latest');
    nonce = nonce > _nonce ? nonce : _nonce;

    console.log('[shakeit-blockchain-api][sendMatic()] Preparing to send MATIC to user...');

    await this.web3.eth.sendTransaction( transactionConfig: {
        from: this.adminAddress,
        to: userAddress,
        nonce: nonce,
        value: amountToSend,
        gas: 8000000,
    })
        .then(function (receipt : TransactionReceipt ) {
            nonce+=1;
            console.log('[shakeit-blockchain-api][sendMatic()] Sent 0.0002 MATIC to Address: ' + userAddress +
                ' with Transaction Hash: ' + receipt.transactionHash);
        });
}
```

Figura 6.12 - Lógica usada para enviar MATIC ao utilizador

- Por fim, uma vez que possuímos a custódia sobre as carteiras digitais dos utilizadores, é enviada uma transação para o nosso *smart contract*, assinada com a chave privada dos mesmos. Nessa transação, é invocada a função **setApprovalForAll** do *smart contract*, mencionada na secção 6.3.2 do presente capítulo. Aqui, são concedidas permissões ao endereço do Operador para efetuar transferências em nome do utilizador. A lógica que permite efetuar esta operação poder ser vista através do excerto de código apresentado na Figura 6.13.

```
async setApproval(userAddress, userKey){
    console.log('[shakeit-blockchain-api][setApproval()] Sending setApproval() Transaction...');

    const nonce_user = await this.web3.eth.getTransactionCount(userAddress, defaultBlock: 'latest');

    const tx = {
        'from': userAddress,
        'to': this.contractAddress,
        'nonce': nonce_user,
        'gas': 500000,
        'maxPriorityFeePerGas': 2999999987,
        'data': this.contract.methods.setApprovalForAll(this.adminAddress, approved: true).encodeABI()
    };

    const signedTx = await this.web3.eth.accounts.signTransaction(tx, userKey);
    const transactionReceipt = await this.web3.eth.sendSignedTransaction(signedTx.rawTransaction);

    if(transactionReceipt){
        console.log('[shakeit-blockchain-api][setApproval()] Set Approval done, with Transaction Hash: '
            + transactionReceipt.transactionHash + '\n');
    }
}
```

Figura 6.13 - Lógica usada para delegar as transações dos utilizadores

Concluindo, para realizar as etapas anteriormente descritas, este endpoint recebe no pedido HTTP o endereço público e a chave privada de um utilizador, em formato JSON. De seguida, são feitas verificações para ver se algum dos parâmetros está em falta ou se o endereço fornecido é inválido. Depois de efetuadas as operações necessárias, em caso de sucesso é devolvida uma mensagem de confirmação, com o código *HTTP 200 (OK)*. Caso contrário, é devolvida a mensagem de erro e o código *HTTP 400 (Bad Request)*.

6.5.4 POST /api/mint

Este é o *endpoint* responsável por dar *mint* de um NFT, ou seja, criar um NFT. Como tal, este endpoint recebe um *array* de *bytes* da imagem, o número de identificação (*token_id*) que será associado ao NFT, o nome, a descrição e a quantidade de *tokens* que serão criados para esse *token_id*. Após verificar se todos os parâmetros foram fornecidos, é iniciado o processo do *mint*, começando pelo *upload* da imagem para o IPFS, representado no excerto da Figura 6.14.

```
// Upload an image to IPFS
async uploadImage(imageFile) {
    const buffer = Buffer.from(imageFile);
    const base64String = buffer.toString( encoding: 'base64');

    const file = new Moralis.File( name: "image.png", data: {base64 : base64String });
    await file.saveIPFS( options: {useMasterKey:true});

    console.log('[shakeit-blockchain-api] Uploaded image to IPFS at ' + file.ipfs());

    return file.ipfs();
}
```

Figura 6.14 - Lógica usada no *upload* da imagem para o IPFS

Depois, com o *link* do IPFS obtido através do método da Figura 6.14, é criado um ficheiro JSON contendo esse *link*, o nome, e a descrição que serão associados ao NFT. Esta lógica pode ser consultada na Figura 6.15.

```
// Upload metadata object to IPFS
async uploadMetadata(imageURL, name, description) {
    const metadata = {
        'name': name,
        'description': description,
        'image': imageURL
    };

    const btoa = function (str) {
        return Buffer.from(str, encoding: 'binary').toString( encoding: 'base64');
    };

    const file = new Moralis.File( name: 'file.json', data: {base64 : btoa(JSON.stringify(metadata))});
    await file.saveIPFS( options: {useMasterKey:true});

    console.log('[shakeit-blockchain-api] Uploaded metadata to IPFS at ' + file.ipfs());

    return file.ipfs();
}
```

Figura 6.15 - Lógica usada no *upload* da metadata para o IPFS

Por fim, com o *link* do IPFS obtido através do método da Figura 6.15, é construída a transação que será enviada ao *smart contract*, juntamente com os outros parâmetros que serão enviados para a função *mint* do contrato. Finalmente, a transação é assinada com a chave privada do operador e enviada para a *Blockchain*.

O *endpoint* deve devolver uma mensagem de confirmação e o código *HTTP 200 (OK)*. Em caso de erro, devolve a mensagem de erro e o código *HTTP 400 (Bad Request)*.

6.5.5 POST /api/transfer

Este *endpoint* é usado para transferir um NFT de um endereço para outro. Como tal, no pedido HTTP são enviados os seguintes parâmetros: endereço público do remetente, endereço público do destinatário, o *token_id* do NFT que se pretende transferir, e a quantidade que irá ser transferida. Depois de verificar se algum dos parâmetros está em falta ou se algum dos endereços fornecidos é inválido, é iniciada a lógica da transferência. Começa-se por construir a transação, representada na Figura 6.16, em que é invocado o método **safeTransferFrom** do *smart contract*. Por fim, a transação é assinada e enviada pelo operador.

```
const tx = {
    'from': this.adminAddress,
    'to': this.contractAddress,
    'nonce': nonce,
    'gas': 500000,
    'maxPriorityFeePerGas': 2999999987,
    'data': this.contract.methods.safeTransferFrom(fromAddress, toAddress, tokenID, transferAmount, 0x00).encodeABI()
};
```

Figura 6.16 - Transação construída para a transferência de um NFT

Na resposta ao pedido, o *endpoint* deve devolver a *hash* da transação efetuada e o código *HTTP 200 (OK)*. Em caso de erro, devolve a mensagem de erro e o código *HTTP 400 (Bad Request)*.

6.5.6 POST /api/transfer/batch

Quanto ao presente *endpoint*, o mesmo é responsável por realizar transferências de NFTs em lote. Assim, quando se pretende enviar múltiplos NFTs, apenas é enviada uma transação, poupando tempo e o dinheiro que estaria associado a cada uma das diferentes transferências.

Este *endpoint* funciona de maneira semelhante ao anterior, à exceção de que recebe um *array* de *token_ids* e um *array* com as respetivas quantidades que serão transferidas. Depois de verificar se algum dos parâmetros está em falta ou se algum dos endereços fornecidos é inválido, é construída a transação, na qual é invocado o método **safeBatchTransferFrom** do *smart contract*. Por fim, a transação é assinada e enviada pelo operador.

Na resposta ao pedido, o *endpoint* deve devolver a *hash* da transação efetuada e o código *HTTP 200 (OK)*. Em caso de erro, devolve a mensagem de erro e o código *HTTP 400 (Bad Request)*.

6.5.7 GET /api/getCollection/:address

O objetivo deste *endpoint* é retornar uma lista com os NFTs presentes num determinado endereço público, o qual é enviado juntamente com o pedido HTTP. Em primeiro lugar, é feita uma verificação para confirmar se o endereço fornecido é válido. Depois, recorrendo à API do Moralis, é feito um pedido para receber a lista de NFTs nesse endereço. A lógica usada está disponível na Figura 6.17.

```
async getNFTCollection(address){

    const options = {
        chain: 'mumbai',
        address: address,
        token_address: this.contractAddress
    };

    const nfts = await Moralis.Web3API.account.getNFTsForContract(options);

    const metadata = [];

    Array.prototype.forEach.call( nfts.result, argArray: e => {
        let aux = {"metadata": e.token_uri, "amount": e.amount, "token_id": e.token_id};
        metadata.push(aux);
    });

    return metadata;
}
```

Figura 6.17 - Lógica usada para ver os NFTs num determinado endereço

Por fim, o *endpoint* irá devolver uma resposta com a lista de NFTs (se existir) em formato JSON e o código *HTTP 200 (OK)*. Em caso de erro, devolve a mensagem de erro e o código *HTTP 400 (Bad Request)*.

6.5.8 GET /api/getOperatorCollection

Por último, o presente *endpoint* tem como objetivo devolver a lista de NFTs presentes na carteira do operador. A lógica usada é a mesma que a do *endpoint* anterior, no entanto, o endereço público usado é o do operador.

A resposta devolvida pelo *endpoint* será lista de NFTs (se existir) em formato JSON e o código *HTTP 200 (OK)*. Em caso de erro, devolve a mensagem de erro e o código *HTTP 400 (Bad Request)*.

6.5.9 Considerações finais

Por fim, é importante salientar alguns dos mecanismos criados durante o desenvolvimento do presente serviço (*Blockchain Gateway*), nomeadamente quanto ao controlo de falhas. Dado que grande parte dos *endpoints* previamente mencionadas lidam com tarefas assíncronas e que dependem da *Blockchain*, é importante conseguir mitigar algum imprevisto fora do nosso controlo.

No Shake It, dado que todas as transações que envolvem a *Blockchain* são enviadas pela mesma carteira (do operador), teve que ser implementada uma maneira de enviar as transações sequencialmente. Isto foi feito porque a mesma carteira não pode enviar múltiplas transações em simultâneo, sem a transação anterior ser completada.

Deste modo, foi implementado um *lock*, ou seja, um mecanismo que bloqueia uma secção do código. Na secção bloqueada apenas pode entrar uma *thread*, bloqueando o acesso às restantes. Depois dessa *thread* executar a sua tarefa, o *lock* é libertado. Na imagem da Figura 6.18 é apresentado um exemplo de onde o *lock* é usado.

```
try {
    // pause execution with lock
    await lock.acquire()

    console.log('[shakeit-blockchain-api][/api/transfer] Received request to Transfer NFT. Params: [from] '
        + params.from + ' [to] ' + params.to + ' [token_id] ' + params.token_id + ' [amount] ' + params.amount)

    await transfer.transferNFT(params.from, params.to, params.token_id, params.amount)
        .then(r => res.status( code: 200).send(r))
} catch (e) {
    console.error(e)
    res.status( code: 400).send( body: 'error')
} finally {
    // release lock
    lock.release()
}
```

Figura 6.18 - Exemplo de utilização do *lock*

Por fim, outra medida utilizada para assegurar que não são feitas múltiplas transações em simultâneo, foi o uso do *nonce*. O *nonce* é um número pseudoaleatório que está associado a cada transação, e não pode ser replicado. A Figura 6.19 apresenta um excerto de código onde esta lógica é aplicada.

```
// get current nonce perceived by blockchain
const _nonce = await this.web3.eth.getTransactionCount(this.adminAddress, defaultBlock: 'latest');
nonce = nonce > _nonce ? nonce : _nonce;

const tx = {
    'from': this.adminAddress,
    'to': this.contractAddress,
    'nonce': nonce,
    'gas': 500000,
    'maxPriorityFeePerGas': 2999999987,
    'data': this.contract.methods.safeTransferFrom(fromAddress, toAddress, tokenID, transferAmount, 0x00).encodeABI()
};

const signedTx = await this.web3.eth.accounts.signTransaction(tx, this.adminKey);
const transactionReceipt = await this.web3.eth.sendSignedTransaction(signedTx.rawTransaction);

// increase nonce
nonce+=1;
```

Figura 6.19 - Lógica usada para controlar o *nonce*

Primeiramente, é guardada no servidor uma instância do nonce (representado na Figura 6.19 como *nonce*). Depois, através da API do Web3JS é obtido o valor do último *nonce* usado, de acordo com a *Blockchain* – que devolve o valor do *nonce* para o último bloco publicado, não tendo em consideração qualquer transação pendente.

Por fim, são comparados os dois *nonces* e o que for maior é o que será utilizado no envio da transação. Finalmente, o valor do *nonce* é incrementado depois de a transação ser enviada.

6.6 Backoffice

Como vimos na arquitetura do projeto, detalhada no Capítulo 5, o *Backoffice* é uma das aplicações dentro da *WIT Gamification Engine*. Apenas um administrador tem acesso a esta aplicação, e é onde estão disponíveis as funcionalidades que gerem as coleções que estão ativas no Shake It, as probabilidades, os diferentes baralhos de cartas, entre outras.

Para ser possível a realização dos requisitos funcionais à frente mencionados, o estagiário fez algumas alterações ao *Backoffice*, detalhadas ao longo da presente secção.

6.6.1 Lançamento de uma coleção de NFTs

Na solução já existente do Shake It, existe um fluxo que tem de ser cumprido antes de uma coleção ficar ativa, isto é, antes de ser lançada. Esse fluxo é apresentado na Figura 6.20.

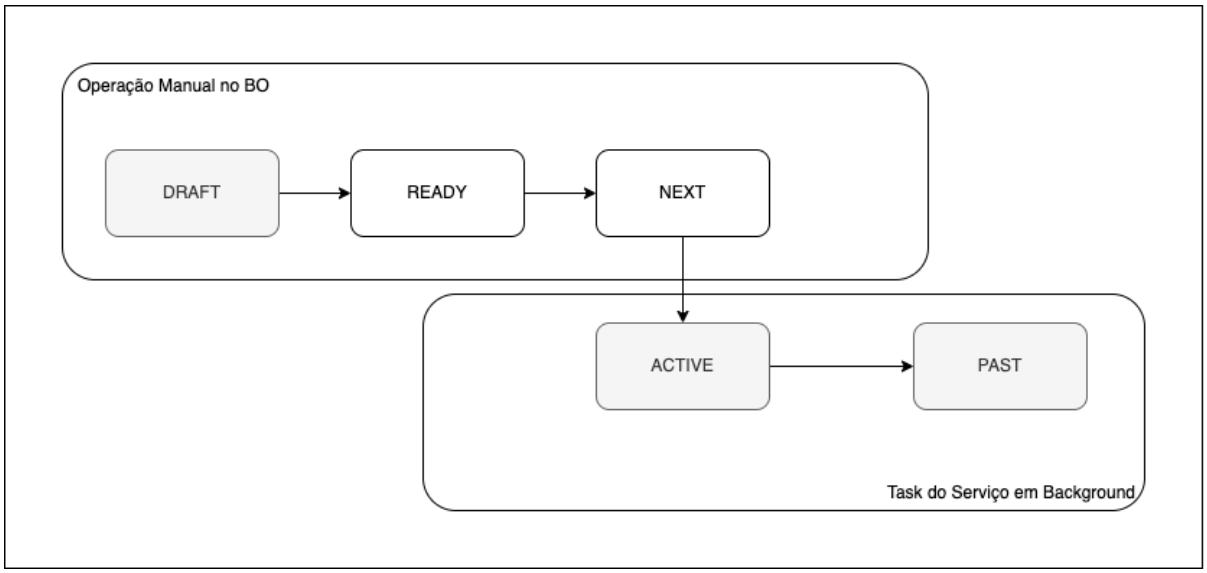


Figura 6.20 - Fluxo dos estados de uma coleção

Inicialmente, a uma coleção recém-criada é-lhe atribuído o estado *DRAFT*. Depois, quando a mesma reúne todas as condições necessárias para que possa ser lançada, o seu estado passa para *READY*. Posteriormente, quando quisermos preparar a coleção para a mesma ser lançada automaticamente, alteramos o seu estado para *NEXT*. Nesse estado, o operador consegue definir a data de início e de fim da respetiva coleção e, consoante essas datas, a coleção passa para *ACTIVE*, quando chegar a altura para ser lançada. Quando a mesma chega ao fim, o estado passa automaticamente para *PAST* e o ciclo continua.

Agora, no contexto do estágio e de modo a incorporar a funcionalidade de criar os NFTs, foram adicionados 3 novos estados:

- **MINT** – Este estado é selecionado manualmente, e inicia o processo de criar a coleção de NFTs, de acordo com o baralho de cartas selecionado. Depois de selecionar esta opção, o estado passa automaticamente para *MINTING*;

- **MINTING** – Este estado indica que a coleção de NFTs ainda está a ser criada;
- **MINTED** – Quando a coleção de NFTs é criada, o estado passa automaticamente para *MINTED*, indicando ao operador que pode continuar e selecionar as datas de início e fim da respetiva coleção.

O novo fluxo é apresentado na imagem da Figura 6.21.

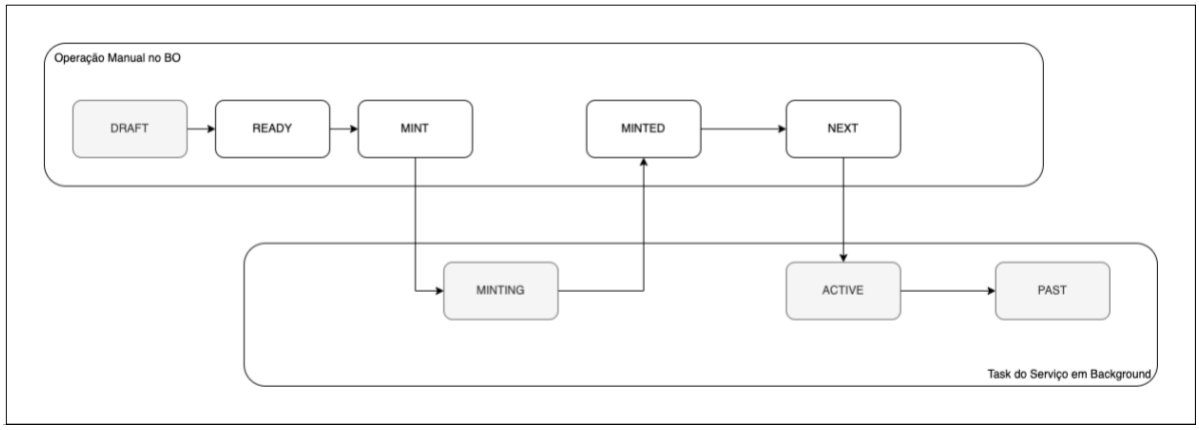


Figura 6.21 - Fluxo criado pelo estagiário

Concluindo, quanto à operação de *MINT*, quando esta é selecionada pelo operador é criada uma tarefa assíncrona no *Backoffice*. Esta tarefa é responsável por obter a lista de cartas da coleção selecionada e enviar um pedido HTTP ao *Blockchain Gateway*, para o *endpoint /api/mint*. Nesse pedido são enviados os parâmetros necessários para a criação de cada um dos NFTs.

6.6.2 Ver coleção de NFTs de um utilizador

Para este requisito funcional foi criado um novo ecrã no *Backoffice* e o respetivo separador no menu lateral, intitulado de *Search User Wallets* (Figura 6.22).

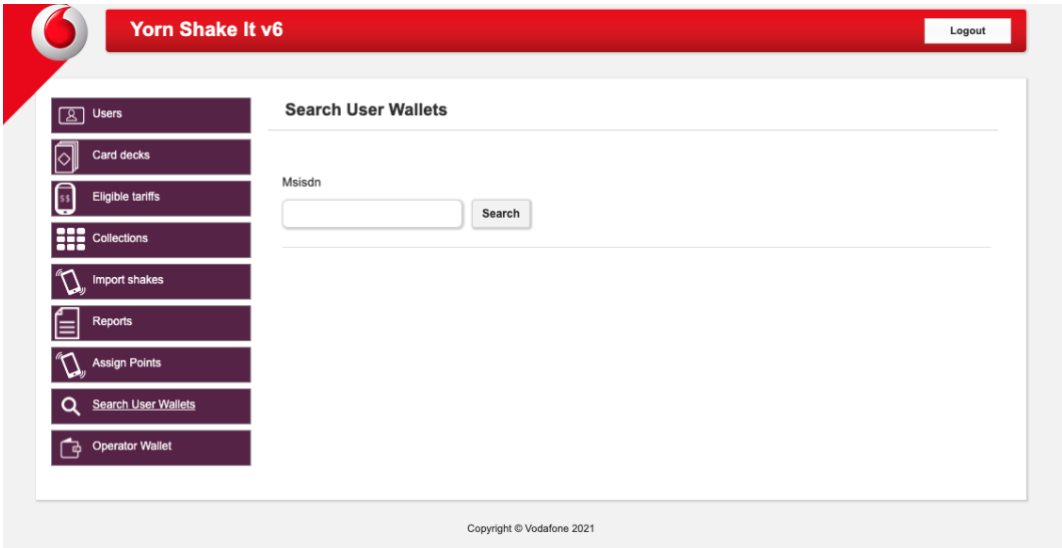


Figura 6.22 - Ecrã do *Backoffice* onde se pode pesquisar pelos NFTs de um utilizador

Neste ecrã, o operador pode consultar os NFTs que estejam presentes na carteira de um utilizador, e faz esta pesquisa através do número de telemóvel do mesmo (*Msisdn*). Para obter esta lista, é feito um pedido HTTP ao *endpoint /api/getCollection/:address* do *Blockchain Gateway*.

Na Figura 6.23 é exemplificado o resultado desta pesquisa, no caso de um utilizador com NFTs na sua carteira.

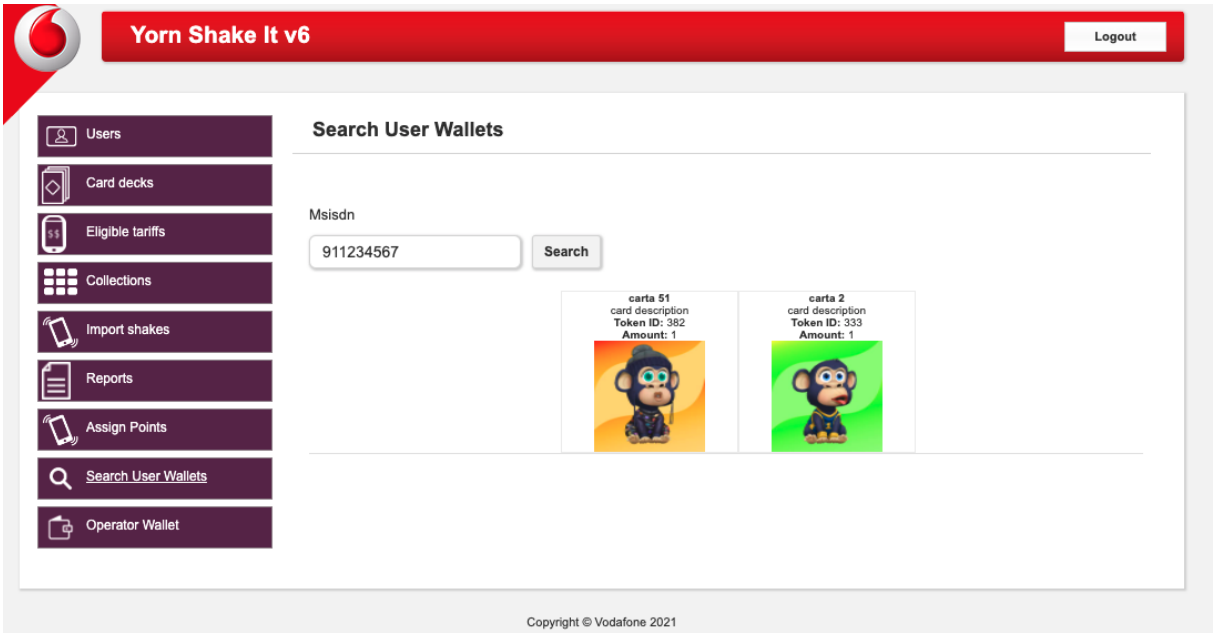


Figura 6.23 - Exemplo: Ver NFTs de um utilizador

Para cada um dos NFTs apresentados nesta pesquisa é indicado o nome, a descrição, o *token_id*, a quantidade de *tokens* que o utilizador possuí e, por fim, a respetiva imagem.

Caso o utilizador não tenha NFTs na sua carteira digital, a aplicação mostra uma mensagem com essa informação. Além disso, é apresentada uma mensagem de erro caso o número de telemóvel não seja reconhecido ou o endereço público associado ao mesmo seja inválido.

6.6.3 Ver coleção de NFTs do operador

Por último, para este requisito funcional foi seguida a mesma abordagem que para o requisito anterior, descrito na secção 6.6.2. O estagiário adicionou um novo ecrã ao *Backoffice* e o respetivo separador no menu lateral, intitulado de *Operator Wallet*.

Com esta funcionalidade, o objetivo pretendido é apresentar ao operador todos os NFTs que se encontram na sua carteira, e as respetivas informações de cada um. Para tal, é feito um pedido HTTP ao *endpoint /api/getOperatorCollection* do *Blockchain Gateway*.

A Figura 6.24 mostra o ecrã previamente descrito, e a respetiva coleção de NFTs do operador.

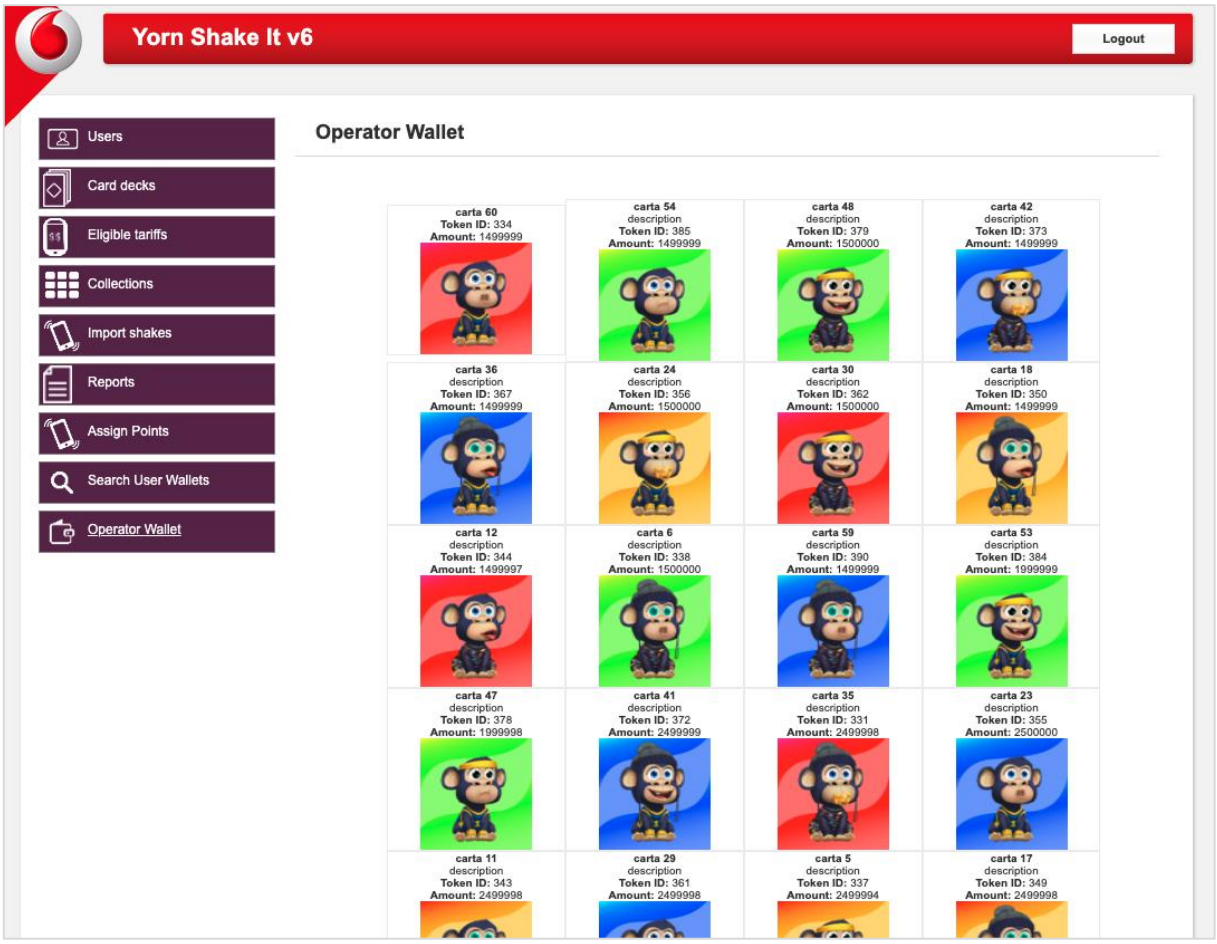


Figura 6.24 - Ecrã onde são apresentados os NFTs do operador

6.7 Web Client e *Services*

Os outros dois componentes da *WIT Gamification Engine*, detalhados no Capítulo 5, são o *Web Client* (*frontend*) e os *Services* (*backend*). Os mesmos estiveram sujeitos a alterações por parte do estagiário, para possibilitar a realização dos requisitos mencionados ao longo da presente secção.

6.7.1 Criação da carteira digital e delegação de transações

Sempre que um utilizador entra no Shake It é feita uma verificação para confirmar se o mesmo está ou não registado na aplicação. Assim, no caso de um utilizador não registado, foi adicionada a funcionalidade que permite criar uma carteira digital para o mesmo, composta pelo respetivo endereço público e chave privada.

Para tal, no componente do *Services* responsável por esta verificação, é enviado um pedido HTTP para o *Blockchain Gateway*, para o *endpoint /api/getAccount*. Este cria a carteira digital e a mesma é enviada como resposta ao pedido.

Depois de receber o endereço público e a chave privada devolvidos pelo *Blockchain Gateway*, é cumprido o mecanismo que permite a delegação das transações dos utilizadores para o operador. Pelas razões enunciadas na secção 6.5.3, esta é uma etapa essencial e a mesma deve ser realizada para que todas as transações provenientes da carteira de um utilizador possam ser enviadas pelo operador, para que este cubra esses custos.

Deste modo, é novamente enviado um pedido HTTP para o *Blockchain Gateway*, desta vez para o *endpoint /api/register*. No mesmo, todo o fluxo mencionado na secção 6.5.3 é cumprido e é devolvida uma resposta de confirmação.

Por fim, quando os passos anteriores forem completados, a carteira digital é atribuída ao utilizador e é guardada na base de dados, salientando que a chave privada é encriptada previamente.

6.7.2 Abertura de *shakes*

Para o primeiro requisito funcional, depois de receber uma carta ao abrir um *shake*, é esperado que o utilizador receba o NFT correspondente.

Na solução existente do Shake It, quando um utilizador abre um *shake* o *Web Client* envia um pedido ao microsserviço responsável pela abertura dos *shakes*, e aí é selecionada uma carta de forma aleatória e a mesma é atribuída ao utilizador.

Deste modo, e para que se pudesse enviar o respetivo NFT ao utilizador, a este microsserviço foi adicionada uma tarefa assíncrona cujo propósito é enviar um pedido HTTP para o *Blockchain Gateway*, para o mesmo invocar a função de transferência do nosso *smart contract*.

Este novo fluxo é apresentado na Figura 6.25.

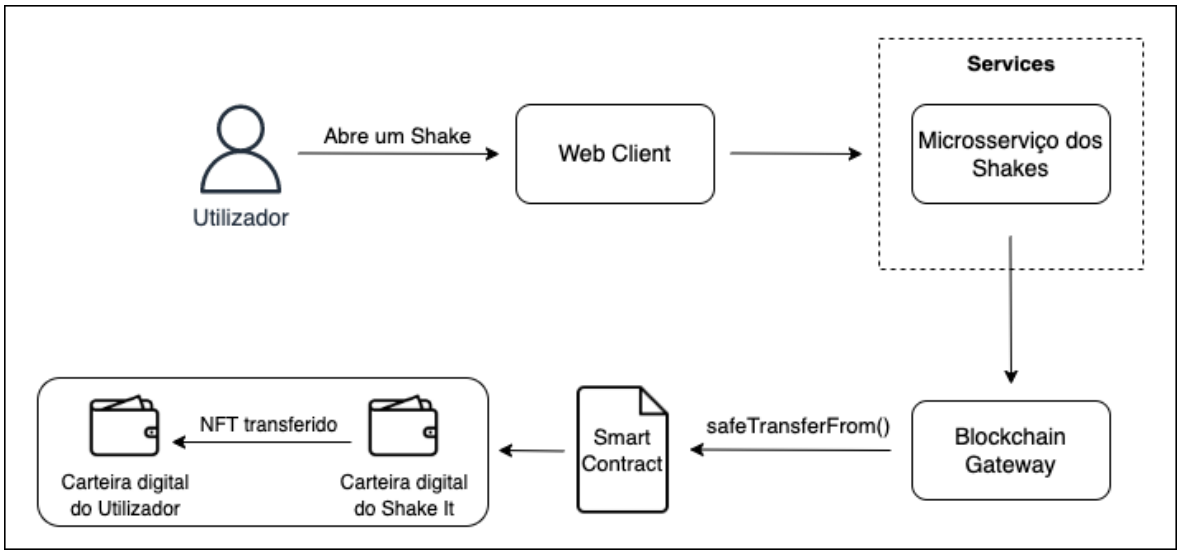


Figura 6.25 - Abertura de um shake

Depois de o NFT ser transferido da carteira principal do Shake It para a carteira do utilizador, a *hash* da respetiva transação é devolvida como resposta ao pedido HTTP do microsserviço mencionado anteriormente. Isto é feito para que, se desejar, o utilizador possa consultar mais detalhes sobre os NFTs que tem colecionados, como irá ser apresentado na próxima subsecção. Essa *hash* é a prova de que o NFT foi de facto enviado, e através dela é possível obter informações como: a data e hora da transação; o endereço público da carteira de onde o NFT foi enviado; o número do bloco onde se encontra registada a transação; o endereço do *smart contract* que realizou a transferência, entre outras.

No que toca ao *frontend*, a funcionalidade descrita na presente subsecção não sofreu qualquer alteração visual, não afetando deste modo a usabilidade do jogo. Além disso, apesar de os tempos de comunicação com a *Blockchain* serem altos comparativamente a um pedido HTTP leve, como toda esta operação é realizada de forma assíncrona não existe qualquer impacto no utilizador, e não é utilizado nenhum tipo de espera ativa.

6.7.3 Ver detalhes de NFTs colecionados

Para a presente funcionalidade pretendia-se que o utilizador, ao navegar para o menu da coleção, pudesse selecionar qualquer NFT e ver os respetivos detalhes. Nomeadamente, o número de identificação atribuído ao seu NFT, o tipo de *token standard* herdado pelo *Smart contract* (ERC-1155), o endereço do mesmo, e, por fim, a *hash* da transação mencionada na secção anterior.

A alteração aqui descrita, realizada no *Web Client*, pode ser consultada na Figura 6.26.

Figura 6.26 - Detalhes de um NFT

Depois, ao clicar sobre o endereço do *smart contract* ou sobre a *hash* da transação, o utilizador é redirecionado para a respetiva página no Polygon Scan [104], mencionado na secção 6.3.3. Aqui, o mesmo pode consultar mais informações acerca do seu NFT e do contrato em si, diretamente num *blockchain explorer* onde os dados registados são inalteráveis e públicos.

Quanto à última opção, designada na Figura 6.26 como "*View NFT on OpenSea*", este é outro botão que redireciona o utilizador para a página do *OpenSea* [110], onde o seu NFT é apresentado (Fig. 6.27). Este botão foi implementado para que numa iteração futura sobre este projeto, em que a carteira do utilizador seja associada à aplicação, o utilizador possa ir diretamente para a sua carteira no *OpenSea* e ver os NFTs que tem colecionados, assim como efetuar transações autonomamente, fora do contexto do Shake It.

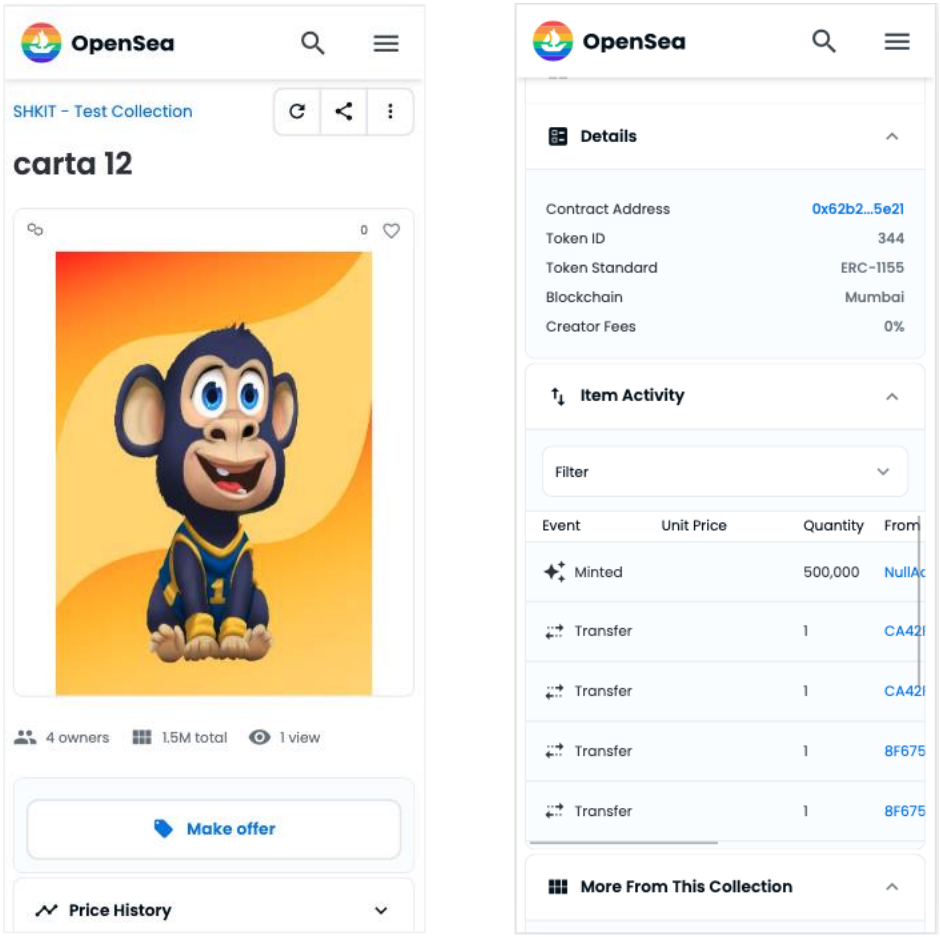


Figura 6.27 - *OpenSea*

6.7.4 Troca de cartas repetidas

Quanto à troca de cartas repetidas, na solução existente do Shake It é possível trocar cartas repetidas por *shakes*, por pontos, e com amigos.

Ao longo da presente subsecção serão apresentadas essas funcionalidades e as respetivas alterações que tiveram que ser feitas. Em todas elas, as modificações foram realizadas no componente *Services*, pelo que o *frontend* não sofreu qualquer alteração visual.

Troca de 4 cartas repetidas por 1 shake

Para este requisito, após obter a lista com as 4 cartas que serão transacionadas, é enviado um pedido HTTP para o *endpoint /api/transfer/batch* do *Blockchain Gateway*, com os respetivos parâmetros. Aqui, de forma a que se consigam transferir os 4 NFTs numa só transação, é invocada a função *safeBatchTransferFrom* do *smart contract*.

É de salientar que esta transferência de NFTs da carteira do utilizador para a carteira do operador só é possível graças à delegação das transações, descrita anteriormente na secção 6.7.1.

O diagrama de sequência da Figura 6.28 demonstra o fluxo seguido para este requisito funcional.

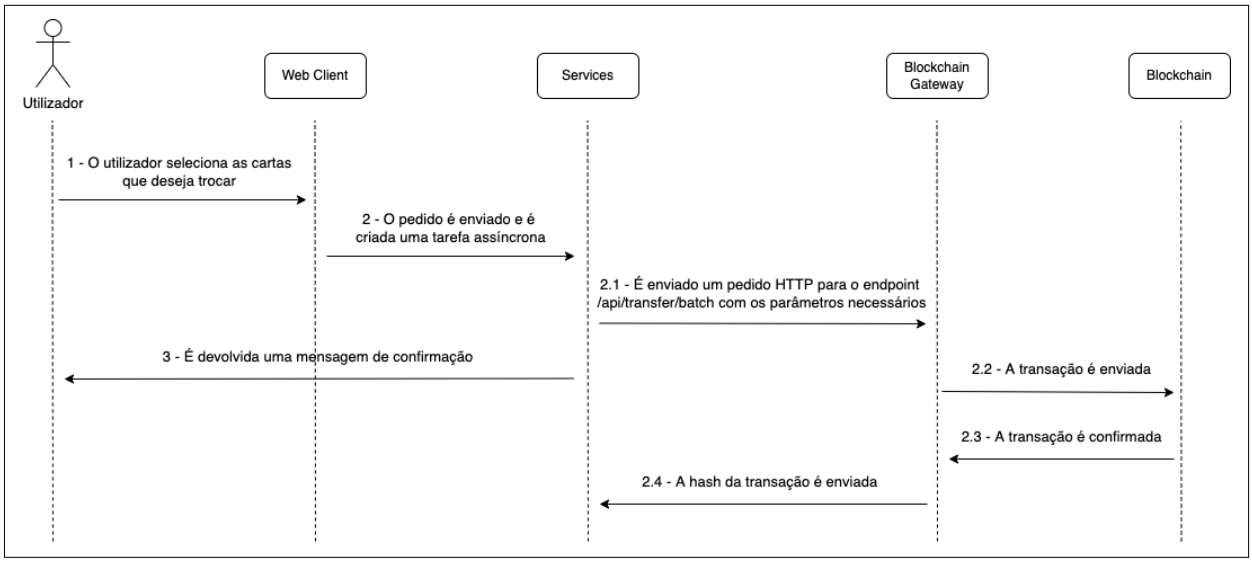


Figura 6.28 - Diagrama de sequência para a troca de 4 NFTs por 1 *shake*

Troca de cartas repetidas por pontos ou com amigos

Por último, quanto às funcionalidades de troca de cartas repetidas por pontos ou com amigos, a lógica seguida é idêntica à mencionada anteriormente. Distingue-se pelo facto de o *endpoint* do *Blockchain Gateway* ao qual é realizado o pedido HTTP do *Services*, ser diferente do *endpoint* usado na funcionalidade anterior. Neste caso, como apenas é transacionado um NFT por operação, é usada a função *safeTransferFrom* do *smart contract*.

6.7.5 Histórico de transações

Por fim, e apesar de não constar na lista de requisitos inicial, a equipa optou por implementar a presente funcionalidade para que o utilizador conseguisse ver, de forma autónoma, um histórico de tudo o que foi realizado na sua carteira digital.

Além disso, dado que parte das funcionalidades descritas nas secções anteriores não dispõem de apoio visual, este requisito torna-se útil para que o utilizador possa acompanhar em tempo real uma determinada transação, ou para comprovar que a mesma foi efetuada.

Para esta funcionalidade, foi adicionado ao *Web Client* um botão no perfil do utilizador, que quando pressionado redireciona-o para a sua página do *Polygon Scan*. Aqui pode consultar o seu endereço público, um histórico de todas as transações que foram efetuadas com o seu endereço e também o número de *tokens* que possui. (Fig. 6.29)

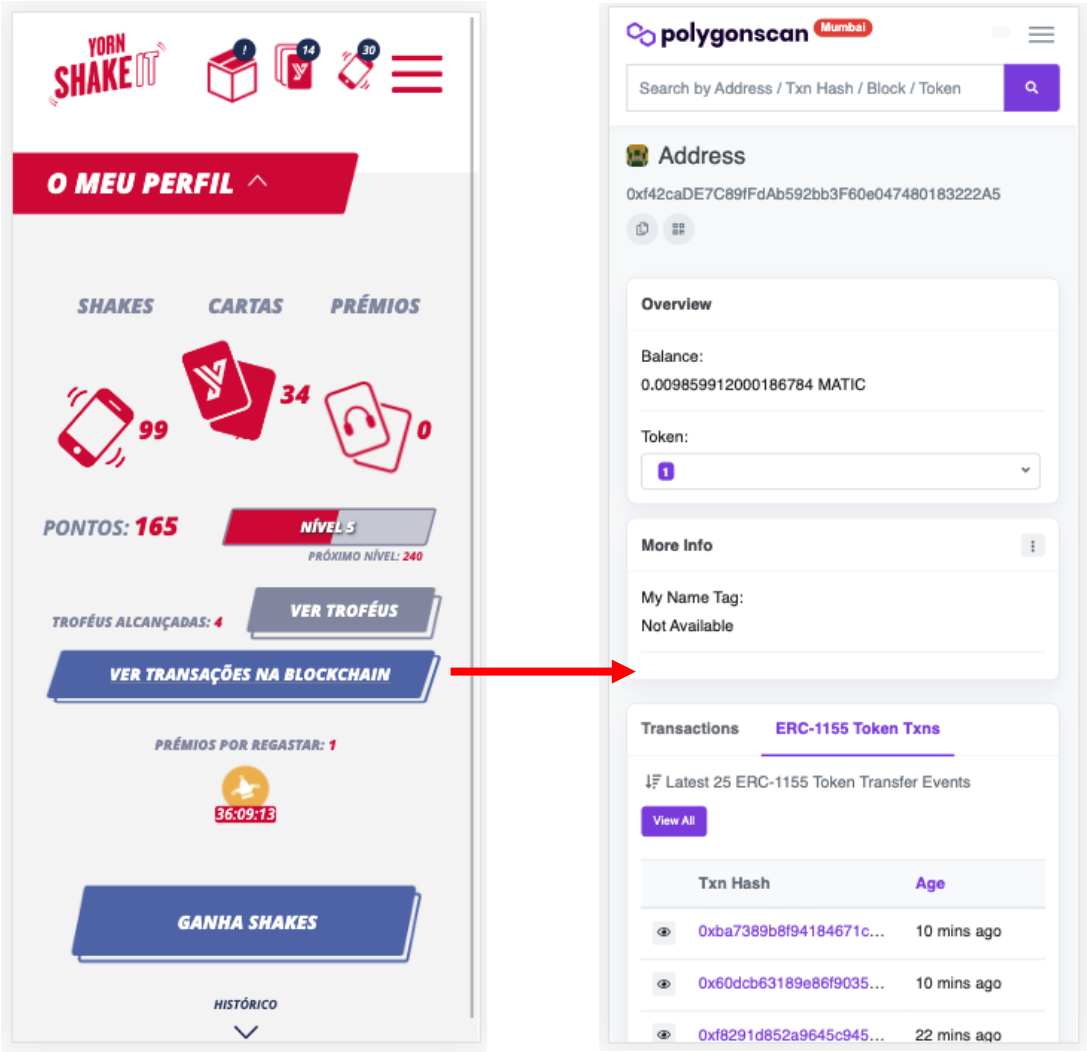


Figura 6.29 - Botão do histórico de transações

Capítulo 7
Testes e Validações

Os testes de *software* servem como um controlo de qualidade e são, geralmente, realizados na etapa final de um projeto de *software*. O seu propósito é simples, e resume-se à verificação de se o produto obtido corresponde aos requisitos esperados e às necessidades do projeto, e para garantir que o produto de *software* está livre de *bugs*. Além disto, caso se identifique algum resultado indesejado ou que algum requisito esteja ausente, é importante detetar estes problemas antecipadamente, para que se possam realizar as devidas alterações antes de entregar o produto final.

Neste capítulo, são apresentados os testes utilizados para validar a implementação descrita no capítulo 6 e o cumprimento dos requisitos funcionais e não funcionais mencionados no capítulo 4.

7.1 Testes unitários

Nesta secção são descritos os testes unitários efetuados ao *Blockchain Gateway*, desenvolvido pelo estagiário durante a fase de implementação, recorrendo às *frameworks* Jest [107] e Supertest [108].

O objetivo dos testes unitários é isolar pequenas partes do sistema e verificar se elas funcionam conforme esperado. Os testes são executados num ambiente isolado e é fornecido um *input* para efeitos de simulação e o respetivo *output* esperado. Após a execução, é feita uma comparação entre o *output* obtido e o *output* esperado e, se se corresponderem, o teste será considerado bem sucedido.

7.1.1 *Endpoint* 1 - */api/transfer*

Para este *endpoint*, o qual é responsável por interagir com o *smart contract* para efetuar transferências de NFTs, é fundamental que o mesmo receba todos os parâmetros necessários e que os mesmos sejam válidos. Deste modo foi desenvolvido o seguinte *script* (Figuras 7.1 e 7.2).

```
const metadata = {
    from: "OPERATOR",
    to: "0xf42caDE7C89fFdAb592bb3F60e047480183222A5",
    token_id: "371",
    amount: "5"
}

describe("POST /api/transfer", () => {
    describe("Given a from-address, to-address, token-id and an amount", () => {
        jest.setTimeout(timeout);

        test("should respond with a 200 status code", async () => {
            const response = await request(app).post("/api/transfer").send(metadata)
            expect(response.statusCode).toBe( expected: 200)
        })
        test("should specify text/html in the content type header", async () => {
            const response = await request(app).post("/api/transfer").send(metadata)
            expect(response.headers['content-type']).toEqual(expect.stringContaining( str: "text"))
        })
        test("response has a transaction hash", async () => {
            const response = await request(app).post("/api/transfer").send(metadata)
            expect(response).toBeDefined()
        })
    })
```

Figura 7.1 - Teste com parâmetros válidos

Na Figura 7.1 é testado um cenário onde este *endpoint* recebe um pedido com os parâmetros corretos para a sua execução, nomeadamente os endereços públicos do remetente e do destinatário, o ID do NFT e a quantidade que pretendemos transferir. Assim, sendo estes atributos válidos, a resposta devolvida deverá:

- retornar um código *HTTP 200 (OK)*;
- especificar *Content-type: text/html* nos *Headers* da resposta;
- retornar a *hash* da transação.

```
277
278     describe("When any of the parameters is missing", () => {
279         test("should respond with a status code of 400", async () => {
280             const bodyData = [
281                 {from: "OPERATOR", to: "0xf42caDE7C89fFdAb592bb3F60e047480183222A5", token_id: "371"},
282                 {from: "OPERATOR", to: "0xf42caDE7C89fFdAb592bb3F60e047480183222A5", amount: "5"},
283                 {from: "OPERATOR", token_id: "371", amount: "5"},
284                 {to: "0xf42caDE7C89fFdAb592bb3F60e047480183222A5", token_id: "371", amount: "5"},
285                 {}
286             ]
287             for (const body of bodyData) {
288                 const response = await request(app).post("/api/transfer").send(body)
289                 expect(response.statusCode).toBe( expected: 400)
290             }
291         })
292     })
293
294     describe("Given an invalid ETH address", () => {
295         jest.setTimeout(timeout);
296         test("should respond with a status code of 400", async () => {
297             const bodyData = [
298                 {from: "OPERATOR", to: "0xtoBadAddress", token_ids: "371", amounts: "5"},
299                 {from: "0xfromBadAddress", to: "OPERATOR", token_ids: "371", amounts: "5"}
300             ]
301
302             for (const body of bodyData) {
303                 const response = await request(app).post("/api/transfer").send(body)
304                 expect(response.statusCode).toBe( expected: 400)
305             }
306         })
307     })
308 })
```

Figura 7.2 - Testes com parâmetros inválidos ou em falta

A Figura 7.2 representa um cenário onde este *endpoint* recebe um endereço público inválido (linha 278) ou quando pelo menos um dos parâmetros está em falta (linha 294). Para estes testes, a resposta deverá:

- retornar um código *HTTP 400 (Bad Request)*;

Depois de executar o *script* anterior, obtivemos os resultados apresentados na Figura 7.3.

```
  POST /api/transfer
    Given a from-address, to-address, token-id and an amount
      ✓ should respond with a 200 status code (4829 ms)
      ✓ should specify text/html in the content type header (9441 ms)
      ✓ response has a transaction hash (9829 ms)
    When any of the parameters is missing
      ✓ should respond with a status code of 400 (30 ms)
    Given an invalid ETH address
      ✓ should respond with a status code of 400 (10 ms)

Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
```

Figura 7.3 - Resultado do teste unitário ao *endpoint /api/transfer*

Com base nos resultados da Figura 7.3, concluímos que os testes unitários para o ***Endpoint* 1** foram bem sucedidos.

7.1.2 *Endpoint* 2 - */api/transfer/batch*

Como mencionado no capítulo 6, este *endpoint* é responsável por interagir com o *smart contract* para efetuar transferências de NFTs em lote e, portanto, é fundamental que o mesmo receba todos os parâmetros necessários e que os mesmos sejam válidos. Deste modo foi desenvolvido o seguinte *script* (Figuras 7.4 e 7.5).

```
const metadata_batch = {
    from: "OPERATOR",
    to: "0xf42caDE7C89fFdAb592bb3F60e047480183222A5",
    token_ids: [371,336],
    amounts: [5,3]
}

describe("POST /api/transfer/batch", () => {
    describe("Given a from-address, to-address, token-ids and amounts", () => {
        jest.setTimeout(timeout);

        test("should respond with a 200 status code", async () => {
            const response = await request(app).post("/api/transfer/batch").send(metadata_batch)
            expect(response.statusCode).toBe( expected: 200)
        })
        test("should specify text/html in the content type header", async () => {
            const response = await request(app).post("/api/transfer/batch").send(metadata_batch)
            expect(response.headers['content-type']).toEqual(expect.stringContaining( str: "text"))
        })
        test("response has a transaction hash", async () => {
            const response = await request(app).post("/api/transfer/batch").send(metadata_batch)
            expect(response).toBeDefined()
        })
    })
```

Figura 7.4 - Testes com parâmetros válidos

Na Figura 7.4 é testado um cenário onde este *endpoint* recebe um pedido com os parâmetros corretos para a sua execução, nomeadamente os endereços públicos do remetente e do destinatário, os IDs dos NFTs e a quantidade que pretendemos transferir para cada um dos NFTs. Assim, sendo estes atributos válidos, a resposta devolvida deverá:

- retornar um *código HTTP 200 (OK)*;
- especificar *Content-type: text/html* nos *Headers* da resposta;
- retornar a *hash* da transação.

```
209  describe("When any of the parameters is missing", () => {
210      test("should respond with a status code of 400", async () => {
211          const bodyData = [
212              {from: "OPERATOR", to: "0xf42caDE7C89fFdAb592bb3F60e047480183222A5", token_ids: [371,336]},
213              {from: "OPERATOR", to: "0xf42caDE7C89fFdAb592bb3F60e047480183222A5", amounts: [5,3]},
214              {from: "OPERATOR", token_ids: [371,336], amounts: [5,3]},
215              {to: "0xf42caDE7C89fFdAb592bb3F60e047480183222A5", token_ids: [371,336], amounts: [5,3]},
216              {}
217          ]
218          for (const body of bodyData) {
219              const response = await request(app).post("/api/transfer/batch").send(body)
220              expect(response.statusCode).toBe( expected: 400)
221          }
222      })
223  })
224
225  describe("Given an invalid ETH address", () => {
226      jest.setTimeout(timeout);
227
228      test("should respond with a status code of 400", async () => {
229          const bodyData = [
230              {from: "OPERATOR", to: "0xtoBadAddress", token_ids: [371,336], amounts: [5,3]},
231              {from: "0xfromBadAddress", to: "OPERATOR", token_ids: [371,336], amounts: [5,3]}
232          ]
233
234          for (const body of bodyData) {
235              const response = await request(app).post("/api/transfer/batch").send(body)
236              expect(response.statusCode).toBe( expected: 400)
237          }
238      })
239  })
240 })
```

Figura 7.5 - Testes com parâmetros inválidos ou em falta

A Figura 7.5 representa um cenário onde este *endpoint* recebe um endereço público inválido (linha 225) ou quando pelo menos um dos parâmetros está em falta (linha 209). Para estes testes, a resposta deverá:

- retornar um código *HTTP 400 (Bad Request)*;

Depois de executar o *script* anterior, obtivemos os resultados apresentados na Figura 7.6.

```
  POST /api/transfer/batch
    Given a from-address, to-address, token-ids and amounts
      ✓ should respond with a 200 status code (5560 ms)
      ✓ should specify text/html in the content type header (9995 ms)
      ✓ response has a transaction hash (10193 ms)
    When any of the parameters is missing
      ✓ should respond with a status code of 400 (31 ms)
    Given an invalid ETH address
      ✓ should respond with a status code of 400 (12 ms)

Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
```

Figura 7.6 - Resultado do teste unitário ao *endpoint /api/transfer/batch*

Com base nos resultados da Figura 7.6, concluímos que os testes unitários para o ***Endpoint* 2** foram bem sucedidos.

7.1.3 *Endpoint* 3 - */api/mint*

Quanto ao *endpoint* 3, este é responsável por interagir com o *smart contract* para dar *mint* de um NFT e, portanto, é fundamental que o mesmo receba todos os parâmetros necessários e que os mesmos sejam válidos.

```
const metadata_mint = {
    image: "image",
    token_id: "0",
    name: "Unit Testing - test name",
    description: "Unit Testing - test description",
    amount: "1"
}

describe("POST /api/mint", () => {
    describe("Given an image, token_id, name, description and amount", () => {
        jest.setTimeout(timeout);

        test("should respond with a 200 status code", async () => {
            const response = await request(app).post("/api/mint").send(metadata_mint)
            expect(response.statusCode).toBe( expected: 200)
        })
        test("should specify text/html in the content type header", async () => {
            const response = await request(app).post("/api/mint").send(metadata_mint)
            expect(response.headers['content-type']).toEqual(expect.stringContaining( str: "text"))
        })
        test("response confirms that the mint was successful", async () => {
            const response = await request(app).post("/api/mint").send(metadata_mint)
            expect(response.text).toEqual(expect.stringContaining( str: "Minted NFT with ID"))
        })
    })
```

Figura 7.7 - Testes com parâmetros válidos

Na Figura 7.7 é testado um cenário onde este *endpoint* recebe um pedido com os parâmetros corretos para a sua execução, nomeadamente: a imagem que queremos incluir no NFT e que será carregada para o IPFS; o ID que queremos atribuir ao NFT; o nome e a descrição; e, por fim, a quantidade de NFTs que queremos criar. Assim, sendo estes atributos válidos, a resposta devolvida deverá:

- retornar um código *HTTP 200 (OK)*;
- especificar *Content-type: text/html* nos *Headers* da resposta;
- retornar uma mensagem de confirmação de que a transação foi bem sucedida.

```
describe("When any of the parameters is missing", () => {
    test("should respond with a status code of 400", async () => {
        const bodyData = [
            {image: "image", token_id: "1", name: "name", description: "description"},
            {image: "image", token_id: "1", name: "name", amount: "1"},
            {image: "image", token_id: "1", description: "description", amount: "1"},
            {image: "image", name: "name", description: "description", amount: "1"},
            {token_id: "1", name: "name", description: "description", amount: "1"},
            {}
        ]
        for (const body of bodyData) {
            const response = await request(app).post("/api/mint").send(body)
            expect(response.statusCode).toBe( expected: 400)
        }
    })
})
```

Figura 7.8 - Teste com parâmetros em falta

A Figura 7.8 representa um cenário onde este *endpoint* não recebe pelo menos um dos parâmetros necessários. Para estes testes, a resposta deverá:

- retornar um código *HTTP 400 (Bad Request)*;

Depois de executar o *script* anterior, obtivemos os resultados apresentados na Figura 7.9.

```
 POST /api/mint
   Given an image, token_id, name, description and amount
     ✓ should respond with a 200 status code (6925 ms)
     ✓ should specify text/html in the content type header (9980 ms)
     ✓ response confirms that the mint was successful (9898 ms)
   When any of the parameters is missing
     ✓ should respond with a status code of 400 (35 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
```

Figura 7.9 - Resultado do teste unitário ao *endpoint /api/mint*

Com base nos resultados da Figura 7.9, concluímos que os testes unitários para o ***Endpoint* 3** foram bem sucedidos.

7.1.4 *Endpoint* 4 - */api/getCollection/:address*

Este *endpoint* é responsável por retornar uma lista de NFTs que estejam num determinado endereço público, o qual é dado como parâmetro quando é feito o pedido HTTP.

Para testar este *endpoint*, foi desenvolvido o *script* da Figura 7.10.

```
describe("GET /api/getCollection/:address", () => {
    describe("Given an address", () => {
        test("should respond with a 200 status code and should specify json in the content type", (done) => {
            request(app)
                .get("/api/getCollection/0xCA42F06E9f3Db574a9925e1da9FA041A3Bf34DF4")
                .expect("Content-Type", /json/)
                .expect(200)
                .end((err, res) => {
                    if (err) return done(err);
                    return done();
                });
        });
    })

    describe("Given an invalid ETH address", () => {
        test("should respond with a status code of 400", (done) => {
            request(app)
                .get("/api/getCollection/0xbadAddress")
                .expect(400)
                .end((err, res) => {
                    if (err) return done(err);
                    return done();
                });
        })
    })
})
```

Figura 7.10 - *Script* de testes usado no *endpoint* 4

No primeiro teste (linha 100) da Figura 7.10, é testado um cenário onde é recebido um endereço público válido como parâmetro. Neste caso, a resposta ao pedido deverá:

- retornar um código *HTTP 200 (OK)*;
- especificar *Content-type: application/json* nos *Headers* da resposta.

No segundo teste (linha 114) da Figura 7.10, é testado um cenário onde o endereço recebido não é um endereço *Ethereum* válido.

Depois de executar o *script* anterior, obtivemos os resultados apresentados na Figura 7.11.

```
  GET /api/getCollection/:address
    Given an address
      ✓ should respond with a 200 status code and should specify json in the content type (612 ms)
    Given an invalid ETH address
      ✓ should respond with a status code of 400 (7 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
```

Figura 7.11 - Resultado do teste unitário ao *endpoint /api/getCollection/:address*

Com base nos resultados da Figura 7.11, concluímos que os testes unitários para o ***Endpoint* 4** foram bem sucedidos.

7.1.5 *Endpoint* 5 - */api/getAccount/*

Este *endpoint* é responsável por retornar uma nova conta *Ethereum*, ou seja, um endereço público e a respetiva chave privada. Para testar este *endpoint*, foi desenvolvido o *script* da Figura 7.12.

```
describe("GET /api/getAccount", () => {
    test("should respond with a 200 status code and should specify json in the content type", (done) => {
        request(app)
            .get("/api/getAccount")
            .expect("Content-Type", /json/)
            .expect(200)
            .end((err, res) => {
                if (err) return done(err);
                return done();
            });
    });
})
```

Figura 7.12 - *Script* de teste usado no *endpoint* 5

Aqui, é testado um cenário onde é feito um pedido regular a este *endpoint*. A resposta ao pedido deverá:

- retornar um código *HTTP 200 (OK)*;
- especificar *Content-type: application/json* nos *Headers* da resposta.

Depois de executar o script anterior, obtivemos os resultados apresentados na Figura 7.13.

```
GET /api/getAccount
  ✓ should respond with a 200 status code and should specify json in the content type (108 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
```

Figura 7.13 - Resultado do teste unitário ao *endpoint /api/getAccount*

Com base nos resultados da Figura 7.13, concluímos que o teste unitário feito ao ***Endpoint* 5** foi bem sucedido.

7.1.6 *Endpoint* 6 - */api/getOperatorCollection/*

Este *endpoint* é responsável por devolver uma lista com a metadata dos NFTs presentes na carteira do Operador. Para testar este *endpoint*, foi desenvolvido o *script* da Figura 7.14.

```
describe("GET /api/getOperatorCollection", () => {
    test("should respond with a 200 status code and should specify json in the content type", (done) => {
        request(app)
            .get("/api/getOperatorCollection")
            .expect("Content-Type", /json/)
            .expect(200)
            .end((err, res) => {
                if (err) return done(err);
                return done();
            });
    });
})
```

Figura 7.14 - *Script* de testes usado no *endpoint* 6

Aqui, é testado um cenário onde é feito um pedido regular a este *endpoint*. A resposta ao pedido deverá:

- retornar um código *HTTP 200 (OK)*;
- especificar *Content-type: application/json* nos *Headers* da resposta.

Depois de executar o *script* anterior, obtivemos os resultados apresentados na Figura 7.15.

```
  GET /api/getOperatorCollection
    ✓ should respond with a 200 status code and should specify json in the content type (516 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
```

Figura 7.15 - Resultado do teste unitário ao *endpoint /api/getOperatorCollection*

Com base nos resultados da Figura 7.15, concluímos que o teste unitário feito ao ***Endpoint* 6** foi bem sucedido.

7.1.7 *Endpoint* 7 - */api/getContractAddress/*

Este *endpoint* é responsável por devolver o endereço do *Smart Contract*. Para testar este *endpoint*, foi desenvolvido o *script* da Figura 7.16.

```
describe("GET /api/getContractAddress", () => {
    test("should respond with a 200 status code and should specify text/html in the content type", (done) => {
        request(app)
            .get("/api/getContractAddress")
            .expect("Content-Type", /text/)
            .expect(200)
            .end((err, res) => {
                if (err) return done(err);
                return done();
            });
    });
})
```

Figura 7.16 - *Script* de testes usado no *endpoint* 7

Aqui, é testado um cenário onde é feito um pedido regular a este *endpoint*. A resposta ao pedido deverá:

- retornar um código *HTTP 200 (OK)*;
- especificar *Content-type: text/html* nos *Headers* da resposta.

Depois de executar o *script* anterior, obtivemos os resultados apresentados na Figura 7.17.

```
  GET /api/getContractAddress
    ✓ should respond with a 200 status code and should specify text/html in the content type (25 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
```

Figura 7.17 - Resultado do teste unitário ao *endpoint /api/getContractAddress*

Com base nos resultados da Figura 7.17, concluímos que o teste unitário feito ao ***Endpoint* 7** foi bem sucedido.

7.1.8 *Endpoint* 8 - */api/register/*

Por fim, e como mencionado no capítulo 6, este *endpoint* é responsável por lidar com a questão da delegação de transações, e, como tal, recebe como parâmetros o endereço público e a chave privada do utilizador em causa. Para testar este *endpoint*, foi desenvolvido o seguinte *script* (Fig. 7.18).

```
const register = {
    userAddress: "0x8f6756a685f1F0Dd985FF6Fe40A8Ac43Ca724609",
    userKey: "0x79bbbe5b13151d60fab91070c2e19d5416ed005517cf814e02aa55dec20ac61c"
}

describe("POST /api/register", () => {
    describe("Given an address and a private key", () => {
        jest.setTimeout(timeout);

        test("should respond with a 200 status code", async () => {
            const response = await request(app).post("/api/register").send(register)
            expect(response.statusCode).toBe( expected: 200)
            expect(response.text).toEqual(expect.stringContaining( str: "Sent Matic and Set Approval for user with address"))
        })
        test("should specify text/html in the content type header", async () => {
            const response = await request(app).post("/api/register").send(register)
            expect(response.headers['content-type']).toEqual(expect.stringContaining( str: "text"))
        })
    })
```

Figura 7.18 – Testes com parâmetros válidos

Na Figura 7.18 é testado um cenário onde este *endpoint* recebe um pedido com os parâmetros válidos, nomeadamente um endereço público e a respetiva chave privada. Assim, a resposta devolvida deverá:

- retornar um código *HTTP 200 (OK)*;
- especificar *Content-type: text/html* nos *Headers* da resposta;

```
    describe("When any of the parameters is missing", () => {
        test("should respond with a status code of 400", async () => {
            const bodyData = [
                {userAddress: "0x8f6756a685f1F0Dd985FF6Fe40A8Ac43Ca724609"},
                {userKey: "0x79bbbe5b13151d60fab91070c2e19d5416ed005517cf814e02aa55dec20ac61c"},
                {}
            ]
            for (const body of bodyData) {
                const response = await request(app).post("/api/register").send(body)
                expect(response.statusCode).toBe( expected: 400)
            }
        })
    })

    describe("Given an invalid ETH address", () => {
        test("should respond with a status code of 400", async () => {
            const response = await request(app).post("/api/register").send({
                userAddress: "0xBadAddress",
                userKey: "0x79bbbe5b13151d60fab91070c2e19d5416ed005517cf814e02aa55dec20ac61c"
            })
            expect(response.statusCode).toBe( expected: 400)
        })
    })
```

Figura 7.19 - Testes com parâmetros inválidos ou em falta

A Figura 7.19 representa um cenário onde este *endpoint* recebe um endereço *Ethereum* inválido (linha 27) ou quando pelo menos um dos parâmetros está em falta (linha 41).

Para estes testes, a resposta deverá:

- retornar um código *HTTP 400 (Bad Request)*;

Depois de executar o *script* anterior, obtivemos os resultados apresentados na Figura 7.20.

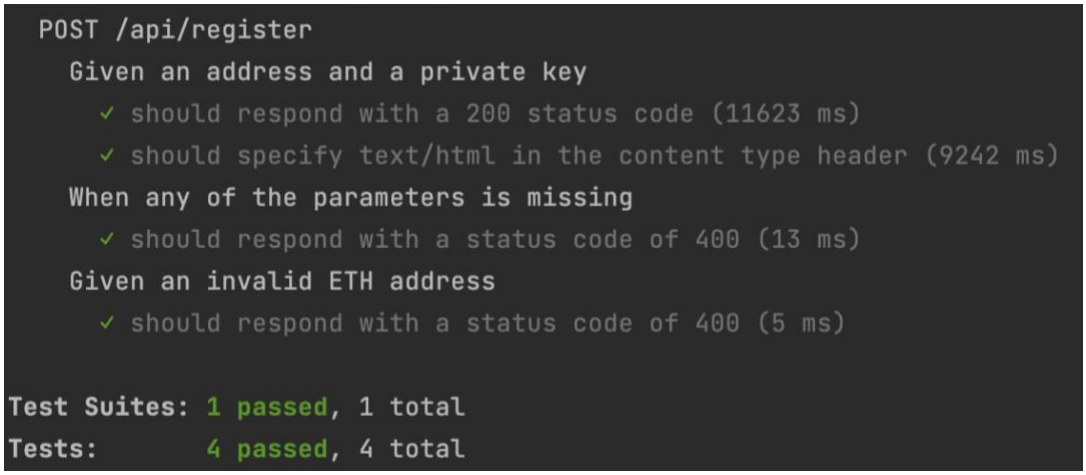


Figura 7.20 - Resultado do teste unitário ao *endpoint /api/register*

Com base nos resultados da Figura 7.20, concluímos que os testes unitários para o ***Endpoint* 8** foram bem sucedidos.

7.1.9 Sumário

Na Tabela 7.1 é apresentado um resumo de todos os testes unitários realizados, indicando o número de testes realizados e passados.

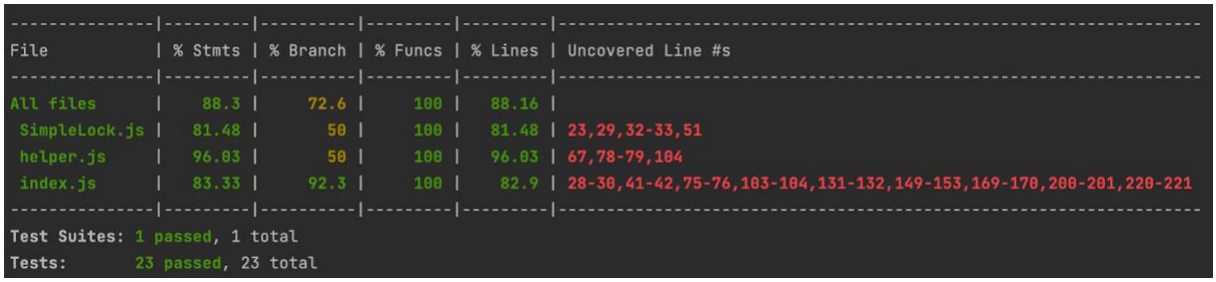
Tabela 7.1 - Testes unitários

Endpoint	**Número de testes efetuados**	**Número de testes passados**
1 - /api/transfer	5	5
2 - /api/transfer/batch	5	5
3 - /api/mint	4	4
4 - /api/getCollection/:address	2	2
5 - /api/getAccount	1	1
6 - /api/getOperatorCollection	1	1
7 - /api/getContractAddress	1	1
8 - /api/register	4	4

7.2 Code coverage

Depois de realizar os 23 testes da secção anterior, todos bem sucedidos, foi calculada a percentagem de *Code coverage*. O *Code Coverage* é uma métrica que ajuda a perceber a quantidade de código fonte que é testado.

Para este cálculo, foi usada a mesma *framework* que nos testes unitários, a Jest. Esta ferramenta não só mede a quantidade de linhas de código que são executadas durante a concretização dos testes mencionados na secção anterior, como também regista quais as linhas de código que não foram visitadas.



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	88.3	72.6	100	88.16	
SimpleLock.js	81.48	50	100	81.48	23,29,32-33,51
helper.js	96.03	50	100	96.03	67,78-79,104
index.js	83.33	92.3	100	82.9	28-30,41-42,75-76,103-104,131-132,149-153,169-170,200-201,220-221

Test Suites: 1 passed, 1 total
Tests: 23 passed, 23 total

Figura 7.21 - Resultados do teste de *Code Coverage*

Como podemos observar pelos resultados da Figura 7.21, os testes realizados cobrem 88.16% do código desenvolvido. Apesar do que se pretende atingir serem valores elevados, como é o caso, uma percentagem alta não garante que o código não tenha falhas.

Assim, a percentagem de *Code Coverage* não deve ser considerada como um indicador absoluto da qualidade da implementação porque apenas significa que, neste caso, 88.16% do código fonte foi testado, mas não assegura a qualidade dos testes que foram executados.

Não obstante, é também através desta métrica que conseguimos perceber quais as partes da nossa *codebase* que não são abrangidas pelos testes. Com esta informação podemos melhorar a qualidade dos testes unitários e aumentar a quantidade de cenários que os mesmos alcançam, havendo menos espaço para erros.

7.3 Validação dos Requisitos funcionais

Por forma a validar o cumprimento dos requisitos funcionais presentes no capítulo 4, no decorrer do estágio foram feitas pequenas demonstrações dos mesmos.

Na Tabela 7.2 é apresentada a validação dos requisitos funcionais.

Tabela 7.2 - Validação dos requisitos funcionais

ID	Requisito Funcional	Comportamento esperado
Backoffice		
RF1	Lançar coleção de NFTs	**Sim**
RF2	Ver coleção de NFTs da operadora	**Sim**
RF3	Ver coleção de NFTs de um utilizador	**Sim**
Web Client		
RF4	Receber NFT ao abrir um Shake	**Sim**
RF5	Ver detalhes sobre os NFTs colecionados	**Sim**
RF6	Trocar NFTs por Shakes	**Sim**
RF7	Trocar NFTs por pontos	**Sim**
RF8	Trocar NFTs com amigos	**Sim**
Carteiras digitais		
RF9	Criar uma *custodial wallet* para cada utilizador	**Sim**
RF10	Delegar uma transação	**Sim**
RF11	Associar uma *non-custodial wallet*	**-**
Prémios		
RF12	Receber NFT como prémio	**-**

Todos os requisitos com prioridade elevada (***Must Have***) foram implementados.

Outros requisitos com prioridades mais baixas não foram atendidos, nomeadamente os requisitos RF11 e RF12, descritos no capítulo 4. O primeiro, e dada a sua baixa prioridade (***Could Have***), não foi implementado por questões de tempo. Quanto ao último, ficou definido no primeiro semestre que o mesmo não iria ser implementado nesta versão do projeto (prioridade ***Won't Have***), mas que poderia ser implementado futuramente.

7.4 Validação dos Requisitos não funcionais

Nesta secção é demonstrada a forma como foram validados os requisitos não funcionais incluídos no projeto, tendo em conta o modo como foram especificados no capítulo 4.

7.4.1 RNF1 - Modularidade

Como descrito no capítulo 5, e dado que o Shake It segue uma arquitetura de microsserviços, manteve-se para este estágio a mesma abordagem. Assim, parte dos requisitos implementados

pelo estagiário foram desenvolvidos no seu próprio módulo, e a outra parte nos módulos já existentes, respeitando desta forma o atributo de qualidade em questão.

7.4.2 RNF2 - Desempenho

Como detalhado no capítulo 4, foram testados dois cenários para o desempenho: um onde o foco são os *endpoints* responsáveis por comunicarem com a *Blockchain* (**Cenário 1**), e outro onde o foco são os restantes *endpoints* (**Cenário 2**).

Esta distinção foi feita para que se pudesse fazer uma análise mais precisa dos tempos de resposta obtidos para cada pedido HTTP. Dado que no Cenário 1 o tempo de resposta para cada pedido está dependente da *Blockchain*, os valores esperados são significativamente maiores. Isto deve-se ao facto de que o tempo de comunicação com a *Blockchain* é alto e é algo que não é possível mitigar através da nossa aplicação. No entanto, considerando que estas tarefas (como a transferência de um NFT, por exemplo) são feitas de forma assíncrona, o utilizador não fica ativamente à espera de resposta, e portanto a relevância do tempo de resposta para este cenário é relativa.

Para efetuar os testes de desempenho para cada um dos cenários foi utilizada a ferramenta Apache JMeter [109], variando o número de pedidos feitos em cada conexão de maneira a medir os tempos de resposta do sistema. É de notar que foram simuladas 10 conexões, sendo que este valor permaneceu fixo para cada um dos casos em cada um dos cenários.

Os tempos obtidos encontram-se registados nas Tabelas 7.3 e 7.4.

Tabela 7.3 - Cenário 1: Desempenho

Cenário 1 – *Blockchain endpoints*					
Número de pedidos por conexão	**Tempo (ms)**			**Número total de pedidos**	***Throughput* (pedidos/minuto)**
	Mín.	**Médio**	**Máx.**		
1	4104	20700	45037	10	7.6
2	4574	33309	50026	20	9.8
5	5431	45371	51069	50	11.9
10	8078	80640	112382	100	8.2

Analisando os resultados obtidos para o Cenário 1, é possível observar que os valores não variam consideravelmente quando se aumenta o número de pedidos por cada uma das 10 conexões em simultâneo. O pior resultado foi de aproximadamente 112 segundos, para o caso em que 10 utilizadores, simultaneamente, realizam 10 pedidos ao servidor, o que, dentro do contexto do Shake It, é um cenário bastante improvável.

Ainda assim, e assumindo a especificação definida no capítulo 4, todos os tempos de resposta são inferiores a 2 minutos e podemos por isso assumir este requisito não funcional como válido, para o Cenário 1.

Tabela 7.4 - Cenário 2: Desempenho

Cenário 2 – *Endpoints*					
Número de pedidos por conexão	**Tempo (ms)**			**Número total de pedidos**	***Throughput* (pedidos/segundo)**
	Mín.	**Médio**	**Máx.**		
10	1	3	39	100	21.6
20	1	3	51	200	43.6
50	1	2	36	500	106.8
100	1	2	30	1 000	205
200	1	3	144	2 000	417.5
500	1	12	192	5 000	519
1000	1	10	78	10 000	707.2

Quanto aos resultados obtidos para o Cenário 2 (Tabela 7.4), os mesmos foram igualmente positivos. Visto que o pior caso foi de aproximadamente 0.2 segundos e sabendo que, geralmente, o período de atenção normal de um utilizador ronda os 2 e os 4 segundos, podemos concluir que o sistema, sob circunstâncias normais, cumpre o presente requisito não funcional para o Cenário 2.

7.4.3 RNF3 - Escalabilidade

Pelas mesmas razões levantadas no ponto anterior (RNF2 - Desempenho), para testar a escalabilidade foram simulados 2 cenários diferentes, usando novamente a ferramenta Apache JMeter.

Aqui, foi testado o comportamento do sistema em situações de grande *stress*, aumentando o número de conexões em simultâneo e o número de pedidos feitos por conexão. Para podermos validar este requisito, a análise dos resultados será feita através da comparação entre os valores obtidos com os tempos conseguidos nos testes de desempenho. É de referir que, para obter uma comparação o mais rigorosa possível, o número de pedidos por conexão manteve-se o mesmo entre os cenários 1 e 3 e entre os cenários 2 e 4.

Os tempos obtidos encontram-se registados nas tabelas 7.5 e 7.6.

Tabela 7.5 - Cenário 3: Escalabilidade

Cenário 3 – *Blockchain endpoints*						
Número de pedidos por conexão	**Número de conexões em simultâneo**	**Tempo (ms)**			**Número total de pedidos**	***Throughput* (pedidos/minuto)**
		Mín.	**Médio**	**Máx.**		
1	**10**	4104	20700	45037	10	7.6
2	**15**	7238	115475	156889	30	5.9
5	**20**	10694	123330	200251	100	8.7
10	**25**	10133	272376	582682	250	5.3

Como podemos observar pelos tempos de resposta obtidos na Tabela 7.5, os resultados não são positivos, e não cumprem com a métrica estipulada no capítulo 4, que diz que o tempo de resposta não se deverá desviar mais de 50% acima dos valores obtidos nos testes de desempenho. Deste modo, não podemos validar a Escalabilidade para o cenário 3.

Uma possível explicação para estes resultados é o facto de as especificações da máquina onde foram realizados os testes não serem suficientemente poderosas computacionalmente. Outra explicação poderá ser o facto de a infraestrutura onde está alojado o *node* RPC que está a ser usado para a comunicação com a *Blockchain* não fornecer um desempenho que permita obter os resultados que se pretendiam.

Todavia, como mencionado anteriormente, os tempos de resposta obtidos, apesar de acentuados, não impactam os utilizadores da aplicação, dado que são tarefas assíncronas. Além disso, tendo em conta a prioridade deste atributo de qualidade (*Could Have*), o facto de não ter sido validado, não apresenta um problema crítico para o sucesso do projeto, para esta fase.

Tabela 7.6 - Cenário 4: Escalabilidade

Cenário 4 - *Endpoints*						
Número de conexões	**Número de pedidos por conexão**	**Tempo (ms)**			**Número total de pedidos**	**Throughput (pedidos/segundo)**
		Mín.	**Médio**	**Máx.**		
10	**10**	1	3	16	100	22.1
20	**20**	1	3	25	400	83
40	**50**	1	3	141	2 000	404.4
50	**100**	1	32	237	5 000	676.6

75	**200**	1	81	804	15 000	717.4
100	**500**	1	150	867	50 000	623.1
200	**1000**	1	292	1533	200 000	669

Por fim, iremos agora analisar os resultados obtidos para o cenário 4 (Tabela 7.6). Aqui, o pior tempo de resposta foi de 1.533 segundos, para o caso em que 200 utilizadores em simultâneo realizam 1000 pedidos ao servidor. Deste modo, podemos concluir que o presente requisito não funcional é cumprido para o cenário 4.

7.5 Considerações finais

Para terminar o presente capítulo, é importante mencionar que a fase de testes é crucial para qualquer projeto. Esta etapa garante que o produto desenvolvido é testado formalmente para garantir o atendimento dos requisitos e a qualidade do produto final.

Para esta tese, os testes efetuados foram importantes para assegurar o atendimento de todos os requisitos, com a qualidade pretendida, e cada um prestou um papel diferente:

- os testes unitários foram essenciais para assegurar o funcionamento correto dos *endpoints* criados, em vários cenários diferentes;
- quanto ao *code coverage*, este foi útil porque nos ajudou a perceber a quantidade de código que é realmente abrangido pelos testes;
- os testes funcionais garantiram que todos os requisitos funcionais obrigatórios foram atendidos e que os mesmos se comportam como esperado;
- por fim, os testes não funcionais permitiram-nos representar as características que o sistema desenvolvido suporta.

Capítulo 8
Conclusão

O presente capítulo apresenta algumas reflexões e considerações finais tiradas pelo estagiário no decorrer da dissertação, sobre o processo e o produto desenvolvido, mencionando ainda os desafios que foram encontrados e o conhecimento que foi adquirido.

Por fim, é mencionado algum do trabalho futuro que poderá ser implementado para melhorar o produto desenvolvido ao longo do estágio, indicando as funcionalidades que poderiam ser adicionadas.

8.1 Reflexões finais

O estagiário, com o terminar do estágio curricular, trabalhou e aprendeu com vários profissionais da área de Engenharia de *Software*, podendo assim experienciar os conceitos fundamentais e as metodologias que sustentam uma empresa desta área.

Quanto ao trabalho realizado, evidenciado ao longo deste documento, o estagiário começou por planear o ano curricular. Esta foi uma tarefa de grande importância não só para organizar a ordem e priorização das tarefas, como também para assegurar o cumprimento das mesmas dentro do período de tempo definido para o estágio.

Depois, deu-se início à pesquisa por algum enquadramento do tema e por mais conhecimento na área, através do levantamento do Estado da Arte. Neste, foi possível perceber o contexto em que estão inseridas as motivações deste projeto e possibilitou a criação de uma base de conhecimentos que permitisse o desenvolvimento do mesmo. Deste modo, considerando que antes do estágio o aluno não estava familiarizado com as tecnologias contíguas à *Blockchain*, a investigação realizada tornou-se num período de aprendizagem bastante valioso.

Para terminar o primeiro semestre, o estagiário procedeu à especificação dos requisitos e ao desenho da arquitetura de *software*. Estas fases permitiram organizar e estruturar todo o desenvolvimento efetuado, por forma a cumprir com os requisitos levantados e com a arquitetura delineada.

De seguida, depois da fase de desenvolvimento do projeto e perto do final do estágio, seguiu-se a fase de testes e validações. Esta etapa permitiu que a solução implementada fosse validada consoante os requisitos funcionais e não funcionais previamente definidos, tendo também em consideração os objetivos estabelecidos inicialmente.

Por fim, e considerando os critérios de sucesso definidos para este estágio (Capítulo 4), o mesmo pode ser considerado um sucesso. Todos os requisitos funcionais e não funcionais classificados com prioridade obrigatória foram implementados, e as respetivas validações foram atestadas. A entidade acolhedora, a WIT Software, e os orientadores que acompanharam o aluno durante este projeto, forneceram todas as condições para que o sucesso do projeto fosse concretizado, providenciando o apoio e os recursos necessários ao mesmo, garantindo ainda que a solução desenvolvida ia de encontro ao pretendido.

Finalmente, e como referido anteriormente, o desconhecimento ao tema foi para o estagiário um grande desafio, no entanto, o contacto com as tecnologias utilizadas contribuiu para o seu

enriquecimento profissional. Outro desafio encontrado foi a larga *codebase* sobre a qual o aluno realizou parte da implementação, uma vez que o mesmo, no seu percurso académico, nunca tinha tido contacto com um projeto desta dimensão. Após um pequeno período de adaptação, estes desafios foram ultrapassados e o aluno desenvolveu um grande conjunto de aptidões e capacidades ao trabalhar num ambiente profissional, auferindo mais conhecimento sobre as práticas, ferramentas e tecnologias utilizadas pela empresa e no mercado de trabalho.

8.2 Trabalho futuro

Este documento descreve uma versão do projeto que, como referido na secção anterior, inclui todos os requisitos que permitiram cumprir os objetivos do estágio. Contudo, existe sempre margem de progressão, pelo que a presente secção tem como objetivo apresentar um conjunto de funcionalidades que poderiam ser integradas neste projeto, para que o mesmo possa evoluir.

Primeiramente, os dois requisitos funcionais definidos com prioridade "*Could Have"* e "*Won't Have"* no Capítulo 4 que não foram implementados, podiam ser atendidos. Além disso, traria mais valor ao sistema se a segurança fosse incluída nos atributos de qualidade. Isto poderia ser aplicado à forma como são armazenadas as chaves privadas dos utilizadores na solução atual e também ao *Blockchain Gateway*, adicionando métodos de autenticação e autorização.

Outra possibilidade, e talvez a mais pertinente dado o contexto do projeto, seria implementar o mecanismo do sorteio das cartas no *smart contract*. Desta forma, a aleatoriedade na atribuição de cartas podia ser comprovada por qualquer pessoa, e consequentemente, promovendo mais transparência aos utilizadores do jogo, dado que esse mecanismo estaria registado na *Blockchain*. No entanto, esta seria uma tarefa bastante complexa dada a natureza determinística da *Blockchain*, que não permite a geração de números completamente aleatórios, em segurança. Deste modo, teriam que ser usados recursos externos à *Blockchain*, como por exemplo, um *oracle* [111].

Referências

[1] WIT Software. https://www.wit-software.com/. (Acedido a 29-12-2021).

[2] Starbucks Rewards. https://www.starbucks.com/rewards. (Acedido a 28-10-2021).

[3] Formation. How Starbucks Became #1 in Customer Loyalty. https://formation.ai/blog/how-starbucks-became-1-in-customer-loyalty/. (Acedido a 28-10-2021).

[4] Nike Run Club. https://www.nike.com/pt/nrc-app. (Acedido a 28-10-2021).

[5] Duolingo. https://pt.duolingo.com/. (Acedido a 28-10-2021).

[6] Epidemiology of HIV/AIDS. https://en.wikipedia.org/wiki/Epidemiology_of_HIV/AIDS. (Acedido a 28-10-2021).

[7] HIV.gov. The Global HIV/AIDS Epidemic. https://www.hiv.gov/hiv-basics/overview/data-and-trends/global-statistics. (Acedido a 28-10-2021).

[8] Yu-kai Chou. The 10 best social products that use Gamification to literally save the world. https://yukaichou.com/gamification-examples/top-10-gamification-examples-human-race/#.WvHlm1SpnyU. (Acedido a 28-10-2021).

[9] FoldIt. https://fold.it/. (Acedido a 28-10-2021).

[10] Shake It. https://www.yorn.net/yorn/shake-it.html. (Acedido a 28-10-2021).

[11] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. http://bitcoin.org/bitcoin.pdf. (Acedido a: 05-12-2021).

[12] Thomas McGovern. How many Blockchains are there in 2022? https://earthweb.com/how-many-blockchains-are-there/. (Acedido a: 03-07-2022).

[13] Ben McCarthy. Salesforce Blockchain for Dummies. https://www.salesforceben.com/salesforce-blockchain-for-dummies/. (Acedido a: 22-11-2021).

[14] P2P networks What are they, what are their advantages and disadvantages and what are the types?. https://www.informatique-mania.com/en/linternet/p2p/. (Acedido a: 22-11-2021).

[15] Matthew Allen. Como o blockchain pode afetar em breve a vida cotidiana. https://www.swissinfo.ch/por/transa%C3%A7%C3%B5es-digitais_como-o-blockchain-pode-afetar-em-breve-a-vida-cotidiana/43643288. (Acedido a: 22-11-2021).

[16] Kathleen E. Wegrzyn, Eugenia Wang. Permissionless vs. Permissioned Blockchains. https://www.foley.com/en/insights/publications/2021/08/types-of-blockchain-public-private-between. (Acedido a: 05-12-2021).

[17] Christine Parizo. What are the 4 different types of blockchain technology? https://searchcio.techtarget.com/feature/What-are-the-4-different-types-of-blockchain-technology. (Acedido a: 05-12-2021).

[18] Ethereum. https://ethereum.org/en/. (Acedido a: 05-12-2021).

[19] Solana. https://solana.com/. (Acedido a: 05-12-2021).

[20] Cardano. https://cardano.org/. (Acedido a: 05-12-2021).

[21] Chainlink. https://chain.link/. (Acedido a: 05-12-2021).

[22] Dai. https://makerdao.com/en/. (Acedido a: 05-12-2021).

[23] USDC. https://www.centre.io/usdc. (Acedido a: 05-12-2021).

[24] OpenSea. https://opensea.io/. (Acedido a: 05-12-2021).

[25] Decentraland. https://decentraland.org/. (Acedido a: 05-12-2021).

[26] Alyssa Hertig. Which Crypto Projects Are Based on Ethereum? https://finance.yahoo.com/news/crypto-projects-based-ethereum. (Acedido a: 05-12-2021).

[27] What are smart contracts on blockchain? https://www.ibm.com/topics/smart-contracts. (Acedido a: 05-12-2021).

[28] Beeple's "Everydays: The First 5000 Days". https://adamfard.com/blog/nfts-21-examples. (Acedido a: 05-12-2021).

[29] CryptoPunk 7523. https://www.larvalabs.com/cryptopunks/details/7523. (Acedido a: 05-12-2021).

[30] ConsenSys. Ethereum Has 4x More Developers Than Any Other Crypto Ecosystem. https://consensys.net/blog/developers/ethereum-has-4x-more-developers-than-any-other-crypto-ecosystem/. (Acedido a: 05-12-2021).

[31] Anatoly Yakovenko. Tower BFT: Solana's High Performance Implementation of PBFT. https://medium.com/solana-labs/tower-bft-solanas-high-performance-implementation-of-pbft-464725911e79. (Acedido a: 05-12-2021).

[32] Anatoly Yakovenko. Proof of History: A Clock for Blockchain. https://medium.com/solana-labs/proof-of-history-a-clock-for-blockchain-cf47a61a9274. (Acedido a: 05-12-2021).

[33] ChangeNOW.io . How does Solana work? https://medium.com/coinmonks/whats-the-deal-with-solana-a-guide-into-the-skyrocketing-cryptocurrency-5f5dbaf68d94. (Acedido a: 05-12-2021).

[34] Solana. https://solana.com/. (Acedido a: 05-12-2021).

[35] Etherchain. https://etherchain.org/. (Acedido a: 05-12-2021).

[36] https://bitinfocharts.com/comparison/ethereum-transactionfees.html#3y. (Acedido a: 05-12-2021).

[37] edChain. POW vs. PoS: which is better? https://medium.com/@EdChain/pow-vs-pos-a-comparison-of-two-blockchain-consensus-algorithms-f3effdae55f5. (Acedido a: 05-12-2021).

[38] Werner Vermaak. A Deep Dive Into Cardano [Updated]. https://coinmarketcap.com/alexandria/article/a-deep-dive-into-cardano. (Acedido a 05-12-2021).

[39] Anupam Varshney. Cardano Roadmap and Outlook. https://coinmarketcap.com/alexandria/article/cardano. (Acedido a 05-12-2021).

[40] Rahul Nambiampurath. Cardano Smart Contracts to Hit Public Testnet on Sept 12. https://finance.yahoo.com/news/cardano-smart-contracts-hit-public-103919542.html. (Acedido a: 05-12-2021).

[41] Haskell. https://pontem.network/posts/comparison-of-the-top-10-smart-contract-programming-languages-in-2021. (Acedido a: 09-12-2021).

[42] Cosmos. https://cosmos.network/. (Acedido a: 09-12-2021).

[43] Inter-Blockchain Communication Protocol. https://ibcprotocol.org/. (Acedido a: 09-12-2021).

[44] Michelle Lim. What is Cosmos — the 'internet of blockchains'? https://forkast.news/what-is-cosmos-the-internet-of-blockchains/. (Acedido a: 09-12-2021).

[45] Cosmos SDK. https://v1.cosmos.network/sdk. (Acedido a: 09-12-2021).

[46] Osmosis. https://osmosis.zone/. (Acedido a: 09-12-2021).

[47] Cronos. https://cronos.crypto.org/. (Acedido a: 09-12-2021).

[48] Terra. https://www.terra.money/. (Acedido a: 09-12-2021).

[49] Polygon. https://polygon.technology/. (Acedido a: 05-12-2021).

[50] Andrew Saunders. Sidechain Blockchain: Advantages & Disadvantages. https://medium.com/@andrew_23660/sidechain-blockchain-advantages-disadvantages-2e7f773a64fe. (Acedido a: 05-12-2021).

[51] Yash Kamal Chaturvedi. All about Polygon (MATIC) in One Shot. https://medium.datadriveninvestor.com/all-you-need-to-know-about-polygon-matic-a6c61dbc2d3c. (Acedido a: 05-12-2021).

[52] Tezos. https://tezos.com/. (Acedido a: 09-12-2021).

[53] Stellar. https://www.stellar.org/. (Acedido a: 09-12-2021).

[54] Polkadot. https://polkadot.network/. (Acedido a: 09-12-2021).

[55] Werner Vermaak. A Deep Dive Into Tezos. https://coinmarketcap.com/alexandria/article/a-deep-dive-into-tezos. (Acedido a: 09-12-2021).

[56] Stellar (XLM) Cryptocurrency FAQ. https://bitni.com/page/stellar-xlm-faq. (Acedido a: 09-12-2021).

[57] Michele Mostarda. Gamification in the days of blockchain: 10 applications, 20+ techniques and strategies. https://hardest.medium.com/gamification-in-the-days-of-blockchain-10-applications-20-techniques-and-strategies-1ecea64526cc. (Acedido a 05-11-2021).

[58] Valeonti F, Bikakis A, Terras M, Speed C, Hudson-Smith A, Chalkias K. Crypto Collectibles, Museum Funding and OpenGLAM: Challenges, Opportunities and the Potential of Non-Fungible Tokens (NFTs). Applied Sciences. 2021; 11(21):9931. https://doi.org/10.3390/app11219931. (Acedido a 14-01-2022).

[59] InterPlanetary File System. https://ipfs.io/. (Acedido a 14-01-2022).

[60] Socios.com. https://www.socios.com/. (Acedido a: 09-12-2021).

[61] CryptoKitties. https://www.cryptokitties.co/. (Acedido a: 09-12-2021).

[62] Tamagotchi. https://tamagotchi.com/pt/inicio/. (Acedido a: 09-12-2021).

[63] CryptoKitties. Technical details. https://www.cryptokitties.co/technical-details. (Acedido a: 09-12-2021).

[64] NonFungible. https://nonfungible.com/market/history/cryptokitties?filter=saleType%3D&length=10&sort=blockTimestamp%3Ddesc&start=0. (Acedido a: 09-12-2021).

[65] Gods Unchained. https://godsunchained.com/. (Acedido a: 09-12-2021).

[66] Yu-Gi-Oh. https://www.yugioh-card.com/. (Acedido a: 09-12-2021).

[67] Pokemon TCG. https://www.pokemon.com/us/pokemon-tcg/play-online/. (Acedido a: 09-12-2021).

[68] Zed Run. https://zed.run/. (Acedido a 03-11-2021).

[69] CryptoSlam. https://cryptoslam.io/zed-run/sales/summary. (Acedido a: 03-11-2021).

[70] CoinMarketCap. https://coinmarketcap.com/currencies/ethereum/. (Acedido a: 03-11-2021).

[71] Manifesto para o Desenvolvimento Ágil de Software. http://agilemanifesto.org/. (Acedido a 06-01-2022).

[72] Simon Kneafsey. A Short History Of Scrum. https://www.thescrummaster.co.uk/scrum/short-history-scrum/. (Acedido a 06-01-2022).

[73] Sandra Petrova. Adopting Agile: The Latest Reports About The Popular Mindset. https://adevait.com/blog/remote-work/adopting-agile-the-latest-reports-about-the-popular-mindset. (Acedido a 06-01-2022).

[74] Lavanya, N. & Malarvizhi, T. (2008). Risk analysis and management: a vital key to effective project management. https://www.pmi.org/learning/library/risk-analysis-project-management-7070. (Acedido a 06-01-2022).

[75] Moscow Prioritisation. https://www.agilebusiness.org/page/ProjectFramework_10_MoSCoWPrioritisation. (Acedido a 13-01-2022).

[76] Shomik Sem Bhattacharjee. https://gadgets.ndtv.com/cryptocurrency/news/signal-co-founder-moxie-marlinspike-nft-poop-emoji-blockchain-twitter-2576697. (Acedido a 19-01-2022).

[77] Jonas Bostoen. A Hands-on Introduction to IPFS. https://medium.com/coinmonks/a-hands-on-introduction-to-ipfs-ee65b594937. (Acedido a 30-01-2022).

[78] D. Yaga, P. Mell, N. Roby, K. Scarfone. Blockchain Technology Overview. https://nvlpubs.nist.gov/nistpubs/ir/2018/nist.ir.8202.pdf . (Acedido a 22-06-2022).

[79] Kevin Linden. What is Proof-of-History?. https://anycoindirect.eu/en/blog/what-is-proof-of-history. (Acedido a 22-06-2022).

[80] ISO/IEC 25010:2010. Systems and software engineering — Systems and software quality requirements and evaluation (SQuaRE) — System and software quality models. Switzerland, 2010. (Acedido a 21-06-2022).

[81] Moralis. https://moralis.io/. (Acedido a 24-06-2022).

[82] Ethereum.org. Introduction to Web3. https://ethereum.org/en/web3/. (Acedido a 24-06-2022).

[83] Spring Boot. https://spring.io/projects/spring-boot. (Acedido a 26-06-2022).

[84] Java. https://www.java.com/. (Acedido a 26-06-2022).

[85] HTML/CSS. https://html.com/. (Acedido a 26-06-2022).

[86] AngularJS. https://angularjs.org/. (Acedido a 26-06-2022).

[87] JavaScript. https://developer.mozilla.org/en/JavaScript. (Acedido a 26-06-2022).

[88] Apache Tomcat. https://tomcat.apache.org/. (Acedido a 26-06-2022).

[89] Oracle Database. https://www.oracle.com/pt/database/12c-database/. (Acedido a 26-06-2022).

[90] NodeJS. https://nodejs.org/. (Acedido a 26-06-2022).

[91] ExpressJS. https://expressjs.com/. (Acedido a 26-06-2022).

[92] Web3JS. https://web3js.readthedocs.io/. (Acedido a 26-06-2022).

[93] Solidity. https://docs.soliditylang.org/. (Acedido a 26-06-2022).

[94] IntelliJ IDEA. https://www.jetbrains.com/idea/. (Acedido a 27-06-2022).

[95] GitLab. https://about.gitlab.com/. (Acedido a 27-06-2022).

[96] Docker. https://www.docker.com/. (Acedido a 27-06-2022).

[97] Jenkins. https://www.jenkins.io/. (Acedido a 27-06-2022).

[98] Docker docs. Dockerfile reference. https://docs.docker.com/engine/reference/builder/. (Acedido a 27-06-2022).

[99] Docker docs. Overview of Docker Compose. https://docs.docker.com/compose/. (Acedido a 27-06-2022).

[100] Apache Maven. https://maven.apache.org/. (Acedido a 27-06-2022).

[101] Remix IDE. https://remix-project.org/. (Acedido a 27-06-2022).

[102] OpenZeppelin | docs. ERC 1155. https://docs.openzeppelin.com/contracts/3.x/api/token/erc1155. (Acedido a 27-06-2022).

[103] OpenZeppelin | docs. ERC 721. https://docs.openzeppelin.com/contracts/3.x/api/token/erc721. (Acedido a 27-06-2022).

[104] Polygon Scan Mumbai. https://mumbai.polygonscan.com/. (Acedido a 27-06-2022).

[105] L. Lesavre, P. Varin, D. Yaga. Blockchain Networks: Token Design and Management Overview. https://nvlpubs.nist.gov/nistpubs/ir/2021/NIST.IR.8301.pdf. (Acedido a 27-06-2022).

[106] Polygon Technology | Docs. Mumbai PoS Testnet. https://docs.polygon.technology/docs/develop/network-details/network/. (Acedido a 27-06-2022).

[107] Jest. https://jestjs.io/. (Acedido a 29-06-2022).

[108] Supertest. https://www.npmjs.com/package/supertest. (Acedido a 29-06-2022).

[109] Apache JMeter. https://jmeter.apache.org/. (Acedido a 29-06-2022).

[110] OpenSea. https://opensea.io/. (Acedido a 29-06-2022).

[111] Chainlink. What is a Blockchain Oracle? https://chain.link/education/blockchain-oracles. (Acedido a 01-07-2022).

[112] Messari. https://messari.io/. (Acedido a 01-07-2022).

[113] Bootstrap. https://getbootstrap.com/. (Acedido a 03-07-2022).

Apêndices

0

Apêndice A

A.1 Planeamento

Nesta secção estão representados os diagramas de *Gantt* utilizados para o planeamento do primeiro e segundo semestre.

0

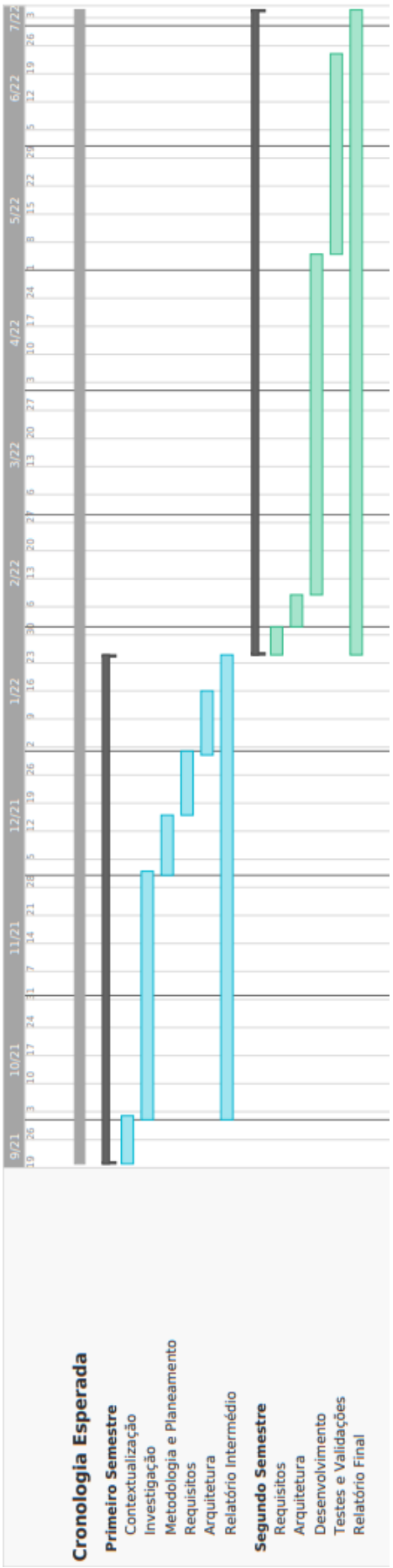


Figura A.1 - Diagrama de Gantt com a cronologia esperada para ambos os semestres.

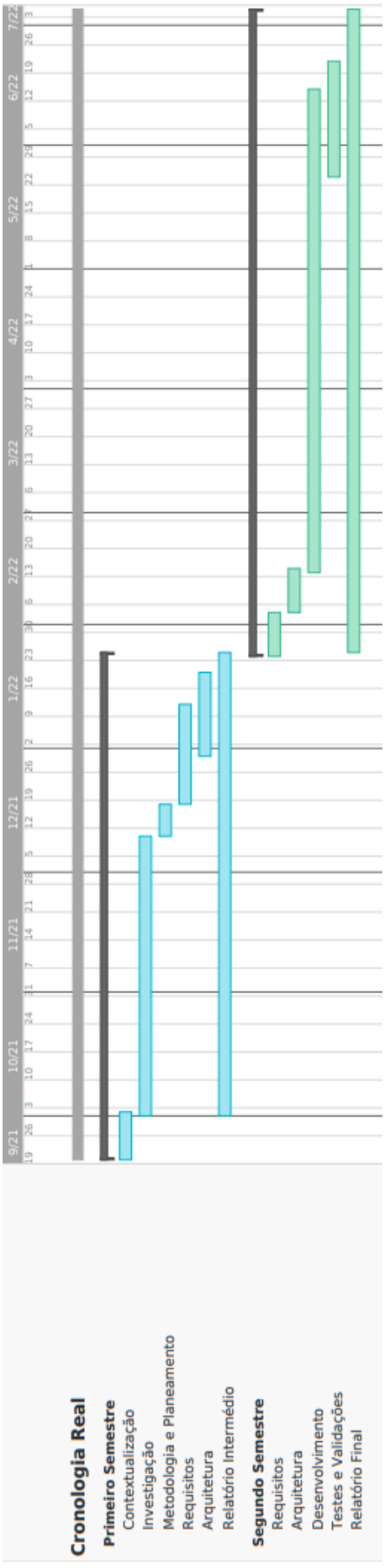


Figura A.2 - Diagrama de Gantt com a cronologia obtida, para ambos os semestres.

0

Apêndice B

B.1 Levantamento de Requisitos

Na presente secção encontram-se descritos alguns dos cenários que foram considerados no decorrer do processo de levantamento dos requisitos funcionais.

Cenário 1 (Modelo adotado durante o estágio)

Descrição: Todas as cartas do jogo passam a ser *Non-Fungible Tokens* (NFTs), ou seja, ao abrir um Shake, receberia-se um NFT, por exemplo. O mesmo aconteceria com as restantes funcionalidades, como por exemplo ao receber uma carta repetida ou ao realizar uma troca de cartas.

Pontos a considerar: Dada a grande quantidade de cartas que são transacionadas no Shake It, é importante ter os custos em conta, dependendo da *Blockchain* a ser usada.

Cenário 2

Descrição: Dividir os prémios já existentes por raridades (por exemplo: comum, incomum, raro, lendário) e nas categorias mais baixas dar um NFT em conjunto com o prémio.

Pontos a considerar: O NFT poderia depois ter algum propósito secundário, dentro de algum contexto do jogo.

Cenário 3

Descrição: Criar uma criptomoeda para o Shake it, e em todas as transações de troca de cartas recebe-se essa moeda. Depois, em cada temporada do Shake It, cria-se um número bastante limitado de NFTs e um mercado para os mesmos, e quem tiver moedas suficientes pode comprá-los.

Pontos a considerar: Um mercado de NFTs poderia tornar-se algo ambicioso e complexo de mais para desenvolver, tendo em conta a duração do estágio.

Cenário 4

Descrição: Criar um número limitado de NFTs e distribuí-los de forma aleatória através das mecânicas do jogo, como troca de cartas ou quando se acerta numa pergunta ao abrir shakes.

Pontos a considerar: Para além disto, deve ser dada alguma utilidade aos NFTs e vantagens para quem os possui.

Cenário 5

Descrição: Cada temporada tem um número limitado de NFTs (relacionados com algum tema, por exemplo). Quem conseguir colecionar pelo menos um, poderá votar em certas

decisões dentro do jogo, como o tema da próxima temporada, próximos prémios, entre outras coisas. Quantos mais NFTs um utilizador possuir, maior o seu poder de voto.

Pontos a considerar: Teria que se desenvolver um mecanismo para fazer a distribuição dos NFTs de forma justa e aleatória; Teria de existir um mercado para comprar e vender os NFTs.

Cenário 6

Descrição: Por cada *achievement* completado (por exemplo: se abrirmos 10 *shakes*, ganhamos um *achievement* chamado "Shaker") ganha-se um NFT que represente esse *achievement*. Depois, esse NFT (ou um conjunto de NFTs) pode ser trocado por prémios mais comuns, como comunicações ou acessórios e outros brindes.

Pontos a considerar: Poderia ser criado um sistema de compra e venda de NFTs, para motivar os utilizadores a colecionar mais cartas, e consequentemente a jogar mais.

Cenário 7

Descrição: O prémio de uma das categorias da caderneta pode ser um NFT, relacionado com um artista de música ou com outro tema interessante, e só seria lançada uma quantia limitada de NFTs todos os meses. As primeiras pessoas a conseguirem completar essa categoria, receberiam o NFT.

Pontos a considerar: Teria de se arranjar uma maneira para dar utilidade a esses NFTs.

0

Apêndice C

C.1 Arquitetura

Nesta secção, na Figura C.1, encontra-se disponível o diagrama da Arquitetura do presente estágio.

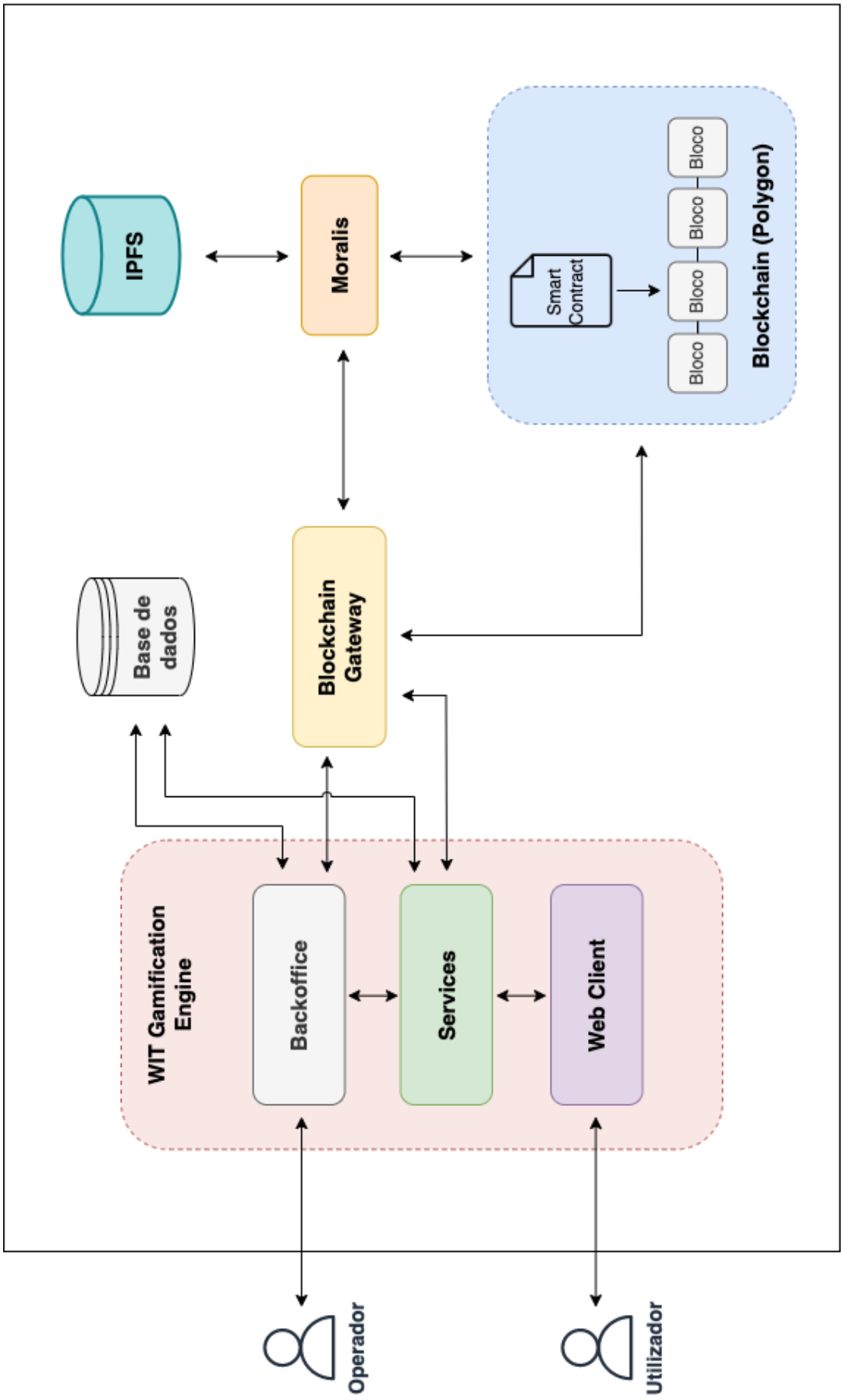


Figura C.1 - Arquitetura do projeto

0

Apêndice D

D.1 *Smart Contract*

Nesta secção, na Figura D.1, é apresentado o *Smart Contract* implementado no decorrer do segundo semestre, durante a fase de desenvolvimento do estágio.

0

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.4;

import "@openzeppelin/contracts/token/ERC1155/ERC1155.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract ShakeItNFT is ERC1155, Ownable {

    mapping (uint256 => string) private _uris;

    constructor() ERC1155("") {}

    function setURI(string memory newuri) public onlyOwner {
        _setURI(newuri);
    }

    function mint(address account, uint256 id, uint256 amount, string memory tokenURI)
        public
        onlyOwner
    {
        _mint(account, id, amount, "");
        setTokenURI(id, tokenURI);
    }

    function uri(uint256 tokenId) override public view returns (string memory){
        return(_uris[tokenId]);
    }

    function setTokenURI(uint256 tokenId, string memory newuri) public onlyOwner {
        _uris[tokenId] = newuri;
    }
}
```

Figura D.1 - *Smart Contract*