Universidade de Trás-Os-Montes e Alto-Douro

Escola de Ciência e Tecnologia
Mestrado em Engenharia Informática

Player Market Cap: Aplicação móvel para negociação de ativos digitais usando tecnologia *blockchain*

Nelson Miguel Alves Gomes

Relatório de Estágio Realizado no Âmbito do Mestrado em Engenharia Informática

Sob Orientação do Professor Doutor Vítor Filipe

Vila Real, 2020

Universidade de Trás-Os-Montes e Alto-Douro

Escola de Ciência e Tecnologia
Mestrado em Engenharia Informática

Player Market Cap: Aplicação móvel para negociação de ativos digitais usando tecnologia *blockchain*

Nelson Miguel Alves Gomes

Relatório de Estágio Realizado no Âmbito do Mestrado em Engenharia Informática

Sob Orientação do Professor Doutor Vítor Filipe

Relatório de estágio apresentado por Nelson Miguel Alves Gomes à Universidade Trás-Os-Montes e Alto-Douro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Informática, sob orientação do Professor Doutor Vítor Filipe, Professor Associado com Agregação do Departamento de Engenharias da Universidade de Trás-Os-Montes e Alto-Douro.

Resumo

O presente relatório descreve o trabalho desenvolvido durante o estágio na empresa Brandit, que focou o desenvolvimento da aplicação móvel “Player Market Cap”. O principal objetivo desta aplicação é permitir aos utilizadores a negociação de ativos digitais numa rede *blockchain*. Os ativos digitais são representativos de jogadores de futebol, e os utilizadores poderão comprar, guardar e vender os ativos, podendo obter lucros a partir da sua valorização ao longo do tempo. Para além disto, a aplicação tem outros intuitos como permitir a gestão de contas de utilizador ou apresentar os conteúdos de uma forma cativante.

O desenvolvimento da aplicação móvel (nível de *front-end*) foi complementado por tarefas de desenvolvimento a nível de *back-end*, como a implementação de uma rede *blockchain*, a criação de uma base de dados e a implementação de uma API (*Application Programming Interface*), de modo a suportar todas as funcionalidades pretendidas para a aplicação.

Palavras-chave: blockchain, ativos digitais, aplicação móvel, front-end, back-end

Abstract

This report describes the tasks done during the internship at Brandit, which has been focused on the development of a mobile application "Player Market Cap". The main purpose of the application is to allow users to trade digital assets on a blockchain network. These digital assets are representative of football players. The app allows users to buy, store and sell these assets, making profits from their appreciation over time. In addition, the application has other purposes such as enabling user account management or displaying the content in an engaging way.

The mobile app development was complemented by back-end development such as the implementation of a blockchain network, the creation of a database and the implementation of an Application Programming Interface (API), with the purpose of enabling all the planned features of the application.

Keywords: blockchain, digital assets, mobile app, front-end, back-end

Agradecimentos

Ao Professor Doutor Vítor Filipe, orientador deste trabalho, pela sua grande disponibilidade, dedicação, confiança e apoio durante todo o período de estágio.

À Brandit, empresa onde decorreu o estágio, por esta oportunidade de desenvolvimento pessoal e profissional, em especial ao Senhor António Martins pela atribuição deste projeto e por me proporcionar todas as condições necessárias ao seu desenvolvimento, bem como ao Tiago Coelho, pela vasta transmissão de experiência e conhecimentos e por estar sempre disponível para o esclarecimento de todas e quaisquer dúvidas que foram surgindo durante o estágio.

À minha família e amigos, em especial aos meus pais, José Gomes e Alice Alves, pelo apoio e motivação transmitidos ao longo deste período, fulcrais para que pudesse aqui chegar.

A todos um sincero obrigado!

Índice

Lista de figuras

Lista de tabelas

Lista de siglas e acrónimos

API – *Application Programming Interface*

CPU – *Central Processing Unit*

CRUD - *Create, Read, Update, Delete*

DEX -*Decentralized Exchange*

FTP – *File Transfer Protocol*

HTTP - *Hypertext Transfer Protocol*

HTTPS - *Hypertext Transfer Protocol Secure*

IDE - *Integrated Development Environment*

JSON – *Javascript Object Notation*

LPOS – *Leasing Prove of Stake*

NPM – *Node Packcage Manager*

ORM – *Object-Relation Mapping*

P2P – *Peer to Peer*

PL/SQL - *Procedural Language/Structured Query Language*

PMC – *Player Market Cap*

REST - *Representational State Transfer*

SDK- *Software Development Kit*

SGBD – *Sistema de Gestão de Base de Dados*

SFTP – *SSH File Transfer Protocol*

SQL - *Structured Query Language*

SSH – *Secure Shell*

SSL/TLS – *Secure Sockets Layer/Transport Layer Security*

TCP/IP - *Transmission Control Protocol/Internet Protocol*

URL – *Uniform Resource Locator*

XML - *Extensible Markup Language*

1- Introdução

1.1 - Entidade acolhedora

A Brandit[1] é uma empresa sediada em Barcelos e fundada em 2007 por António Martins e Pedro Araújo. De uma pequena *startup* de *software* rapidamente cresceu e tornou-se numa empresa com parceiros e clientes por todo o mundo, incluindo nomes como Sony, Apple, Real Madrid, Bosch, Puma, entre muitos outros.

Atualmente possui uma equipa com cerca de 20 profissionais distribuídos por áreas como o desenvolvimento de *software,* design gráfico e *web* e produção de conteúdos multimédia.

No que à área de *software* diz respeito, a Brandit destaca-se no desenvolvimento para o setor do futebol, sendo reconhecida pela criação de aplicações móveis e aplicações web como o Tactical Soccer[2], o Mourinho Tactical Board[3], o Scouting System[4] ou mesmo aplicações customizadas e exclusivas para alguns dos clubes de maior renome a nível nacional e internacional.

Contando já com centenas de projetos desenvolvidos, a criatividade, a inovação e a alta qualidade do produto final são as marcas que a equipa da Brandit procura vincar em cada trabalho desenvolvido.

1.2 - Motivação e contexto

A *blockchain* é atualmente uma tecnologia que tem suscitado interesse por parte de várias empresas, com vista à sua possível adoção de modo a melhorar soluções desenvolvidas ou a conceber produtos inovadores [1].

Este interesse também acabou por gerar-se ao nível da Brandit, que sendo uma empresa com um espírito empreendedor e que está sempre na procura de usar novas tecnologias que possam trazer mais valia aos seus produtos e desenvolvimentos.

[1] https://www.brandit.pt

[2] https://www.tacticalsoccer.pt

[3] https://www.mourinhotacticalboard.com

[4] https://scoutingsystem.com

Deste modo, e depois de um estudo aprofundado da tecnologia *blockchain* por parte dos responsáveis da Brandit, surgiu a vontade de investir no desenvolvimento de um projeto com base nesta tecnologia.

Sendo a empresa fortemente ligada com o setor do futebol, onde estão presentes no mercado com diversas aplicações e onde tem uma grande rede de contactos, acabou por ser idealizada uma aplicação relacionada com esta área.

O "Player Market Cap*"* surge assim, como uma aplicação móvel que terá como principal objetivo a negociação de ativos digitais (mais conhecidos como *tokens*). Estes *tokens* serão representativos de jogadores de futebol e o seu valor variará de acordo com as movimentações de mercado (compras e vendas) na plataforma. A infraestrutura em que a aplicação está assente, ao nível dos métodos de negociação e valorização dos ativos digitais, comportar-se-á de forma semelhante às plataformas de negociação (*trading*) de criptomoedas, que também tem como base a tecnologia *blockchain*, e tirará assim partido das suas principais funcionalidades.

1.3 - Objetivos

Na perspetiva do estagiário, este estágio apresenta desafios que permitem adquirir novos conhecimentos e a oportunidade de solidificar e pôr em prática conhecimentos previamente adquiridos no percurso académico, sendo acompanhado e direcionado por profissionais experientes que lhe indicam as melhores práticas e metodologias de desenvolvimento ao longo do estágio, tendo-lhe sido atribuído o desenvolvimento da aplicação "Player Market Cap".

Na perspetiva do projeto atribuído, inclui-se o desenvolvimento de *software* tanto a nível *back-end* (do servidor), como a nível *front-end* (da aplicação móvel), uma vez que o conteúdo apresentado na aplicação móvel será gerado a partir do servidor e as interações do utilizador com a aplicação móvel serão também nele executadas. Ao nível do *back-end*, os objetivos passam pela implementação e gestão de uma rede *blockchain*, pela implementação de uma base de dados e pela criação de uma API (*Application Programming Interface*) para habilitar comunicação do lado do servidor com a aplicação móvel. Ao nível do *front-end*, o objetivo passa pelo desenvolvimento de uma aplicação móvel, disponível para os sistemas Android e iOS.

1.4 - Estrutura do documento

O presente documento é composto por cinco capítulos. No primeiro capítulo é feita uma introdução, com uma breve descrição da entidade acolhedora, dos objetivos do estágio, da idealização do projeto e motivos da sua execução.

No segundo capítulo enquadra-se o planeamento do estágio, com um cronograma geral das tarefas realizadas e a sua distribuição ao longo do tempo. É também referenciada a metodologia utilizada para o desenvolvimento do projeto.

No terceiro capítulo é efetuada uma apresentação da tecnologia *blockchain*, começando com uma revisão de leitura de alguns dos principais conceitos, seguida de uma descrição de algumas aplicações que tem como base a *blockchain* e por fim uma listagem das sua principais vantagens e desafios da tecnologia. Este capítulo dispõe também de um ponto que perspetiva o desenvolvimento de aplicações móveis, no que toca aos sistemas operativos destinados, tipos de aplicações e de *frameworks* de desenvolvimento.

O quarto capítulo detalha o projeto "Player Market Cap", referenciando as funcionalidades esperadas da aplicação, enunciando a arquitetura do projeto e apresentando as diferentes fases de desenvolvimento do projeto. De forma a apresentar estas fases de desenvolvimento de uma forma mais organizada, este último ponto foi subdividido em dois tópicos: desenvolvimento *back-end* e desenvolvimento *front-end*. Na fase de *back-end* são descritas as diferentes tarefas, métodos e tecnologias usadas para a implementação do projeto ao nível do servidor, enquanto que na fase de *front-end* estas são descritas para a implementação ao nível da aplicação móvel.

No quinto capítulo é feita a apresentação dos resultados do desenvolvimento comparativamente ao que se pretendia da aplicação em termos de funcionalidades.

No sexto e último capítulo são apresentadas as conclusões retiradas deste trabalho e lançadas considerações de trabalho futuro.

2 - Planeamento e Metodologia

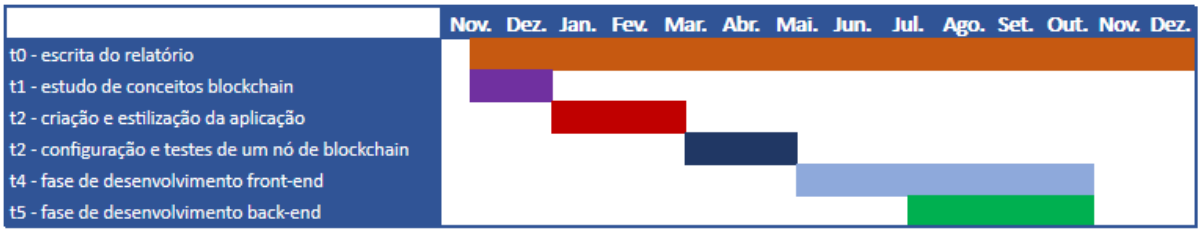
2.1 - Planeamento

O estágio decorreu entre 12 de novembro de 2018 e 25 de outubro de 2019, sendo interrompido durante 14 dias, nos quais a empresa esteve encerrada por motivos de férias gerais.

Na Tabela 1, encontra-se uma representação da distribuição das diferentes tarefas (***t***) realizadas ao longo do tempo.

A **t0** não faz parte da elaboração do projeto em si, mas sim uma referência à escrita do presente relatório ao longo de todo o estágio.

Tabela 1 - Cronograma de tarefas



No que diz respeito às tarefas realizas, a primeira tarefa (**t1**) foi reservada a uma aprendizagem dos principais conceitos da *blockchain,* com um estudo da documentação associada ao tema, bem como de vídeos instrucionais de forma a perceber as mecânicas desta tecnologia. Nesta fase foi possível realizar alguns testes de operações numa rede de testes *(Ethereum testnet),* como por exemplo transferências entre diferentes contas, compra e venda de *tokens* ou mesmo criação de simples exemplos de *smart contracts*. Isto permitiu ter uma visão muito mais clara do funcionamento de uma rede *blockchain* e das condições impostas para que isso aconteça.

A segunda tarefa (**t2**) representa uma fase inicial do desenvolvimento da aplicação, no que diz respeito ao seu *design.* Tendo já os *mockups* sido desenvolvidos por parte dos *designers* da empresa aquando do início desta tarefa, procedeu-se à criação e estilização dos diferentes ecrãs da aplicação, ainda sem qualquer tipo de funcionalidades. A realização desta etapa permitiu ter um melhor entendimento do que se pretendia da aplicação "Player Market Cap" e um primeiro contacto com os ambientes e ferramentas de desenvolvimento de aplicações móveis.

A terceira tarefa (**t3**) é relativa à configuração e génese de um nó de rede privada, sendo detalhada no ponto **3.1.2** do capítulo **4.** Nesta fase foram realizados testes de modo

a verificar a operacionalidade da rede e criação de alguns protótipos de mecanismos a implementar numa fase mais avançada.

A tarefa 4 (**t4**), representa a parte mais extensa do projeto, com o desenvolvimento da parte de *back-end*, no qual se inclui o desenvolvimento da base de dados, da API (*Application Programming Interface*) que servirá a aplicação e fará a comunicação entre esta e a rede *blockchain* implementada.

Por fim, a tarefa (**t5**), que acabou por ter um desenvolvimento em paralelo com a fase **t4**, dedicada à elaboração do *front-end*, onde se incluí o desenvolvimento completo da aplicação móvel (tanto para Android como iOS), com comunicação integral ao *back-end* e algumas redefinições dos ecrãs criados na tarefa **t2,** devido a atualizações de *software* que entretanto se verificaram ao nível da tecnologia usada para o seu desenvolvimento ou reformulações que entretanto foram necessárias.

2.2 - Metodologia

O desenvolvimento do projeto foi orientado segundo uma metodologia baseada no modelo de desenvolvimento em cascata (*waterfall*), no qual as diferentes etapas de desenvolvimento ocorrem sequencialmente e dependentes da etapa predecessora para se realizar [2]. Esta foi a metodologia definida por ser amplamente conhecida e utilizada pela empresa em diversos projetos. Na Figura 1, encontra-se esquematizada a metodologia, utilizada ao longo do desenvolvimento.

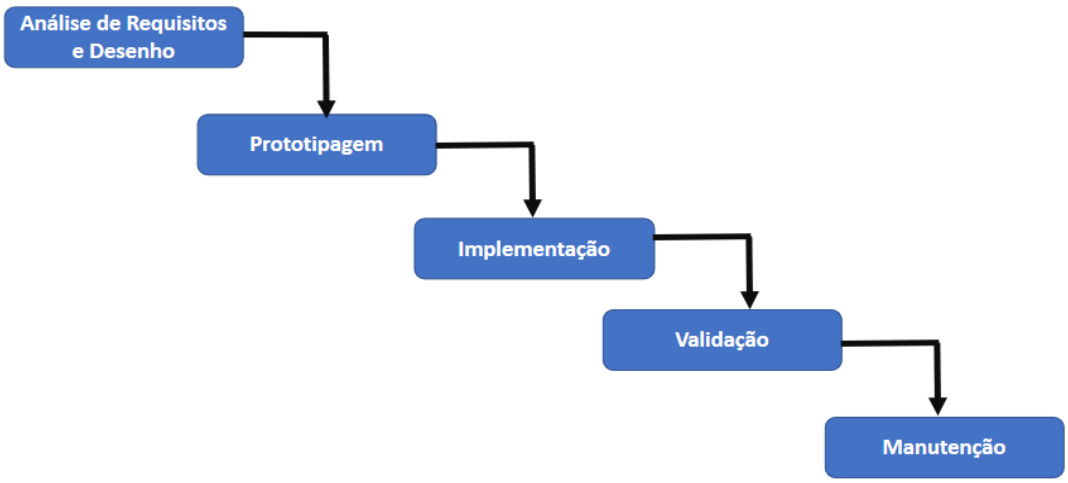


Figura 1 - Metodologia de desenvolvimento aplicada

Na etapa inicial é feita uma análise de requisitos e desenho do sistema de modo a determinar quais as necessidades do sistema, tanto a nível aplicacional, como de servidor.

De seguida, foram elaborados protótipos da aplicação, de forma a simular a aparência que esta terá e as suas principais funcionalidades.

A fase de implementação, envolve todo o processo de desenvolvimento do projeto, implementando a arquitetura definida para o sistema. Após terminada esta fase, segue-se a validação que confere o que foi desenvolvido com o plano inicial e caso haja contratempos segue-se uma correção, que pode ser ao nível da implementação ou mesmo da definição de requisitos.

Com o término de todo este fluxo, o produto do desenvolvimento do projeto, entra no período de manutenção, no qual pode haver necessidade de intervenções, como por exemplo atualizações de algum *software* de suporte do sistema.

De realçar que este relatório se remete às fases de implementação e validação, uma vez que no momento em que o estagiário iniciou funções, a aplicação já se encontrava especificada e também já tinha os protótipos desenvolvidos por *designers* da empresa.

3 – Estado da arte

3.1 - *Blockchain*

O termo *blockchain* surgiu da apresentação da Bitcoin ao público em 2008, com a divulgação do artigo "Bitcoin: A Peer-to-Peer Electronic Cash System" [3] por parte de Satoshi Nakamoto (pseudónimo da pessoa ou conjunto de pessoas criadoras do Bitcoin). Este artigo visava a introdução de uma tecnologia inovadora que possibilita a realização de transações financeiras na internet, prevenindo operações inválidas, dando confiança no mecanismo transacional e evitando ao mesmo tempo que sejam precisas entidades reguladoras para legitimar as transações [4].

O *Bitcoin* nasceu assim, como uma moeda totalmente digital, de alcance global e que não é controlada por governos ou qualquer instituição financeira como bancos, bancos centrais ou fundos monetários privados, visto que está inserida na *blockchain* que cria um mercado autorregulado de transações digitais criptografadas [4].

Segundo Satoshi, a solução para evitar operações inválidas, como por exemplo o "gasto duplo", no qual o utilizador gastaria mais de uma vez as mesmas moedas digitais ou transferência em que um utilizador diz que enviou 100 *Bitcoins* para alguém quando na verdade só enviou 10, passa por um esquema de rede distribuído em conjunto com assinaturas digitais. A rede insere a data e a hora nas transações através de um algoritmo de *hash* e regista-as numa cadeia contínua de blocos encriptados, formando por sua vez registos que a partir daí não poderão ser alterados sem refazer toda a rede e estrutura da *blockchain* [4].

Para que uma nova transação (legítima) e respetivo registo na cadeia possam ser realizados é necessário um processo de validação, que culmina com uma atualização ao nível de um dos computadores da rede e que em questão de segundos é sincronizada e copiada por toda a rede [4].

Na figura 2 podem verificar-se as fases que precedem a execução de uma transação na *blockchain*. Inicialmente é feita a criação da transação (uma transferência de *bitcoins*, por exemplo) inserindo-a num bloco que é enviado para toda a rede. Seguidamente é necessário esperar uma resposta por parte de cada um dos nós (pontos de comunicação) de rede, que podem validar ou não a transação. Por fim, a confirmação e execução da transação depende do consenso das respostas obtidas de todos os nós.

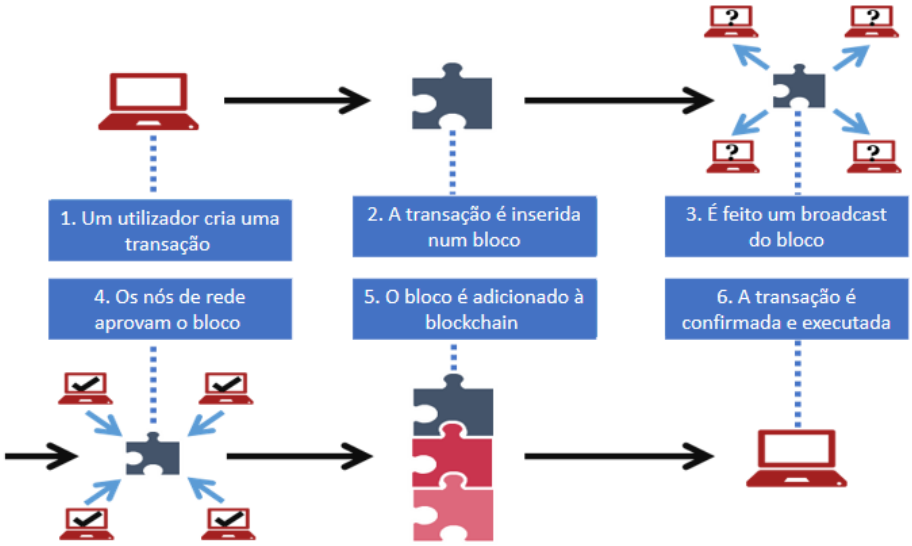


Figura 2 - Execução de transação na blockchain - Adaptada de [6]

Sendo assente numa estrutura distribuída, toda a lógica da *blockchain* está armazenada numa grande quantidade de nós de rede, à qual qualquer pessoa se poderá juntar, desde que cumpra os requisitos impostos nas configurações geradoras da rede. Deste modo o desaparecimento ou perda de ligação de um computador não afeta o funcionamento do sistema [5].

A *blockchain* pode ser definida como uma base de dados, mas com um fator diferenciador: o sistema funciona como um "livro de registos" que devido às suas mecânicas pode ser considerado como inviolável, "inderrubável" e extremamente eficiente [5]. Neste "livro" estão presentes todas as transações que foram processadas e confirmadas pela rede, agrupadas em sucessivos blocos.

Na figura 3 está representada de forma simplificada o processo de encadeamento que é feito na *blockchain*. Sempre que uma transação é confirmada, acaba por ser agregada a outras transações dentro de um bloco. Este bloco vai inserir também informações próprias, como por exemplo a data e hora atuais.

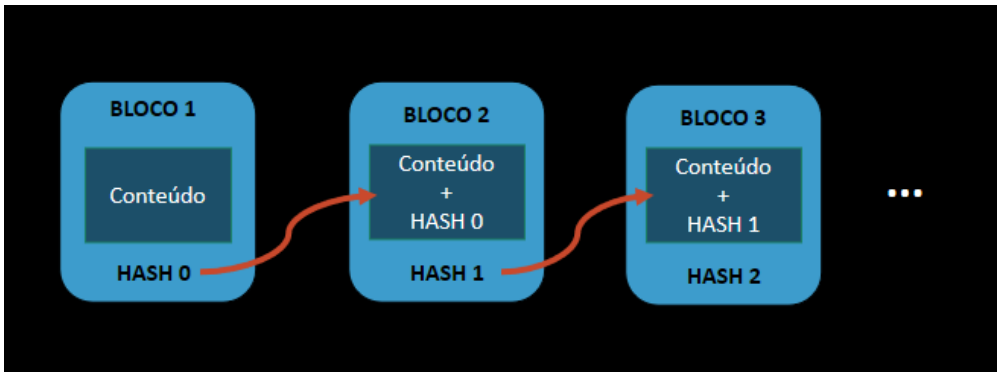


Figura 3 - Processo de encadeamento na blockchain - Adaptada de [5]

Com o conteúdo do bloco é executado um algoritmo de *hash*, que consiste numa função matemática que recebe uma grande quantidade de informações e as transforma num código com um valor representativo do conteúdo do bloco, sendo o código gerado será a assinatura digital do bloco.

Esta assinatura digital vai ser inserida no bloco posterior, fazendo parte do conteúdo do bloco 2, que irá obter a sua assinatura digital através do mesmo método do bloco 1. Esta mecânica é sucessivamente desempenhada sempre que um novo bloco for inserido na *blockchain*.

Depois da submissão de um bloco na rede, se houver alguma informação alterada, o código que o algoritmo de *hash* gera será completamente diferente do que foi gerado no momento da submissão do bloco e, portanto, essa alteração será invalidada [5].

Em suma, a *blockchain* foi o mecanismo de rede divulgado por Satoshi para criar a *Bitcoin* e serviu de inspiração para o lançamento das inúmeras criptomoedas que existem atualmente, com replicações do código base e posteriores alterações para inserção de melhorias ou mecânicas inovadoras. De qualquer forma, a *Bitcoin* continua a ser a criptomoeda mais popular e com mais mercado.

Enquanto que uma parte das criptomoedas se baseiam num sistema de *blockchain* mais virado para o aspeto monetário e respetivas transações financeiras, surgiram já algumas redes de blockchain como a Ethereum[5]que permitem que para além das transações financeiras, utilizadores com conhecimentos de programação criem "*dAPPs"*, que são aplicações descentralizadas de "código-aberto" que estão armazenados na *blockchain* e operam automaticamente [7].

[5] https://ethereum.org/

A tecnologia em si é tão interessante que se constatou que poderia ser usada noutros sistemas para além dos da área financeira, como os comerciais, governamentais e eleitorais e nesse sentido tem havido um grande interesse por parte de várias entidades como bancos, seguradoras, corretoras de ações, empresas de segurança ou governos que pretendem desenvolver soluções mais eficientes com a *blockchain* [5].

3.1.1 - Conceitos associados à *Blockchain*

3.1.1.1 - *Nodes, miners* e *fees*

Um *node* (ou nó) serve como um ponto de redistribuição ou um terminal de comunicação da rede. De uma forma simples, é representado por um dispositivo físico, ou em alguns casos virtual, onde uma mensagem pode ser criada, recebida ou transmitida [8].

Num contexto de *blockchain*, qualquer dispositivo que se conecte à rede pode ser definido como um node, nem que seja apenas para obter um mínimo de informações, visto que comunica com outros dispositivos da rede para este efeito. Há, no entanto, dois tipos de nodes que se destacam numa rede blockchain: os *full nodes* e os *mining nodes* [8].

Os *full nodes* são responsáveis pelo suporte e segurança de uma *blockchain.* Fazem constantes replicações (cópias) dos blocos e transações que vão sendo inseridas na rede, podendo armazenar uma cópia completa ou reduzida de toda a informação armazenada na *blockchain*. Entram nos processos de verificação dos blocos, assegurando que as regras algorítmicas estipuladas para a sua validação são seguidas [8]. Estes *nodes* podem ser estabelecidos por qualquer pessoa, através de variadas implementações de *software,* tendo, no entanto, requisitos mínimos, que dependem do tipo de rede *blockchain.* Por exemplo, para um *node* de Bitcoin, são necessários 2 Gb de memória RAM, algum espaço livre para armazenamento no disco (na ordem dos 200 GB), uma conexão de *internet* de alta velocidade e elevada largura de banda para executar *uploads.* Para além disso, um *full node* deve estar ativo pelo menos 6 horas todos os dias, sendo preferencial que esteja ativo continuamente (24 horas por dia) [8].

Os *mining nodes* ou *miners* são na sua estrutura semelhantes aos *full nodes,* sendo que correm paralelamente programas de *software* específico para poderem fazer parte do processo de mineração (processo de adição de blocos à rede). Para correr estes *softwares* específicos, é necessário um grande investimento em *hardware* especializado. Este tipo

de *nodes* tira o máximo partido do poder computacional, para chegar ao objetivo de ser o *miner* de um bloco de transações. Isto é feito, através de *softwares* que têm de resolver problemas criptográficos bastante complexos, recorrendo a sucessivas tentativas-erro. Este processo é feito por todos os *mining nodes* de uma *blockchain*, sendo, portanto, bastante competitivo, pois a mineração de um bloco é assegurada por um apenas um dos *nodes*. Ao concluir este processo, os restantes *nodes* limitam-se a verificar e transmitir o bloco, sendo que o *miner* do bloco recebe uma recompensa em forma criptomoedas na sua conta pessoal [8].

Esta recompensa, incentiva a que existam os *mining nodes*, de forma a tornar a rede mais segura, pois quanto mais *miners* existirem mais segura será a rede [9]. O seu valor pode ser gerado de raiz ou constituído pelo conjunto das *fees* das transações do bloco, que são taxas de valor bastante reduzido que são pagas pelos criadores de cada transação que é feita na rede [3].

3.1.1.2 - Carteiras e *balances*

Numa *blockchain*, uma carteira é um par criptográfico, composto por uma chave privada (*private key*) e uma chave pública (*public key*), e permitem correlacionar transações ou ordens com os seus intervenientes. As chaves públicas e privadas são *strings* de base58 que correspondem a uma e apenas uma entidade [10].

A chave pública representa o endereço de uma entidade na rede *blockchain*, sendo usada por exemplo numa transferência como a referência de destino, enquanto que a chave privada é o que garante a posse desse endereço e dos ativos (*balances*) a ele associados [10].

De forma a tornar estas representações ainda mais simples, a chave pública pode ser representada por um *address*, que é uma versão *string* em base58 mais reduzida e a chave privada pode ser representada por uma *seed phrase* ou *seed recovery phrase*, que é um conjunto de um determinado número palavras do dicionário inglês com uma ordem específica, cuja configuração depende do método de conceção desta *seed* [10]. Por exemplo em caso de uso do método BIP39, o dicionário é composto por 2048 palavras, providenciando um imenso número de combinações possíveis, que é tanto maior quanto maior for o número de palavras usadas na *seed*. Na tabela 2 podem verificar-se exemplos de representações do formato de um *address* e de uma *seed* (com base em 15 palavras).

Tabela 2 - Representação simplificada de informações de uma carteira

Address	3P9KR33QyXwfTXv8kKtNGZYtgKk3RXSUk36
Seed phase	*uncle push human bus echo drastic garden joke sand warfare sentence fossil title color combine*

Cada carteira pode possuir diferentes conjuntos de ativos digitais (*tokens)* com diversas quantidades. Estes ativos podem representar criptomoedas, usadas como meio de negociação na rede de acordo com as condições de mercado ou como *tokens*, também conhecidos como *assets,* que podem representar algo do mundo real ou virtual, como um produto ou uma marca para serem negociados na *blockchain* [10].

3.1.1.3 - Métodos de validação

A validação de um bloco e respetiva adição à *blockchain* rege-se por métodos de validação, que foram desenvolvidos para prevenir ataques de negação de serviço (DDoS), spams na rede ou outros abusos do sistema. Existem diversos métodos destacando-se entre eles o *Prove of Work* e o *Prove of Stake.*

O *Prove of Work* foi introduzido como o método de validação pioneiro, sendo o método usado pelo Bitcoin e por muitas outras plataformas de *blockchain.* O algoritmo por trás deste processo baseia-se no tempo e poder de processamento dos *miners* para fazer a validação das transações, apresentando problemas computacionais complexos que estes devem solucionar para comprovar a sua legitimidade como *miners,* bem como das transações do bloco. Este método tem como desvantagens exigir a aquisição de *hardware* muito caro e consumir enormes quantidades de energia, acabando por não ser a solução mais eficiente na atualidade [11].

O *Prove of Stake* surgiu em 2011 e trouxe uma nova visão para a validação de transações, consistindo numa seleção pseudoaleatória do *node* que será eleito para validar o próximo bloco. Esta seleção pode basear-se em diversos fatores que dependem das configurações de implementação da *blockchain*, podendo ser baseada por exemplo no número de criptomoedas que um utilizador possui ou o período de tempo desde que este as possui [11].

No *Prove of Stake* os *nodes* que quiserem fazer parte do processo têm que reservar uma certa quantidade de criptomoedas, não as podendo transferir ou gastar, sendo que quanto maior for a sua quantidade, maior também será a probabilidade de ser o nó escolhido. Esta reserva é denominada de *stake* e serve como "prova de participação" no

processo, evitando que os proprietários dos *nodes* sintam motivação para defraudar a rede, uma vez que quando uma transação fraudulenta é detetada, o *node* corre o risco de perder a *stake* e perder o direito de participar na validação [11].

Neste método de validação o *node* escolhido, valida o bloco de transações e insere-o na *blockchain*, sendo recompensado com o conjunto das *fees* das transações do bloco, proporcionando uma maior eficiência energética, quando comparada com o *Prove of Work*, uma vez que não há necessidade de um grande número de *nodes* competirem por validar e adicionar o mesmo bloco à rede [11].

3.1.1.4 - Redes públicas e privadas

A *blockchain* surgiu como uma rede pública, permitindo a possibilidade de qualquer pessoa aderir e poder contribuir para o seu funcionamento, participando em atividades como a leitura, criação ou validação das transações. O modo de operação incentiva a que participantes se juntem à rede e assim ajudem a manter a sua natureza distribuída e "autogovernada", ou seja, livre da autoridade de terceiros [12].

A *blockchain* pública é onde se dão todas as transações e validações de caráter real, sendo que existem também réplicas destas implementações de rede denominadas de *testnet*, onde os utilizadores ou programadores podem fazer diversos tipos de teste sem terem a preocupação de lidar com criptomoedas reais ou de algum modo perturbar ou danificar a rede, uma vez que todas as transações e *tokens* da rede têm um valor fictício [13].

A rede privada é uma implementação de *blockchain* com características e funcionalidades similares a uma rede pública, embora como o nome indica não seja aberta a todos. Neste tipo de implementação, um participante apenas se pode juntar à rede através de um convite específico por parte de um operador já presente na rede ou por algum protocolo implementado nas configurações de rede [12].

Em suma, nesta rede os utilizadores não têm os mesmos direitos para executar as funções padrão da *blockchain* como a criação de transações, validação de blocos ou replicação dos *nodes* a não ser que estas sejam disponibilizadas pelo criador da rede [12].

Este tipo de rede pode trazer benefícios ao nível da velocidade de validação e escalabilidade da rede, na medida em que com a limitação de acesso a dados e funções, permite a necessidade de um menor tempo para atingir um consenso na validação das

transações, supondo que com limite de acessos o número de *nodes* na rede será menor [13].

Como desvantagens pode referir-se que no caso de uma rede privada ser composta por um reduzido número de *nodes* e estes pertencerem à mesma entidade há o risco de a *blockchain* desligar, por exemplo devido a uma falha de energia. Também o controlo que ser exercido sobre quem pode aceder às informações de rede pode ser visto por este prisma, pois pode acabar por contrariar os pressupostos de transparência da *blockchain* [14].

3.1.1.5 – *Order Book*

O *order book* é uma lista de ordens de compra e venda de ativos que faz parte da *blockchain.* Nesta lista são inseridas todas as intenções de compra ou venda, incluindo para cada uma delas uma referência da conta interveniente, o número de unidades a transacionar e o preço a que o comprador ou vendedor o pretendem fazer [15].

Os registos no *order book* permitem combinar ordens que possuam valores coincidentes relativamente às unidades da ordens e valores iguais ou superiores ao preço estabelecido na ordem de venda. Esta combinação de ordens pode ser feita manual ou automaticamente, dependendo da configuração do nó de *blockchain,* originando transações entre a conta do comprador e a do vendedor e vice-versa [15].

3.1.1.6 - Wallet e Exchange

Uma *wallet* é uma ferramenta, normalmente disponibilizada através de aplicações *web* ou móveis, com um ambiente seguro e criptografado onde os utilizadores podem interagir com as diferentes *blockchains*, podendo aceder aos seus *balances*, enviar ou receber os diferentes tipos de criptomoedas. Isto é possível recorrendo às chaves públicas e às chaves privadas, que fazem a ligação aos recursos das diversas redes. Numa *wallet* é também possível criar mais que uma conta na mesma rede *blockchain*, gerando novos pares de chaves públicas e privadas conforme a necessidade [16].

Já uma *exchange* é uma ferramenta, que pode também ser disponibilizada como aplicação *web* ou móvel onde os utilizadores podem fazer negociações de criptomoedas ou *assets.* É também um meio onde pode ser feita a permuta entre moedas oficiais e criptomoedas, sendo assim realizada a monetização das criptomoedas (transição de valor entre o "mundo real" e as redes de *blockchain*) [16].

3.1.2 - Vantagens, limitações e desafios da *blockchain*

O uso da tecnologia *blockchain* tem como maior objetivo a remoção da necessidade de uma terceira entidade para verificar a legitimidade e validação de uma transação. Isto traz benefícios como a remoção do "erro-humano" na verificação, que passa a ser feita computacionalmente na própria rede, a redução de custo por não envolver verificação por terceiras entidades. A sua estrutura distribuída torna-a mais difícil de defraudar, tornando as transações seguras e eficazes e permite transparência das informações da rede [17].

Como limitações do uso desta tecnologia pode-se referir a ineficiência no caso da implementação do método de validação em *Prove of Work*, devido aos elevados gastos energéticos que este requer. No entanto este problema tem vindo a ser superado com o surgimento de novos métodos de validação como o *Prove of Stake* e as suas derivações. Uma outra limitação, que de certo modo acaba por ser o que dá segurança à conta de um utilizador na *blockchain*, é que a chave privada não é armazenada em nenhum local da rede, sendo por isso da responsabilidade do utilizador saber e guardar de forma segura as suas credenciais. No caso de perda ou esquecimento destes dados os *balances* de conta não poderão ser recuperados. A recuperação de conta pode ser feita com a inserção da *seed phrase*, que representa de uma forma mais acessível a chave privada que os utilizadores devem guardar e é também apresentada ao utilizador no momento em que a sua conta foi concebida [17].

Por fim, aquele que é o maior desafio da *blockchain,* por envolver criptomoedas e monetização da rede, tem a ver com questões políticas e regulamentares que são inexistentes ou pouco esclarecedoras em vários países (como é o caso de Portugal[6]), criando diversas dificuldades em integrar a tecnologia em aplicações ou modelos de negócio [17].

3.1.3 - Aplicações *web* e móveis com base em *blockchain*

Desde o surgimento da *blockchain*, diversas aplicações têm vindo a ser concebidas com base nesta tecnologia. O tipo de aplicações mais comuns são as *wallets* e as *exchanges*, que são disponibilizadas tanto para plataformas *web* e móveis, como por

[6] https://www.bportugal.pt/page/moedas-virtuais

exemplo a Coinbase[7], representada na figura 4. Estas aplicações permitem ao utilizador fazer a gestão dos seus *balances* nas diversas contas que possa possuir e realizar *trading* de criptomoedas ou *tokens* de acordo com as condições de mercado, realizando investimentos que se podem revelar benéficos ou não ao longo do tempo, dependendo da valorização das referidas criptomoedas ou *tokens* no mercado [18].

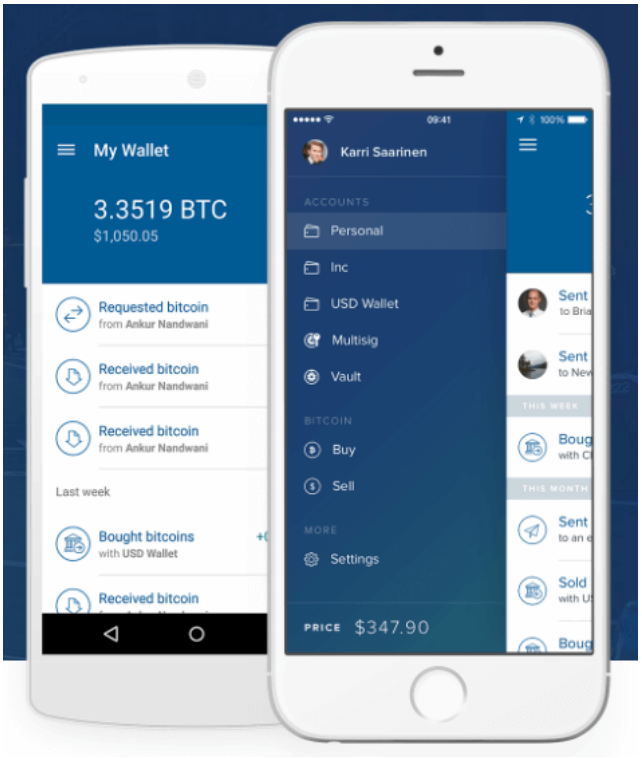


Figura 4 - Aplicação móvel da Coinbase[8]

Surgiram também aplicações com uma abordagem de compra e venda de ativos digitais (*tokens)* inovadora, como o caso da aplicação CryptoKitties[9], que é conhecida como um "jogo em *blockchain*".

Esta aplicação está implementada na rede pública de *blockchain* Ethereum e permite o *trading* de *tokens*, que representam "gatos virtuais". Cada "gato virtual" tem caraterísticas únicas e apenas um proprietário, que o pode comprar, colecionar, vender, destruir ou reproduzir (em conjunto com um "gato virtual" de género diferente). Na figura 5 pode verificar-se um exemplo de um dos ecrãs da aplicação, onde é apresentada a página individual de um *token*, podendo por exemplo consultar-se as caraterísticas de "gato" e o seu valor de mercado [19].

[7] https://www.coinbase.com/

[8] https://www.coinbase.com/mobile?locale=pt

[9] https://www.cryptokitties.co/

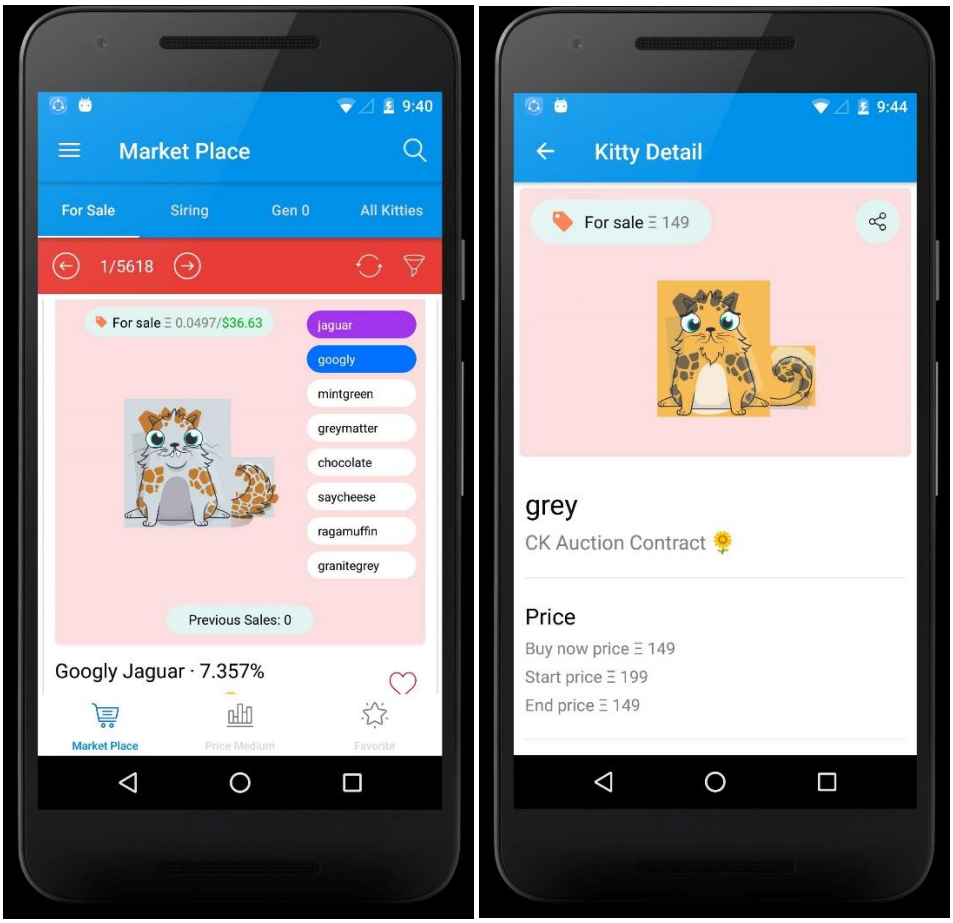


Figura 5 - Ecrãs ilustrativos da aplicação CryptoKitties[10]

Apesar de ter um conceito mais lúdico, acabou por se tornar uma aplicação bastante popular, com um elevado número de utilizadores e transações. Em outubro de 2018, a CryptoKitties alcançou a marca da criação de 1 milhão de "gatos" e 3.2 milhões de transações registadas na *blockchain* [20]. Dependendo das caraterísticas e raridade do "gato" os valores podem também ascender a elevados números, tendo sido já realizadas transações por cerca de 100 mil dólares [19].

3.2 – Desenvolvimento de Aplicações Móveis

As aplicações móveis associadas ao uso dos *smartphones* é sem dúvida uma das maiores tendências tecnológicas nos dias de hoje, daí o grande interesse no seu desenvolvimento por parte das empresas. Estima-se que cerca de 5 mil milhões de pessoas possuem *smartphone* [21], o que leva também a que o uso e descarregamento de

[10] https://apkpure.com/crypto-kitties-icat/com.cryptokitties

aplicações seja elevado. Apenas para o ano de 2018, calcula-se que tenham sido feitos cerca de 194 mil milhões de *downloads* de aplicações [22].

3.2.1 – Sistemas Android e iOS

No que ao mercado de *smartphones* diz respeito, existem vários sistemas operativos disponíveis, embora domínio seja claro para apenas dois deles: Android e iOS, como se pode verificar na figura 6, que apresenta a distribuição de mercado dos sistemas operativos móveis para o ano de 2018, segundo dados da StatsCounter GlobalStats [23].

Devido a esta distribuição, em que duas das plataformas detém mais de 96% do mercado, o desenvolvimento de aplicações móveis por parte das empresas na sua generalidade foca-se apenas em Android e iOS.

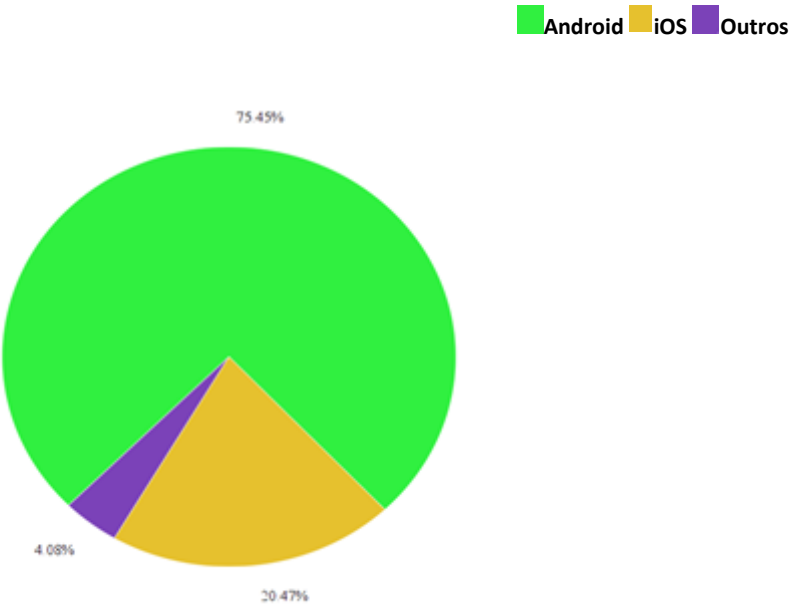


Figura 6 - Distribuição de mercado dos sistemas operativos para dispositivos móveis – Adaptada de [23]

O domínio por parte do Android é justificado pela sua versatilidade no que toca aos dispositivos em que pode correr, sendo utilizado pela maior parte das marcas fabricantes de *smartphones.* Já o iOS, funciona apenas em dispositivos da marca Apple, que apresentam regularmente grandes quantidade de vendas.

Na tabela 3, pode-se verificar uma listagem e comparação de caraterísticas destas duas plataformas.

Tabela 3 - Caraterísticas dos sistemas operativos Android e iOS

Sistemas Operativos Móveis	Android	iOS
Proprietário	Google LLC	Apple Inc.
Loja de aplicações	Google Play Store	App Store
Linguagem de programação	JAVA e/ou Kotlin	Swift e Objective-C
SDK	Android Studio	Xcode
Plataformas de desenvolvimento suportadas	Linux, Windows, Mac OS X	Mac OS X
Página oficial	http://www.android.com/	http://www.apple.com/ios

No que toca aos SDK (*kit* de desenvolvimento de *software*), ambas as plataformas fornecem ao programador uma interface gráfica para elaboração dos ecrãs da aplicação, ferramentas de *debug* e um emulador que permitem testar as aplicações sem necessidade de usar um dispositivo físico. Quanto às plataformas suportadas, o iOS "obriga" os programadores a possuir o sistema operativo Mac OS X para que o desenvolvimento das aplicações possa ser realizado, enquanto que o Android pode ser desenvolvido em qualquer uma das principais plataformas existentes.

3.2.2 – Aplicações nativas, *web* e híbridas

Para o desenvolvimento de aplicações móveis, podem-se seguir diferentes abordagens, que resultam em diferentes tipos de aplicações que podem ser nativas, *web* ou híbridas.

Aplicações nativas

Este tipo de aplicações é desenvolvido numa linguagem de programação exclusiva para um tipo de sistema operativo, como Java para o Android ou Swift para iOS e está presente nas lojas de aplicações das respetivas plataformas. Cada plataforma possui ferramentas e elementos de interface próprios, resultando numa diferença não apenas ao nível do código, mas também estética e comportamental, não havendo garantias de que o que funciona numa plataforma funcionará na outra [24].

Como são programadas em exclusivo para cada sistema operativo, acabam por ser mais rápidas e confiáveis que os outros tipos de aplicação e podem fazer facilmente uso de todos os recursos dos smartphones como o GPS ou a câmera [24].

Aplicações *web*

Uma aplicação *web*, na verdade não é uma aplicação, mas sim um site desenvolvido em exclusivo para *smartphone*, portanto não está presente nas lojas de aplicações. Este site faz um reconhecimento do tipo de dispositivo que acede ao site e adapta-se a este, sendo executado através de um navegador web. São programados com recurso a HTML5, CSS e JavaScript e acabam por ser eficientes em casos de aplicações mais simples no qual o objetivo passa por pouco mais que apresentar conteúdo estático [24].

Aplicações híbridas

Este tipo de aplicações é uma mistura entre aplicações nativas e web, fazendo uso de tecnologias web (HTML5, CSS e JavaScript) para o seu desenvolvimento, acompanhado de uma quantidade reduzida de código nativo, que permite aceder às funcionalidades mais específicas do *smartphone*. Esta parte nativa das aplicações assegura também que esta possam estar presentes nas lojas de aplicações [24].

A aplicação híbrida é mais simples e rápida de desenvolver, visto que a "parte web" da aplicação acaba por ser igual para os diferentes sistemas operativos, mas comparada com uma aplicação nativa acaba por ter uma performance mais lenta [24].

3.2.3 – *Frameworks* de desenvolvimento multiplataforma

Baseadas no conceito das aplicações híbridas, surgiram várias *frameworks* que permitem o desenvolvimento para diferentes plataformas tendo como base apenas uma estrutura de código. Estas introduzem camadas de abstração que fazem um mapeamento entre as *plugins* e código da *framework* com as *plugins* e código nativos das plataformas, sendo que o código da proveniente da *framework* pode ser desenvolvido em linguagens para além de HTML5, CSS e JavaScript [25]. Permitem também aceder diretamente aos emuladores das plataformas e deste modo simular e testar a aplicação em tempo real.

Embora as primeiras versões destas ferramentas tivessem desempenhos irregulares, com o passar do tempo têm vindo a receber constantes aperfeiçoamentos e, consequentemente, a afirmar-se como alternativas cada vez mais válidas.

Entre as várias *frameworks* existentes verifica-se que algumas são criadas e suportadas pelas maiores entidades tecnológicas como o React Native[11] por parte do Facebook, o Flutter[12] por parte da Google ou o Xamarin[13] por parte da Microsoft [26].

[11] https://facebook.github.io/react-native/

[12] https://flutter.dev/

[13] https://dotnet.microsoft.com/apps/xamarin

4 - Aplicação Player Market Cap

4.1 – Funcionalidades da aplicação

O principal propósito da aplicação "Player Market Cap" será a negociação de ativos digitais, ou seja, a execução por parte dos utilizadores de operações de compra e venda destes ativos digitais, baseada em investimentos com a expectativa de obtenção de lucros a partir da sua valorização a curto, médio ou longo prazo. Os ativos digitais são representativos de jogadores de futebol, são geridos a partir de uma *blockchain* e o seu valor é autorregulado tendo em conta mecanismos baseados na lei da procura e da oferta disponibilizados também por parte da *blockchain*.

Além disso, a *blockchain* automatiza as negociações entre utilizadores, emparelhando e executando prontamente ordens de compra com ordens de venda, e vice-versa, que se igualem nos valores transacionais. De realçar que para que uma venda de um ativo (jogador de futebol) se realize será necessária a existência de uma contraordem com valores de compra iguais ou superiores. No caso de não se concretizar no momento, o utilizador pode deixar a ordem em aberto (ativa) até que se consume ou efetuar o seu cancelamento quando desejar.

A aplicação disponibilizará também funcionalidades relativas à gestão de contas de utilizador, à gestão de carteiras na *blockchain* ou à apresentação de detalhes de jogadores

De seguida é apresentada uma lista com os principais requisitos funcionais da aplicação para um melhor entendimento destas funcionalidades.

Requisitos funcionais da aplicação:

1. A aplicação deverá permitir o registo de utilizador através do preenchimento de formulários com os seus dados pessoais;

2. A aplicação deverá permitir ao utilizador definir um login e password para acesso à sua conta;

3. A aplicação deverá permitir e requisitar ao utilizador a validação do email para que possa efetuar login;

4. A aplicação deverá permitir ao utilizador a definição de um pin pessoal para acesso à conta cada vez que minimize ou feche a aplicação;

5. A aplicação deverá permitir ao utilizador, no momento do primeiro login, gerar uma carteira na *blockchain* ou associar uma carteira previamente gerada;

6. A aplicação deverá permitir ao utilizador a recuperação e redefinição da password de acesso à sua conta no caso de esquecimento;

7. A aplicação deverá permitir ao utilizador a alteração dos dados pessoais, incluindo a password e pin criados;

8. A aplicação deverá apresentar ao utilizador uma lista de jogadores (*tokens*), acompanhados de detalhes (como nome ou clube) e respetivas variações de preço (para compra/venda por unidade);

9. A aplicação deverá permitir ao utilizador selecionar jogadores como favoritos e visualizar a lista filtrada dos selecionados;

10. A aplicação deverá apresentar ao utilizador uma lista filtrada com jogadores dos quais possua *tokens* em carteira na *blockchain*;

11. A aplicação deverá permitir ao utilizador a submissão de ordens de compra ou venda de jogadores (*tokens*) na *blockchain* para uma ou mais unidades, de acordo com os preços designados no momento da submissão;

12. A aplicação deverá permitir ao utilizador a gestão das suas ordens de compra e venda submetidas na *blockchain*, permitindo o cancelamento de ordens que ainda não se tenham consumado/exercido;

13. A aplicação deverá apresentar ao utilizador um histórico das ordens que foram exercidas/consumadas na *blockchain*;

14. A aplicação deverá permitir ao utilizador realizar atividades de leasing à *blockchain*;*

15. A aplicação deverá apresentar ao utilizador uma lista dos leases ativos na *blockchain;**

16. A aplicação deverá permitir ao utilizador cancelar um lease a qualquer momento; *

17. A aplicação deverá apresentar ao utilizador uma lista com os lucros resultantes das atividades de leasing realizadas; *

18. A aplicação deverá apresentar ao utilizador o saldo em carteira na *blockchain* (em unidades da criptomoeda em vigor na rede *blockchain*);

19. A aplicação deverá permitir ao utilizador a gestão de valor da sua carteira na blockchain, viabilizando depósitos ou levantamentos; **

20. O sistema deverá designar as variações nos preços de compra e venda dos jogadores (*tokens*) de acordo com os valores aplicados nas ordens mais recentes que tiverem sido executadas;

21. O sistema deverá permitir aos administradores a gestão dos jogadores que são listados ao nível da aplicação, adicionando ou removendo jogadores ou editando informações de jogadores já criados; ***

Notas importantes:

* A atividade de leasing é uma operação concebida em específico a partir da plataforma *blockchain* implementada, que fundamentalmente consiste numa alocação (concessão temporária) de unidades da criptomoeda em vigor na *blockchain* e que resulta em lucros contínuos ao longo do tempo de atividade para os utilizadores. Mais detalhes acerca desta operação são apresentados no ponto 4.3.1.2.

** As funcionalidades de depósito ou levantamento de valor para a *blockchain* envolvem serviços complexos e externos à aplicação que serão desenvolvidos futuramente, fora do âmbito deste estágio. Deste modo este requisito não será concebido nesta versão da aplicação. Para testar algumas das funcionalidades da aplicação será atribuído um saldo "fictício" às contas dos utilizadores, possibilitando a realização de operações;

*** O controlo por parte do administrador sobre os jogadores é apenas feito ao nível da lista apresentada na aplicação, uma vez que depois de criado um *token*, este não poderá ser removido da *blockchain*;

4.2 - Arquitetura do projeto

A arquitetura de projeto da aplicação "Player Market Cap" é baseada no modelo cliente-servidor, que efetua uma comunicação e troca constante de pedidos e respostas entre o cliente e o servidor, de modo a disponibilizar ao cliente o acesso a recursos ou serviços de rede indispensáveis para o funcionamento da aplicação.

Na figura 7 é apresentado um esquema da arquitetura do software composto pelo cliente e o servidor com a representação dos seus principais componentes.

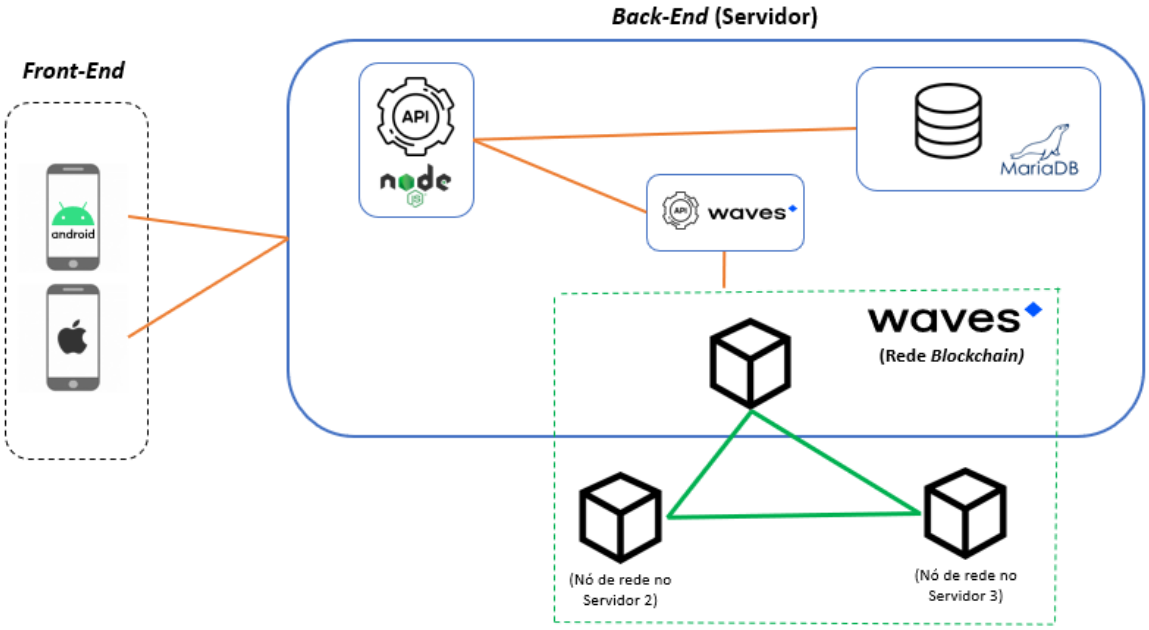


Figura 7 - Esquema da arquitetura do projeto

O cliente consiste numa aplicação móvel, disponível para as plataformas Android e iOS, responsável pela parte de apresentação de informação e interação com o utilizador. É a partir da aplicação móvel que os utilizadores exercem pedidos ao servidor para que se executem as operações pretendidas.

O servidor é responsável por responder aos pedidos por parte da aplicação móvel e executar ou não as operações dos pedidos mediante determinadas circunstâncias. O servidor é composto por 3 componentes:

- API (*Application Programming Interface*) - habilita realização de pedidos por parte da aplicação móvel ao servidor, e deste modo executar operações ao nível da base de dados ou da rede *blockchain*;
- Base de dados - permite o armazenamento de dados importantes para o funcionamento da aplicação;
- Nó de rede *blockchain* – nó principal (de génese), a partir do qual se executam as transações ou consulta de dados na *blockchain*;

A rede *blockchain* é responsável pela lógica de negociação dos ativos (*tokens*), disponibilização de carteiras aos utilizadores para armazenamento dos seus saldos e ativos, entre outras funcionalidades. A *blockchain*, em virtude das suas caraterísticas, está em execução em conjunto com outros nós de rede, implementados noutro servidores, que vão replicando automaticamente todos os dados gerados ao nível do nó principal. Isto permite que a informação monetária da aplicação, correspondente às carteiras dos

utilizadores e respetivos saldos não se percam caso o servidor principal seja comprometido.

4.3. Desenvolvimento do projeto

Neste subcapítulo é feita a descrição das diferentes implementações e configurações realizadas ao longo do desenvolvimento do projeto.

De forma a proporcionar uma melhor compreensão e organização é feita uma divisão entre o desenvolvimento *Back-end,* relacionado com o desenvolvimento do lado do servidor e o desenvolvimento *Front-end*, relacionado com o desenvolvimento do lado do cliente, mais especificamente da aplicação móvel.

Esta divisão não está necessariamente organizada por ordem cronológica, visto que houve fases de desenvolvimento alternadas e/ou paralelas entre as duas vertentes mencionadas, como por exemplo, a fase de criação e estilização da aplicação (desenvolvimento do *front-end*) que antecedeu o desenvolvimento do *back-end* ou a fase de desenvolvimento da API que foi acompanhada por alguns testes ao nível da aplicação.

De salientar ainda que o desenvolvimento de código ao longo do projeto foi sendo acompanhado com versionamento através da tecnologia GitLab[14], permitindo o armazenamento seguro do código produzido, a visualização do histórico do desenvolvimento (com detalhes sobre as alterações feitas desde a primeira versão até à mais recente) e a possibilidade de recuperação e utilização de uma versão do código mais antiga e estável.

4.3.1 - Desenvolvimento *Back-end*

4.3.1.1 - Ambiente de implementação

Inicialmente a implementação do *back-end* foi realizada numa máquina virtual a partir do *software* Virtual Box[15], que permitiu a execução do sistema operativo Ubuntu Server[16].

Nesta máquina virtual foi possível fazer a replicação e configuração do nó de *blockchain* (apresentada no ponto 4.3.1.2) e também realizar os principais testes de

[14] https://about.gitlab.com/

[15] https://www.virtualbox.org

[16] https://ubuntu.com/server

operações na rede com recurso à biblioteca PyWaves[17] uma biblioteca Python composta de métodos especificamente orientados à rede *blockchain* implementada.

Foi também implementada uma base de dados simples, através do uso de SQLite[18], com o intuito de complementar os métodos que a *blockchain* fornece e deste modo criar novas mecânicas para a aplicação posteriormente.

A implementação ao nível da máquina virtual foi importante para a fase inicial, mas revelou algumas limitações de performance na velocidade de processamento e acesso à rede, e por isso, foi feita uma migração da implementação do *back-end* para um servidor.

Na nova implementação, o servidor foi também configurado com o sistema operativo Ubuntu Server, que permitiu uma gestão facilitada, devido a este ser um sistema completo, seguro, estável, gratuito e acompanhado por uma extensa documentação e comunidade para tirar dúvidas que pudessem surgir ao longo das implementações no servidor [27].

A gestão do servidor foi realizada a partir de um terminal local (na máquina pessoal) com linhas de comandos (instruções) do tipo Linux e de uma conexão SSH (*Secure SHell*) que permite um acesso criptografado ao servidor. Para além disso, fez-se uso do *software* Transmit 5, para o envio ou edição de ficheiros ao nível do servidor. Este *software* recorre ao protocolo FTP (*File Transfer Protocol*) para fazer a transmissão de ficheiros entre a máquina local e o servidor ou vice-versa, e tal como se verifica na figura 8 apresenta uma *interface* amigável e sincronizada com os dois ambientes (máquina local do lado esquerdo e servidor do lado direito), permitindo diversas funcionalidades como por exemplo a alteração do nome de ficheiros ou o envio de pastas.

À semelhança da máquina virtual, no servidor também foi feita a replicação e configuração de um nó de *blockchain*, acompanhada da implementação de uma base de dados, bem como a integração de uma API (*Application Programming Interface*), que serão detalhados nos pontos seguintes.

[17] https://github.com/PyWaves/PyWaves/

[18] https://www.sqlite.org/index.html

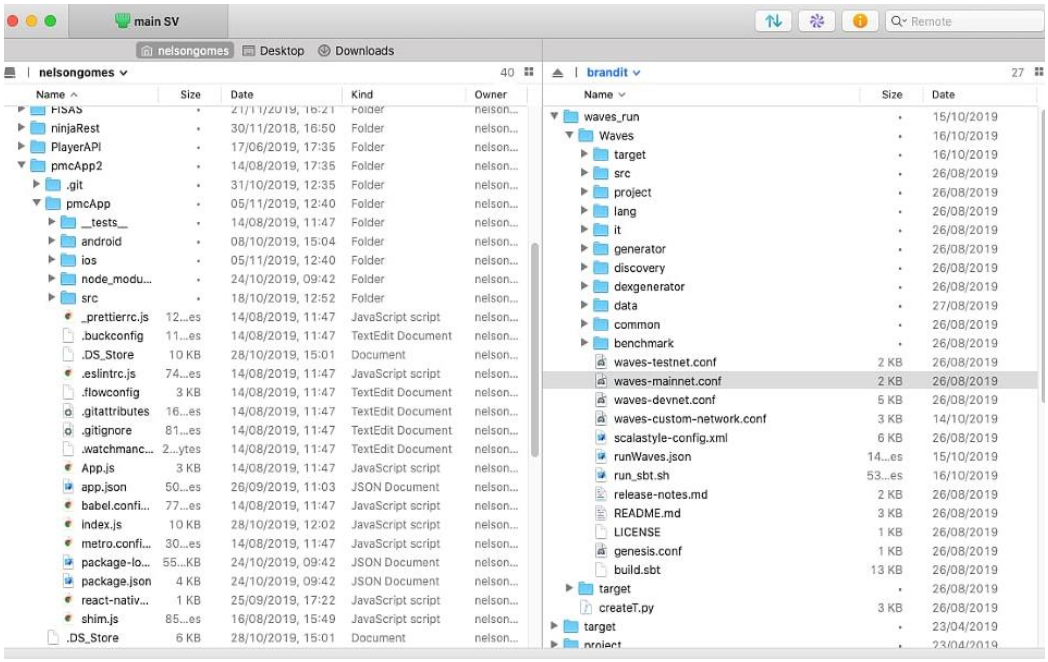


Figura 8 - Interface gráfica do software Transmit 5

4.3.1.2 – Implementação e configuração da *blockchain*

A implementação do nó de *blockchain* começou com a replicação e transferência para o servidor de um conjunto de ficheiros presentes num repositório oficial da plataforma de rede escolhida.

A escolha recaiu sobre a Waves[19], uma plataforma de *blockchain* com caraterísticas inovadoras e que se enquadravam naquilo que era pretendido para a aplicação móvel a desenvolver. Entre estas caraterísticas destacam-se:

- A existência de uma DEX (*Decentralized Exchange*) interna, que entre outras funcionalidades permite a negociação automática de compras e vendas de ativos;
- A diversidade de bibliotecas de suporte para os programadores, disponíveis em diversas linguagens de programação (como Python, JavaScript, C#, Java, Kotlin, etc.);
- Os valores reduzidos de *fees* a pagar nas transações;
- A possibilidade de efetuar *leasing* a um nó de rede;
- A integração do protocolo de rede Waves-NG, que melhora consideravelmente a velocidade de processamento das transações;

[19] https://wavesplatform.com/

- A disponibilidade de uma API por defeito, que permite aceder à rede blockchain para consulta de informações de uma forma simples e eficaz;

No Anexo A podem encontrar-se mais detalhes acerca da plataforma Waves e das suas caraterísticas.

Configuração da blockchain

No passo seguinte, procedeu-se à sua configuração seguindo os passos presentes na documentação oficial da plataforma. A rede *blockchain* definida para implementação é do tipo privado, criada de raiz, e, portanto, todos os procedimentos foram feitos de acordo com o estipulado na documentação oficial da plataforma Waves [28].

Entre estes procedimentos, que envolvem por exemplo a instalação de algum *software*, é de realçar a configuração do ficheiro "waves-custom-network.conf", que é composto por blocos onde se definem várias propriedades da *blockchain*.

Na figura 9 destaca-se a área para definição do tipo de rede aplicando o termo "CUSTOM" no campo "*type",* e a inicialização de carteiras com algum saldo através dos campos "*initial-balance"* e "*transactions"* inseridos na área de configuração denominada de *"genesis"*. A atribuição de saldos às carteiras dos utilizadores foi feita com o propósito de permitir a execução de operações, como por exemplo a criação (*issue*) de novos *tokens* ou submissão de ordens de compra e venda na rede.

```
# Waves node settins
waves {
  # data storage folder
    directory=data

  logging-level = DEBUG

  blockchain {
    type: CUSTOM
    custom {
      address-scheme-character: "W"
      # various parameters of network consensus
      functionality {···
      }
      genesis {
         average-block-delay: 60000ms
         initial-base-target: 153722867
         timestamp: 1554824232446
         block-timestamp: 1554824232446
         signature: "4x38UkHbmb7WP9HWB7rWJh8awV4dEyFyaGgQ5HTWBvbCaLYLPFiGPrY3ocsGDJhYtrN2uAcq9rQujMUzZ7iPJmAX"
         initial-balance: 10000000000000000
         transactions = [
           {recipient: "3PK2xXeAhq3ADCrWx8XeqEJdCEDMmEmMmjK", amount: 2000000000000000},
           {recipient: "3PKyrZcKtQTb6HTh83Ws4R3FCGJCBCfgBTq", amount: 2000000000000000},
           {recipient: "3PPPfz6QFQc9gKUbWCZ7RN6YHkA3J2auvdh", amount: 2000000000000000},
           {recipient: "3P6cJ1kc46iZfv2wt7r6xQQcZdUci2vBczC", amount: 2000000000000000},
           {recipient: "3P6GiXp3Kjm6V16ySeZykgJyJZJhnyFd7vf", amount: 2000000000000000}
         ]
       }
    }
  }
```

Figura 9 - Configuração do tipo de rede e área de genesis

Na figura 10 são apresentadas as áreas dedicadas à configuração do nó de rede e da API respetivamente, sendo possível definir o nome do nó ("*node-name*") e a porta de acesso para o nó de rede, bem como a porta de acesso à API e respetivos métodos.

```
network {
  bind-address = "0.0.0.0"
  port = 6869
  known-peers = []
  node-name = "PlayerMarketCap"
  declared-address = "127.0.0.1:6869"
}
```

```
rest-api {
  enable = yes
  bind-address = "0.0.0.0"
  port = 6863
  cors = yes
}
```

Figura 10 - Configuração de nó de rede e API da blockchain

Na figura 11 é apresentada a configuração relativa à Waves DEX (*matcher*), onde se habilita o seu funcionamento através do campo "*enable"*, e se declara um endereço através do campo "*account"* (de uma conta) que tem a responsabilidade de receber e armazenar todas as ordens de compra/venda submetidas na DEX. O endereço também determina que o local onde são depositados os *fees* das transações executadas pela DEX.

O *matcher* também possui uma API, com porta definida através do campo "*port"*, com os mesmos princípios da API de rede, que permite consultar diversas informações acerca do *order-book.*

```
# Matcher settings
  matcher {
    # Enable/disable matcher
    enable = yes

    # Matcher's account address
    account = "3PK2xXeAhq3ADCrWx8XeqEJdCEDMmEmMmjK"

    # Matcher REST API bind address
    bind-address = "0.0.0.0"

    # Matcher REST API port
    port = 6864

    price-assets = [
      "WAVES"
    ]
  }
```

Figura 11 - Configuração do matcher (Waves DEX) da blockchain

Estando completamente definida a estrutura do ficheiro de configuração de acordo com o pretendido, a rede é executada e "montada" com o comando sbt "node/run

waves-custom-network.conf" através do terminal da máquina em que o nó será implementado (no servidor). A execução bem-sucedida do comando sbt, irá gerar *logs* sucessivos acerca da geração de microblocos na rede a cada 5 segundos, conforme a lógica implementada pelo protocolo Waves-NG (figura 12).

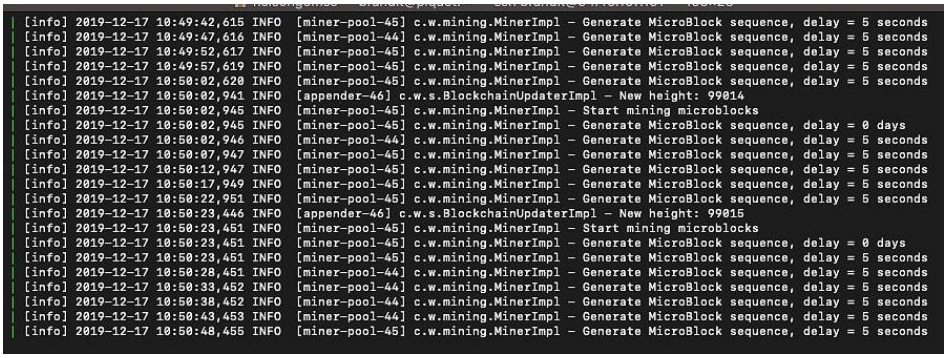


Figura 12 - Nó de rede blockchain em execução

Durante a configuração da *blockchain*, são ainda apresentadas informações relativas às contas geradas (endereço da conta, a *seed* e *seed phrase*, a chave pública e a chave privada), como representado na figura 13 para duas contas. Estas informações são críticas e devem ser guardadas de forma segura para efeitos de consulta e uso das carteiras criadas.

Figura 13 - Informações relativas às contas geradas na rede

De salientar ainda que para se adicionar novos nós à rede implementada, o processo de configuração na nova máquina deverá ser semelhante ao detalhado acima, tendo como diferenças a inclusão do endereço de rede do nó gerador da rede no campo "*known-peers*" apresentado na figura 10 e a exclusão da área de configuração "*genesis*" presente na figura 9.

Realização de testes e operações na *blockchain*

Com o nó em execução e para que fosse possível executar operações com as contas criadas para o efeito, procedeu-se à instalação da biblioteca PyWaves[20], que fornece uma interface para o terminal, a partir da qual se podem executar comandos diretamente à rede *blockchain.* Esta biblioteca pode também ser importada e executada num ficheiro Python, disponibilizando a mesma sintaxe de código e respetivas funcionalidades.

Com a PyWaves foi possível executar diversos testes e atividades de gestão de rede, como por exemplo a geração de novos endereços na rede, a criação de novos *tokens* (jogadores), a execução de ordens de compra e venda de *tokens* ou a execução de *lease* de Waves à rede.

Alguns exemplos destas operações acompanhadas do respetivo código necessário para a sua execução podem ser encontrados no Anexo B.

***Fees* e conceção do mecanismo de leasing**

Dado que a rede a implementar é do tipo privada, existirá um controlo sobre os nós que poderão juntar-se à rede e executar a validação e mineração das transações. O princípio será de que estes nós de rede pertençam à empresa, e, deste modo todo o retorno das *fees* seja assegurado pela mesma entidade.

A partir desta lógica foi decidido que as *fees* aglomeradas em toda a rede *blockchain* pela empresa, seriam para distribuir pelos utilizadores que executem operações de *lease* aos nós de rede, sendo estas distribuídas de acordo com a quantidade que os utilizadores alocaram a esta atividade num determinado período de tempo.

Esta ideia surgiu com o intuito de fornecer um método de incentivo à monetização da rede (depósitos de valor na *blockchain*) por parte dos utilizadores.

4.3.1.3 - Implementação da base de dados

A implementação de uma base de dados surgiu da necessidade de complementar a *blockchain*, possibilitando armazenar e apresentar informações personalizadas relativas às contas dos utilizadores da aplicação "Player Market Cap", aos *tokens* dos jogadores de futebol ou aos lucros provenientes das operações de *leasing*.

[20] https://github.com/PyWaves/PyWaves

Para esta implementação fez-se uso do *software* MariaDB[21], um SGBD (Sistema de Gestão de Base de Dados) do tipo relacional, dos mais populares do mercado.

A escolha deveu-se a diversas vantagens que esta tecnologia oferece, entre as quais se destacam:

- A escalabilidade facilitada, pois permite replicação de dados entre servidores;
- O bom desempenho em servidor com distribuições do tipo Linux (caso do Ubuntu Server);
- Baixa quantidade de bugs e falhas, muito devido à sua entrega atualizações de segurança regulares;
- Ser gratuita;
- O uso de PL/SQL (*Procedural Language/Structured Query Language*), que possibilita o desenvolvimento de *queries* que são armazenadas, compiladas e executadas ao nível do servidor onde a base de dados está implementada;

Na figura 14 é apresentada uma esquematização da base de dados criada, sendo esta composta pelas seguintes tabelas e finalidades:

Users – dedicada ao armazenamento de dados relativos às contas de utilizadores da aplicação móvel como o *email*, nome ou *password.* É composta também pelos campos "secretToken" e "isValid" relativos à confirmação e validação de email da conta e pelo campo "wavesAddress" que servem de referência à carteira do utilizador na *blockchain*;

Players – dedicada a armazenar dados adicionais de um jogador(*token*) a apresentar na aplicação móvel, como o nome, posição, clube ou nacionalidade. É também constituída pelo campo "address" referente ao endereço do *token* na blockchain;

favStatus – possibilita que um utilizador defina um jogador como parte da sua lista de favoritos;

leaseCalcs – armazena diariamente os endereços de utilizador com *leasing* ativo e a quantidade alocada à operação nesse mesmo dia;

profits – armazena informações sobre os endereços e quantidade de Waves que estes receberam após a operação de distribuição das *fees* de um determinado período;

[21] https://mariadb.org/

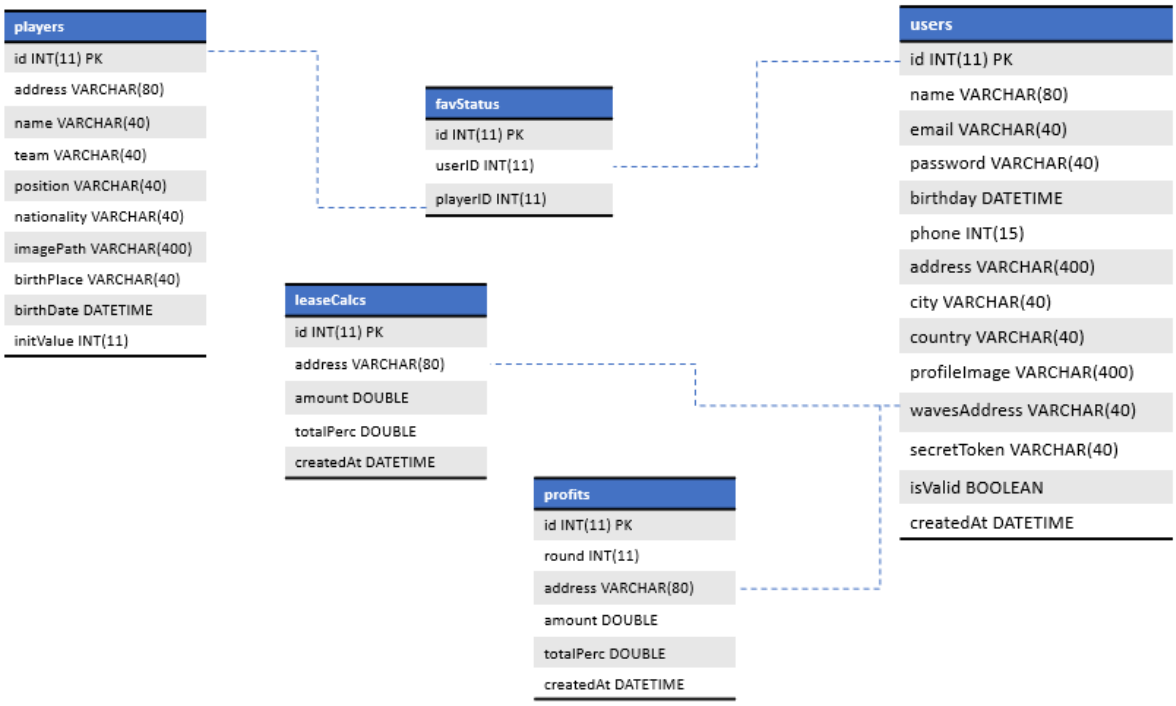


Figura 14 - Esquematização das tabelas criadas na base de dados

Como complemento à implementação da base de dados fez-se uso do *software* SequelPro[22], que efetua uma conexão SSH ao servidor, permitindo funcionalidades como a rápida consulta do conteúdo e estrutura de cada uma das tabelas da base de dados ou a execução de *queries*, através de uma interface gráfica simples e intuitiva, tal como se pode observar na figura 15.

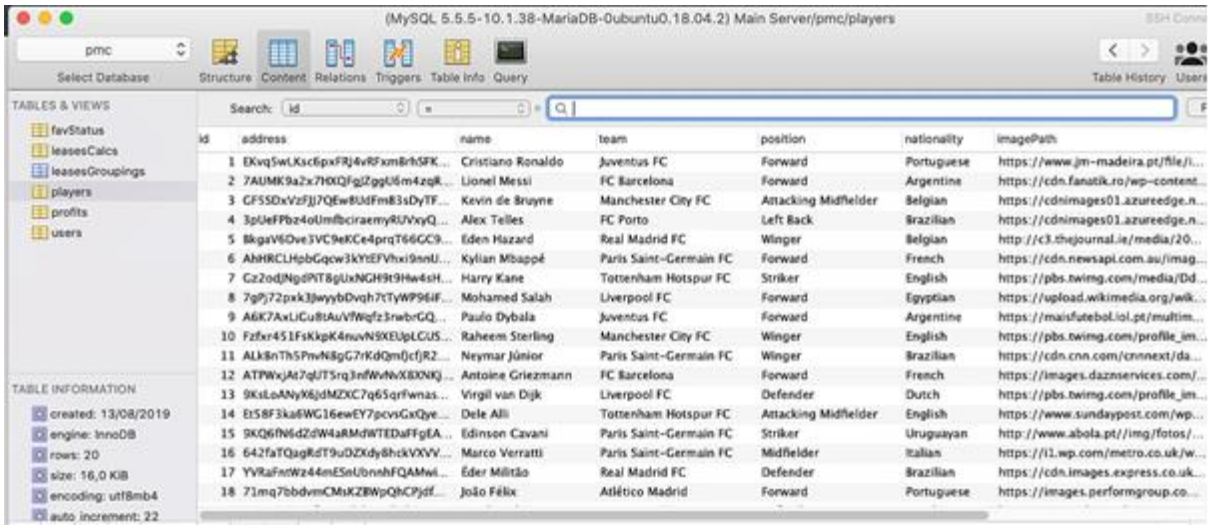


Figura 15 - Interface gráfica do software SequelPro

[22] https://www.sequelpro.com/

4.3.1.4 – Desenvolvimento da API

A conceção da API surgiu com o intuito de estabelecer comunicação entre a aplicação móvel e o servidor, mais especificamente entre a aplicação móvel e a base de dados ou a *blockchain*. Tal como visto, a *blockchain* Waves, possui uma API por defeito que é bastante útil para a realização de testes, mas como para certos pedidos os dados não são retornados para a aplicação móvel no formato e condições pretendidas, faz-se o seu tratamento ao nível desta nova API.

Esta API do tipo RESTful (*Representational State Transfer*), tem como base pedidos HTTP para executar operações CRUD (*Create, Read, Update, Delete*) que correspondem à criação, leitura, atualização/edição e remoção de dados respetivamente.

Na tabela 4 é apresentada uma correspondência entre os métodos de requisição padrão do protocolo HTTP e a operações CRUD que se espera que estes realizem, bem como o *response code* expectável por parte do servidor aos pedidos, que depende do sucesso ou insucesso da operação.

Para o método HTTP GET, no caso de a operação ser executada com sucesso, o *response code* é acompanhado por um *response body*, que é composto pelos dados "lidos" a partir do servidor, geralmente em formato JSON (*JavaScript Object Notation*) ou XML (*Extensible Markup Language*).

Tabela 4 - Métodos de requisição HTTP (Adaptado de https://restfulapi.net/http-methods/#get)

Método de requisição HTTP	**Operação CRUD**	**Response Code**	
POST	*Create*	201 (*Created*)	--
GET	*Read*	200 (Ok)	404 (*Not Found*)
PUT/PATCH	*Update*	200 (Ok) ou 204(*No Content*)	404 (*Not Found)*
DELETE	*Delete*	200 (Ok)	404 (*Not Found*)

A escolha da tecnologia para desenvolvimento da API recaiu sobre Node.js[23], que é um interpretador (*runtime*) de JavaScript que possibilita o processamento, renderização e execução de programas em sistemas que não suportam esta linguagem de programação nativamente [29].

Desta forma, aplicações com base em Node.js podem ser executadas em múltiplos sistemas operativos como OS X, Microsoft Windows ou Linux, permitindo aos programadores desenvolver aplicações em JavaScript diretamente do lado do servidor.

A possibilidade de usar JavaScript como linguagem de programação ao nível de *back-end,* paralelamente ao desenvolvimento *front-end* nesta mesma linguagem foi um dos principais motivos para a escolha de Node.js como plataforma para desenvolvimento da API.

De realçar que o Node.js tem ainda proveitos como a sua eficiência de execução e a grande variedade de recursos (pacotes de *software*) disponíveis para *download* e uso, presentes no NPM[24] (*Node Package Manager*), um repositório *online* para publicação de projetos de código aberto Node.js [29].

No anexo C, são apresentados mais alguns detalhes sobre o funcionamento e vantagens da tecnologia Node.js.

De forma a auxiliar o desenvolvimento e execução da API fez-se uso de diversos pacotes de *software* entre os quais se destacam: o Express.js, o Sequelize, o PM2, o Nodemailer, o bcrypt e o Randomstring.

Em seguida é feita uma descrição de cada um destes pacotes de *software* e da sua utilidade para o projeto.

Express.js[25]

O Express.js é uma *framework* para Node.js que simplifica o desenvolvimento de aplicações *web.* Com o uso desta *framework,* foi possível definir todos os métodos a invocar por parte da aplicação e disponibilizá-los para consumo *web* com a criação de um servidor, também a partir do Express.js.

Na figura 16 é apresentada a estrutura de código de um método simples, que efetua uma pesquisa (leitura de dados) à base dados. No exemplo apresentado, inicialmente é

[23] https://nodejs.org/en/

[24] https://www.npmjs.com

[25] https://expressjs.com/

invocada a função “get” da *framework*, que representa o tipo de método HTTP a executar. Esta função recebe como parâmetros um “URL endpoint” (necessário para a distinção de outros métodos que estejam presentes no código) e os campos “req” e “res” que representam respetivamente as estruturas da pesquisa e da resposta, e sob os quais serão executadas as operações pretendidas para o método dentro da função.

```
// 1. get lista de players
app.get("/players", (req,res) => {
    Player.findAll().then(players => {
        res.send(players);
    }).catch((err) => {
        return err
    })
})
```

Figura 16 - Definição de método com o Express.js

Após a definição de todos os métodos pretendidos, também foi incluído ao nível do código uma instrução semelhante à da figura 17, que representa a execução de um *web server* a partir do Express.js, viabilizando assim o uso destes métodos em ambiente externo ao servidor onde a API está implementada.

Para que tal seja possível é apenas necessário invocar a função “listen” desta *framework, que* para este caso, leva como parâmetro único o número da porta pela qual se deve efetuar a comunicação com o servidor.

```
// ================ LISTENER ================ //
// ========================================== //

app.listen(8087, function(){
    console.log("Servidor a rodar na porta 8087");
});
```

Figura 17 - Execução de um web server a partir do Express.js

A partir do momento em que este servidor estiver em execução é possível aceder e fazer uso do método apresentado na figura 16, através de uma URL com o seguinte padrão: **http://nomeDoDomínio:8087/players**, sendo que o nome do domínio poderá ser também o IP do servidor de implementação.

De forma a melhor organizar os métodos da API, foi feita uma divisão por 2 ficheiros de configuração, cada um em execução numa porta diferente. Na tabela 5 são apresentados os métodos destinados à gestão de dados dos jogadores e *tokens* respetivos,

sendo que o destino destes pedidos será a *blockchain* ou a base de dados, mais especificamente a tabela "Players".

Já na tabela 6, os métodos apresentados relacionam-se com a gestão das contas dos utilizadores e das suas operações na rede *blockchain*, tendo estes como destino a *blockchain* ou a base de dados.

Tabela 5 - Métodos para gestão de jogadores (tokens)

URL endpoint	Parâmetros	Método HTTP	Destino do pedido	Descrição
/players	---	GET	Base de dados	Retorna lista total de jogadores e as suas caraterísticas
/players	email (utilizador)	GET	Base de dados	Retorna lista de jogadores acompanhado do estado "isFav" para um utilizador
/setFav	email (utilizador) id (jogador)	POST	Base de dados	Define jogador como parte da lista de favoritos
/unsetFav	email (utilizador) id (jogador)	DELETE	Base de dados	Remove jogador da lista de favoritos
/player	id (jogador)	GET	Base de dados	Retorna caraterísticas individuais de um jogador
/tokenBalance	address (conta de utilizador) address (*token* do jogador)	GET	Blockchain	Retorna o saldo específico de um jogador para um determinado utilizador
/tokenPrice	address (*token* do jogador)	GET	Blockchain ou Base de dados	Retorna preços para compra e venda de um jogador (*token*)

Tabela 6 - Métodos para gestão de utilizadores

URL endpoint	Parâmetros	Método HTTP	Destino do pedido	Descrição
/checkEmail	email (utilizador)	GET	Base de dados	Verifica se email já está registado (retorna *true* ou *false*)
/login	email password	GET	Base de dados	Validação de login (retorna *true* ou *false*)
/regist	name email password hash* birthday phone address city country secretToken*	POST	Base de dados	Regista utilizador na base de dados e envia email de confirmação de conta
/resend	email (utilizador) secretToken*	PUT	Base de dados	Reenvia email de confirmação
/validate	secretToken*	PUT	Base de dados	Valida email para determinada conta
/getUser	email (utilizador)	GET	Base de dados	Retorna dados de conta de um determinado utilizador
/updateUser	email (utilizador) name birthday city country address phone	PUT	Base de dados	Atualiza dados de conta de um determinado utilizador
/resetPass	email (utilizador)	GET	Base de dados	Envia email para proceder com alteração da password
/changePass	email, password hash* password hash*	PUT	Base de dados	Executa alteração da password

/getProfit	address (conta de utilizador)	GET	Base de dados	Retorna lista de profits para conta de utilizador
/getLeasings	address (conta de utilizador)	GET	Blockchain	Retorna lista de leasings de um utilizador
/getBalance	address (conta de utilizador)	GET	Blockchain	Retorna saldo (em Waves) em carteira para um determinado utilizador
/getOrders	address (conta de utilizador)	GET	Blockchain	Retorna a lista de todas as ordens compra/venda submetidas por um determinado utilizador
/cancelOrder	id (ordem na blockchain)	POST	Blockchain	Cancela ordem ativa na blockchain

*****Parâmetro gerado automaticamente ao nível do método requisitado

Tal como se pode constatar, os métodos que tem como destino a *blockchain* tem como intuito apenas a consulta de informações, uma vez que para a execução de operações, como por exemplo a submissão de uma ordem de compra, pode ser feita diretamente a partir da aplicação móvel, recorrendo a bibliotecas de *sofware* para o desenvolvimento *front*-end, detalhadas no próximo subcapítulo (4.3.2).

Estes métodos com consulta de informações da *blokchain* basicamente são compostos por requisições à API nativa da plataforma Waves e fazem o tratamento das respostas a essas requisições, retornando à aplicação móvel informação customizada de acordo com os objetivos da aplicação.

Sequelize[26]

O Sequelize é um ORM (*Object-Relational Mapper*) para Node.js que suporta bases de dados PostgreSQL, MariaDD, MySQL, SQLite e MSSQL e que é responsável pelo mapeamento dos dados relacionais (tabelas, colunas e linhas da base de dados) para objetos JavaScript, sendo que a execução de operações CRUD na base de dados com recurso a este módulo é realizada de forma bastante simples [30].

[26] https://sequelize.org/

Para fazer uso deste ORM foi necessário configurar um ficheiro de conexão à base de dados, que recebe parâmetros como o nome da base de dados a que se pretende aceder, um *username* e *password* do servidor, o tipo de base dados (neste caso MariaDB) ou a hora local.

Depois disto foi configurado um ficheiro de mapeamento para cada uma das tabelas existentes, como representado na figura 18, para um mapeamento à tabela "Profits". Este ficheiro inclui uma invocação do ficheiro de conexão, acompanhado pelo método "*define*", no qual devem ser listados todos os campos da tabela e o tipo de variável definido.

```
const db = require('./db');

// definição dos tipos de dados  da bd
const Profit = db.sequelize.define('profits', {
    address: {
        type: db.Sequelize.STRING
    },
    amount: {
        type: db.Sequelize.DOUBLE
    },
    totalPerc: {
        type: db.Sequelize.DOUBLE
    }
},
)

module.exports = Profit
```

Figura 18 - Ficheiro de mapeamento para tabela "Profits"

Os módulos criados a partir destes ficheiros de mapeamento podem ser importados para estruturas de código e, deste modo o Sequelize disponibiliza os métodos HTTP para manipulação de dados das respetivas tabelas.

Na figura 19 é apresentada a estrutura de código para a consulta de todos os jogadores presentes na base de dados. Para tal o módulo "Player" que representa a tabela "Players" é invocado e faz uso da função "findAll" (requisição do tipo GET) para retornar a lista de dados em formato JSON ou uma mensagem de erro em caso de haver alguma falha.

```
Player.findAll().then(players => {
    res.send(players);
}).catch((err) => {
    return err
})
```

Figura 19 - Uso de módulo Sequelize para consulta de dados

PM2 [27]

O PM2 é um gestor de processos para Node.js que permite a gestão de aplicações em ambiente de produção, garantindo a sua execução 24/7, sendo que uma vez feita a execução de uma aplicação através deste programa e até ordem em contrário, o PM2 faz um controlo sobre o estado e disponibilidade das aplicações e reinicia-as automaticamente em caso de necessidade.

Este *software* pode ser executado pelo terminal de comandos e para que uma aplicação se associe ao seu ambiente apenas é necessário executar a seguinte instrução:

"pm2 start AppFile.js"

✓ Sendo "AppFile.js" substituída pelo nome do ficheiro a executar

Na figura 20 está representada a lista de programas em Node.js em execução no servidor, num determinado instante, sendo apresentados detalhes sobre o seu estado de funcionamento e gasto de recursos computacionais que cada um acarreta.

Name	id	mode	status	↺	cpu	memory
API_OWN_RATES	1	fork	online	0	0.2%	78.0 MB
API_PMC_PLAYERS	2	fork	online	90	0.4%	102.3 MB
API_PMC_USERS	9	fork	online	97	0.4%	77.5 MB
BinanceExchange	4	fork	online	29	0.4%	110.5 MB
ExchangesEmailAlerts	6	fork	online	0	0.2%	56.9 MB
PoloniexExchange	3	fork	online	8	0.5%	106.3 MB
calcOwnRates	8	fork	online	0	0.4%	78.0 MB
coinBaseAPI	7	fork	online	0	0.4%	79.4 MB
forexAPI	10	fork	online	1	0.2%	98.2 MB
marketPrices	0	fork	online	0	0.7%	77.4 MB
totalWallet	5	fork	online	0	0.4%	57.5 MB

Use `pm2 show <id|name>` to get more details about an app

Figura 20 - Aplicações Node.js em execução através do PM2

[27] https://pm2.keymetrics.io/

bcrypt[28]

A bcrypt é uma biblioteca que suporta a encriptação de *passwords* ao nível da base de dados, permitindo o seu armazenamento criptografado e impossível de decifrar e a validação de *logins*, comparando a *password* inserida com a versão criptografada da base de dados.

A encriptação de uma *password* é bastante simples, sendo apenas necessário invocar a função "*hash"* da biblioteca, que a recebe como parâmetro.

Para se efetuar a validação de uma password no login*,* faz-se uso da função "*compare*", que faz parte da biblioteca e que recebe como parâmetros a password a testar e a "*hash"* retornada da base de dados. Esta função retorna "*true"* no caso de a *password* ser válida ou *"false"* no caso de não o ser e permite que se definam ações dependendo do tipo de resposta.

Nodemailer[29] & RandomString[30]

O Nodemailer é um módulo para envio gratuito de emails através de aplicações Node.js. A sua configuração exige como parâmetros um *email* emissor e respetiva *password (*que na maior parte dos casos não é a *password* real, mas sim uma representação pré-definida pelos serviços de email). O Nodemailer, ao nível desta aplicação teve utilidade para efetuar a validação de conta (*email*) dos utilizadores.

Por seu lado, a biblioteca RandomString, tal como indica o seu nome tem como finalidade gerar uma *string* aleatória. Com o uso desta biblioteca em conjunto com o Nodemailer foi possível definir um mecanismo para a validação das contas de utilizadores, apresentado na figura 21.

Para validar a conta, e posteriormente fazer se habilitar a fazer *login*, o utilizador deverá aceder à conta do email com que se registou e clicar no *link* disponibilizado por uma mensagem automaticamente enviada para esse email no momento do registo.

Tendo efetuado esta validação nas 24 horas procedentes ao registo o utilizador será reencaminhado para uma página HTML com uma mensagem referente ao sucesso

[28] https://www.npmjs.com/package/bcrypt

[29] https://www.nodemailer.com

[30] https://www.npmjs.com/package/randomstring

da operação. Se houver alguma falha, será reencaminhado para uma página HTML referente ao erro e terá a possibilidade de gerar um novo *link* de validação.

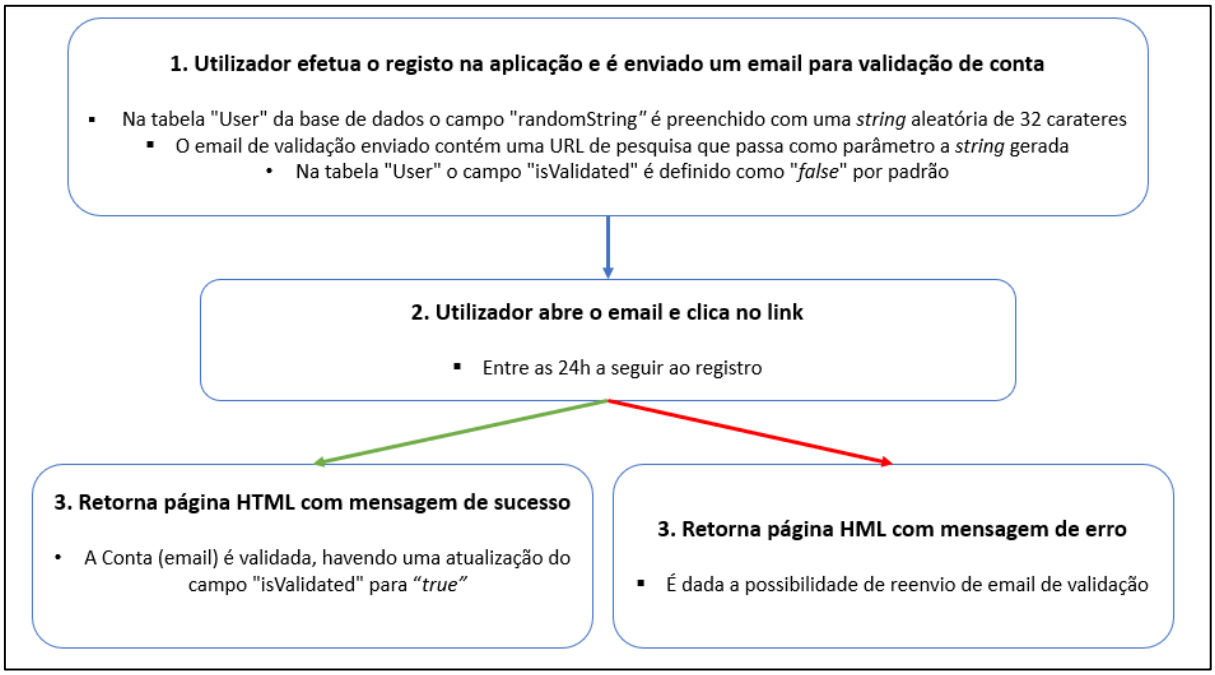


Figura 21 - Mecanismo para validação de conta de utilizador

Ao longo do desenvolvimento da API, foram sendo realizados diversos testes através do *software* Postman[31], que permite simular todos os tipos de pedidos do protocolo HTTP, com possibilidade de passagem de parâmetros e apresentação dos resultados dos pedidos no formato especificado.

Na figura 22 é apresentada a interface deste *software*, com a apresentação dos resultados de um pedido em formato JSON, sendo possível verificar os resultados e propriedades que um pedido retorna.

[31] https://www.getpostman.com/

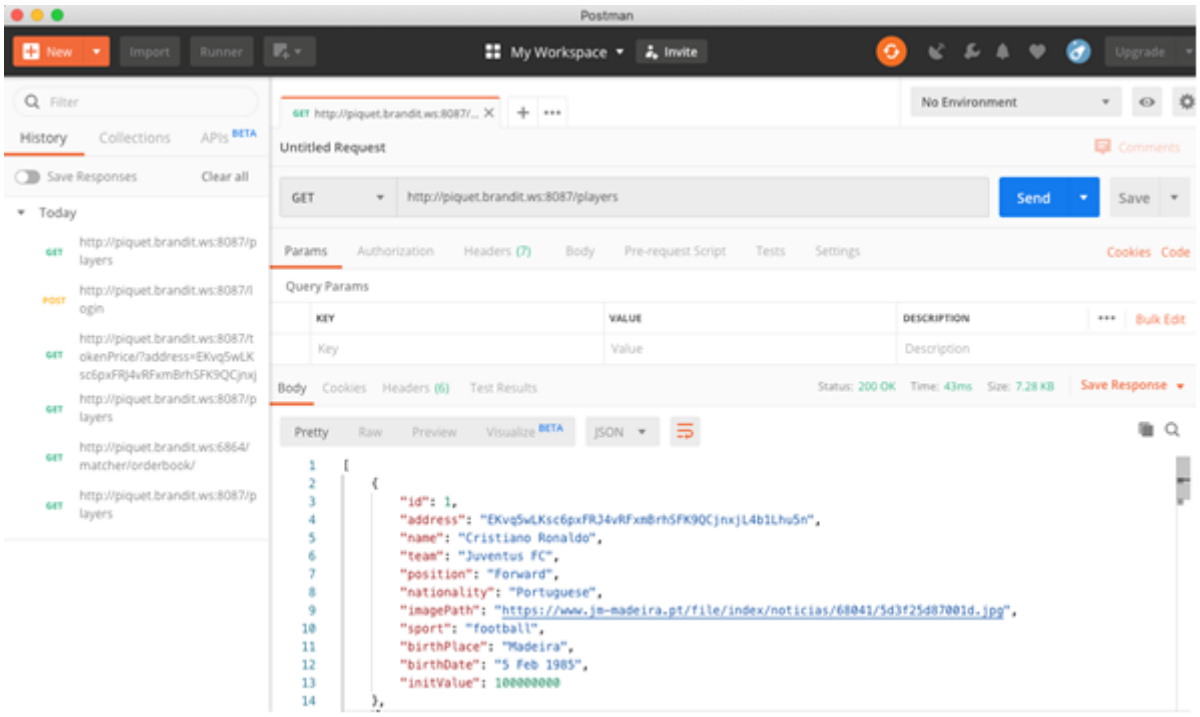


Figura 22 - Interface gráfica do software Postman

Configuração da comunicação da API com HTTPS

De forma a providenciar uma comunicação segura com o servidor, fez-se a implementação de uma "extensão" adicional de segurança, denominada HTTPS (*Hyper Text Transfer Protocol Secur*e) [31].

Esta extensão tem como base o protocolo HTTP e a emissão e verificação de certificados do protocolo SSL/TLS (*Secure Sockets Layer/Transport Layer Security*) para comprovar que um método/URL é fidedigno.

Resumidamente o objetivo do SSL/TLS é tornar segura a transmissão de informações sensíveis como dados pessoais (como por exemplo os de *login*), surgindo como alternativa à transferência de dados em "texto simples" uma vez que criptografa esses dados e os torna mais difíceis intercetar e decifrar por parte de *hackers* [31].

Para se proceder à implementação do HTTPS, foi definida uma "autoridade de certificação" para a emissão de certificados SSL/TLS, tendo esta escolha recaído sobre a Let's Encrypt[32], que fornece esta emissão de forma gratuita e pouco complexa [32].

Foi também necessário proceder à instalação de 2 *softwares*, referenciados de seguida e com uma breve descrição dos seus propósitos.

[32] https://letsencrypt.org/

Nginx [33]**–** serve como um *proxy* reverso através do qual os pedidos à API são endereçados e através do qual é realizada a configuração e a implementação do HTTPS;

Certbot [34]- ferramenta de *software* gratuita e "*open source*" que permite a obtenção e gestão dos certificados Let's Encrypt, procedendo ao carregamento da configuração do Nginx e fazendo uma renovação automática dos certificados a cada 60 dias, uma vez que a cada emissão, os certificados Let's Encrypt possuem um limite de validade de cerca de 90 dias.

Ao nível da configuração do Nginx seguiu-se documentação oficial [33], realçando-se a edição de um ficheiro pertencente a este *software* no qual foi necessário apontar o domínio do servidor através do campo "server_name" e definir os *endpoints* do *proxy* com estruturas de código semelhantes à da figura 23 para cada um dos métodos da API.

```
server {
    server_name nomeDoDomínio;
      ...
      location /players {
        proxy_pass http://localhost:porta/players;
      }
...
```

Figura 23 - Exemplo de estrutura de código da configuração do Nginx

Para concluir a implementação do HTTPS executou-se no terminal de comandos a instrução "sudo certbot --**nginx** -d nomeDomínio", que permite a execução do Certbot com as configurações do Nginx e a ativação dos protocolos SSL/TLS com os certificados Let's Encrypt.

A partir desse momento, o acesso aos métodos da API através de HTTPS pode ser feito com URLs semelhantes à seguinte: https://nomeDomínio/endpointProxy.

[33] ttps://www.nginx.com/

[34] https://certbot.eff.org/

4.3.2 - Desenvolvimento *Front-end*

Relativamente à parte do front-end do desenvolvimento do projeto, procedeu-se à elaboração da aplicação móvel para os sistemas Android e iOS, seguindo uma abordagem de desenvolvimento multiplataforma através do uso de React Native[35], uma *framework* para desenvolvimento de aplicações móveis criada pelo Facebook que tem como base a popular biblioteca de desenvolvimento *web* React.

O React Native proporciona todas as vantagens inerentes ao desenvolvimento multiplataforma, como a reutilização de código, uma vez que toda a estrutura de código desenvolvida foi feita em JavaScript e desenvolvida simultaneamente para as plataformas Android e iOS, sendo posteriormente convertida para linguagem nativa dos sistemas operativos correspondentes [34]. Outra grande virtude desta *framework* é a funcionalidade "*Fast Refresh*", que permite um *debug* do código desenvolvido de forma bastante rápida, e deste modo visualizar ao nível da aplicação (em execução no emulador/simulador) todas as alterações realizadas no código de forma imediata [35], tal como se pode verificar na figura 24, em que o desenvolvimento de código é acompanhado por uma emulação de um dispositivo "iPhone Xs" em tempo real.

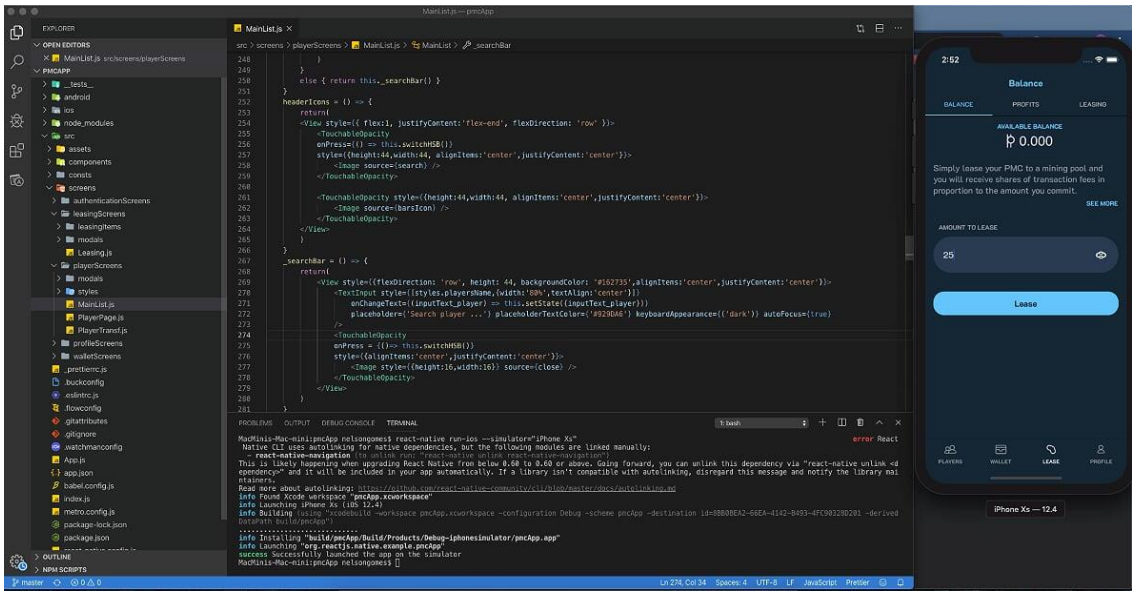


Figura 24 - Desenvolvimento de código acompanhado de emulação em tempo real

O desenvolvimento foi realizado numa máquina com sistema operativo MacOS, permitindo o uso do *software* Xcode[36], um IDE (*Integrated Development Environment*)

[35] https://facebook.github.io/react-native/

[36] https://developer.apple.com/xcode/

exclusivo para plataformas deste tipo que é composto por vários recursos e ferramentas para o desenvolvimento e gestão de projetos relacionados com sistemas operativos Mac. Permitiu também o uso do *software* Android Studio[37], com ferramentas e recursos semelhantes ao Xcode, mas para projetos e aplicações do tipo Android.

Seguindo a documentação oficial [36] iniciou-se o projeto da aplicação móvel, resultando na estrutura apresentada na figura 25. Nesta estrutura a maior parte dos ficheiros são inerentes à *framework* React Native, destacando-se as pastas "ios" e "android" com ficheiros e configurações necessários para que a aplicação se execute nestas plataformas e a pasta "node_modules" onde são armazenadas e instaladas todas as dependências do projeto, como por exemplo bibliotecas externas. Já na pasta "src" está incluída toda a estrutura de código desenvolvida, com os ficheiros correspondentes a cada ecrã da aplicação, bem como as imagens e ícones necessários para a aplicação.

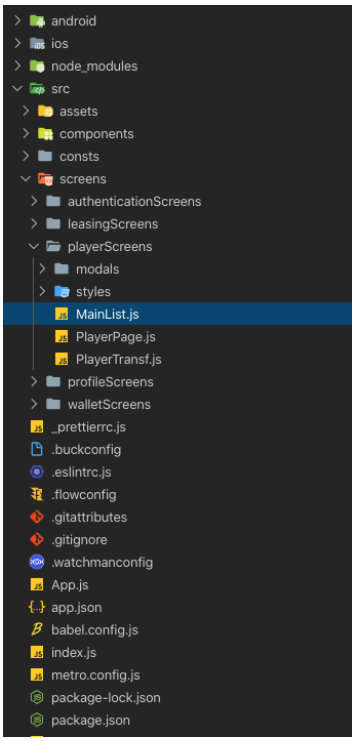


Figura 25 - Estrutura de projeto da aplicação móvel

A cada ecrã da aplicação corresponde um ficheiro JavaScript. Durante o desenvolvimento destes ficheiros procurou-se que a estrutura de código fosse o mais semelhante possível, sendo que em cada ficheiro a organização foi segmentada com base nos seguintes tipos de elementos:

[37] https://developer.android.com/studio/

- *Imports* (invocações) de módulos, componentes, bibliotecas ou ficheiros necessários para o desenvolvimento da lógica do ecrã;
- Inicialização de variáveis/constantes do ficheiro (*props* e *state*);
- Funções/Eventos – funções que executam operações ou apresentam itens no ecrã de acordo com determinadas circunstâncias;
- Estruturação da página/ecrã – com os diferentes tipos de componentes;
- Estilos - podem ser feitos no mesmo ficheiro ou em ficheiro externo;

4.3.2.1 - Estruturação de página/ecrã e componentes

A construção de um ecrã é baseada em componentes, que são nada mais que os elementos de interface do React Native que posteriormente são diretamente convertidos para os elementos nativos das plataformas. A estruturação dos componentes assemelha-se à estrutura de uma página HTML, com uma hierarquização bem definida. Na tabela 7 são apresentadas as componentes básicas que foram recorrentemente utilizadas em comparação com as *tags* normalmente utilizadas em HTML.

Tabela 7 - Componentes do React Native e equivalência de tags HTML

<View>	
<Text>	<label>, <title>
<ListView>	,
<Button>	<button>
<Image>	<img>
<TextInput>	<input>

Outro tipo de componente a destacar é o **<ScrollView>**, que permite que o utilizador faça Scroll e visualize conteúdos que vão para além dos limites do ecrã.

4.3.2.2 - Estilos

A estilização foi feita de acordo com o *design* elaborado pelos colaboradores da Brandit e os *designs* para os diferentes ecrãs foram disponibilizados em ficheiros de formato Sketch[38]. A partir destes ficheiros foi possível inspecionar os estilos e elementos dos ecrãs com o uso do *software* InVision[39], copiar a sintaxe CSS (*Cascading Style Sheets*) correspondente e ainda fazer *download* de ícones ou imagens específicas a incluir na aplicação. Ao nível do React Native os princípios na criação dos estilos são bastante semelhantes a CSS para a *web*, diferenciando-se apenas pelo uso de Camel Case na referenciação dos nomes (por exemplo com backgroundColor em vez do habitual background-color).

Ao nível da estilização realça-se o uso e configuração da propriedade *flex*, que permite que a aplicação se torne responsiva e ajuste automaticamente tamanhos e espaçamentos entre as diferentes componentes.

4.3.2.3 - *Props* e *State*

Os *props* e o *state* são respetivamente constantes e variáveis para controlo das componentes dos ecrãs e da informação neles disponibilizada. Os *props* são herdados de ecrãs anteriores (*parents*) e tem um valor constante durante o ciclo de vida de um ecrã. O *state* pode ser inicializado com um determinado valor ou com valor nulo e poderá sofrer alterações ao longo da utilização de um ecrã por parte de um utilizador. A título de exemplo, temos o ecrã de login, que tem as propriedades de *email* e *password* que correspondem a um *state* (variáveis) com nome idêntico e que são inicializadas como nulas. Estas propriedades e respetivo *state* serão alteradas através do preenchimento dos respetivos campos no ecrã, e quando o utilizador carrega no botão para efetuar a validação do login fará uso delas.

4.3.2.4 – Programação funcional da aplicação

Cada ficheiro é também composto por diversas funções que podem ser destinadas à renderização condicional do ecrã ou à execução de eventos.

As funções destinadas à renderização condicional têm o fim de apresentar componentes ou uma estilização específica tendo em conta as *props* ou o *state* do ficheiro.

[38] https://www.sketch.com/

[39] https://www.invisionapp.com/

Por exemplo, na renderização de uma lista de jogadores (figura 26), cada jogador é constituído pela propriedade "isFav" que representa se este pertence à lista de favoritos do utilizador. Neste caso a função de renderização verifica o estado (*state*) desta propriedade para cada jogador e caso este esteja na lista de favoritos apresenta uma estrela com preenchimento. Caso não esteja na lista apresenta uma estrela sem preenchimento.

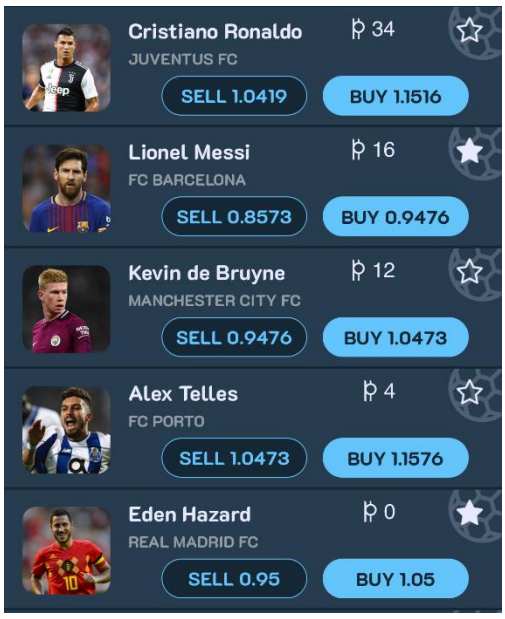


Figura 26 - Exemplo de renderização condicional num ecrã da aplicação

Já as funções direcionadas a responder a eventos, podem executar diversos tipos de operações como por exemplo a alteração de estado de uma variável, a navegação para outro ecrã ou a submissão de dados para o lado do servidor. No exemplo da figura 26, este tipo de funções pode ser executado quando um utilizador pressiona o botão "Sell", navegando para outra página ou quando pressiona um ícone com formato de estrela, adicionando ou retirando um jogador da sua lista de favoritos (executado no lado do servidor) e consequentemente alterando o próprio estado (*state*) da propriedade "isFav". Esta alteração resulta numa nova renderização, na qual o preenchimento da estrela será também alterado.

De realçar também, que para cada ecrã são feitas sucessivas pesquisas ao servidor com pequenos intervalos de tempo. Estas pesquisas são habilitadas através de funções que fazem uso dos métodos disponibilizados pela API desenvolvida para consultar e apresentar no ecrã dados relativos por exemplo a variações nos preços dos *tokens* dos jogadores ou a alterações da propriedade "isFav" referenciada nos exemplos apresentados.

4.3.2.5 – Pacotes de *software* utilizados

À semelhança do Node.js, o React Native pode utilizar o NPM como gestor de módulos e para fazer a instalação de pacotes de *software* úteis ao desenvolvimento da aplicação, entre os quais se destacam o "react-native-navigation[40]", o "react-native-keychain[41]"e o "waves-transactions[42]".

react-native-navigation

O "react-native-navigation" é uma biblioteca que habilita a navegação nativa para iOS e Android entre os diferentes ecrãs, que pode incluir passagem de parâmetros específicos. A sua configuração é bastante simples, requerendo apenas a criação de um ficheiro no qual são instanciados os ficheiros correspondentes a cada ecrã. Para executar a navegação basta fazer uso das funções inerentes a esta biblioteca, entre as quais se realçam as seguintes:

- "**Push"** – permite a navegação para um determinado ecrã (deve incluir-se como parâmetro a instância do ficheiro correspondente);
- "**PopTo**" – navega para o ecrã anterior (leva como parâmetro uma propriedade referente ao ecrã anterior);
- "**PopToRoot**" – volta para a página inicial da aplicação;

waves-transactions

A "waves-transactions" é uma biblioteca desenvolvida em específico para execução de transações em plataformas de *blockchain* Waves, sem a necessidade de recorrer aos métodos criados na API. Esta biblioteca permite uma comunicação segura e direta à *blockchain* e a execução de operações através da invocação das suas funções. Estas funções recebem diversos parâmetros para que possam ser executadas com sucesso, entre os quais se destacam o endereço de rede da *blockchain*, o endereço do *matcher*/Waves DEX e a *seed* da carteira Waves do utilizador na *blockchain*.

Na tabela 8 são apresentadas as funções da biblioteca de que se faz uso ao nível da aplicação móvel e os parâmetros que estas recebem.

[40] https://www.npmjs.com/package/react-native-navigation

[41] https://www.npmjs.com/package/react-native-keychain

[42] https://www.npmjs.com/package/@waves/waves-transactions

Tabela 8 - Funções da biblioteca waves-transactions

Função	Parâmetros recebidos
submitOrder	Endereço do **matcher** (Waves DEX); ***Seed*** da carteira do utilizador; **amountAsset** – endereço do *token* (correspondente a um jogador) a transacionar; **amount** – quantidade de *tokens* a transacionar; **price** – preço por unidade (*token* de jogador) em *tokens* Waves a transacionar; **orderType** – identificação do tipo de ordem a submeter: compra (“buy”) ou venda (“sell”);
cancelOrder	Endereço do **matcher** (Waves DEX); **OrderID;** ***Seed*** da carteira do utilizador;
lease	**Endereço de rede** da *blockchain;* ***Seed*** da carteira do utilizador; **amount** – número de unidades (*tokens*) a transacionar; **recipient** – endereço que irá receber o lease (por defeito será correspondente ao *matcher*);

De realçar ainda que esta biblioteca disponibiliza métodos para gestão de carteiras na *blockchain* diretamente a partir da aplicação móvel. Com o uso do método “seedUtils.seed.create” é possível fazer a criação de uma nova carteira na *blockchain*, retornando e apresentando ao utilizador a respetiva *seed* (de 15 palavras), que deve ser guardada de forma segura. Através da *seed* é feita uma associação localmente (ao nível do *smartphone*) entre a carteira na *blockchain* e a conta do utilizador na aplicação PMC (“Player Market Cap”).

react-native-keychain

O "react-native-keychain" é uma biblioteca que possibilita o armazenamento local encriptado (ao nível do *smartphone*) de diversos dados como o *login*s e *passwords*, fazendo uso dos sistemas de gestão Keychain para o iOS e KeyStore para o Android.

O propósito do uso desta biblioteca foi o armazenamento do *email*, da *password*, da *seed* da carteira Waves e de um *pin* de utilizador para acesso à aplicação. Este armazenamento destes dados visa facilitar a experiência do utilizador a diferentes níveis. A criação e armazenamento de um *pin* (de 4 dígitos) permite que cada vez que o utilizador minimize ou feche a aplicação possa aceder novamente sem que tenha de introduzir novamente os dados de login (*email* e *password*), mas sim o *pin*. Por sua vez, o armazenamento da *seed*, permite que esta seja introduzida ou gerada apenas uma vez (no momento do primeiro *login*) e encriptada, sendo posteriormente consultada e usada de forma segura através do uso de funções da biblioteca "react-native-keychain".

5 - Resultados

Após dado como concluído o desenvolvimento da aplicação, que incluiu a aprovação de todo o desenvolvimento tanto a nível *front-end* como *back-end* por parte do gestor de projeto, procedeu-se à geração dos ficheiros para a instalação da aplicação para ambas as plataformas: APK (Android Package) para Android e .ipa (iOS App Store Package) para iOS. Após a instalação verificou-se que a aplicação executou de igual forma em ambas as plataformas e que todas as funcionalidades têm um desempenho em conformidade com o que era pretendido para a aplicação.

De seguida são apresentados alguns dos ecrãs da aplicação "Player Market Cap" desenvolvidos, complementados pelo(s) requisitos funcionais a que dão resposta e uma pequena descrição.

Requisitos: 1. A aplicação deverá permitir o registo de utilizador através do preenchimento de formulários com os seus dados pessoais; 2. A aplicação deverá permitir ao utilizador definir um login e password para acesso à sua conta; 3. A aplicação deverá permitir e requisitar ao utilizador a validação do email para que possa efetuar login;
Descrição: Na figura 27 pode verificar-se o ecrã de login, que requer o *email* e *password* para acesso à aplicação e através do qual é possível fazer uma recuperação de *password* (envia email de recuperação). Na figura 28 são apresentados os ecrãs de registo com respetivos campos para preenchimento. Através do 2º ecrã de registo é possível abrir uma lista com os termos e condições da aplicação, que o utilizador deve validar para que possa proceder.

Figura 27 - Ecrã de login

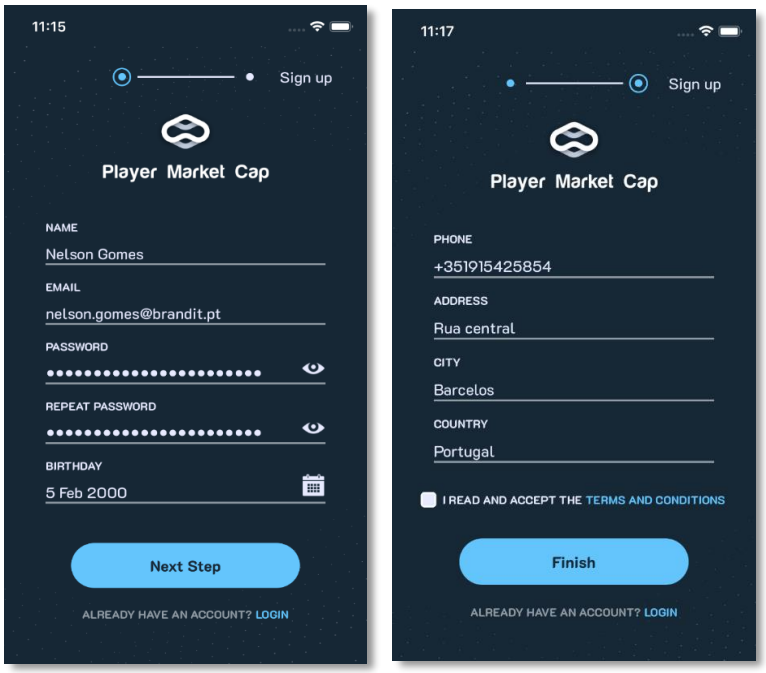


Figura 28 - Ecrãs de registo

Requisito:

4. A aplicação deverá permitir ao utilizador a definição de um pin pessoal para acesso à conta cada vez que minimize ou feche a aplicação;

Descrição: No momento do primeiro *login*, o utilizador pode definir como alternativa de autenticação a inserção de um pin de 4 dígitos, que se realiza através dos ecrãs exibidos na figura 29.

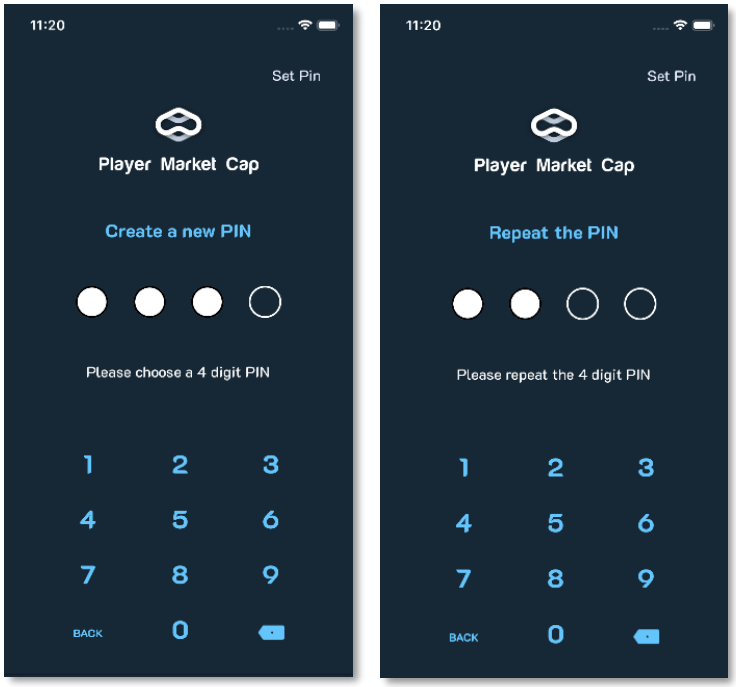


Figura 29 - Definição de PIN pessoal

Requisitos:

5. A aplicação deverá permitir ao utilizador, no momento do primeiro login, gerar uma carteira na *blockchain* ou associar uma carteira previamente gerada;

Descrição: Também no momento do primeiro *login* no dispositivo, existe uma fase relativa à configuração da carteira na *blockchain* (figura 30), que pode ser gerada de raiz ou por associação no caso de o utilizador já possuir uma carteira previamente criada que pretenda recuperar. No caso de o utilizador gerar uma carteira, é-lhe retornada a uma *seed* de 15 palavras, representativa da nova carteira, que ele deve guardar de forma segura (figura 31). No caso de o utilizador pretender associar uma carteira, deve introduzir a *seed* de 15 palavras, que se pressupõe que tenha guardado anteriormente (figura 32).

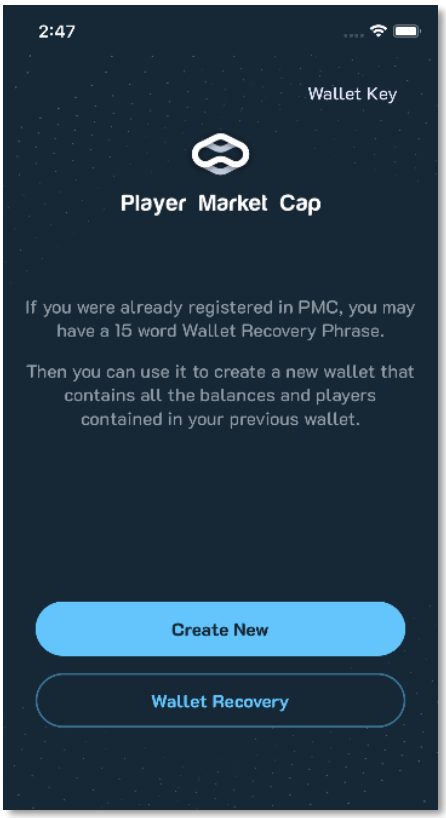


Figura 30 - Configuração de carteira

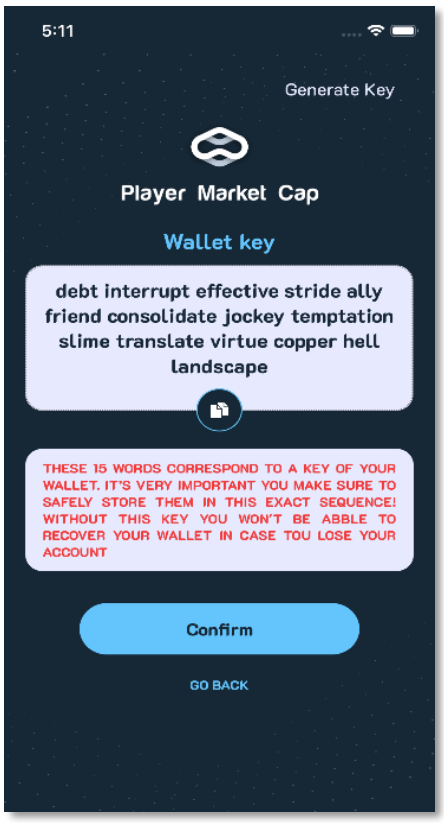


Figura 31 – Ecrã gerador de carteira

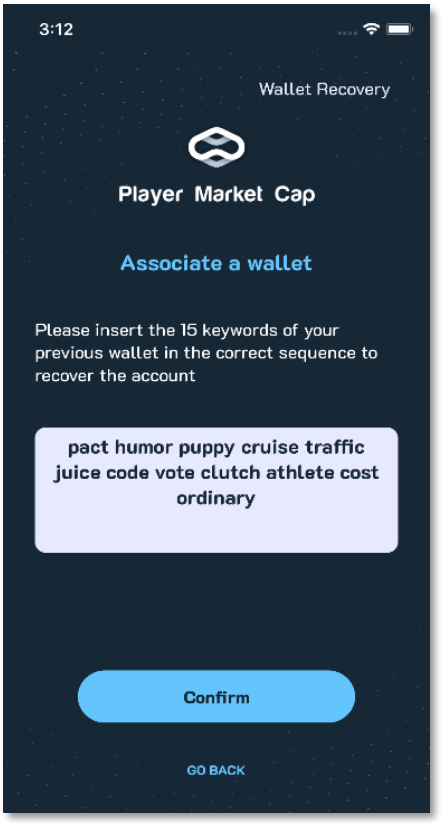


Figura 32 – Ecrã para recuperação de carteira

Requisitos:

6. A aplicação deverá permitir ao utilizador a recuperação e redefinição da password de acesso à sua conta no caso de esquecimento;

7. A aplicação deverá permitir ao utilizador a alteração dos dados pessoais, incluindo a password e pin criados;

18. A aplicação deverá apresentar ao utilizador o saldo em carteira na *blockchain* (em unidades da criptomoeda em vigor na rede *blockchain*);

Descrição: Na figura 33 são apresentados ecrãs relativos ao perfil e gestão de conta de utilizador, onde são apresentados os seus dados pessoais e se pode fazer realizar a sua edição. Pode verificar-se também que é apresentado o saldo em carteira, com uma taxa de atualização constante.

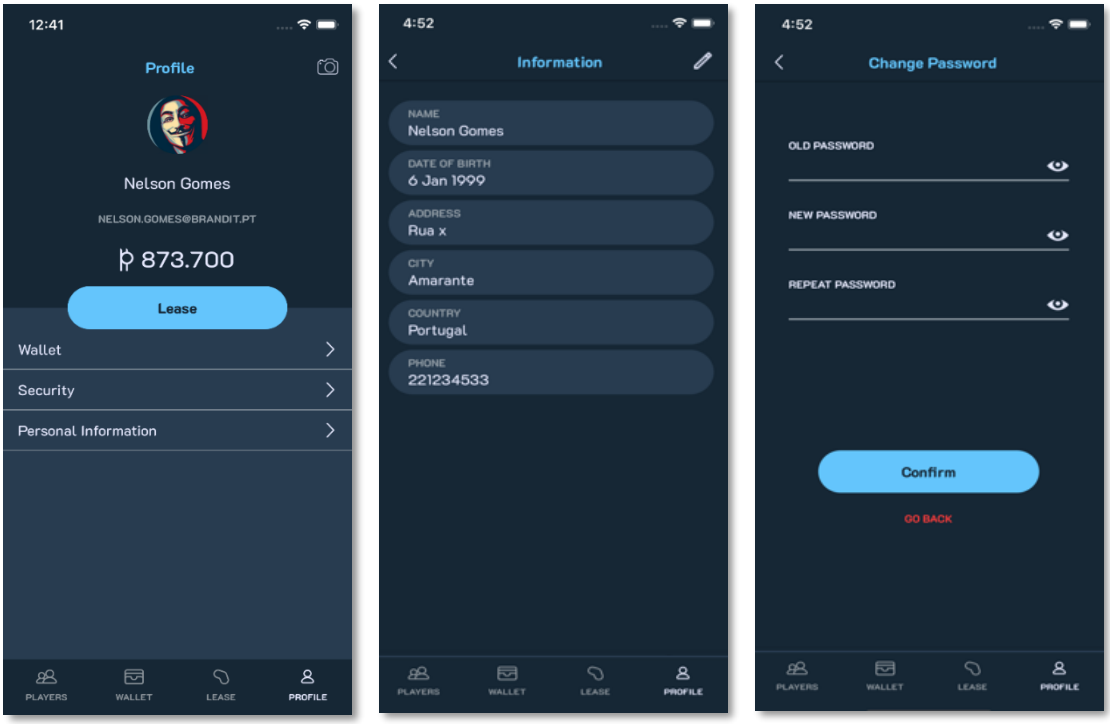
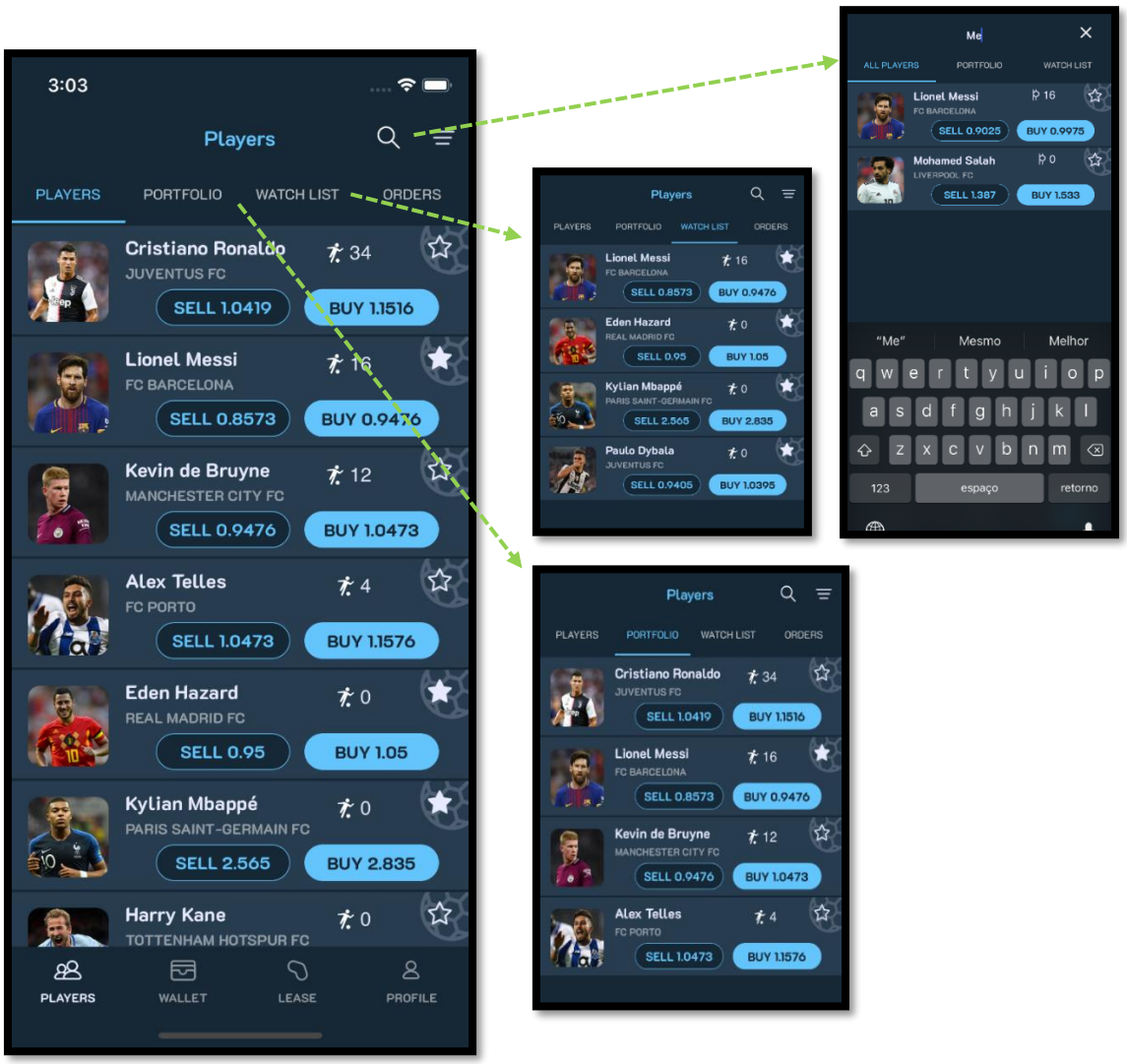


Figura 33 - Ecrãs para gestão de conta e edição de dados

Requisitos:

8. A aplicação deverá apresentar ao utilizador uma lista de jogadores (*tokens*), acompanhados de detalhes (como nome ou clube) e respetivas variações de preço (para compra/venda por unidade);

9. A aplicação deverá permitir ao utilizador selecionar jogadores como favoritos e visualizar a lista filtrada dos selecionados;

10. A aplicação deverá apresentar ao utilizador uma lista filtrada com jogadores dos quais possua *tokens* em carteira na *blockchain*;
Descrição: Na figura 34 é apresentado o ecrã principal relativo à listagem dos jogadores, que são acompanhados dos preços de compra e venda e número de unidades em carteira (que são regularmente atualizados de modo a conferir alterações de mercado). Ao nível deste ecrã é possível fazer uma filtragem dos dados, pressionando os diferentes campos da barra supeior do ecrã. Deste modo pode-se apresentar uma lista com jogadores dos quais o utilizador possui ativos (*tokens*) em carteira, uma lista de jogadores favoritos (a funcionalidade de adicionar/remover favoritos é realizada pressionando o ícone em forma de estrela do jogador pretendido) ou uma lista resultante da pesquisa por nome.
 *Figura 34 - Listagem de jogadores e respetivos filtros*

Requisitos:

11. A aplicação deverá permitir ao utilizador a submissão de ordens de compra ou venda de jogadores (*tokens*) na *blockchain* para uma ou mais unidades, de acordo com os preços designados no momento da submissão;

18. A aplicação deverá apresentar ao utilizador o saldo em carteira na *blockchain* (em unidades da criptomoeda em vigor na rede *blockchain*);

Descrição: A partir da lista de jogadores é possível abrir a página individual de um jogador (figura 35 à esquerda), na qual são apresentados mais detalhes pessoais (como a nacionalidade, a data e local de nascimento ou o pé preferido). É também apresentado um gráfico relativo às variações do seu valor nos últimos dias (ainda com valores redundantes devido a não haver grandes movimentações). Nos ecrãs destinados à compra ou venda dos ativos (*tokens* do jogador) são exibidas informações atualizadas relativas aos preços de compra e venda, saldo em carteira e *tokens* do jogador em carteira, de modo a que o utilizador possa fazer uma gestão do que pretende transacionar. Existem ainda campos para inserção, por parte do utilizador, dos montantes que pretende transacionar, que refletem momentaneamente os custos ou lucros estimados para a transação (figura 35 à direita).

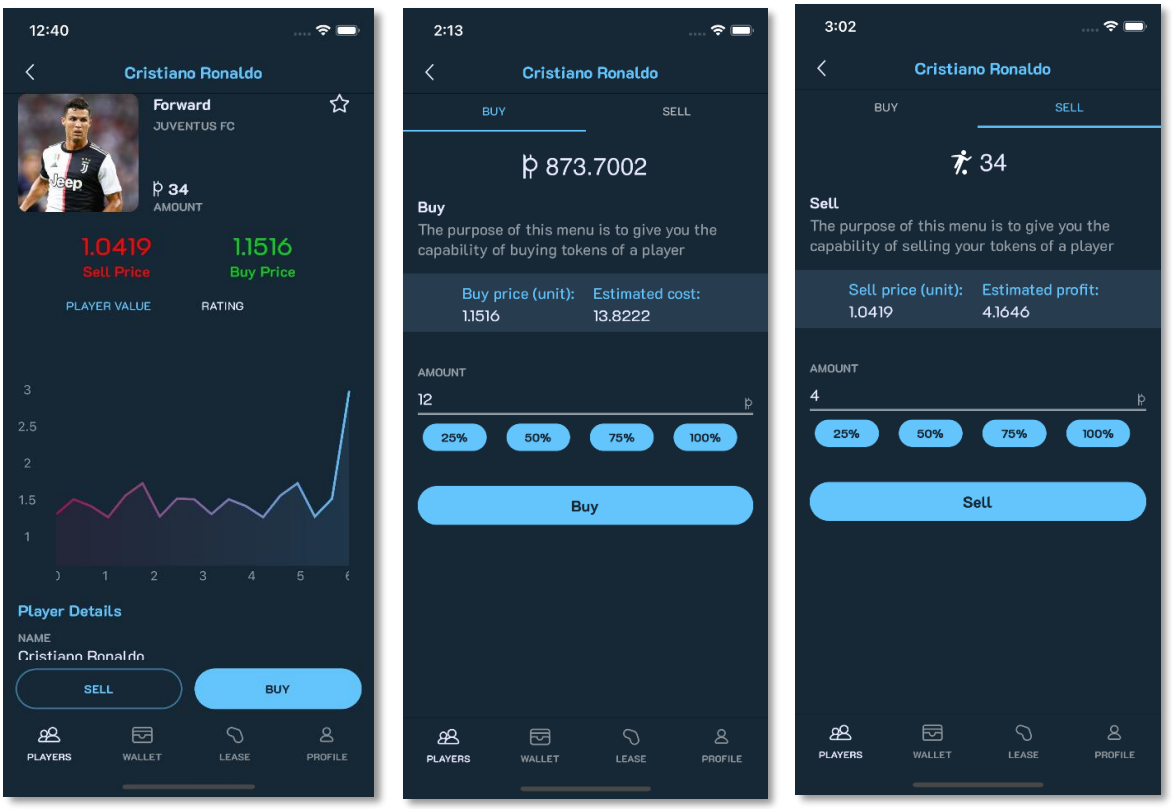


Figura 35 - Ecrãs para gestão de compra e venda de ativos de um jogador

Requisitos:

12. A aplicação deverá permitir ao utilizador a gestão das suas ordens de compra e venda submetidas na *blockchain*, permitindo o cancelamento de ordens que ainda não se tenham consumado/exercido;

13. A aplicação deverá apresentar ao utilizador um histórico das ordens que foram exercidas/consumadas na *blockchain*;

Descrição: Agregada à listagem dos jogadores, existe também uma área onde é possível fazer uma gestão das ordens de compra e venda submetidas, à qual se acede a partir da barra superior. Na figura 36, pode-se observar uma lista das ordens ativas na *blockchain*, com informações relativas ao ativo (*token*) e aos valores em negociação. É também disponibilizada a opção de cancelamento destas mesmas ordens. Também se pode analisar dados relativos ao histórico de ordens que já foram consumadas e que assim sendo já não se encontram ativas.

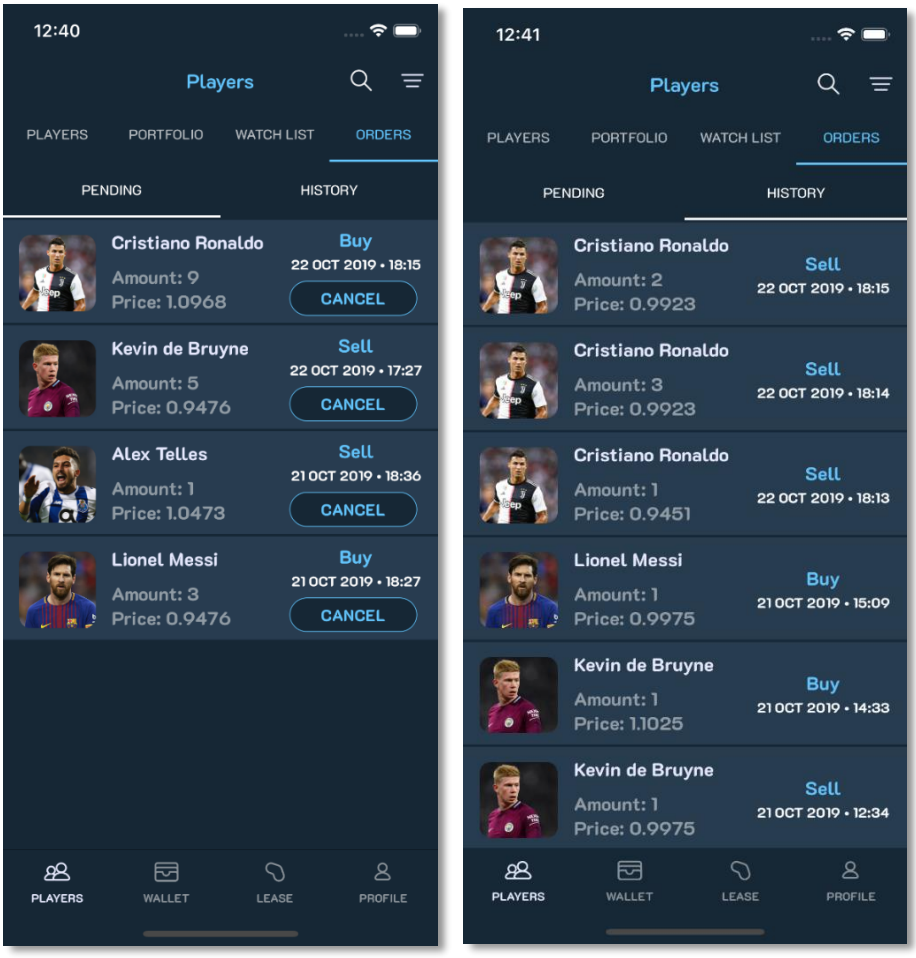


Figura 36 - Ecrãs para gestão de ordens submetidas na blockchain

Requisitos:

14. A aplicação deverá permitir ao utilizador realizar atividades de leasing à *blockchain*;

15. A aplicação deverá apresentar ao utilizador uma lista dos leases ativos na *blockchain;*

16. A aplicação deverá permitir ao utilizador cancelar um lease a qualquer momento;

17. A aplicação deverá apresentar ao utilizador uma lista com os lucros resultantes das atividades de leasing realizadas;

18. A aplicação deverá apresentar ao utilizador o saldo em carteira na *blockchain* (em unidades da criptomoeda em vigor na rede *blockchain*);

Descrição: Na figura 37 é apresentado o ecrã através do qual o utilizador insere um montante e deste modo pode executar uma operação de lease à rede. A partir deste ecrã é também possível abrir uma lista de detalhes relativos à operação.

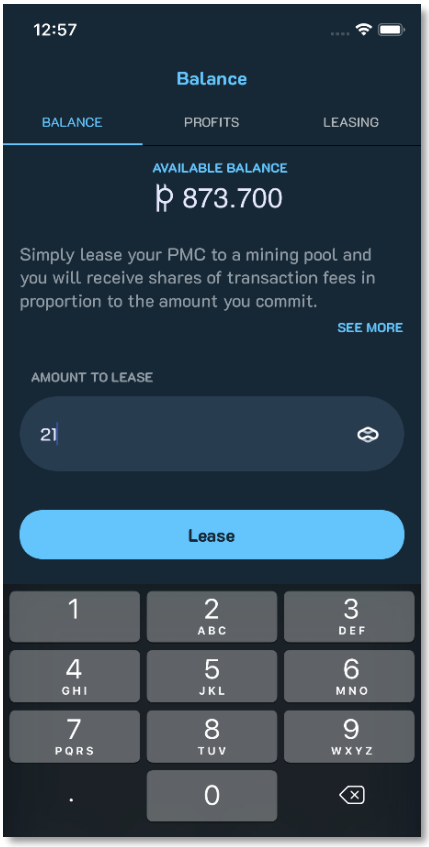


Figura 37 - Ecrã para execução de operação de lease

Requisitos: 16. A aplicação deverá permitir ao utilizador cancelar um lease a qualquer momento; 17. A aplicação deverá apresentar ao utilizador uma lista com os lucros resultantes das atividades de leasing realizadas;
Descrição: Na figura 38 são apresentadas as listas relativas ao processo de leasing. No 1º ecrã é apresentada a lista dos leases ativos, com detalhes relativos ao momento da operação, ao valor e a possibilidade de cancelamento imediato de cada uma das atividades de lease. No 2º ecrã são apresentados os lucros recebidos ao longo do tempo de atividade, com detalhes relativos à data e hora da transferência e ao valor recebido.

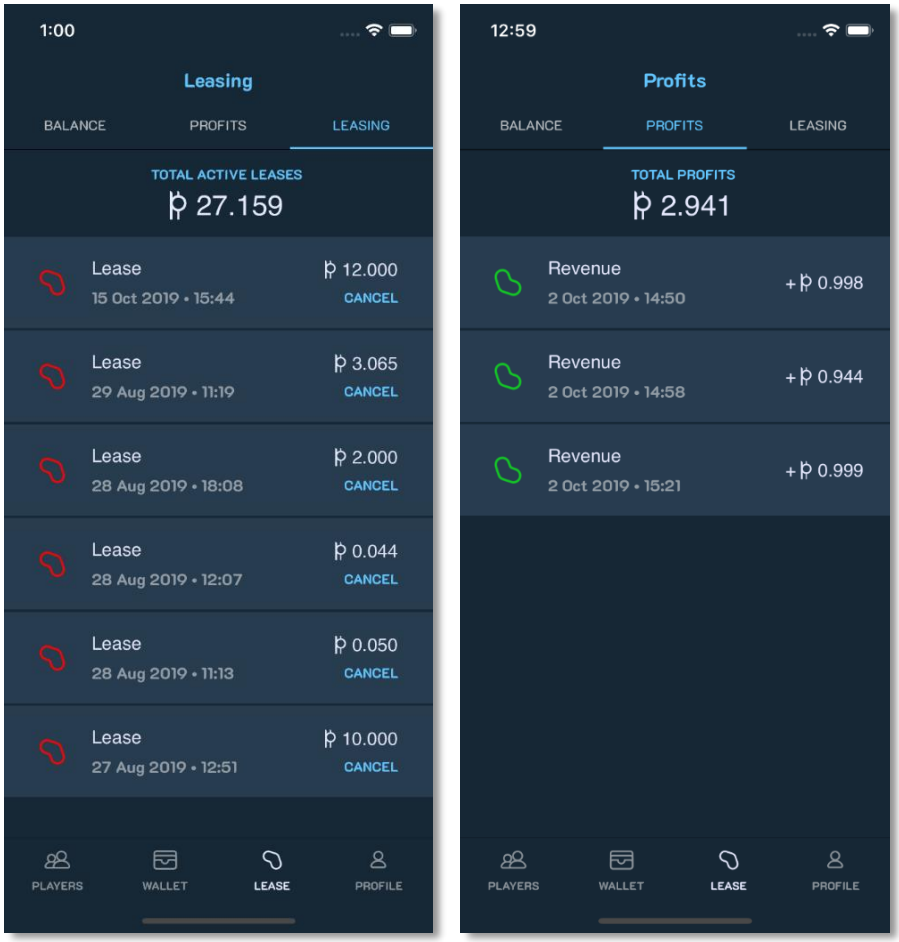


Figura 38 - Lista de leases ativos e lista de lucros relativos ao leasing

De realçar ainda que todas as operações relacionadas com transações na *blockchain*, como a submissão de uma ordem de compra de um ativo, o cancelamento de uma ordem ou a execução de leasing carecem de uma confirmação por parte do utilizador (1º ecrã da figura 39). Depois de confirmada a operação, é retornada uma mensagem

relativa ao sucesso ou insucesso na sua execução (2º ecrã da figura 39) e é realizada a atualização dos saldos em carteira em poucos segundos.

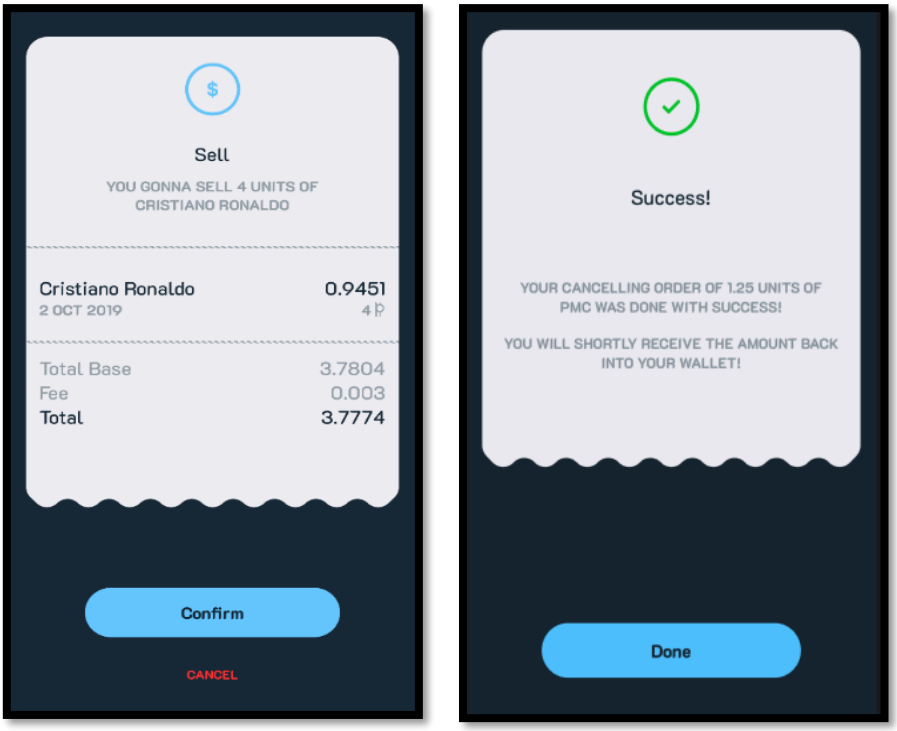


Figura 39 - Ecrã para confirmar transação e mensagem de sucesso

6 - Conclusões

Os objetivos deste estágio inerentes ao desenvolvimento da aplicação "Player Market Cap", envolveram desenvolvimento não só a nível aplicacional (*front-end*), como também a nível de servidor (*back-end*), fazendo uso de diversos tecnologias e metodologias de programação. O desenvolvimento da aplicação foi realizado de modo a dar resposta aos requisitos definidos. Nos testes efetuados verificou-se que as respostas do servidor aos pedidos estão em conformidade com o previsto, resultando em execuções com sucesso por parte das funcionalidades da aplicação.

A implementação e uso da *blockchain* para a parte negocial da aplicação revelou-se benéfica, uma vez que automatiza e disponibiliza uma grande quantidade de operações e métodos de negociação, reduzindo bastante o desenvolvimento requerido para implementar um sistema deste tipo de raiz, que seria complexo e dispendioso.

De realçar também que o desenvolvimento da aplicação móvel com base em React Native e o desenvolvimento da API com base em Node.js trouxe grande eficiência ao desenvolvimento de código, uma vez que foi possível desenvolver três componentes distintas (aplicação para iOS, aplicação para Android e API) tendo apenas JavaScript como linguagem de programação base, que de outro modo poderia acarretar a aprendizagem e aplicação de três tecnologias diferentes.

6.1 - Trabalho Futuro

Como trabalho futuro existem alguns aspetos que poderão ser desenvolvidos e que acrescentariam valor à aplicação e ao seu funcionamento no geral. De seguida são enumerados alguns desses aspetos:

- Habilitar a funcionalidade de levantamentos e depósitos, através da qual os utilizadores poderão realizar movimentos entre a carteira *blockchain* associada à aplicação e o exterior (possivelmente *Exchanges*);
- Conceber um mecanismo que reconheça as ordens de compra/venda submetidas pelos utilizadores que se encontrem ativas na Waves DEX, e deste modo se possam realizar automaticamente "contraordens" que preencham os requisitos negociais;
- Complementar a gestão dos preços de compra e venda dos ativos digitais (jogadores) tendo em conta o seu desempenho desportivo num determinado

período (por exemplo durante a semana), promovendo variações de preço positivas para aqueles que mais se destaquem;

- Adicionar à aplicação um mecanismo de notificações, relativas por exemplo a oportunidades de compra ou venda de ativos digitais, de modo a incentivar os utilizadores a executar operações;

Bibliografia

[1] - Harper, C. (2018). *PwC Global Survey: Corporate Interest in Blockchain on the Rise*. Disponível em: https://bitcoinmagazine.com/articles/pwc-global-survey-corporate-interest-blockchain-rise

[2] - Bassil, Y. (2012). *A simulation model for the waterfall software development life cycle*. arXiv preprint arXiv:1205.6904.

[3] - Nakamoto, S. (2009). *Bitcoin: A peer-to-peer electronic cash system.*

[4] - Crosby, M., Pattanayak, P., Verma, S., & Kalyanaraman, V. (2016). *Blockchain technology: Beyond bitcoin*. Applied Innovation, *2*(6-10), 71.

[5] - Alecrim, Emerson. (2017). *O que é Blockchain: significado e funcionamento.* Disponível em: https://www.infowester.com/blockchain.php

[6] EzPack. (s.d.). *EzPack is introducing a pre-paid system secured by blockchain payment system*. Disponível em: https://www.ezpackwater.com/pre-paid-blockchain-water-billing-systems

[7] - Christidis, K., & Devetsikiotis, M. (2016). *Blockchains and smart contracts for the internet of things*. Ieee Access, *4*, 2292-2303.

[8] - Binance Academy. (2019). *What are nodes?*. Disponível em: https://www.binance.vision/pt/blockchain/what-are-nodes

[9] Xu, X., Pautasso, C., Zhu, L., Gramoli, V., Ponomarev, A., Tran, A. B., & Chen, S. (2016). *The blockchain as a software connector*. *In 2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA) (pp. 182-191).* IEEE.

[10] - Waves Platform. (s.d.). *Waves Docs: Blockchain*. Disponível em: https://docs.wavesplatform.com/en/blockchain/.html

[11] - Bentov, I., Lee, C., Mizrahi, A., & Rosenfeld, M. (2014*). Proof of Activity: Extending Bitcoin's Proof of Work via Proof of Stake*. IACR Cryptology ePrint Archive, *2014*, 452.

[12] - Seth, S. (2018). *Public, private, permissioned blockchains compared*. Disponível em: https://www.investopedia.com/terms/p/permissioned-blockchains.asp

[13] – Zainuddin, A. (s.d.). *Public vs Private Blockchain: What's the difference?*. Disponível em: https://masterthecrypto.com/public-vs-private-blockchain-whats-the-difference

[14] - Gabison, G. (2016). *Policy considerations for the blockchain technology public and private applications*. SMU Sci. & Tech. L. Rev., 19, 327.
[15] - Warren, W., & Bandeali, A. (2017). *0x: An open protocol for decentralized exchange on the Ethereum blockchain*.
[16] - Bertolucci, G. (2018). *Exchange não é wallet*. Disponível em: https://livecoins.com.br/exchange-nao-e-wallet/
[17] – Fortney, L. (2019). *Blockchain Explained*. Disponível em: https://www.investopedia.com/terms/b/blockchain.asp#advantages-and-disadvantages-of-blockchain
[18] – Rosic, A. (2017). *How to Invest in Cryptocurrencies: The Ultimate Beginners Guide.* Disponível em: https://blockgeeks.com/guides/how-to-invest-in-cryptocurrencies/
[19] – Cheng, E. (2017). *Meet CryptoKitties, the $100,000 digital beanie babies epitomizing the cryptocurrency mania.* Disponível em: https://www.cnbc.com/2017/12/06/meet-cryptokitties-the-new-digital-beanie-babies-selling-for-100k.html
[20] – Takahashi, D. (2018). *CryptoKitties explained: Why players have bred over a million blockchain felines?.* Disponível em: https://venturebeat.com/2018/10/06/cryptokitties-explained-why-players-have-bred-over-a-million-blockchain-felines/
[21] - GSM Association. (2017). *Number of Mobile Subscribers Worldwide Hits 5 Billion.* Disponível em: https://www.gsma.com/newsroom/press-release/number-mobile-subscribers-worldwide-hits-5-billion/
[22] – Statista. (s.d.*). Number of mobile app downloads worldwide from 2016 to 2018 (in billions)*. Disponível em: https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/
[23] – StatsCounter GlobalStats. (s.d.). *Mobile Operating System Market Share WorldWide*. Disponível em: https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-201801-201812-bar
[24] - Madureira, D. (2017). *Aplicativo nativo, web app ou aplicativo híbrido.* Disponível em: https://usemobile.com.br/aplicativo-nativo-web-hibrido/#web
[25] - Heitkötter, H., Hanschke, S., & Majchrzak, T. A. (2012). *Evaluating cross-platform development approaches for mobile applications. In International Conference*

on Web Information Systems and Technologies (pp. 120-138). Springer, Berlin, Heidelberg.

[26] – Hales, A. (2019). *Top 10 Best Mobile App Development Frameworks in 2019-20.* Disponível em: https://hackernoon.com/top-10-best-mobile-app-development-frameworks-in-2019-612b95cf930f

[27] - Alexander, P. J., & Radhakrishnan, N. (2015, March). Remote lab implementation on an embedded web server. In *2015 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2015]* (pp. 1-5). IEEE.

[28] – Waves Platform. (s.d.). *Private Waves Network*. Disponível em: https://docs.wavesplatform.com/en/waves-node/private-waves-network.html

[29] - Delfino, P. (s.d.). *Node.js: Entenda o que é e como funciona essa tecnologia*. Disponível em: https://e-tinet.com/linux/node-js

[30] – Orlandi, C. (2017). *Configurando o ORM Sequelize no NodeJS com ExpressJS*. Disponível em: https://blog.rocketseat.com.br/nodejs-express-sequelize/

[31] - Ariane, G. (2019). *O que é SSL/TLS e HTTPS?*. Disponível em: https://www.hostinger.com.br/tutoriais/o-que-e-ssl-tls-https/

[32] - Virdó, H. & Juell, K. (2018). *How To Secure Nginx with Let's Encrypt on Ubuntu 18.04*. Disponível em: https://www.digitalocean.com/community/tutorials/how-to-secure-nginx-with-let-s-encrypt-on-ubuntu-18-044

[33] - Nginx. (s.d.). *Configuring HTTPS Servers*. Disponível em: http://nginx.org/en/docs/http/configuring_https_servers.html

[34] – Becker, L. (2019). O que é React Native? Disponível em: https://www.organicadigital.com/blog/o-que-e-react-native/

[35] – Facebook Inc. (s.d.). *Debugging*. Disponível em: https://facebook.github.io/react-native/docs/debugging

[36] - Facebook Inc. (s.d.). *Getting started*. Disponível em: https://facebook.github.io/react-native/docs/getting-started

[37] – Buchko S. (2018). *What Is the Waves Platform? | The Ultimate Guide*. Disponível em: https://coincentral.com/waves-platform-beginner-guide

[38] – Nunes M. (2018). *O que é a Plataforma Waves*. Disponível em: https://livecoins.com.br/o-que-e-a-plataforma-waves

[39] – Capuano, T. (2019). *Mining by Leasing Proof of Stake (LPoS) Waves*. Disponível em: https://www.publish0x.com/waves-beach/mining-leasing-proof-stake-lpos-waves-xxynez

[40] - Waves Platform. (s.d.). *DEX protocol*. Disponível em: https://docs.wavesplatform.com/en/waves-dex/dex-protocol.html

[41] - Waves Platform. (s.d.). *Transfers and Gateways*. Disponível em: https://docs.wavesplatform.com/en/waves-client/wallet-management.html

[42] - Patel, P. (2018). *What exactly is Node.js?* Disponível em: https://www.freecodecamp.org/news/what-exactly-is-node-js-ae36e97449f5/

[43] - Opus Software. (2018). *Node.js – O que é, como funciona e quais as vantagens*. Disponível em: https://www.opus-software.com.br/node-js/

[44] - Waves Platform. (s.d.). *Scalability Limits and Challenges in Current Blockchain Systems*. Disponível em: https://docs.wavesplatform.com/en/blockchain/waves-protocol/waves-ng-protocol.html

Anexos

Anexo A – Plataforma Waves

A plataforma Waves foi fundada em 2016 por Sasha Ivanov, com o objetivo de se tornar uma "blockchain para as pessoas" e tendo com principal propósito reinventar o empreendedorismo através de uma infraestrutura partilhada na qual qualquer pessoa pode facilmente criar, armazenar, negociar ou gerir o seu próprio *token* (ativo digital) [37].

Desde o seu lançamento, esta plataforma tem vindo a crescer substancialmente, com introdução contínua de novas funcionalidades e melhorias que lhe permitem afirmar-se cada vez mais como uma solução de *blockchain* inovadora e robusta [37]. Esta afirmação já valeu à empresa por trás da plataforma, parcerias com empresas de como a Delloite ou com a Bolsa de Valores de Moscovo, com o fim de desenvolver soluções baseadas em tecnologia *blockchain* [38].

Tal como a maior das plataformas de *blockchain*, a Waves possui a sua própria representação de valor em forma de criptomoeda, que desde o seu surgimento chegou a atingir uma cotação máxima de cerca de 16.29 euros em dezembro de 2017. De momento, este valor tem vindo a baixar (sendo 4.12 euros o valor máximo atingido em 2019) por diversas circunstâncias de mercado, mas a Waves mantém-se de forma consistente entre 100 criptomoedas com mais e melhor mercado (segundo dados do portal CoinMarketCap[43]).

Como plataforma para implementação, a Waves disponibiliza no seu repositório oficial ficheiros que possibilitam a implementação de nós de *blockchain* com base em rede públicas, *testnet* ou redes privadas, existindo documentação que detalha os passos a seguir para que tal seja possível.

Em comum para implementação nestes diferentes tipos de rede, a plataforma é integrada com funcionalidades inovadoras apresentadas de seguida.

Leasing Proof of Stake (LPoS)

O LPoS, tal como o nome indica, tem é um método de validação semelhante ao *Prove of Stake,* que permite ao *mining node* com maior número Waves em carteira, ter a maior probabilidade ser selecionado para gerar o próximo bloco de transações e desta forma receber uma *reward* (conjunto das *fees* das transações processadas) [39].

[43] https://coinmarketcap.com/currencies/waves/

A inovação neste método é que dá a possibilidade a todos os utilizadores da rede de participar na validação do bloco, fazendo um *leasing* dos seus *tokens* Waves a um *mining node* e desta forma receber percentagem da *reward* obtida por esse *mining node*, de acordo com a quantidade de Waves que alocou nesse *leasing* [39].

Este processo de *leasing*, consiste numa "espécie de empréstimo" de *tokens* Waves a um endereço de rede (de um *mining node*), com o objetivo de este aumentar a sua quantidade total de Waves em carteira. Este leasing pode ser cancelado a qualquer momento, sendo que enquanto ativo, os *tokens* alocados à operação estão impossibilitados do uso por parte do seu real proprietário, bem como do uso por parte do *mining node* [39].

Protocolo Waves-NG

Este protocolo de rede desenvolvido pela Waves tem o propósito de aumentar a velocidade de processamento das transações. A ideia do protocolo é a de criar continuamente "microblocos" de transações em vez de um bloco com um grande conjunto de transações, tornado assim o processo de confirmação das transações mais eficiente.

Após definido um *mining node* pelo mecanismo de seleção da *blockchain*, este gera um bloco sem transações (*key block)* e distribui-o pela rede. De seguida, a cada 5 segundos é gerado um microbloco de transações (com estrutura semelhante a um bloco normal, mas com um número de transações reduzido) que é inserido no *key block* e continuamente distribuído por toda a rede [40].

Na figura 40, é apresentada uma análise comparativa efetuada pela Waves das velocidades de processamento das transações que a plataforma pode atingir com a integração deste protocolo, quando comparada com outras plataformas *blockchain*.

O Waves-NG também integra mecanismos para deteção de fraudes ou inconsistências na geração dos microblocos e respetivo *mining* que tornam todo o processo confiável [40].

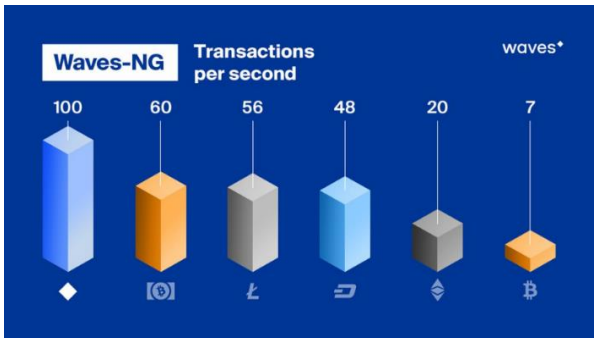


Figura 40 - Análise comparativa de transações por segundo em diversas plataformas blockchain [42]

Waves DEX

A DEX (também conhecida como *matcher*) é uma *exchange* integrada na plataforma Waves que permite aos utilizadores submeter as suas ordens de compra e venda de ativos e negociá-los de forma automática.

Como é possível verificar na figura 41, as ordens de compra e venda são submetidas num *Order Book*, que as organiza pelo tipo (*buy/sell*) e numa disposição de acordo com a data de submissão da ordem. Após esta submissão, a DEX possui mecanismos que fazem uma combinação (*match*) entre ordens tendo em conta fatores como as quantidades de ativos e de Waves a transacionar por compradores e vendedores e a sua ordem na "fila de negociação" (de acordo com a data) [40].

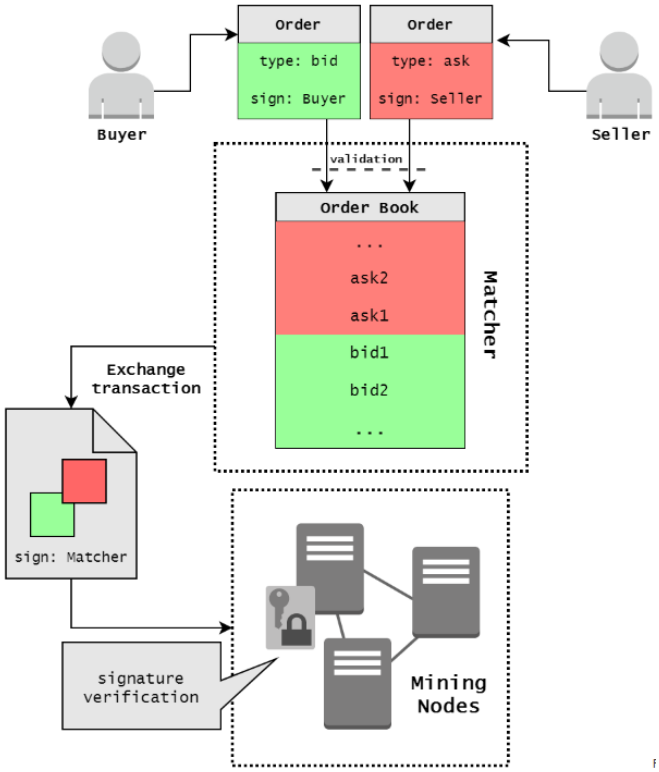


Figura 41 - Submissão de ordens no order book da Waves DEX [44]

Quando existe um *match* entre duas ordens, a transação é verificada e submetida automaticamente na *blockchain*, executando-se assim a transferência dos *tokens (*Waves e ativos digitais) entre as contas envolvidas na negociação e o pagamento de *fees* pela execução da transação por parte de ambas [40].

A DEX possibilita também o cancelamento de ordens submetidas, desde que estas ainda não tenham encontrado e executado um *match* com uma ordem inversa.

Waves API

A Waves possui uma API por defeito para consulta de dados na *blockchain*, que podem ser relativos por exemplo, às configurações de rede, às carteiras de utilizadores, a blocos e transações específicos ou às ordens de compra e venda na rede.

Esta API foi integrada com a *framework* Swagger[44], permitindo a descrição, consumo e visualização dos seus métodos ao nível de um *browser web*, o que facilita bastante na constatação do que um determinado pedido à API deve retornar e na realização de testes.

Fees

A nível das *fees*, a Waves destaca-se por cobrar taxas baixas e competitivas, sendo 0.003 Waves o valor normal de cobrança por executar uma transação.

Dependendo do tipo de transação este valor pode variar destacando-se os seguintes tipos e valores de fees:

Issue* de *tokens	0.001
Exchange* (compra ou venda) de *tokens	0.003
Lease	0.001
Cancel Lease	0.001
Transfer	0.001

De salientar também que para o caso de se efetuar a compra de uma grande quantidade de *tokens* a taxa de *fees* será igual à da compra de apenas 1 unidade, portanto, apesar de a taxa já ser baixa, quanto maior quantidade de *tokens* se transacionar de uma vez só, mais proveitoso será.

Métodos de depósito e levantamento

Para se efetuarem depósitos ou levantamentos de valor nesta plataforma há diversos métodos disponíveis, como transferências com intermédio de outras *blockchains* ou *exchanges*, que é a maneira de depósito mais comum na maior parte das plataformas [41].

[44] https://nodes.wavesnodes.com/api-docs/index.html

Está também disponível a transferência bancária ou pagamento por cartão de crédito, embora para estes casos a sua viabilidade pode depender da legislação em vigor no país do utilizador [41].

Anexo B - Operações na *blockchain* com PyWaves

Neste anexo são apresentados alguns excertos do código usado para executar algumas das operações com a biblioteca PyWaves, sendo que em comum para todos, pode verificar-se que as primeiras linhas de código são reservadas à importação da biblioteca e às declarações dos endereços de rede da *blockchain* e do *matcher* (Waves DEX).

Na figura 42 está representado um excerto relativo à criação de um novo endereço na rede, sendo para este caso apenas necessário invocar a função "Address", vinculada à PyWaves.

```
import pywaves as pw

pw.setChain('node="http://piquet.brandit.ws:6863', chain='custom', chain_id = 'W')
# generate a new address
newAddress = pw.Address()
```

Figura 42 - Geração de novo endereço na rede

No excerto da figura 43, é feita uma consulta das informações (saldo de Waves e *tokens*) de uma determinada conta. Para tal é necessário declarar o endereço dessa conta e de seguida apresentá-lo ao nível do terminal com a instrução "*print*", retornando dados relativos por exemplo aos Waves ou tokens em carteira.

```
import pywaves as pw
pw.setNode(node = 'http://piquet.brandit.ws:6863', chain = 'custom', chain_id = 'W')

myAddress = pw.Address('3PK2xXeAhq3ADCrWx8XeqEJdCEDMmEmMmjK')
print(myAddress)
```

Figura 43 - Consulta de informações relativos a um endereço de conta

Na figura 44 é apresentada a estrutura de código necessária para a criação de um *token* na rede. Primeiramente é definida a conta que irá ser a criadora e proprietária dos *tokens*, através de referência à sua chave privada. Para o *token* em específico podem definir-se propriedades como o seu nome, a sua descrição, a quantidade a emitir ou o número de casas decimais.

```
import pywaves as pw
pw.setNode(node = 'http://piquet.brandit.ws:6863', chain = 'custom', chain_id = 'W')
pw.setMatcher(node = 'http://piquet.brandit.ws:6864')

myAddress = pw.Address(privateKey='CL6Bh3BkEuRnpigsB64vYCWcmaEYT8rxZcHLSqutNPXG')
#Carteira 3FfcVpaWELcMZfcsJfZKn1vTFGdTqhbfrym

myToken = myAddress.issueAsset( name = "JOGX",
                                description = "Jogador X",
                                quantity = 1000000000000, #100 milhoes
                                decimals = 4 )
```

Figura 44 - Criação de token na rede

Na figura 45 é apresentado um excerto da criação dos *tokens* representativos de alguns jogadores que estão disponíveis para negociação na rede, sendo respetivamente composto pelos seus nomes e descrições, por uma quantidade de venda de 100 milhões para cada *token* e por uma quantidade de casas decimais equivalente a 8, que com uma justificação simplificada, implica que cada unidade de token de um jogador corresponda a 1 unidade de Waves.

```
import pywaves as pw
pw.setNode(node = 'http://piquet.brandit.ws:6863', chain = 'custom', chain_id = 'W')
pw.setMatcher(node = 'http://piquet.brandit.ws:6864')

myAddress = pw.Address(privateKey='HCcVKv8oAvzt4qmxsHd8erGfQTwH5f5eH445JREbM8sR') #contaGen 2

myToken = myAddress.issueAsset(name = "CRONALDO", description = "Cristiano Ronaldo", quantity = 1000000000000, decimals = 8)
myToken2 = myAddress.issueAsset(name = "LMESSI", description = "Lionel Messi", quantity = 1000000000000, decimals = 8)
myToken3 = myAddress.issueAsset(name = "KDBRUYNE", description = "Kevin De Bruyne", quantity = 1000000000000, decimals = 8)
myToken4 = myAddress.issueAsset(name = "ATELLES", description = "Alex Telles", quantity = 1000000000000, decimals = 8)
myToken5 = myAddress.issueAsset(name = "EHAZARD", description = "Eden Hazard", quantity = 1000000000000, decimals = 8)
myToken6 = myAddress.issueAsset(name = "KMBAPPE", description = "Kylian Mbappe", quantity = 1000000000000, decimals = 8)
myToken7 = myAddress.issueAsset(name = "HKANE", description = "Harry Kane", quantity = 1000000000000, decimals = 8)
myToken8 = myAddress.issueAsset(name = "MSALAH", description = "Mohamed Salah", quantity = 1000000000000, decimals = 8)
myToken9 = myAddress.issueAsset(name = "PDYBALA", description = "Paulo Dybala", quantity = 1000000000000, decimals = 8)
myToken10 = myAddress.issueAsset(name = "RSTERLING", description = "Raheem Sterling", quantity = 1000000000000, decimals = 8)
myToken11 = myAddress.issueAsset(name = "NEYMAR", description = "Neymar Junior", quantity = 1000000000000, decimals = 8)
myToken12 = myAddress.issueAsset(name = "AGRIEZMANN", description = "Antoine Griezmann", quantity = 1000000000000, decimals = 8)
myToken13 = myAddress.issueAsset(name = "VVDIJK", description = "Virgil van Dijk", quantity = 1000000000000, decimals = 8)
myToken14 = myAddress.issueAsset(name = "DALLI", description = "Dele Alli", quantity = 1000000000000, decimals = 8)
myToken15 = myAddress.issueAsset(name = "ECAVANI", description = "Edinson Cavani", quantity = 1000000000000, decimals = 8)
myToken16 = myAddress.issueAsset(name = "MVERRATTI", description = "Marco Verratti", quantity = 1000000000000, decimals = 8)
myToken17 = myAddress.issueAsset(name = "EMILITAO", description = "Eder Militao", quantity = 1000000000000, decimals = 8)
myToken18 = myAddress.issueAsset(name = "JFELIX", description = "Joao Felix", quantity = 1000000000000, decimals = 8)
myToken19 = myAddress.issueAsset(name = "MDLIGT", description = "Matthijs de Ligt", quantity = 1000000000000, decimals = 8)
myToken20 = myAddress.issueAsset(name = "SRAMOS", description = "Sergio Ramos", quantity = 1000000000000, decimals = 8)
```

Figura 45 - Criação de grupo de tokens (jogadores) na rede

Já na figura 46 é apresentada a submissão de uma ordem de venda na rede (mais propriamente na Waves DEX), sendo para este caso necessário declarar a conta que vai negociar o *token* (através da chave privada), o *token a negociar* (através do respetivo endereço) e o "*AssetPair*" (que será composto precisamente pela conta e *token* referidos). Depois disto é executada a função "*sell*", referenciando o "*AssetPair*", a quantidade de *token* a negociar e o preço a que este deve ser negociado.

Se a intenção for executar uma operação de compra de *token*, o processo é semelhante, tendo apenas de se substituir na estrutura de código o termo "*sell*" pelo termo "*buy*" para que a função respetiva seja invocada.

```
import pywaves as pw
pw.setNode(node = 'http://piquet.brandit.ws:6863', chain = 'custom', chain_id = 'W')
pw.setMatcher(node = 'http://piquet.brandit.ws:6864')
myAddress = pw.Address(privateKey='CL6Bh3BkEuRnpigsB64vYCWcmaEYT8rxZcHLSqutNPXG')

CRONALDO = pw.Asset('61f4RYZPNBCXSYYghRgUbXp3ApMSnDBSwCR261EJwjzw')
A_Pair = pw.AssetPair(CRONALDO, pw.WAVES)

myOrder = myAddress.sell(assetPair = A_Pair, amount = 10, price = 150000000)
print(myOrder)
```

Figura 46 - Submissão de ordem de venda na rede

Anexo C - Node.js

O Node.js foi criado em 2009 por Ryan Dahl e surgiu com o propósito de que o JavaScript pudesse ser utilizado no lado do servidor, permitindo aos programadores desenvolver aplicações de rede mais eficientes e escaláveis [opus].

Esta tecnologia teve como base para o seu desenvolvimento o V8 JavaScript Engine, um interpretador de JavaScript em C++ desenvolvido pela Google para o seu navegador *web:* Google Chrome [42].

O uso de Node.js no lado do servidor inclui diversos benefícios, entre os quais se destacam:

- O uso da mesma linguagem no *front-end* e *back-end,* permitindo versatilidade aos programadores;
- Ser leve e eficiente, uma vez que tem um baixo consumo de recursos computacionais;
- A disponibilidade de pacotes de *software* presentes no repositório NPM;

A leveza e eficiência do Node.js derivam da sua execução *single-thread,* sendo requisitada apenas uma tarefa (*thread*) para executar o código, enquanto que noutras linguagens a execução é *multi-thread* [42].

Basicamente num servidor com linguagens mais tradicionais, por cada pedido recebido é criada uma *thread* para tratá-la, sendo alocados recursos computacionais (como memória RAM) para cada uma delas. Como estes recursos são limitados, a criação de *threads* não será infinita e quando um certo limite for atingido, os novos pedidos terão de aguardar pela conclusão dos antigos [43].

Na figura 47, é apresentado o modelo de gestão de *threads* do Node.js comparativamente aos modelos tradicionais.

No Node.js é apenas usada uma *thread* para tratar de todas as requisições. Esta *thread,* denominada de *Event Loop,* está continuamente em execução e à espera de novos pedidos, que assim que recebidos são inseridos numa “pilha de eventos”, que suporta pedidos simultâneos, uma vez que gere todos os processos de forma assíncrona [43].

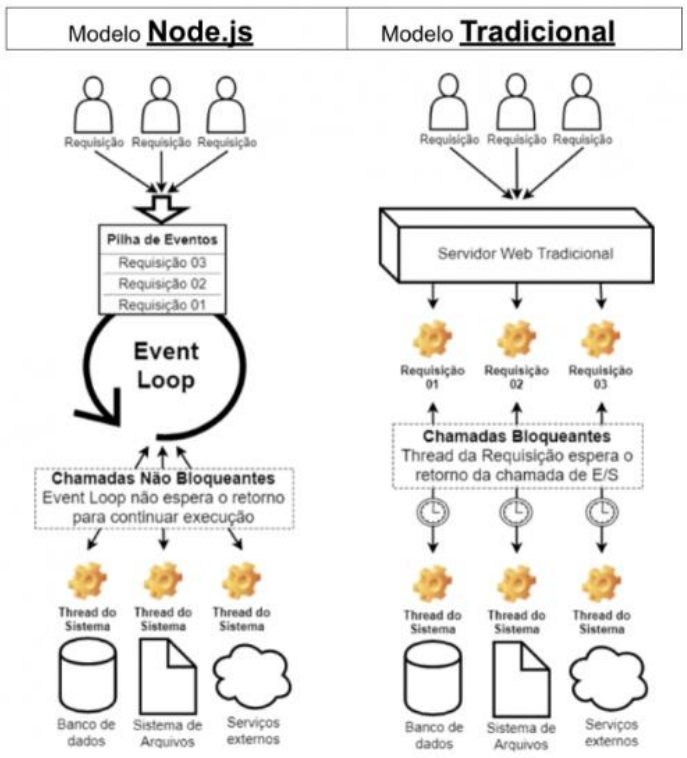


Figura 47 - Modelo de gestão de threads do Node.js e modelo de gestão de threads tradicional [43]

Repositório NPM

O NPM (*Node Package Manager*) é o gestor de pacotes do Node.js e é o maior repositório *online* de *softwares*, como bibliotecas elaboradas pela comunidade de utilizadores que resolvem a maior parte dos problemas no desenvolvimento de aplicações, tornando a programação mais rápida e eficiente [42].

Para além disso pode ser usado como utilitário da linha de comandos, permitindo uma fácil interação com o repositório, instalação de pacotes e gestão de versões e dependências de *software* através de instruções bastante simples.

O pacote de *software* mais conhecido é o Express.js, que permite o desenvolvimento de aplicações *web* [43]. Apesar de para este caso o Node.js possuir por defeito módulos que permitem as mesmas funcionalidades do Express.js, as configurações são geralmente mais complexas, sendo, portanto, um exemplo no qual os uso de um pacote de software externo pode ser proveitoso.