Capítulo 2

Tokens Não Fungíveis (NFTs): Conceitos, Aplicações e Desafios

Ronan D. Mendonça (UFV), Josué N. Campos (UFV), Luiz F. M. Vieira (UFMG), Marcos A. M. Vieira (UFMG), Alex Borges Vieira (UFJF), José A. M. Nacif (UFV)

Abstract

Nowadays, we have been surprised by the popularization of Non-Fungible Token (NFT) markets due to their great success in the blockchain community. NFTs are records of asset ownership held by smart contracts on a blockchain. These records can be traded, transferred and rented through an exchange of ownership. The purpose of this work is to introduce the main concepts, applications and challenges related to NFTs. We present a theoretical view of the technologies involved in the creation and maintenance of NFTs, along with decentralized applications and marketplaces that trade the tokens. Moreover, we list the practical issues for using token standards and creating a market for trading and renting NFTs.

Resumo

Atualmente fomos surpreendidos com a popularização de mercados de Token Não-Fungível (NFT) devido ao seu grande sucesso obtido diante da comunidade blockchain. Os NFTs são registros da propriedade de ativos realizados por contratos inteligentes em uma blockchain. Estes registros podem ser negociados, transferidos e alugados por meio da troca de titularidade. O objetivo deste capítulo é apresentar os principais conceitos, aplicações e desafios relacionados aos NFTs. Apresentamos uma visão teórica das tecnologias envolvidas para a criação e manutenção dos NFTs, juntamente a aplicações descentralizadas e marketplaces que fazem negociações dos tokens. Além disso, apresentamos as questões práticas para a utilização dos padrões de tokens e a criação de mercado para comercialização e aluguel dos NFTs.

2.1. Introdução

Plataformas e aplicações baseadas em *blockchains*, em sua essência, são desenvolvidas para permitir transações financeiras entre usuários anônimos. Isso é possível com o uso de criptografia assimétrica, componente que faz parte da tecnologia *blockchain*, onde os usuários se identificam pelas suas chaves públicas e usam as chaves privadas secretas para autenticar as transações que eles emitem em favor de outros usuários [Wood 2014]. As *blockchains* armazenam as transações realizadas em forma de blocos interconectados por meio de *hashes* criptográficos e, um dos principais benefícios dessa tecnologia é que os dados são mantidos distribuídos pelos entes participantes da rede, sem a necessidade de uma autoridade central [Hewa et al. 2021a].

A relevância dada por pesquisadores e desenvolvedores à tecnologia *blockchain* aumentou significativamente a busca por novas formas de explorar suas características. De fato, as plataformas *blockchains* populares atuais, como Bitcoin e Ethereum, têm diversas funcionalidades com potencial para alavancar novos negócios. Por exemplo, aplicações financeiras descentralizadas (*DeFis*) [Schär 2020] oferecem serviços[1] para pagamentos, empréstimos, doações ou *royalties* por bens digitais (e.g., *tokens não fungíveis*). Há também serviços para análise de perfis dos usuários dessas plataformas, para analisar riscos de crédito e até mesmo para classificar usuários com envolvimento em esquemas fraudulentos [Bartoletti et al. 2020].

Dentre os diversos domínios possíveis de uso das *blockchains*, destaca-se atualmente o domínio dos *tokens*, no qual o uso de aplicações específicas deste domínio visa agregar forma de garantia de posse e representação de bens. Os bens são objetos físicos ou digitais de utilidade variada e que podem ser trocados ou vendidos. Podem ser classificados como um bem fungível e não fungível, sendo que um bem fungível significa que ele pode ser substituído por outro que representa o mesmo valor, e já um bem não fungível não admite substituição e são considerados com valor especial e individual [Fairfield 2021]. Um *token* não fungível (*Non-Fungible Token* (NFT)) é uma tecnologia que permite registrar de maneira distribuída a posse de um bem não fungível. Portanto, o NFT é um token ou certificado que comprova a propriedade de itens exclusivos. São utilizados para provar a propriedade de bens exclusivos como itens colecionáveis ou de investimento, pois se pode revender um NFT e obter lucros com base em seu valor atual[Valeonti et al. 2021]. Um exemplo de bens fungíveis é o próprio dinheiro onde cada nota comum tem o mesmo valor que todas as outras notas. Desta mesma forma, as criptomoedas também se aplicam a este mesmo conceito, onde um Ether e um Bitcoin têm o mesmo valor de outro um Ether e outro um Bitcoin respectivamente. Sendo assim, um item é substituível por qualquer outro item do mesmo tipo e considerados itens fungíveis. Mas para itens que não têm valores iguais a qualquer outro ou não podem ser igualmente trocados, recebem a denominação de itens não fungíveis.

As implementações de NFTs em *blockchains* é uma tecnologia relativamente nova e suas aplicações têm o potencial de mudar a forma de determinar a verificação do direito de posse. Os NFTs trouxeram para as plataformas *blockchains* novas possibilidades que ampliaram o oferecimento de aplicações inovadoras em vários setores como, por exemplo arte, música, moda e bebidas. Atualmente, os NFTs são amplamente conhecidos por

[1] Exemplos de DeFi Ecosystem - https://defiprime.com/ethereum

proporcionarem compras de obras de arte virtuais. Até mesmo um jogador de futebol brasileiro famoso virou notícia ao comprar uma obra de arte digital por meio de NFT em 2021 [2]. Além das obras de artes, NFTs também possibilitam o registro de posse de outros bens (ou a forma de determinar a verificação do direito de posse), como roupas, músicas, terrenos, personagens em jogos, ingressos para shows, entre outros.

Mais precisamente, os NFTs são descritos como registros individuais de posse e são registrados em plataformas *blockchains*, como por exemplo, o Ethereum. Por meio deles é possível definir a propriedade para um determinado item e, assim, atender diversas aplicações como, por exemplo, a certificação de direito autoral e de posse de objetos físicos e digitais [Valeonti et al. 2021]. Um NFT pode ser utilizado para identificar e descrever objetos do mundo real e ou do mundo digital. Tais objetos são representados de forma a possuir propriedades e características únicas e, desta maneira, não podem ser simplesmente trocados ou comparados por outro item deste mesmo objeto pois são únicos. No mundo real, podemos citar como exemplo qualquer objeto físico identificável. Já no mundo digital, toda criação é passível de se tornar um NFT [Nadini et al. 2021]. Em contrapartida, os itens ou *tokens* fungíveis podem ser trocados por outro ou outros itens de mesmo valor que não afetam assim a sua posse ou valorização. Por exemplo, as moedas em circulação nos países e que são permutáveis entre si.

A representação da propriedade de itens exclusivos por meio dos NFTs tornou-se cada vez mais comum ao longo do ano de 2021. Itens digitais ou do mundo real como criações artísticas, objetos colecionáveis, objetos proprietários e até mesmo direitos autorais podem ser identificados por um registro imutável de sua propriedade em *blockchains*. Atualmente, há inúmeros locais, chamados de mercados digitais ou marketplaces, especializados em manter NFTs, assim como plataformas de *blockchain*, oferecem recursos para sua criação e manutenção. Como por exemplo a plataforma Ethereum[3] e os mercados OpenSea[4] e Rarible[5]

Note que os NFTs surgiram juntamente com inúmeras aplicações, provenientes de diversos setores, em consequência da ampla atenção e recente exploração das *blockchains*. Nos últimos meses, as negociações sobre os NFTs alcançaram um crescimento gigantesco, registrando uma movimentação de $ 34.530.649,86 dólares em 2021[Wang et al. 2021]. O crescente volume de negociações de NFTs também impacta na forma como os indivíduos estão lidando com o dinheiro digital, já que os mercados de NFTs utilizam das criptomoedas para compra e venda de seus itens.

Alinhado a esse direcionamento, há um crescente interesse na criação de mercados digitais para a criação e negociação dos NFTs. Uma análise realizada por [Casale-Brunet et al. 2021], apresenta a movimentação de oito mercados, como exemplos de projetos NFTs, registrados na plataforma Ethereum e classificados em categorias de artes, fotos de perfil e metaverso. Estes mercados, endereçados conforme a Tabela 2.1, registraram um volume total de 315.491 ativos e movimentaram cerca de 425.245 transações entre compra, vendas e transferências.

[2]https://einvestidor.estadao.com.br/criptomoedas/investimento-nft-tendencia-famosos/
[3]`https://ethereum.org`
[4]`https://opensea.io`
[5]`https://rarible.com`

Tabela 2.1. Mercados NFTs, Categorias, Endereços dos contratos na plataforma Ethereum e suas movimentações. Adaptada de [Casale-Brunet et al. 2021].

Projeto NFT	Categoria	Endereço do contrato	Ativos	Transações
HashMasks	Arte Digital	0xc2c747e0f7004f9e8817db2ca4997657a7746928	16384	49404
Art Blocks Curated	Arte Digital	0xa7d8d9ef8d8ce8992df33d8b8cf4aebabd5bd270	57873	80660
CryptoPunks	Foto de perfil	0xb47e3cd837ddf8e4c57f05d70ab865de6e193bbb	10000	28576
Bored Ape Yacht Club	Foto de perfil	0xbc4ca0eda7647a8ab7c2061c2e118a18a936f13d	10000	32074
Acclimated Moon Cats	Foto de perfil	0xc3f733ca98e0dad0386979eb96fb1722a1a05e69	10765	14378
CryptoVoxels	Metaverso	0x79986af15539de2db9a5086382daeda917a9cf0c	14872	16607
Decentraland	Metaverso	0xf87e31492faf9a91b02ee0deaad50d51d56d5d4d	177094	174322
Meebits	Metaverso	0x7bd29408f11d2bfc23c34f18275bbf23bb716bc7	20000	292224

As primeiras menções aos conceitos ligados aos NFTs surgiram há alguns anos. Embora somente no ano de 2021 os NFTs ganharam maior atenção, na verdade, eles foram originados pelo trabalho "Overview of Colored Coins" de [Rosenfeld 2012] em 2012. Nesse trabalho surgiu a ideia de se utilizar a plataforma *blockchain* do Bitcoin para ativos colecionáveis digitais, cupons, certificação de propriedade, ações de empresas, etc. A Figura 2.1 mostra a linha do tempo com os principais fatos ocorridos em relação ao surgimento dos NFTs. Após o trabalho primário, de 2012, surgiu em 2014 a plataforma Counterparty utilizando também o Bitcoin e com o intuito de possibilitar aos usuários a criação de suas próprias moedas e ativos negociáveis. Em 2017 surgiram vários outros projetos NFTs importantes, tais como o Cryptopunks[6], CryptoKitties[7], Decentraland[8] e o mercado OpenSea.

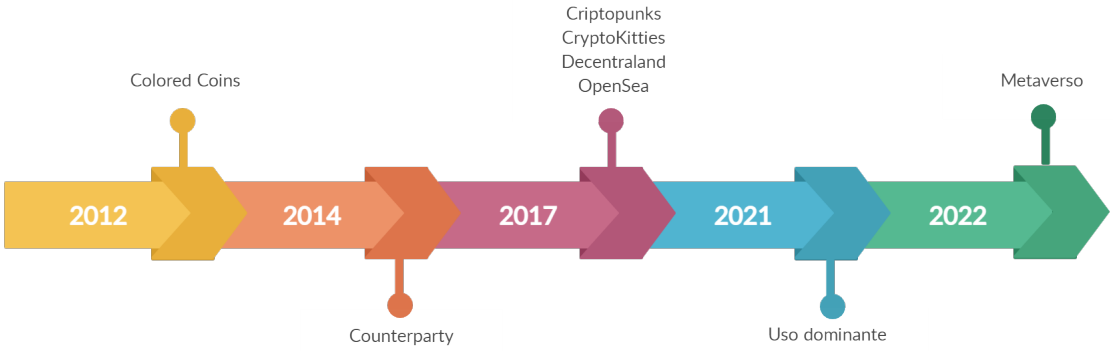

Figura 2.1. Linha do tempo histórica dos NFTs.

Os projetos CryptoPunks e CryptoKitties trabalham com o conceito de arte digital colecionável, que são ativos digitais únicos e nestes casos geradas por algoritmo e armazenados em servidores privados centralizados e com a prova de propriedade armazenada na plataforma *blockchain* Ethereum. CryptoPunks é uma coleção de 10.000 itens de imagens distintas e foi registrado na plataforma Ethereum pelo endereço de contrato

[6] https://www.larvalabs.com/cryptopunks/
[7] https://www.cryptokitties.co
[8] https://decentraland.org

0xb47e3cd837dDF8e4c57F05d70Ab865de6e193BBB. Este contrato serviu de inspiração para o padrão ERC-721 e foi precursor de outros projeto como o CryptoKitties que foi registrado no endereço 0x06012c8cf97BEaD5deAe237070F9587f8E7A266d e que pode ser facilmente verificados pela ferramenta Etherscan[9]. Neste projeto é permitido aos usuários negociar gatos virtuais exclusivos por meio do seu contrato inteligente.

O software Decentraland é uma aplicação que simula um mundo virtual compartilhado e suas transações são verificadas na plataforma Ethereum. Neste mundo virtual é possível criar conteúdos e aplicações que vão desde construções estáticas a jogos interativos. Os NFTs podem ser adicionados às construções ou utilizados nas interações criadas. A integração desta aplicação com os mercados externos de NFTs é possível devido os seus componentes serem construídos em camadas usando contratos inteligentes da plataforma Ethereum. OpenSea e Rarible são mercados para compra e venda de NFTs. Ambos possuem contratos descentralizados e armazenados na blockchain Ethereum. Um mercado NFT é uma camada de interface que torna mais fácil realizar as transações entre a blockchain e os consumidores. As ações mais comuns são criar, vender ou comprar NFTs de diferentes tipos. No entanto, por meio dos mercados também é possível descobrir novos projetos ou obter conhecimento histórico sobre a valorização e negociação de itens.

O cenário do ano de 2021 agilizou os comportamentos relacionados às interações online. Como consequência do trabalho remoto, aumento das compras online e a convivência digital, os NFTs se tornaram uma realidade com um sentido mais justificado já que muitas das interações ocorrem de maneira online. Sendo assim, o avanço desta tecnologia deve conectar esses diferentes mundos, replicando o mundo físico por meio de dispositivos digitais ao mundo virtual. Atualmente esta transição está sendo chamada de metaverso.

A importância dos NFTs para diversos segmentos é retratada através da utilização pelos mercados específicos de cada setor. Cartórios, museus, ateliês, jogos digitais e vários outros setores apresentaram interesses e projetos relacionados a certificação de propriedade utilizando NFTs. Dentre as diversas possibilidades de uso dos NFTs, destaca-se as chances de rentabilidade em suas negociações de compra e venda nos mercados específicos e genéricos. Entretanto, o inerente avanço da quantidade de mercados dos ambientes NFTs e os valores expressivos negociados, requerem soluções que permitam a sua utilização com orçamentos menores [Casale-Brunet et al. 2021]. Neste sentido, a funcionalidade de aluguel para os NFTs, isto é, o uso temporário de um NFT, demonstra ser uma funcionalidade promissora. De fato, é possível alugar com segurança um item e remunerar ao mercado e ao proprietário do *token*. O aluguel de NFTs pode trazer uma renda ao proprietário em um momento que a aquisição pode ser muito dispendiosa para um terceiro. Por parte do locatário, o aluguel de NFTs pode trazer benefícios como o de não ter que investir um grande volume de recursos financeiros na aquisição do item e sua utilização somente quando necessário. Por exemplo, um objeto de um personagem de um jogo online pode ser alugado para um jogador qualquer enquanto esse item NFT não está sendo utilizado por seu dono. Ao ser alugado, o proprietário se torna o locatário daquele bem, e recebe em troca uma rentabilidade paga pelo locador. O jogador que alugou o

[9] `https://etherscan.io`

objeto, ao deixar a plataforma do jogo, pode não ter mais interesse naquele item e assim, devolve ao proprietário original, pagando somente pelo momento que utilizou o objeto.

O ambiente que envolve a criação e negociação dos NFTs é bem mais complexo do que a simples implementação dos padrões estabelecidos para o registro e transferência de posse destes ativos. O arcabouço apresentado pela Figura 2.2 demonstra as partes envolvidas na complexa estrutura utilizada pelos NFTs. Todo o processo que vai desde a criação até a possibilidade de negociação dos *tokens*, utiliza camadas como os próprios *tokens*, os padrões estabelecidos, as bibliotecas implementadas destes padrões e as linguagens utilizadas para esta implementação. Na camada de **Linguagens**, podemos observar uma interdependência com plataforma *blockchain* a qual irá ser desenvolvida a aplicação. Por exemplo, na plataforma Ethereum a linguagem padrão para o desenvolvimento dos contratos é a Solidity. Partindo para a camada **Bibliotecas** encontramos as implementações criadas em uma determinada linguagem para os padrões já estabelecidos. O openZeppelin[10] é um exemplo de biblioteca que disponibiliza, como *Application Programming Interface* (API), interfaces de contratos exaustivamente testados. Os padrões dispostos pela camada **Padrões** descrevem um conjunto de regras para serem seguidos no desenvolvimento dos contratos de finalidades específicas. O ERC-20 e o ERC-721 são exemplos destes padrões e ditam como construir *tokens* fungíveis e não fungíveis na plataforma de *blockchain* Ethereum. Existem vários **Mercados**, camada que também é conhecida como Marketplaces, que surgiram em torno dos NFTs. Eles permitem que os usuários comprem e vendam seus *tokens* por meio de uma interface centralizada. Estes incluem OpenSea, Rarible, CrypoPunks e vários outros. A camada superior e que tem os **tokens** como finalidade, abrange a aplicação dos *tokens* fungíveis e não fungíveis e dentre estes os tangíveis e intangíveis. Para a criação e negociação dos NFTs, torna-se necessário o gasto com seu registro na plataforma e taxas pagas aos mercados. Esse pagamento utiliza normalmente um *token* fungível, como por exemplo o Ether, para efetivação das transações e gerenciados por uma aplicação de carteira. Consequentemente, para obter os valores em criptomoedas, o usuário necessita realizar um investimento com moedas ou qualquer outro ativo tangível por meio das Exchanges.

Os desafios enfrentados pelos NFTs vão além dos problemas já conhecidos para as *blockchains*. Estes desafios devem ser cuidadosamente enfrentados dada a grande aplicabilidade e movimentação financeira dos NFTs. O registro de NFTs pode ser realizado de maneira a atender qualquer tipo de dados, porém os padrões estabelecidos até aqui e as plataformas *blockchains* não fornecem suporte para armazenar conteúdos multimídia ou de grande volume. Uma vez que os conteúdos registrados como um NFT não são armazenados ou mantidos sob custódia da própria plataforma ou contrato, garantir a segurança de acesso apresenta como um desafio para os contratos de NFTs. Os ativos a qual os registros de NFTs representam, podem simplesmente serem alterados em seus locais de origem ou até mesmo não existirem mais, causando uma distorção do que foi negociado e do que se tem em posse. Entre os desafios dos NFTs, discutimos ainda a usabilidade, segurança e legislação na Seção 2.5.

Assim, o objetivo deste capítulo é apresentar e mais detalhes os principais conceitos, aplicações e desafios relacionados aos NFTs. Apresentamos uma visão teórica das

[10] `https://docs.openzeppelin.com/`

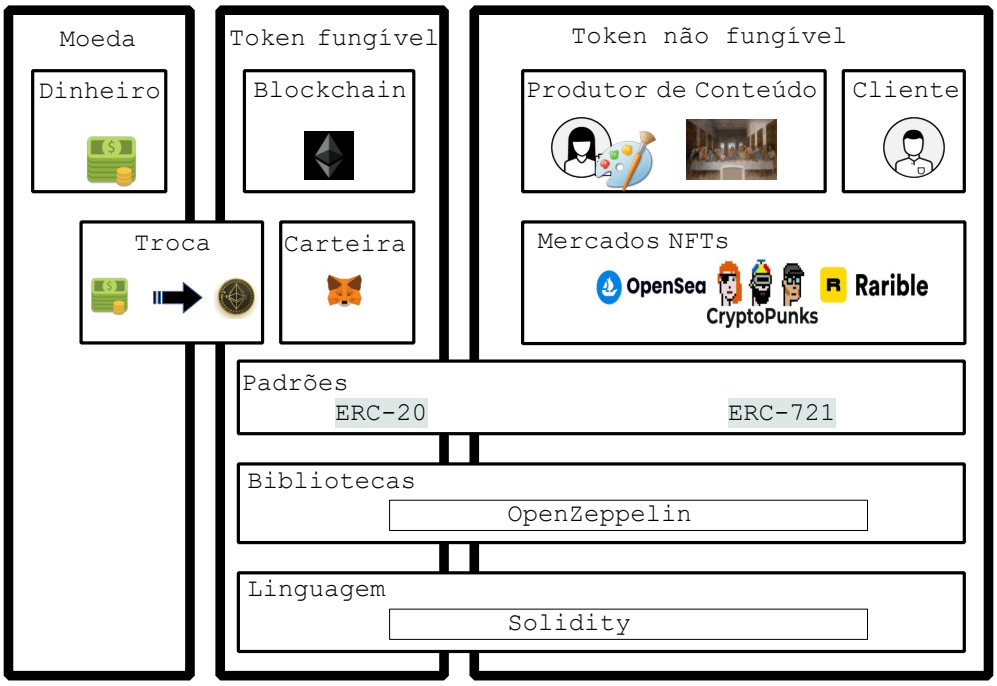

Figura 2.2. Arcabouço NFT.

tecnologias envolvidas para a criação e manutenção dos NFTs e aplicações de Marketplaces que fazem negociações dos *tokens*. Além disso, apresentamos as questões práticas para a utilização dos padrões de *tokens* e a criação de mercado para comercialização dos NFTs. Será abordada ainda a motivação para o estudo da área destacando o avanço tecnológico dos NFTs desde a sua concepção, popularização e utilização de forma autônoma em diversas aplicações. Além disso, será discutido onde e em quais situações utilizar os *tokens*, evidenciando o grande potencial dessa tecnologia em vários setores devido à sua inovação e aplicabilidade. Esse potencial não se restringe apenas a transações financeiras, sendo que há também a utilização em artes, games, certificado de propriedades e várias outras aplicações [Bao and Roubaud 2022]. Serão demonstradas as principais plataformas utilizadas para o desenvolvimento de um NFT e as aplicações descentralizadas (*decentralized applications* – DApps), que são invocadas de acordo com as condições e os códigos dos padrões de contratos NFTs [Wang et al. 2021]. O restante deste capítulo está organizado da seguinte forma. A Seção 2.2 apresenta a visão geral de todos os conceitos envolvidos para um bom entendimento dos NFTs. A Seção 2.3 define os *tokens* e os padrões criados para os mesmos na plataforma Ethereum. As questões de segurança são exploradas na Seção 2.4. A Seção 2.5 elenca os desafios de pesquisa e as perspectivas futuras em relação a usabilidade, privacidade e legislação. A prática envolvendo os padrões de *smart contract* para os *tokens*, bibliotecas e Dapps são desenvolvidas na Seção 2.6. Por fim, a Seção 2.7 conclui o capítulo.

2.2. Visão geral

Esta seção apresenta uma visão geral sobre *tokens* não fungíveis [Wang et al. 2021], cobrindo os temas dos conceitos de sistemas distribuídos, *blockchains*, consenso, padrões

de contratos e plataformas *blockchain* [Chohan 2021]. Primeiramente são apresentadas uma introdução e explicação sobre o funcionamento das *blockchains*. A seguir, são descritos os componentes que formam os contratos inteligentes, como eles trabalham, a definição de termos usados e os passos para o processo de criação e uma apresentação e discussão sobre as plataformas existentes, como Ethereum [Buterin et al. 2014] e Hyperledger [Androulaki et al. 2018]. Por fim, são definidos os conceitos de aplicações e finanças descentralizadas (Dapp, DeFi), oráculos e organizações autônomas descentralizadas (DAO). Finalmente, apresentamos exemplos de aplicações de uso dos NFTs e suas características em relação aos padrões.

2.2.1. Blockchain

Blockchain é uma tecnologia que contém funcionalidades de armazenar registros de transações de maneira imutável e distribuída. A topologia das redes *blockchains* pode se dar de diversas formas, porém na maioria das vezes são concebidas por inúmeros participantes independentes, e que não há necessidade de um controle centralizado. Os dados enviados à rede são organizados em blocos encadeados por meio de *hashes* criptográficos [Nakamoto and Bitcoin 2008]. Esses blocos são compostos por cabeçalho e uma lista de transações conforme ilustra a Figura 2.3 e que demonstra a arquitetura de um bloco e suas interligações apontadas pelas setas.

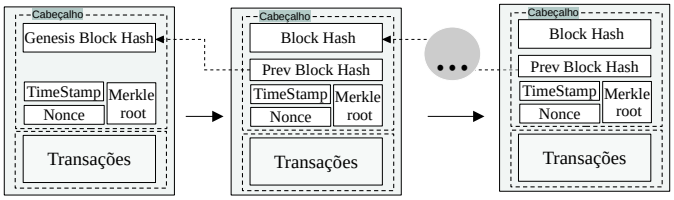

Figura 2.3. Cadeia de blocos.

Cada bloco é interligado ao outro por um endereçamento baseado em um *hash* criptográfico do bloco anterior e ilustrado pela seta pontilhada. À partir do bloco gênesis, isto é, o primeiro bloco da cadeia, o *hash* é calculado contendo o endereço do último bloco. Este processo garante que caso algum bloco da cadeia seja alterado de forma maliciosa, todo o restante será invalidado. As transações são mensagens submetidas ao consenso da *blockchain* e assinadas pelo endereço que as submeteram. Ao serem aceitas, elas alteraram o estado da rede e são armazenadas permanentemente e imutáveis. Além da lista de transações contidas dentro do bloco, o bloco contém os seguintes campos:

- ***Prev Block Hash:*** este campo faz uma referência aos pais, que é um link de um bloco ao seu anterior na cadeia. Todas as informações dentro do bloco anterior serão inseridas em uma função *hash* para obter um valor, então este valor será atribuído ao campo *Prev Block Hash* no novo bloco.
- ***Timestamp***: que contém a hora em que o bloco foi anexado.
- ***Merkle Root:*** contém o valor de *hash* de todas as transações validadas do bloco. Todas as transações são criptografadas em um valor de *hash* que representa a raiz de Merkle.

- ***Nonce:*** campo com valor que é usado pelo protocolo de consenso para comprovar o esforço despendido para anexar o bloco à cadeia.

Existem inúmeros e diversificados problemas que podem ser solucionados com a aplicação da tecnologia *blockchain*. Dentre estes problemas, podemos citar a validação de acesso, integridade e interoperabilidade de dados, rastreabilidade e até mesmo a contatação e verificação de propriedade [Gordon and Catalini 2018].

A utilização do conceito de contratos inteligentes pela tecnologia *blockchain* aumenta as possibilidades de uso e seu funcionamento. Os contratos são implementados em uma determinada linguagem de programação por meio de scripts e armazenados na rede. As regras dos contratos são executadas pela rede da forma como foram estabelecidas.

Os tipos de *blockchains* são classificados basicamente de acordo com a forma de acesso e proteção das transações armazenadas [Monrat et al. 2020]. Os tipos encontrados são as públicas, privadas e de consórcio. As especificações de protocolo de consenso, proteção de acesso às informações e controle da forma de distribuição diferenciam os tipos de *blockchain* e são apresentados na Tabela 2.2.

Tabela 2.2. Tipos de *blockchains*.

	Centralização	**Consenso**	**Acesso a Dados**
Pública	Nenhuma	Livre	Públicos
Privada	Sim	Restrito	Públicos ou restritos
Consórcio	Parcial	Restrito	Públicos ou restritos

Em uma *blockchain* privada as respostas são mais rápidas e seguras, porém o controle é exercido por um proprietário específico e os nós precisam de permissão para ingressarem na rede . A *blockchain* privada é mais rápida, eficiente e segura [Monrat et al. 2020]. Na *blockchain* pública, por sua vez, a rede é totalmente descentralizada e pode conter vários nós e qualquer nó pode se ingressar à rede [Xiao et al. 2020]. Porém, apenas nós sincronizados são utilizados para consenso. Uma *blockchain* de consórcio é composta por nós de organizações específicas que se organizam e controlam quem pode ter acesso à rede. A rede resultante do consórcio é parcialmente descentralizada [Rouhani and Deters 2017, Wang et al. 2018].

Os protocolos de consenso trabalham com algoritmos de forma distribuída, com o intuito de gerar um novo bloco e ser aceito entre os nós da rede. Os algoritmos consistem basicamente na solução de um dilema para obter um consenso entre os nós da rede sobre a validade de uma transação. O objetivo destes algoritmos é obter o consenso entre os nós validadores e ignorar qualquer tentativa de obstrução. A comparação entre os mecanismos de consenso seguem os critérios de eficácia energética, entrada de novos nós e tolerância a falhas. Um estudo mais aprofundado desses algoritmos pode apresentar mais detalhes sobre a capacidade de armazenamento e o tempo gasto para gerar os blocos em cada algoritmo. A escolha do mecanismo de consenso pode ter um impacto significativo no desempenho da solução pretendida. Portanto, deve-se considerar os requisitos da aplicação referente ao armazenamento de dados e tempo de resposta.

2.2.2. Contratos Inteligentes

Contratos Inteligentes (*Smart Contracts*) são programas auto executáveis e auto impositivos que funcionam de acordo com condições previamente acordadas [Hewa et al. 2021a]. Esses contratos são capazes de implementar determinadas operações dentro da *blockchain* e funcionam como parte das aplicações descentralizadas. Entre os benefícios da utilização de contratos inteligentes apontados por [Hewa et al. 2021a] estão a eliminação da necessidade de um terceiro confiável para efetuar transações, a integridade e a transparência das transações e a autonomia de execução e a precisão dos contratos, uma vez que são imutáveis e são executados quando as pré condições são atendidas.

Devido ao caráter complementar que os contratos possuem em relação à *blockchain*, eles se tornaram parte essencial das *blockchains* que surgiram após o Bitcoin, proposto por [Nakamoto and Bitcoin 2008]. Com a utilização desses contratos, as *blockchains* tiveram sua capacidade de armazenamento aprimorada. Um contrato permite definir um comportamento para um determinado estado e atender necessidades de aplicações diversas. Nesse sentido, os contratos inteligentes são capazes de executar transações muito mais complexas dentro da *blockchain* do que simplesmente a troca de moedas.

A partir dessas características, inúmeras aplicações podem ser desenvolvidas baseadas no uso de contratos inteligentes como, por exemplo, a certificação de propriedade feita pelos NFTs, aplicações financeiras (gerenciamento de moeda, serviço de garantia, procedimentos de auditoria, empréstimo), aplicações médicas (gestão de informações de saúde, proteção de dados de pesquisa clínica, monitoramento e tratamento automatizado de pacientes, gerenciamento de identidade e controle de acesso, proteção de dados de identidade), aplicações imobiliárias, aplicações de acordos contratuais, aplicações de internet das coisas, aplicações de serviços de telecomunicações, aplicações de gestão de logística, além de aplicações entre diferentes indústrias [Hewa et al. 2021a].

2.2.3. Plataformas *blockchains*

A evolução da tecnologia *blockchain* provocou a criação de novas plataformas e com diferentes características e parâmetros. As plataformas são basicamente a reunião de procedimentos que determinam o funcionamento da *blockchain* mediante os requisitos apresentados pelas aplicações definidas para seu uso. As características ou parâmetros específicos propostos pelas plataformas compreendem em qual domínio ela irá operar e qual tipo de *blockchain* ela implementa. As plataformas são do tipo permissionada ou não permissionada. O mecanismo de consenso adotado pela plataforma também a caracteriza juntamente com a capacidade e suporte de uso de *smart contracts* [Monrat et al. 2020]. A seguir, mencionamos duas destas plataformas e que estão diretamente envolvidas com o desenvolvimento dos NFTs.

Ethereum é uma plataforma popular do tipo pública, livre de permissão e possui o Ether como moeda principal [Buterin et al. 2014]. Ela possui tecnologia que possibilita o desenvolvimento e execução de contratos inteligentes. O uso deste recurso resulta na implementação das chamadas aplicações descentralizadas (DApps) que são executadas dentro da *blockchain*. Nesta plataforma é necessário o gasto de Ethers para realizar a inserção de um Contrato Inteligente que é utilizado como recompensa ao responsável por criar o bloco deste contrato. Por sua vez, uma rede *blockchain* permissionada

se mostra como uma opção à exposição pública da plataforma. Hyperledger Fabric é uma plataforma privada e permissionada que possibilita o desenvolvimento de aplicações que equilibram os gastos de custo e desempenho em relação às redes *blockchain* públicas [Cachin et al. 2016]. Por ser privada, a escrita de transações é controlada por um grupo de membros da rede, e o acesso à leitura de dados e transações são limitados aos membros e usuários autorizados. Neste caso, por ser uma rede permissionada, os usuários da rede são controlados por autenticação de identidade. Os contratos inteligentes são definidos como Chaincode e as chamadas a ele é que podem alterar o estado da rede. O consenso é realizado por uma implementação do algoritmo *Practical Byzantine Fault Tolerance*.

2.2.4. Carteiras

Uma carteira representa um par de chaves que compreende em uma chave pública e uma chave privada [Borkowski et al. 2019] . Normalmente, em *blockchain* é utilizada como um ambiente que permite aos usuários terem controle sobre as transações realizadas com suas criptomoedas e *tokens*. Por meio de uma aplicação de carteira é que o usuário de sistema *blockchain* consegue ingressar à rede e realizar transações [Hasanova et al. 2019]. A inserção e validação de transações normalmente é feita por uma aplicação de carteira que necessita da assinatura das transações por meio da chave privada do usuário. Este processo de assinatura é responsável por decifrar as transações utilizando a chave pública do usuário, procedimento denominado criptografia assimétrica, que proporciona segurança ao constatar a veracidade dos autores das transações [Pal et al. 2021].

A aplicação de carteira deve conter atributos que consideram principalmente a segurança e anonimato do usuário. Dentre estes atributos estão o controle, validação, transparência, vulnerabilidade e privacidade. Sendo que a contenção dos ativos devem ficar no controle do próprio usuário, não havendo possibilidade de congelamento ou perda dos mesmos [Borkowski et al. 2019]. A verificação e validação das transações deve ser realizada pela carteira sem processamento por terceiros, tornando-a independente. O código-fonte da carteira deve ser transparente ao ponto de possibilitar a auditoria e verificação das funções executadas. As vulnerabilidades devem ser contempladas no projeto ao ponto de indicar as fraquezas e solucioná-las. Quanto à privacidade, a carteira deve dispor de ferramentas que inibem o rastreamento do usuário e transações por meio, por exemplo, de anonimização e roteamento de endereços [Pillai et al. 2019].

2.2.5. Oráculos, Aplicações e Finanças descentralizadas

As aplicações descentralizadas ou *Decentralized Applications* (DApps) são um tipo especial de aplicação que são desenvolvidas baseadas em contratos inteligentes. Em outras palavras, os DApps são softwares que dependem de contratos inteligentes implantados em uma plataforma *blockchain*.

As finanças descentralizadas ou *Decentralized Finance* (DeFi) são uma opção ao sistema controlado por processos centralizados e mantidos por instituições financeiras. A DeFi oferece um controle transparente e descentralizado mediante um sistema aberto e criado especificamente para a era digital.

Por muitas vezes, as DApps necessitam recuperar dados externos a uma plata-

forma *blockchain*. Porém os contratos inteligentes são invocados por eventos ou funções externas por participantes do *blockchain* e não são capazes de realizar a recuperação de dados externos. Isto é, extrair dados externos ao da plataforma onde o contrato está implantado não é uma tarefa trivial. Portanto, os dados externos devem ser inseridos nos contratos. Os oráculos são basicamente uma ponte entre a parte interna e externa da plataforma *blockchain*. Estes oráculos precisam necessariamente advir de fontes confiáveis uma vez que inserem um nível de insegurança e complexidade nas DApps. Dado que este processo de consulta aos oráculos torna esta ação centralizada, é aplicado um consenso entre os oráculos para definir a confiança da informação advinda do processo [Stradling and Voorhees 2018].

2.2.6. Organizações autônomas e descentralizadas

Uma organização autônoma descentralizada, (*Decentralized Autonomous Organizations* – DAOs), é constituída de regras e princípios de governança automatizados por contratos inteligentes transparentes, são controlados pelos membros da organização, mas sem interferência centralizada [El Faqir et al. 2020]. O objetivo principal das DAOs é de criar uma forma de governança coletiva utilizando uma infraestrutura descentralizada. As regras e princípios são comumente escritas em linguagem de programação para contratos inteligentes e implantadas em uma *blockchain* pública. Os membros pertencentes a uma organização votam para o consenso de uma determinada proposta em um determinado tempo. O resultado da aprovação da proposta poderá ser consultado após processo da determinação do consenso. As DAOs podem resolver o problema de confiança no compartilhamento de NFTs com outros e reduzir o processo de colaboração remediando falhas da colaboração tradicional e permitindo a colaboração em escala [Jentzsch 2016].

2.2.7. Cenários e aplicações de NFTs

A inovação e criatividade no mundo dos NFTs tem realizado ainda mais avanços aos padrões e as plataformas que os implementam. À medida que o universo NFT evolui, a tecnologia é adotada por diversos cenários e são produzidas inúmeras aplicações que entregam soluções interessantes. Podemos citar vários exemplos de uso interessante que envolvem experiências de jogos *blockchain*, arte digital, registro seguro, propriedade de objetos/colecionáveis digitais e físicos, direito autoral e muito mais. Os casos de uso que envolvem em seus planejamentos o aluguel de NFTs trazem diversos benefícios em seus cenários e podem ser integrados aos mercados. Alguns exemplos reais de cenários que utilizam NFTs incluem:

- Arte digital

  A possibilidade de aluguel de itens para exposições de arte no mundo virtual. Sendo que este possa ter o tempo controlado e/ou expirado por contratos automatizados. A exibição nestes expositores pode ainda ser realizada de forma distinta e selecionada, gerando um rendimento para itens de arte menos conhecidos.

- Metaverso

  O aluguel de terrenos virtuais pode ser comparado ao modo realizado no mundo real. Possibilidades de aluguel de espaço em anúncio virtual são inseridas neste

cenário onde podem ser explorados espaços por tempo de exibição e até mesmo baseado em tráfego naquele local.

- Jogos

  No cenário de jogos já é muito comum a existência de itens escassos e raros que podem aumentar a capacidade e a jogabilidade. Assim estes itens alugáveis podem ajudar a avançar ou superar certos desafios dos jogos em um determinado momento ou fase.

- Domínio (URL)

  Um nome registrado em espaços virtuais pode ser explorado sob a demanda por eles sem a necessidade de venda ou troca de titularidade permanente. Resultando em uma renda passiva por meio do aluguel.

- Mercados

  Por se tratar de uma extensão do padrão ERC-721, a funcionalidades de aluguel dos NFTs podem ser diretamente integradas aos mercados preexistentes como por exemplo o OpenSea.

- Pessoa pública

  As pessoas públicas estão criando NFTs e disponibilizando às suas comunidades para permitirem benefícios como, por exemplo, acesso a fã clubes, exclusividades em visualização de conteúdos e descontos em produtos de marca própria.

2.3. *Tokens*

Nesta seção tratamos dos conceitos de *tokens* e os aspectos das estruturas de um *token*, tais como seus componentes e padrões. Descrevemos os *Ethereum Request for Comments* (ERC), tais como ERC-20, ERC-721 e ERC 1155, que são padrões de contratos criados no nível de aplicação no ecossistema Ethereum para *tokens* e propostos nos *Ethereum Improvement Proposals* [Wang et al. 2021]. Eles são um fator importante para o sucesso dos NFTs pois são padrões abertos que descrevem como construir *tokens* não Fungíveis em *blockchains* e possui um conjunto de regras que facilitam o trabalho do desenvolvedor. Também descrevemos as formas que os padrões podem ser utilizados para troca de propriedade e serem negociados por outros *tokens* ou criptomoedas.

Token pode ter o significado de uma ficha ou um código que representa algo. Na tecnologia, a palavra *token* é atribuída a dispositivos ou sistemas que geram codificações de acesso e autenticação. Este termo significa a representação digital de um ativo quando no universo de uma *blockchain*. Portanto é utilizado para descrever a criação de um registro digital de um ativo tangível, como por exemplo o ouro, dinheiro, obras de arte, imóveis, equipamentos, etc. Ou ainda um ativo intangível como por exemplo software, criptomoedas, artes digitais, etc.

Sendo assim, conforme representados na Figura 2.4, os ativos tangíveis ou intangíveis podem ser protegidos com a ajuda da tecnologia *blockchain* e representados como um ativo digital por um *token* fungível ou não. Podendo representar dinheiro, certificados

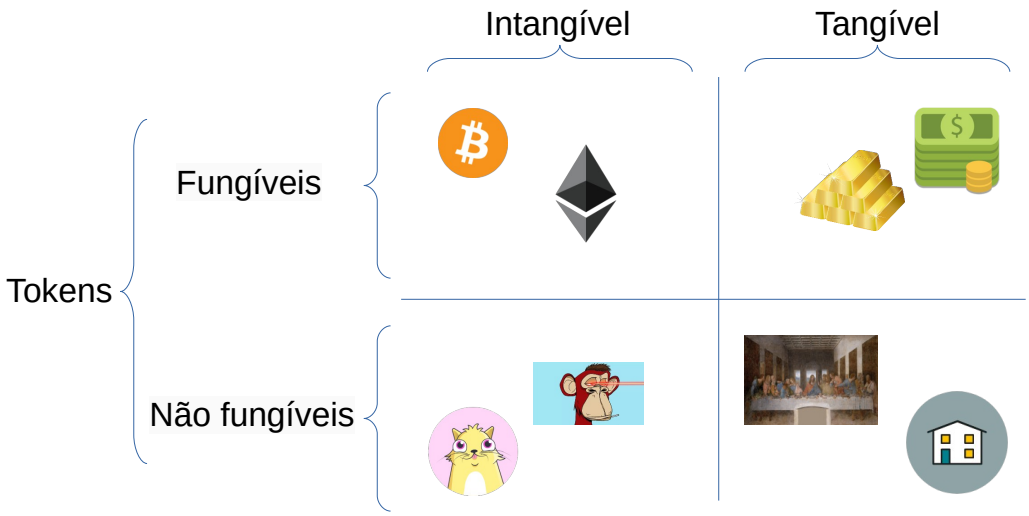

Figura 2.4. *Tokens* Fungíveis e Não fungíveis

de propriedades e outros ativos. As próprias criptomoedas podem ser consideradas como *tokens*. Existem, portanto, vários tipos de *tokens* e se distinguem de acordo com a proposta a qual forem designados. Dentre eles os principais tipos são: *tokens* fungíveis ou de pagamentos, utilidades, não fungíveis e de segurança.

- O *token* fungível (FT) ou de pagamento exerce o papel de dinheiro eletrônico e são operados em transferência e ou pagamentos. Os exemplos mais conhecidos deste tipo de *token* são o Bitcoin e o Ether.

  Definição 1. *Um FT é idêntico a todos os outros tokens em valor, para a mesma classe de tokens e é igualmente intercambiável com qualquer outro dentro de uma determinada classe.*

- Os *tokens* de utilidades são usados no oferecimento de algum benefício ao cliente ou usuário de um serviço ou produto. Eles fornecem utilidades como descontos, acessos a serviços especiais dentre outros.

- Os NFTs, são *tokens* que podem representar algo singular, i.e, não pode ser trocado por outro de mesmo valor como dinheiro porque é único. Eles correspondem à identificação de um item único e vinculado a um proprietário. Em suma, um NFT é um código que contém o registro do objeto e seu proprietário. A maioria dos projetos de NFTs são originados da plataforma Ethereum ou a utilizam para prover garantias. No entanto, existem uma gama de outros projetos baseados em plataformas diferentes e que demonstram a independência da utilização dos NFTs de qualquer plataforma[11].

  Definição 2. *Um NFT é um token exclusivo dentro de uma determinada classe de token e não é igual a nenhum outro NFT dentro de uma determinada classe.*

[11] https://chaindebrief.com/non-ethereum-nft-projects-zilliqa-solana/

- Os *tokens* de segurança são uma nova modalidade de propriedade de ações, títulos e produtos regulados por instituições financeiras. Representam uma participação externa em um ativo e podem ser emitidos em *blockchain* públicas ou permissionadas.

2.3.1. Estrutura e Padrão de *tokens* na plataforma Ethereum

A estrutura de um *token* na plataforma blockchain Ethereum é realizada fundamentalmente em um contrato inteligente. Esta estrutura contém basicamente um mapa de endereços de contas e seus saldos vinculados. O saldo, chamado de *token*, simboliza um valor, que também pode pode ser utilizado na representação de objetos, valores monetários e outras finalidades, conforme mencionadas nos itens da seção anterior. As operações sobre esta estrutura discorrem em transferências da propriedade destes *tokens*. Ao realizar esta transferência, os contratos atualizam o saldo das duas contas envolvidas, creditando-o em uma das contas e debitando na outra.

Os padrões de *tokens* foram sugeridos inicialmente na plataforma Ethereum. Na plataforma Ethereum, quando surge uma nova proposta e que altera um procedimento deve-se fazer por meios de uma Proposta de melhorias no ethereum ou *Ethereum Improvement Proposal* (EIP). Esta proposta consiste em um documento que detalha as alegações de mudanças e as questões técnicas envolvidas. Por meio dos EIPs é possível submeter proposições de comentários, mudanças em protocolos e outros detalhes da Ethereum. No caso dos *tokens*, os documentos são submetidos como solicitação de comentários Ethereum ou *Ethereum Request for Comments* (ERC) que é a forma de definir padrões de uso na Ethereum. As EIPs submetidas passam por um ciclo de vida e recebe um status em cada etapa, que vai desde o rascunho da ideia até a versão final. As EIPs que tratam de conteúdos técnicos e padrões referentes aos contratos inteligentes passarão a ser nomeadas com a sigla ERC após a aprovação na fase final. Existem vários EIPs/ERCs que foram submetidos e aprovados como padrões por meio deste ciclo e que são comumente utilizados para definir tipos de *tokens*. Dentre eles citamos os padrões ERC-20, ERC-721 e ERC-1155 que tratam dos *tokens* fungíveis e *tokens* não-fungíveis respectivamente.

Inicialmente o padrão ERC-20 foi introduzido como uma experiência de proporcionar recursos para padronização de criação de contratos de *tokens*. Este padrão trouxe muitos benefícios, permitindo uma integração de vários *tokens* com uma mesma carteira e listagem em plataformas digitais onde é possível comprar, vender, trocar e guardar os *tokens*. Estes benefícios avançaram para outros padrões de outros tipos de *tokens* conforme apresenta a Tabela 2.3.

2.3.2. Padrão ERC-20

O ERC-20 é um padrão de contratos inteligentes que permite a criação de *tokens* fungíveis no Ethereum, isto é, *tokens* que são iguais e possuem o mesmo valor independente do índice que o representa. O padrão foi proposto por [Vogelsteller and Buterin 2015]. Por meio da implementação da interface deste padrão é possível utilizar inúmeras das funcionalidades pré-estabelecidas. Tais funcionalidades como as de transferência de *tokens* entre contas, verificação o saldo da conta, verificação do total de *tokens* criados e aprovação de *tokens* para utilização de terceiros possibilitam a reutilização e compatibilidade por aplicações como carteiras e exchanges descentralizadas. Há vários *tokens* já implantados

Tabela 2.3. Padrões de *tokens* Ethereum com casos de uso

Padrão de *token*	Definições	Utilidade
ERC 20	Baseado em saldos Valor fungível	Oferta inicial de criptomoedas *token* de segurança *Tokens* de utilidade
ERC 721	Valor não fungível Cada elemento é único	Tokenização de ativos reais Registro de propriedade de ativos colecionáveis e virtuais
ERC 155	Combinação dos padrões ERC-20 e ERC-721 Permite que diferentes *tokens* sejam configurados de um único ponto	Conjunto de ativos heterogêneo

e seguindo o padrão ERC-20 na rede Ethereum. Podemos encontrar em [Charles 2013], implementações de exemplo e que são utilizadas pelas Dapps. A Listagem 2.1 apresenta alguns dos métodos deste padrão e que são relevantes para nossa discussão e prática.

```
1  // SPDX-License-Identifier: UNLICENSED
2  pragma solidity ^0.4.20;
3  interface ERC721{
4  event Transfer(address, address, uint256)
5  event Approval(address, address, uint256)
6  function name() public view returns (string)
7  function symbol() public view returns (string)
8  function decimals() public view returns (uint8)
9  function totalSupply() public view returns (uint256)
10 function balanceOf(address) public view returns (uint256)
11 function transfer(address, uint256) public returns (bool)
12 function transferFrom(address, address, uint256) public returns (bool)
13 function approve(address, uint256) public returns (bool)
14 function allowance(address, address) public view returns (uint256)
15 }
```

Listagem 2.1. Padrão para *token* fungível: EIP-20

2.3.3. Padrão ERC-721

O padrão ERC-721, também conhecido como o padrão de NFTs, é utilizado na identificação de itens únicos e exclusivos. Este tipo de padrão fornece a capacidade ao *token* de realizar funções como certificação de propriedade, identificação de credenciais, identificação de acesso, ingressos para eventos, itens colecionáveis, etc.

Os NFTs possuem as características básicas do *token* ERC-20, porém apresentam um identificador exclusivo para cada registro tornando-os únicos e exclusivos. Uma interface padrão para os NFTs foi apresentada por [Entriken et al. 2018]. Este padrão pode ser considerado como um certificado de posse que permite a implementação de uma API padrão para NFTs utilizando os contratos inteligentes. As funcionalidades básicas para rastrear e transferir NFTs são apresentadas na Lista de código 2.2.

```
1  // SPDX-License-Identifier: UNLICENSED
2  pragma solidity ^0.4.20;
3  interface ERC721{
4    event Transfer(address, address, uint256);
5    event Approval(address, address, uint256);
6    event ApprovalForAll(address, address, bool);
7    function balanceOf(address) external view returns (uint256);
8    function ownerOf(uint256) external view returns (address);
9    function safeTransferFrom(address, address, uint256, bytes) external payable;
```

```
10    function safeTransferFrom(address, address, uint256) external payable;
11    function transferFrom(address, address, uint256) external payable;
12    function approve(address, uint256) external payable;
13    function setApprovalForAll(address, bool) external;
14    function getApproved(uint256) external view returns (address);
15    function isApprovedForAll(address, address) external view returns (bool);
16 }
```

Listagem 2.2. Padrão para *token* não fungível: EIP-721

2.3.4. Padrão ERC-1155

O padrão ERC-1155 é chamado com padrão multi-token e foi idealizado para controlar um número maior de *tokens* simultaneamente. Para abranger *tokens* fungíveis e não fungíveis, o ERC-1155 contém as funções do *token* ERC-20 e do ERC-721 [Radomski et al. 2018]. Por meio de uma interface padrão utilizada por um único contrato inteligente é possível gerenciar vários tipos de *token*. As funcionalidade básica desta interface são apresentadas na Lista de código 2.3.

```
1  // SPDX-License-Identifier: UNLICENSED
2  pragma solidity ^0.5.9;
3  interface ERC1155
4      event TransferSingle(address, address, address, uint256, uint256);
5      event TransferBatch(address, address, address, uint256[], uint256[]);
6      event ApprovalForAll(address, address, bool);
7      event URI(string, uint256);
8      function safeTransferFrom(address, address, uint256, uint256, bytes) external;
9      function safeBatchTransferFrom(address, address, uint256[], uint256[], bytes) external;
10     function balanceOf(address, uint256) external view returns (uint256);
11     function balanceOfBatch(address[], uint256[]) external view returns (uint256[]);
12     function setApprovalForAll(address, bool) external;
13     function isApprovedForAll(address, address) external view returns (bool);
14 }
```

Listagem 2.3. Padrão para Multi *token*: EIP-1155

Os padrões ERC-20 e ERC-721 requerem a implantação de contratos distintos por cada tipo de *token*. O padrão ERC-721, por exemplo, contém um ID do *token* que é um índice que representa um item não fungível desse grupo e é implantado como um único contrato que abrange configurações para todo o grupo de *tokens*. Já o ERC-1155 permite que cada ID de *token* represente um novo tipo de *token* configurável, que pode ter seus próprios metadados, funções e atributos. Tarefas como a transferência de vários tipos de *token* simultaneamente são possíveis por meio das funcionalidades adicionadas a este padrão resultando em economia nos custos de transação.

2.3.5. Negociação de *tokens*

Os *tokens* ERC-721 são negociados de forma diferente das criptomoedas e outros tipos de *tokens*. O valor dos NFTs geralmente depende de sua relação fora da *blockchain* em que são negociados e também a raridade imposta a eles [Rogers et al. 2022]. Alguns NFTs só podem ser comprados com criptomoedas, portanto, é necessário possuir parte dessa criptomoeda e armazená-la em uma carteira digital. A partir daí é possível comprar NFTs por meio de qualquer um dos mercados NFT online, incluindo OpenSea, Rarible e Cryptopunks. Quando ocorre uma transferência de *tokens* fungíveis, por exemplo do tipo ERC-20, são submetidos ao consenso da rede o débito e o crédito nos saldos das contas envolvidas na transação. Já a transferência de NFTs, isto é, o *token* não fungível, ocorre

a mudança da propriedade daquele registro individual e que não será agregado a nenhum outro *token*. A Figura 2.5 apresenta duas funções executadas na transferência e obtenção de dados de um NFT em forma sequencial.

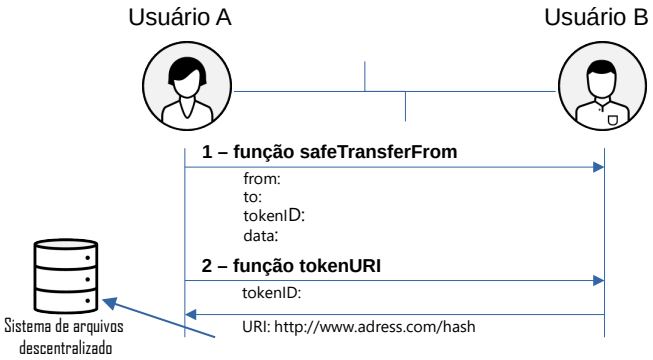

Figura 2.5. Sequência para transação de transferência de NFTs.

Neste caso, os usuários A e B realizam uma transferência da propriedade de um *token* com um ID específico. Onde o usuário A é o atual proprietário do *token* e o usuário B é quem receberá a propriedade do *token*. A transferência do *token* só se dá por meio da execução da função ***saveTransferFrom*** acionado pelo proprietário do *token*. Esta função recebe os dados de identificação do atual proprietário, identificação do usuário para quem vai ser transferido o *token*, juntamente com a identificação do *token* no contrato. A função ***tokenURI*** solicita o retorno do metadado URI armazenado ao criar o registro do *token*.

A compra e venda de NFTs deve ser de comum acordo entre as partes envolvidas, porém a efetivação da transferência só é realizada pelo vendedor. Apesar da iniciativa da compra ser comumente feita pelo comprador nos mercados de *tokens*, o comprador tem simplesmente a função de pagar o valor combinado pela transação. Normalmente os marketplaces realizam a intermediação de todo o procedimento da transferência e o pagamento ao proprietário do NFT.

O aluguel é uma outra modalidade e possibilidade de transação com os NFTs em que é permitido alugar e realizar o pagamento do aluguel de NFTs da mesma forma da venda. Porém, neste caso o NFT recebe o status de alugado, onde o usuário que o alugou pode usá-lo, mas não pode transferi-lo. Somente quando o aluguel terminar e o NFT voltar ao seu proprietário, o NFT poderá ser transferido novamente.

Como o pagamento nestes mercados é realizado normalmente por meio de criptomoedas, existem também mercados de criptomoedas, também chamados de exchanges, que oferecem aos usuários o recurso de poderem comprar e vender as criptomoedas. Este serviço é normalmente incrementado pelas exchanges para oferecerem trocas por moedas correntes tais como, por exemplo, o dólares ou euro. Há três tipos de exchanges de criptomoedas: exchanges centralizadas que são dirigidas por uma empresa ou organização, exchanges descentralizadas que fornecem processos automatizados para negociações entre o próprios usuários e exchanges híbridas que combinam as duas opções [Xia et al. 2020]. Os usuários se dispõem de saldos em sua aplicação de carteira, ao adquirirem as criptomoedas, que poderão ser utilizadas para negociação dos NFTs em seus marketplaces.

2.4. Segurança

Nesta seção serão apresentados os aspectos principais relacionados à segurança no contexto de *smart contract* e consequentemente os NFTs. De fato, nos últimos anos, diferentes estudos tratam das pesquisas relacionadas à segurança nos ambientes voltados aos contratos inteligentes [Rouhani and Deters 2019, Harz and Knottenbelt 2018], discutindo a importância e constante evolução deste aspecto. O conceito de segurança em uma rede está diretamente relacionado aos métodos de ataque, de modo a obter informações sensíveis dos elementos e do próprio estado da rede, e também aos mecanismos de defesa a esses ataques. Além disso, um aspecto fundamental para ambos é a privacidade dos usuários ativos na rede, garantindo que dados sensíveis de identificação e localização sejam preservados [Sayeed et al. 2020].

É importante salientar que os contratos inteligentes para NFTs sejam exaustivamente testados. Os prejuízos causados por falhas podem ser catastróficos se os contratos inteligentes contiverem erros. Portanto, devem ser testados para garantir que façam exatamente o que foi proposto e sem falhas. A qualidade e integridade dos testes devem passar por uma eficiente averiguação de eficácia e demonstrar as possíveis vulnerabilidades existentes [Hartel and Schumi 2020, Hewa et al. 2021b].

2.4.1. Avaliação de segurança

Os NFTs geralmente são seguros, uma vez que usam a tecnologia *blockchain* assim como as criptomoedas. A natureza distribuída das *blockchains* torna os NFTs difíceis de serem atacados. Porém existem vários problemas de segurança que permitem que mineradores ou usuários maliciosos os explorem e ganhem lucro de maneira injusta [Luu et al. 2016].

Boas práticas devem ser seguidas ao criar contratos inteligentes para NFTs para que sejam seguros. Com isso é possível identificar os ataques comumente utilizados e garantir que *tokens* não sejam perdidos devido a erros de programação. Alguns problemas de segurança encontrados em contratos inteligentes e que são comumente explorados por usuários mal intencionados estão descritos abaixo.

Dependência de ordem de transação - Um ataque de dependência da ordem de transação pode gerar um resultado inesperado para os usuários simplesmente se houver invocações simultâneas. Também pode ocorrer de um usuário mal-intencionado explorar as dependências de ordem dos contratos para obter vantagens. Nestes casos, a invocação do contrato para alterar o estado da rede pode não conhecer individualmente o estado, uma vez que outras transações também podem ter sido invocadas. Dado que o minerador do bloco é quem decide a ordem das transações, e consequentemente a ordem das atualizações, o estado final de um contrato pode depender da ordem em que foram invocadas as transações. Exemplificando, caso um item NFT seja anunciado por um preço, o usuário espera pagar esse preço pelo item. Uma tentativa desse ataque poderia alterar o preço do item antes do processamento da transação de compra. Isso demonstra que este contrato depende de um valor que pode ser alterado de acordo com a ordem das transações.

Dependência de *Timestamp* - Este problema de segurança afeta os contratos que utilizam o *timestamp* do bloco como condição para executar alguma tarefa. A definição de data e hora é realizada pelo nó minerador, sendo que este pode variar o valor de tempo

em aproximadamente 900 segundos, que mesmo assim o bloco pode ser validado na rede pelos mineradores. Um exemplo de contrato dependente de *timestamp* é o contrato de aluguel apresentado na Seção 2.6.5. Nele é possível observar que a função de finalizar o contrato pode se dar automaticamente dado um determinado período de expiração baseado no *timestamp* do bloco. Caso o valor do *timestamp* seja determinado de maneira a estender ou encurtar o tempo, dependendo da aplicação e cenário envolvido, os resultados podem ser diferentes do esperado para esta função.

Dependência de terceiros - Um outro grande risco de segurança para os NFTs está relacionado diretamente a confiança nos mercados que comercializam os *tokens*. Existe então a possibilidade de perda do acesso ao *token* se o mercado que hospeda o NFT deixar de existir ou não houver maneira de entrada na plataforma, seja por perda de dados das credenciais de acesso ou mesmo por acesso malicioso, o registro do *token* ficará inutilizado.

Sabemos que a *blockchain* é uma tecnologia dita como segura, mas o que se usa para comunicar com a *blockchain* pode não ser tão seguro assim. A camada de interface normalmente não é implementada de forma distribuída. Na maioria dos casos, estas interfaces são sites comuns ou com apenas algumas variações de tecnologia semelhante a isso. Sendo assim, o desenvolvedor da aplicação deverá implantar uma interface que será utilizada pelos usuários. Consequentemente, um nó será necessário para conectar essa interface à *blockchain*. Desta forma os desenvolvedores são e dependem de entidades centralizadas já que normalmente não executam seus próprios nós da *blockchain* e não criam suas interfaces específicas para cada aplicação. As questões relacionadas às interfaces para a *blockchain* não significa que haja problemas fundamentais com a própria *blockchain*. Isto apenas demonstra que há uma vulnerabilidade em camadas superiores e que devem ser levadas a sério como uma oportunidade disruptiva para que sejam construídas de maneiras mais robustas.

2.4.2. Mecanismos de defesa

A principal característica de segurança apresentada aos usuários pela *blockchain* é a transparência em que as transações são dispostas. Seja qual for o usuário, no caso de redes públicas, usuários sem necessidade de autorização, e em redes privadas, os usuários autorizados, eles conseguem auditar os dados históricos. Desta forma é extremamente difícil esconder algo na *blockchain*, necessitando da criptografia como requisito essencial de segurança dos dados para o funcionamento da tecnologia [Johnson et al. 2019].

A Tabela 2.4 apresenta alguns possíveis problemas de segurança e medidas de defesa que podem ser tomadas para sanar os problemas.

2.5. Desafios

Aqui discutimos os desafios de pesquisa e também os desafios da tecnologia além de algumas possíveis direções de pesquisa. Dada a grande aplicabilidade dos NFTs, a pesquisa apresenta várias possibilidades, desde desenvolvimento de técnicas para melhorar o desempenho até padronizações para prover verificação.

Como desafios de pesquisa encontramos ainda a necessidade de melhorias nos

Tabela 2.4. Potenciais problemas de segurança e soluções correspondentes de NFTs. Adaptada de [Wang et al. 2021]

Vulnerabilidade	Questão de segurança	Solução
Falsificação (Autenticidade)	• Um invasor pode explorar vulnerabilidades de autenticação • Um invasor pode roubar a chave privada de um usuário.	• Uma verificação formal do contrato inteligente. • Usar uma carteira para evitar o vazamento de chave privada.
Adulteração (Integridade)	• Os dados armazenados fora do *blockchain* podem ser manipulados.	• Envio de dados originais e dados de *hash* para o comprador de NFT ao negociar NFTs.
Repúdio (Não repudiabilidade)	• Os dados de *hash* podem ser vinculados ao endereço de um invasor.	• Uso parcial de um contrato com várias assinaturas.
Divulgação de informação (Confidencialidade)	• Um invasor pode facilmente explorar o *hash* e a transação para vincular um determinado comprador ou vendedor de NFT.	• Usar contratos inteligentes que preservam a privacidade em vez de contratos inteligentes para proteger a privacidade do usuário.
Negação de serviço (Disponibilidade)	• Os dados NFT podem ficar indisponíveis se o ativo for armazenado fora do *blockchain*.	• Usando a arquitetura *blockchain* híbrida com algoritmo de consenso fraco.
Elevação de privilégio (Autorização)	• Um contrato inteligente mal projetado pode fazer com que os NFTs percam tais propriedades.	• Uma verificação formal dos contratos inteligentes.

processos de desenvolvimento, como mais suporte para depuração, garantia de segurança, melhores ferramentas para facilitar o desenvolvimento e as dificuldades em realizar testes nas plataformas existentes [Zou et al. 2019]. O alto custo de falhas e a dificuldade de mudanças após implantados, torna os NFTs um desafio ainda maior [Dolgui et al. 2020]. Os direcionamentos e esforços de pesquisas para solucionar os desafios enfrentados pelos NFTs percorre dentre estes outros a revogação de certificados e *tokens*, geração de números aleatórios, paralelização de códigos e execução de aplicações em dispositivos com baixa capacidade de processamento e memória [Kemmoe et al. 2020]. Apesar de ser possível registrar um NFT para qualquer tipo de dados, os padrões estabelecidos até aqui e as plataformas *blockchains* não foram projetados para armazenar conteúdos multimídia e de grande volume. Isso apresenta como um desafio para como os NFTs mantêm protegidos algo que não está sob sua custódia. Entre os desafios da tecnologia, apresentamos a usabilidade, segurança, legislação e eficiência energética. Além destes desafios, os NFTs ainda são sujeitos aos desafios de escalabilidade e interoperabilidade que são inerentes à *blockchain*.

2.5.1. Usabilidade

O trabalho para aumentar a usabilidade e a eficácia das plataformas *blockchains* necessitam ainda de muitos esforços. Várias questões relacionadas a facilidade de uso da tecnologia precisam ser melhor estudadas para permitirem a viabilidade de uso eficaz das aplicações que demandam de muitos recursos para proverem uma interação facilitada [Pillai et al. 2017].

No contexto dos NFTs, a interoperabilidade entre plataformas contribui para a usabilidade ao implicar a capacidade de transferir um ativo entre mercados distintos, mantendo o estado consistente. A interoperabilidade de mercados deve atingir a eficiência de dois tipos, cada um dos quais trazendo considerações distintas porém contribuindo para a usabilidade. A troca de ativos digitais entre os mercados é um dos tipos de interoperabilidade. Ele deveria conter a capacidade de transferir e trocar ativos originários de diferentes mercados sem intermediários confiáveis, como trocas centralizadas. Um exemplo disso seria tornar um NFT originário do mercado Criptopunks negociável em qualquer outro mercado disponível. Outro tipo de interoperabilidade desejada se diz respeito a troca de

informações que mantém a capacidade de fazer algo em um mercado que reflete em outro mercado NFT. Esta troca deve permitir o rastreamento não só de ativos ou itens negociáveis, mas também as operações executadas. Como exemplo, o compartilhamento do histórico de transações de um determinado item contendo valores e negociação.

2.5.2. Privacidade

O risco de exposição de dados é iminente em tecnologias que utilizam a internet como meio de comunicação, uma vez que ficam suscetíveis a ameaças de segurança e privacidade. Sendo a segurança e a privacidade quesitos importantes para aplicações, como NFTs, que envolvem recursos financeiros, estas aplicações demandam mecanismos eficientes para o controle e validação de credenciais, integridade e interoperabilidade de dados. Nesse contexto, a utilização de *blockchain* torna-se atraente, pois essa tecnologia proposta por [Nakamoto and Bitcoin 2008], contempla a inibição de inúmeras vulnerabilidades e pode ser utilizada para garantir segurança e auditabilidade dos dados. As *blockchains* permitem o armazenamento seguro e imutável de dados, com aplicação direta na resolução de problemas de segurança e privacidade.

As pesquisas relacionadas à privacidade ainda não obtiveram aplicabilidades relacionadas ao cenário dos NFTs devido ao alto custo computacional das soluções existentes. No entanto, é necessário prover formas menos custosas para a disponibilização destes métodos, favorecendo assim a adesão pela utilização da tecnologia no intuito de melhorar a segurança e privacidade do usuário [Wang et al. 2021].

2.5.3. Legislação

Existem questões legais em relação aos direitos autorais e proteção de dados que precisam ser abordados como desafio desta aplicação de contratos inteligentes. Os NFTs dificilmente incluem informações relacionadas ao licenciamento ou transferência dos seus direitos [Çağlayan Aksoy and Özkan Üner 2021]. Isto se deve ao fato de o padrão ERC-721 fornecer a possibilidade de apenas criar um link entre o registro do *token* feito pelo criador do NFT e o conteúdo digital a ser representado quando implementado desta forma. Assim, o conteúdo digital é representado no contrato NFT apenas como um índice do *token* e um link para o conteúdo ou item que é representado pelo *token* e são armazenados fora da *blockchain*, normalmente em uma URL pública.

Uma questão que se pode ter é como a legislação pode proteger a criação e venda de um NFT em uma plataforma digital e se ela pode ser considerada como um trabalho artístico. De fato, no direito existe legislação que responde às criações do mundo artístico e somente com o passar do tempo é que será possível verificar se os NFTs serão considerados obras artísticas cabíveis de proteção ou não. Outra questão é que o ato de comprar uma NFT não transfere o direito autoral deste. Sendo assim, a compra transfere apenas o direito de posse, embora seja possível o detentor dos direitos autorais conceder ao comprador uma licença para determinados usos da obra. Muitos questionamentos são realizados acerca dos NFTs, porém é sabido que os instrumentos digitais só tem validade quando o mundo real valida o processo. E as leis que tangem os direitos autorais juntamente aos contratos de licença podem esclarecer muitos destes questionamentos [Çağlayan Aksoy and Özkan Üner 2021, Valeonti et al. 2021].

2.5.4. Eficiência energética

A blockchain, por ser uma rede descentralizada, necessita de um conjunto de nós interligados de forma ponto a ponto para manterem os dados replicados e atualizados. Nesta topologia de rede, o sistema torna-se altamente disponível mesmo que alguns nós saiam da rede ou fiquem inacessíveis. Ainda para manter a segurança, a blockchain utiliza uma estratégia tolerante a falhas, chamada de consenso, para obter o acordo necessário em um único estado de rede. O consenso é um método de tomada de decisão para um conjunto de nós que precisam chegar a um acordo para manter o funcionamento do sistema. Existem várias maneiras de implementar os mecanismos de consenso que geram grandes diferenças técnicas em relação a tolerância à falha e consumo de recursos como tempo e processamento [Xiao et al. 2020].

As principais implementações dos mecanismos de consenso são o *Proof of Work (PoW)* e *Proof of Stake (PoS)*. Não há diferença quanto à forma direta de interação com aplicações de mercados para os ativos digitais, porém há uma mudança em relação à taxas e tempo gastos em transações. O *PoW*, por exemplo, tem uma característica de ser extremamente seguro e descentralizado. Em contrapartida ele consome uma quantidade de tempo e energia excessiva para este trabalho. Isto se deve ao fato de que para a criação de um bloco, a rede depende de nós mineradores com hardwares de alta capacidade para verificar as transações e cunharem um novo bloco. As plataformas Bitcoin e Ethereum utilizam o mecanismo de consenso *PoW*, e no caso da Ethereum cobra altas taxas nas transações para compra e venda de NFTs ou qualquer outra transação. Plataformas *blockchains* que utilizam o mecanismo de consenso *PoS* tem a possibilidade de obter maior desempenho e taxas menores uma vez que a cunhagem dos blocos não dependem de um alto processamento e sim da escolha de mineradores que têm seus saldos tomados por contratos como garantia de boa índole. Consequentemente o consumo de energia em *blockchain* que utilizam o *PoS* é menor do que o *PoW* devido a economia de trabalho de processamento realizada pelos mineradores.

2.6. Prática

Nesta seção abordaremos como projetar, codificar, implantar e executar contratos inteligentes para NFTs e um *marketplace* que manipula NFTs alugáveis. Os contratos desenvolvidos serão construídos por meio da linguagem Solidity e, além da implementação e execução destes contratos, apresentaremos como testá-los e invocá-los a partir de uma aplicação descentralizada desenvolvida na linguagem Python, em um ambiente de desenvolvimento configurável. Por sua vez, neste ambiente de desenvolvimento serão utilizadas as plataformas:

- **Remix IDE:** Desenvolver, compilar e implantar os contratos inteligentes na rede *Blockchain*;
- **Ganache:** Simular um nó da rede *Blockchain Ethereum* localmente;
- **Sistema de Arquivos Interplanetário (IPFS):** Gerar os *hashes* ou CID (*Content Identifier*) dos NFTs criados.

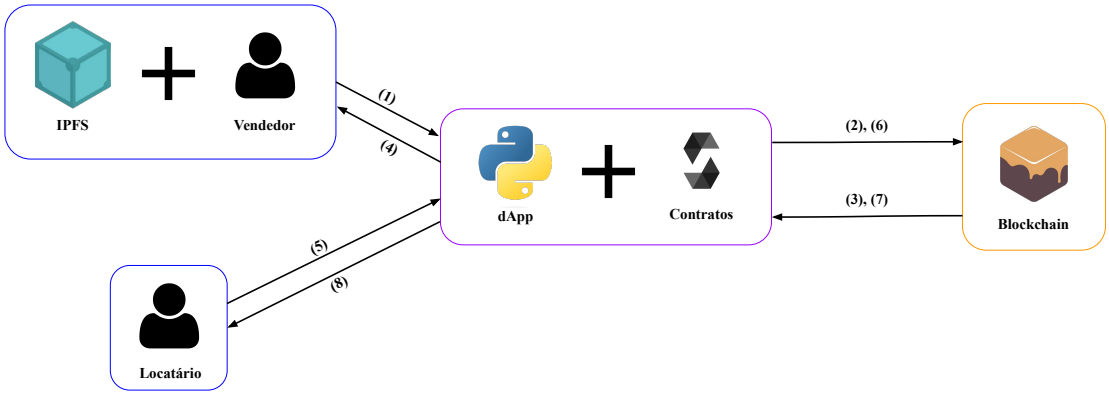

Figura 2.6. Fluxo de criação e aluguel de um NFT.

Conforme a Figura 2.6, ao final da prática teremos o seguinte fluxo: o vendedor enviará o CID gerado pelo IPFS para a *blockchain*, por meio da aplicação descentralizada (*dApp*) desenvolvida em Python (1). Por sua vez, a *dApp* utilizará as funcionalidades do contrato escrito em Solidity, com o objetivo de criar o NFT e disponibilizá-lo para aluguel na *blockchain* simulada pelo Ganache (2). Analogamente, a *dApp* receberá os dados das transações mineradas na *blockchain* (3), para transmiti-los de volta ao vendedor (4). Por outro lado, a pessoa que alugará o *token* utilizará o fluxo de maneira semelhante. O locatário solicitará tanto o aluguel quanto a devolução através da aplicação descentralizada (5), responsável por enviar as transações que serão mineradas (6). Posteriormente, os resultados deste procedimento serão enviados para a *dApp* (7) e, por fim, também poderão ser consultados (8).

Sendo assim, os objetivos desta prática são esclarecer características da programação de NFTs por contratos inteligentes que auxiliam a criação de *tokens* mais eficientes e seguros. Dentre as características mais relevantes a serem discutidas, destacamos a utilização dos padrões estruturais de *tokens* existentes, como as interfaces da OpenZeppelin, assim como os padrões das funções de empréstimo e devolução. Assim, objetivamos abordar os passos para a implementação de uma aplicação real para NFTs, bem como apresentar a utilidade dos *tokens* não fungíveis para controle de propriedade de objetos físicos e digitais.

2.6.1. Configuração do ambiente de desenvolvimento

Nesta subseção, iniciamos o processo de desenvolvimento realizando a configuração do ambiente de desenvolvimento, por meio da instalação e utilização das ferramentas Remix IDE, Ganache e IPFS. Estas ferramentas desempenham o papel de auxiliar na construção e implantação do contrato, bem como na simulação de uma *blockchain* real. Desta forma, a principal vantagem deste ambiente é o fato dele ser facilmente configurável e sem gastos reais, visto que toda execução é realizada localmente.

2.6.1.1. Remix IDE

A primeira ferramenta que abordaremos será o Remix[12], que nada mais é do que uma IDE desenvolvida em ambiente web, utilizada principalmente para desenvolver, compilar, implantar e utilizar contratos inteligentes, como é exibido na Figura 2.7.

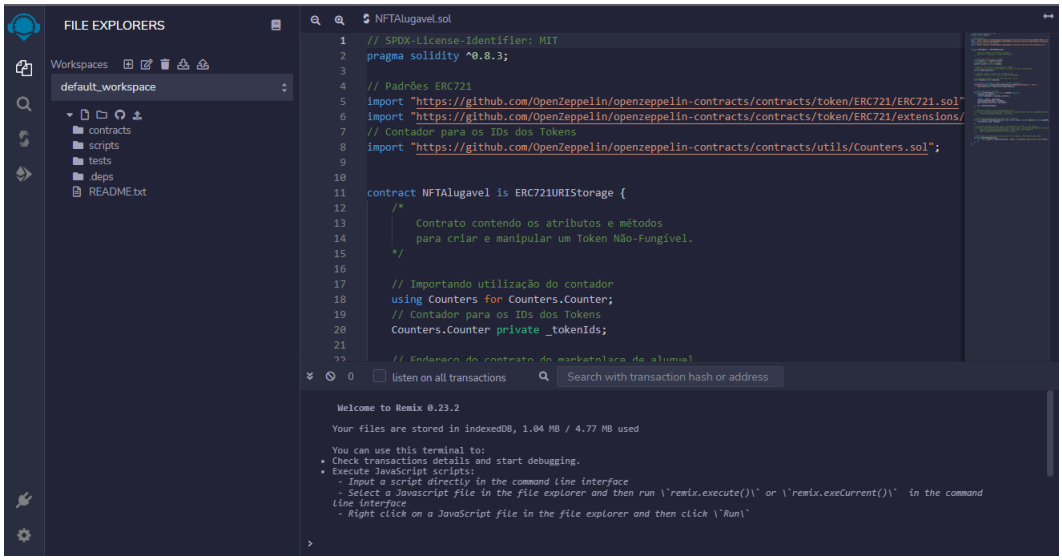

Figura 2.7. Tela principal da ferramenta Remix IDE.

Por meio do acesso a IDE[13], seremos capazes de realizar todas as etapas descritas anteriormente que são necessárias para o funcionamento da rede de NFTs alugáveis. Na primeira aba da esquerda da Figura 2.7 podemos criar novos arquivos da linguagem Solidity para desenvolvermos nossos contratos. Já na terceira aba seremos capazes de compilar os contratos, a fim de verificarmos possíveis erros de sintaxe e/ou de fluxo e, finalmente, na quarta aba poderemos implantá-los na rede *blockchain* de teste que configuraremos a seguir.

2.6.1.2. Ganache

A segunda ferramenta que necessitamos é o Ganache[14], utilizada para simular uma *blockchain Ethereum* localmente. Dessa maneira, é possível configurar a rede, realizar testes e monitorar transações da forma que o usuário desejar. Outra vantagem é a integração com o Remix IDE, além da possibilidade de criar um ambiente de trabalho de forma rápida que engloba a rede *blockchain* juntamente com as contas fictícias para teste, conforme na Figura 2.8.

[12] `https://remix-ide.readthedocs.io/en/latest/file_explorer.html`
[13] `http://remix.ethereum.org/`
[14] `https://trufflesuite.com/docs/ganache/`

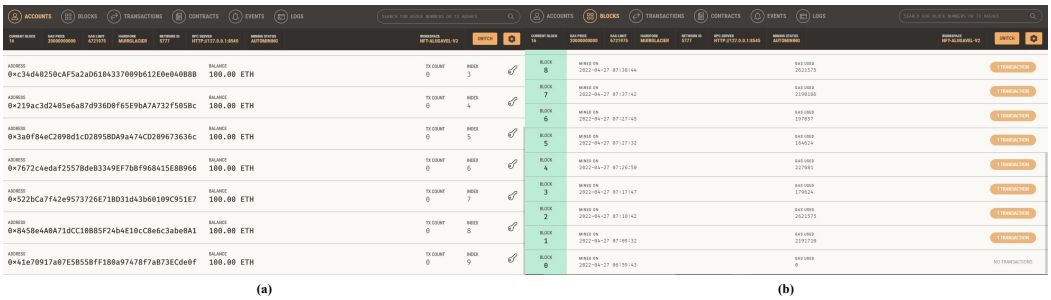

Figura 2.8. (a) Lista de contas criadas automaticamente. (b) Histórico de todas as transações da rede de teste.

2.6.1.3. IPFS - Interplanetary File System

O sistema de arquivos interplanetário, ou IPFS[15], é um sistema de arquivos descentralizado, que visa criar um armazenamento associado ponto-a-ponto e distribuído. Suas principais vantagens são garantir segurança e privacidade dos dados armazenados, e vem sendo utilizado juntamente com as tecnologias de *blockchain* e NFTs, pelo fato da descentralização da ferramenta possibilitar o acesso aos ativos de maneira simplificada.

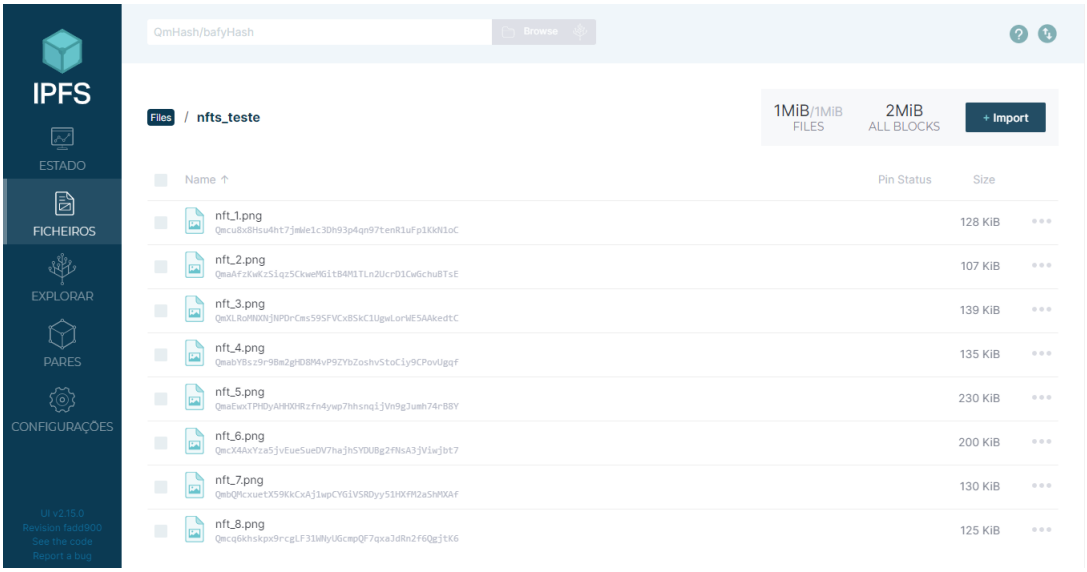

Figura 2.9. Tela que exibe os arquivos de um usuário alocados no IPFS.

Assim sendo, o IPFS torna-se uma solução efetiva para nossa prática, no que diz respeito a armazenar os NFTs e gerar os CIDs, que são o conteúdo das imagens codificadas por *hash*. Apesar desse sistema de arquivos ser utilizado por outras ferramentas, como o Piñata[16], que endereça arquivos na nuvem e podem ser acessados de qualquer lugar, optamos pelo IPFS Desktop[17] propriamente dito, para termos controle local dos

[15] https://docs.ipfs.io/
[16] https://www.pinata.cloud/
[17] https://docs.ipfs.io/install/ipfs-desktop/

NFTs, conforme a Figura 2.9.

2.6.2. Fundamentos da Linguagem Solidity

Nesta subseção apresentaremos os principais conceitos relativos à linguagem de descrição de contratos inteligentes para NFTs. O objetivo é ensinar os fundamentos da linguagem Solidity[18], que é de alto nível e criada por meio de uma combinação de JavaScript, Java e C++, para realizar a interação do conjunto de funções e eventos definidos nos padrões ERC dos *tokens* não fungíveis. A partir das características da linguagem descritas e apresentadas, trabalharemos com as estruturas dos ERC baseados nas funções de propriedades de metadados.

2.6.2.1. Tipos de dados

Os tipos de dados em Solidity apresentam similaridades com a linguagem C++ e JavaScript, no que diz respeito às operações entre eles. Destacamos os principais tipos de dados utilizados durante a prática. O primeiro tipo de dados e o tipo mais básico das linguagens de programação é o tipo booleano, o qual é representado pelos valores verdadeiro ou falso na linguagem Solidity.

```
1 bool verdadeiro = true;
2 bool falso = false;
```

Listagem 2.4. Exemplo de variáveis booleanas em Solidity.

Já o segundo tipo de dados muito utilizado nas linguagens é o tipo inteiro. Diferentemente de outras linguagens, em Solidity é possível especificar o tamanho do inteiro em bits e a sinalização, conforme mostra a Listagem 2.5.

```
1 int8 w = 3; // Inteiro com sinal de 8 bits
2 uint8 x = 1; // Inteiro sem sinal de 8 bits
3 int y = 10; // (int256) Inteiro sem sinal de 256 bits
4 uint256 z = 30; // (uint) Inteiro sem sinal de 256 bits
```

Listagem 2.5. Exemplo de variáveis inteiras em Solidity.

Outros dois tipos de dados existentes em Solidity são os Bytes e Strings, utilizados comumente para armazenar cadeias de caracteres. Assim como o tipo inteiro, os bytes podem ter tamanhos definidos. Por sua vez, variáveis do tipo String podem armazenar dados com a codificação UTF-8 de qualquer comprimento. A diferença dos dois tipos baseia-se na utilização, visto que o tipo Bytes é mais barato computacionalmente, logo, quando o texto pode possuir um tamanho definido, recomenda-se o uso do tipo Bytes em vista do tipo String.

```
1 bytes32 b = "Texto";
2 string s = "Ola, Mundo!";
```

Listagem 2.6. Exemplo de variáveis que armazenam caracteres em Solidity.

Além disso, a linguagem Solidity possui tipos de dados específicos: os mapeamentos e endereços. O tipo de dado mapeamento pode ser associado com a tabela *hash*.

[18] https://docs.soliditylang.org/en/v0.8.9/

Sua sintaxe é simples conforme a Listagem 2.7 e, como o próprio nome diz, mapeia chaves de um tipo para valores do mesmo ou outro tipo.

```
1 mapping(uint256 => bool) mapeamento; // Mapeia chaves do tipo inteiro para valores booleanos
```

Listagem 2.7. Exemplo de mapeamentos em Solidity.

No que diz respeito ao tipo de dado endereço, podemos dizer que este é um dos tipos mais importantes da linguagem, pois com ele é possível armazenar em variáveis os endereços *Ethereum* de contas, transações, contratos, além de realizar chamadas de funções específicas deste tipo. Variáveis do tipo endereço podem possuir o modificador *payable*, significando que o endereço desta variável pode receber transferências de outro endereço, bem como ter o seu balanço de *Ethers* consultado.

```
1 address payable add1 = address(0x123); // Indica que o endereco add1 e pagavel
2 address meuEndereco = address(this); // Armazena o endereco da entidade que chama uma funcionalidade do contrato
3 add1.transfer(10); // A entidade que chama o contrato realiza uma transferencia para o endereco add1
4 meuEndereco.call(...); // Realiza uma chamada de funcao de outro contrato, por exemplo
```

Listagem 2.8. Exemplo de endereços em Solidity.

Finalmente, o último tipo de dado que abordaremos durante a prática será o tipo composto de estrutura, semelhante ao tipo de estrutura existente na linguagem C. Com este tipo composto, podem ser definidas abstrações de uma entidade que possui atributos de tipos variados, desde tipos básicos como inteiros, booleanos, até mesmo tipos como endereços e mapeamentos.

```
1 struct Token{
2     uint id,
3     bool status,
4     bytes32 cid,
5     address payable dono
6 }
```

Listagem 2.9. Exemplo de estruturas em Solidity.

2.6.2.2. Variáveis especiais

Tendo em vista os tipos de dados mais utilizados em Solidity, apresentaremos também variáveis globais e funções reservadas que possuem papéis importantes ao tratar de manipulação de contratos e suas funcionalidades.

A primeira variável que abordaremos é a *block*, visto que como o próprio nome diz, possui valores referentes ao bloco atual da rede *blockchain*. Por exemplo, alguns valores possíveis de serem consultados são o número do bloco e a data/hora em segundos, desde uma época em Unix. Do mesmo modo, a variável *msg* possui valores referentes ao envio da transação, isto é, dados como o endereço de quem envia a transação, bem como o número de *Wei* transferido. Finalmente, a variável global *abi* possui as funcionalidades de decodificar ou codificar funções externas, tal como funções de outros contratos.

```
1 block.number; // Retorna o numero atual do bloco
2 block.timestamp; // Retorna a data e hora atual
3 msg.sender; // Endereco de quem envia a transacao
4 msg.value; // Valor em Wei transferido
5 abi.encodeWithSignature(...); // Codifica uma funcao enviando sua assinatura como primeiro parametro
```

Listagem 2.10. Exemplo de variáveis globais em Solidity.

2.6.2.3. Funções

As funções em Solidity possuem grande importância ao tratarmos de contratos inteligentes, pelo fato de definirem de forma exata as funcionalidades que podem ser realizadas e quem é capaz de acessá-las. Logo, cada função possui um tipo associado, indicando qual o escopo de sua utilização:

- ***public*:** Este tipo indica que a função pode ser acessada por todos;
- ***external*:** Este tipo informa que a função pode ser acessada apenas externamente ao contrato;
- ***internal*:** Este tipo assinala que uma função pode ser acessada apenas pelo próprio contrato, ou por contratos e bibliotecas derivados;
- ***private*:** Este tipo restringe o escopo da função para que apenas o próprio contrato possa acessá-la.

Adicionalmente, funções em Solidity possuem modificadores padrões que definem quais regras a função deve seguir em seu fluxo de execução:

- ***pure*:** Este modificador indica que a função não poderá ler dados de variáveis externas à função e tampouco alterá-las, isto é, funções com este modificador podem trabalhar apenas em seu escopo local e com os parâmetros recebidos;
- ***view*:** Já este modificador restringe uma função para que ela possa ler dados de variáveis externas à ela, porém ainda assim funções com este modificador não podem alterar dados destas variáveis;
- ***payable*:** Por fim, este modificador aponta que a função pode acessar variáveis externas, bem como alterá-las e, em adição, requisita que ao utilizar a função, um pagamento em *Ether* deve ser realizado.

Vale ressaltar que, para uma função conseguir ler e alterar os dados de variáveis externas, porém sem requisitar um pagamento, basta omitir estes modificadores. Porém, cada um deles adiciona uma camada de segurança e coerência importantes para a utilização de um contrato. Na Listagem 2.11 é possível observar o modelo de declaração de uma função.

```
1 function nomeFunc(<parametros>) {public|internal|external|private} [pure|view|payable] returns (<tipo retorno>)
```

Listagem 2.11. Exemplo de função em Solidity.

2.6.2.4. Alocação de dados

Em Solidity, as variáveis além de possuírem tipos associados, podem ser armazenadas de modos diferentes. Semelhantemente com a memória principal e secundária de um computador, existem os modificadores *memory* e *storage*, respectivamente.

Variáveis com o modificador *memory* possuem dados armazenados em tempo de execução do contrato, e normalmente são utilizadas para armazenar parâmetros ou retornos de funções. Por outro lado, o modificador *storage* é o modo de armazenamento padrão da linguagem, logo, variáveis com este modificador possuem dados armazenados de forma persistente e, consequentemente, consomem uma quantidade maior de gás em comparação com as variáveis que possuem o modificador anterior.

```
1 string storage texto = "Persistente"; // Armazenamento persistente
2 bool memory statusRetorno = "False"; // Armazenamento temporario
3 mapping(uint => bool) mapeamento; // Armazenamento persistente omitido
4 delete mapeamento[1]; // Metodo para desalocar um item
```

Listagem 2.12. Exemplo de alocação de dados em Solidity.

2.6.3. Contratos inteligentes

Em sequência, nesta subseção abordaremos os recursos necessários sobre contratos inteligentes em Solidity voltados para a nossa prática. Os contratos assemelham-se com as classes de linguagens de programação orientadas a objetos, isto é, cada contrato em Solidity pode possuir um construtor, variáveis com armazenamento persistente e métodos que manipulam estas variáveis, conforme a Listagem 2.13.

Embora haja similaridade com as classes de linguagens orientadas a objetos, os contratos possuem particularidades intrínsecas, como por exemplo os eventos, manipuladores de erros e modificadores, que proporcionam vantagens de segurança ao tratarmos sobre boas práticas de modelagem de contratos.

```
1  contract Marketplace{
2      address payable donoContrato;
3      uint256 taxaMarketplace = 0.02 ether;
4
5      constructor(){
6          donoContrato = payable(msg.sender);
7      }
8
9      function getTaxaMarketplace() public view returns (uint256) {
10         return taxaMarketplace;
11     }
12
13     function setTaxaMarketplace(uint256 _novaTaxa) external {
14         taxaMarketplace = _novaTaxa;
15     }
16 }
```

Listagem 2.13. Exemplo básico de um contrato em Solidity.

2.6.3.1. Eventos

Os eventos são recursos muito utilizados em contratos, com o objetivo de transmitirem informações para as aplicações descentralizadas. Além disso, os eventos são úteis para

retornarem informações em funções que geram transações na *blockchain*, pelo fato dos dados de um evento poderem ser acessados por meio dos *logs* da transação.

```
1 event RetornaToken(uint256 tokenId); // Declaracao de um evento
2 emit RetornaToken(1); // Modo de disparar um evento
```

Listagem 2.14. Exemplo de eventos em Solidity.

2.6.3.2. Manipuladores de erros

Outro recurso difundido na construção de um contrato são os manipuladores de erros, que são funcionalidades previamente definidas pela linguagem para tratamento de exceção ou para avaliar condições prévias que devem ser atendidas. Nesta prática utilizaremos o manipulador *require*, que avalia se uma condição é falsa, para então disparar uma exceção e finalizar o fluxo principal do escopo em que o manipulador foi chamado. Adicionalmente, uma mensagem customizada pode ser retornada, caso a mesma esteja definida.

```
1 require(msg.value == taxaMarketplace, "Pague exatamente a taxa que o Marketplace exige!");
```

Listagem 2.15. Exemplo de manipuladores de erros em Solidity.

2.6.3.3. Modificadores

No que tange aos modificadores criados internamente em contratos, os mesmos são semelhantes aos modificadores padrões da linguagem vistos em 2.6.2.3. Novos modificadores podem ser criados dentro dos contratos para serem utilizados em determinadas funções e delimitarem o uso das mesmas, como por exemplo, especificar quais funções apenas o dono do contrato poderá invocá-las.

```
1 modifier apenasMarketplace(){
2     require(msg.sender == marketplace, "Apenas o marketplace pode utilizar esse metodo!");
3     _;
4 }
```

Listagem 2.16. Exemplo de modificadores em Solidity.

2.6.4. Padrões ERC e OpenZeppelin

Até o momento, observamos como os contratos podem ser modelados, bem como os seus recursos internos existentes e como manipular estes recursos. No que diz respeito a segurança de um contrato, podemos destacar a utilização de modificadores e tratamentos de erros, porém existem outras técnicas de modelar um contrato inteligente de maneira eficiente e segura.

Com o advento dos NFTs, os padrões ERC tornaram-se referência para a modelagem de contratos de *tokens*, e juntamente com eles surgiram as demandas por segurança. Dessa forma, os padrões da OpenZeppelin[19] introduziram interfaces e contratos devidamente testados para que o processo de desenvolvimento de contratos seguros e eficientes torne-se mais simples.

[19] https://docs.openzeppelin.com/

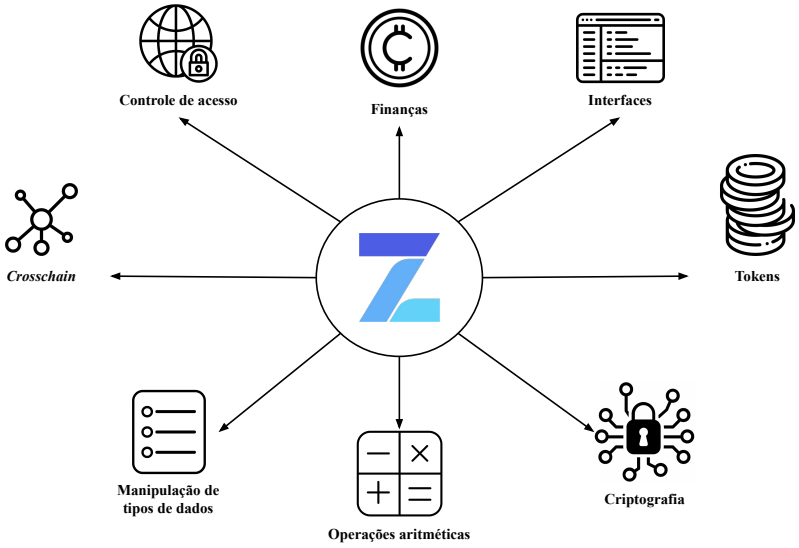

Figura 2.10. Subáreas de segurança abrangidas pela biblioteca OpenZeppelin.

A biblioteca OpenZeppelin conta com contratos e interfaces para diversos tipos de aplicações, além de possuir implementados os próprios padrões ERC. Na Figura 2.10 é possível observar como esta biblioteca perpassa várias aplicações importantes que podem ser desenvolvidas para *blockchain* e NFTs.

2.6.5. Desenvolvendo NFTs baseados nos padrões de *tokens* Ethereum

Nesta subseção, abordaremos o processo de desenvolvimento de *tokens* não fungíveis utilizando algumas das boas práticas de modelagem apresentadas nas seções anteriores. Ao final, planejamos obter um contrato capaz de criar NFTs e com funcionalidades disponíveis de serem utilizadas por meio de uma aplicação descentralizada e independente de plataforma.

O primeiro passo para modelar o contrato de NFTs de forma segura e eficiente, baseia-se em importar os contratos da biblioteca OpenZeppelin que já possuem funcionalidades bem testadas e implementadas. Nesta prática, optamos por utilizar o contrato do padrão ERC-721 para criar os *tokens*, bem como o contrato para armazenar o código da imagem que se tornará um NFT. Utilizamos ainda a biblioteca de contadores para termos controle dos índices dos *tokens* criados.

```
1 import "https://github.com/OpenZeppelin/openzeppelin-contracts/contracts/token/ERC721/ERC721.sol";
2 import "https://github.com/OpenZeppelin/openzeppelin-contracts/contracts/token/ERC721/extensions/ERC721URIStorage.
      sol";
3 import "https://github.com/OpenZeppelin/openzeppelin-contracts/contracts/utils/Counters.sol";
```

Listagem 2.17. Contratos e bibliotecas importadas.

Posteriormente, o contrato inteligente pode ser criado herdando as funcionalidades do contrato da Linha 2 na Listagem 2.17. Visto que esta importação herda o contrato do padrão ERC-721, não precisamos especificar novamente na definição do contrato de NFTs esta herança. Internos ao mesmo, teremos os atributos: o contador de identificador dos *tokens*, com o objetivo de armazenar o índice correto a cada criação de um novo NFT,

o endereço do contrato que representa o *Marketplace* de aluguéis, com a finalidade de demarcar qual *Marketplace* pode gerenciar os NFTs deste novo contrato. Além do mais, deve existir o evento de emitir o identificador de um novo *token* de volta para a aplicação descentralizada e um modificador para demarcar quais funções apenas o *Marketplace* estará elegível para utilizar.

```
1  contract NFTAlugavel is ERC721URIStorage {
2      using Counters for Counters.Counter;
3      Counters.Counter private _tokenIds;
4
5      address marketplace;
6
7      event TokenId(uint256 token_id);
8
9      constructor(address enderecoContratoMarketplace) ERC721("NFTAlugavel", "AFT") {
10         marketplace = enderecoContratoMarketplace;
11     }
12
13     modifier apenasMarketplace(){
14         require(msg.sender == marketplace, "Apenas o marketplace pode utilizar esse metodo!");
15         _;
16     }
17 }
```

Listagem 2.18. Atributos do contrato de NFTs.

Vale ressaltar que, conforme a Listagem 2.18, é possível observar a inicialização do construtor do padrão ERC-721. Este contrato demanda a inicialização do nome dos NFTs, bem como o símbolo destes *tokens*, ambos à cargo do programador. Em seguida, as funções do contrato de NFTs podem ser desenvolvidas, como na Listagem 2.19.

```
1  function criarNovoToken(string memory tokenURI) public {
2      _tokenIds.increment();
3      uint256 newItemId = _tokenIds.current();
4      _mint(msg.sender, newItemId);
5      _setTokenURI(newItemId, tokenURI);
6      approve(marketplace, newItemId);
7      emit TokenId(newItemId);
8  }
9
10 function transferirTokenExpirado(address de, address para, uint256 tokenId) external apenasMarketplace {
11     _transfer(de, para, tokenId);
12 }
```

Listagem 2.19. Funções de criar e retornar um Token Não-Fungível.

A primeira delas (e a mais importante) é a função de criação de um novo *token*, responsável por realizar as chamadas de funções do padrão ERC-721, como por exemplo, as funções de forjar um novo NFT (Linha 4), definir o código CID do novo *token* (Linha 5), bem como permitir o acesso do *Marketplace* ao NFT gerado (Linha 6). Por sua vez, a segunda função aborda o caso de um NFT em estado de alugado, porém com o prazo de aluguel expirado. Em um cenário como este, a função do contrato é chamada apenas pelo próprio *Marketplace* com a finalidade de retornar a titularidade do *token* para o dono de origem e, consequentemente, finalizar o aluguel deste NFT.

De forma complementar, a Listagem 2.20 apresenta o contrato referente ao *Marketplace*, responsável por gerenciar todo o processo de aluguel de um *token*, desde a disponibilização de um item para alugar por parte do vendedor, até o aluguel e finalização por parte do locatário. Os atributos deste contrato assemelham-se com a ideia do contrato

de NFTs, visto que existem os contadores de itens totais, alugados e devolvidos. Neste contrato, há também o atributo que armazena a conta do dono responsável pelo *Marketplace*, bem como a taxa que é cobrada para um vendedor colocar o seu item disponível e, portanto, sendo dessa maneira que o dono do *Marketplace* recebe sobre a utilização do seu sistema. Além disso, uma estrutura é definida para abstrair um item alugável, contendo dados como tempo de aluguel, identificador do *token*, etc. Vale ressaltar que, nesta estrutura há o campo que armazena o endereço do contrato do NFT existente, pelo fato do *Marketplace* ser desenvolvido para hospedar *tokens* de tipos variados, incrementando sua usabilidade. Finalmente, o mapeamento armazena estes itens criados de acordo com seus identificadores, assim como um evento de retorno é emitido após a criação de um item disponível para ser alugado, sendo que neste último existem campos com o modificador *indexed*, indicando que eles podem atuar como filtros em consultas pelo evento nos logs das transações.

```
1  contract MarketplaceAluguel{
2      using Counters for Counters.Counter;
3      Counters.Counter private _itemIds;
4      Counters.Counter private _countItensAlugados;
5      Counters.Counter private _countItensDevolvidos;
6
7      address payable donoContrato;
8      uint256 taxaMarketplace = 0.02 ether;
9
10     struct Item{uint256 itemId; bool statusAlugado; address contratoNFT;
11         uint256 tokenId; address payable vendedor; address locatario;
12         uint256 preco; uint256 expiraEm; bytes32 descricao;
13     }
14
15     mapping(uint256 => Item) private listaItens;
16
17     event ItemCriado(uint256 indexed itemId, bool statusAlugado,
18         address indexed contratoNFT, uint256 indexed tokenId,
19         address vendedor, address locatario,
20         uint256 preco, uint256 expiraEm, bytes32 descricao
21     );
22
23     constructor(){
24         donoContrato = payable(msg.sender);
25     }
26 }
```

Listagem 2.20. Contrato do Marketplace para NFTs alugáveis.

Após a definição dos atributos do contrato *Marketplace*, faz-se necessário definir os métodos que manipulam estes dados, isto é, as funções responsáveis por colocar um NFT disponível para aluguel, alugar um *token* e finalizar um aluguel. Cada uma destas funções possui suas particularidades, devido às verificações que devem ser feitas em conjunto com o fluxo principal de execução.

A função de disponibilizar um NFT para aluguel, conforme mostra a Listagem 2.21, baseia-se em verificar o preço do NFT, a fim de garantir que o *token* possua um preço válido, além de verificar se a taxa de alocação está sendo paga corretamente ao dono do *Marketplace*. Caso os dados da transação atendam aos requisitos, o fluxo principal da função é seguido. O item é criado e a titularidade do NFT é transferida para o *Marketplace*, com o intuito do mesmo ser capaz de permitir um locatário possuir o *token* durante o tempo de aluguel definido. Em adição, o dono real do NFT transfere a taxa ao

dono do *Marketplace* e o evento de retorno para a aplicação descentralizada é emitido.

```
function criaItemAlugavel(address contratoNFT, uint256 tokenId, uint256 preco, uint256 expiraEm, bytes32 descricao
    ) public payable{
    require(preco > 0, "Preco deve ser ao menos de 1 wei!");
    require(msg.value == taxaMarketplace, "Pague exatamente a taxa que o Marketplace exige!");
    _itemIds.increment();
    uint256 itemId = _itemIds.current();
    listaItens[itemId] = Item(itemId, false, address(contratoNFT), tokenId, payable(msg.sender), address(0),
    preco, expiraEm, descricao);

    IERC721(contratoNFT).transferFrom(msg.sender, address(this), tokenId);
    donoContrato.transfer(taxaMarketplace);
    emit ItemCriado(itemId, false, contratoNFT, tokenId, msg.sender, address(0), preco, expiraEm, descricao);
}
```

Listagem 2.21. Função de criar um NFT alugável.

Já a função de alugar um item disponível - Listagem 2.22 demanda que um NFT não esteja alugado, além de analisar se o *token* requerido pertence ao *Marketplace*, com o objetivo de garantir o sucesso na transferência para o locatário. A última verificação aborda o valor transferido, obrigando o locatário a realizar o pagamento exato pelo aluguel do NFT. O fluxo principal desta função segue a lógica de um processo de compra comum: o pagamento é feito ao vendedor, para logo em seguida os dados do item serem alterados, como o endereço do locatário, o status de *token* alugado e o prazo de expiração.

```
function alugarItem(address contratoNFT, uint256 itemId) public payable{
    uint256 preco = listaItens[itemId].preco;
    uint256 tokenId = listaItens[itemId].tokenId;
    require(!listaItens[itemId].statusAlugado, "Este token ja esta alugado!");
    require(IERC721(contratoNFT).ownerOf(tokenId) == address(this), "Token nao disponivel!");
    require(msg.value == preco, "Pague exatamente o valor do aluguel!");
    listaItens[itemId].vendedor.transfer(msg.value);
    IERC721(contratoNFT).transferFrom(address(this), msg.sender, tokenId);
    listaItens[itemId].expiraEm = listaItens[itemId].expiraEm + block.timestamp;
    listaItens[itemId].locatario = msg.sender;
    listaItens[itemId].statusAlugado = true;
    _countItensAlugados.increment();
}
```

Listagem 2.22. Função de alugar um NFT.

Concluindo as três funções principais do *Marketplace*, a função de finalizar um aluguel, como consta na Listagem 2.23, é responsável por verificar se um NFT está realmente alugado, bem como se o referido *token* ainda está em período de aluguel. É importante destacar que, o locatário de um NFT é capaz de finalizar o aluguel antes do prazo definido e, por outro lado, qualquer pessoa é capaz de finalizar o aluguel de um item após o prazo expirado. Sendo assim, com os requisitos atendidos, a função transfere o NFT de volta ao vendedor e exclui o item referente a este *token*, ou seja, caso o vendedor desejar colocar o mesmo item para aluguel novamente, uma nova taxa deve ser paga.

```
function finalizaAluguel(uint256 itemId) external {
    Item storage itemAlugado = listaItens[itemId];
    require(itemAlugado.statusAlugado, "Este token nao esta alugado!");
    require(msg.sender == itemAlugado.locatario || block.timestamp >= itemAlugado.expiraEm,
            "Este token ainda esta no periodo de aluguel!");
    itemAlugado.statusAlugado = false;
    (bool sucessoTransferencia, ) = (itemAlugado.contratoNFT).call(
        abi.encodeWithSignature(
            "transferirTokenExpirado(address,address,uint256)",
            itemAlugado.locatario,
```

```
            itemAlugado.vendedor,
            itemAlugado.tokenId
        )
    );
    require(sucessoTransferencia, "Nao foi possivel transferir o NFT de volta para o vendedor!");
    _countItensDevolvidos.increment();
    delete listaItens[itemId];
}
```

Listagem 2.23. Função de finalizar o aluguel de um NFT.

2.6.6. Aplicação descentralizada cliente

Nesta subseção, é implementado a *dApp* responsável por consumir as funcionalidades dos contratos criados anteriormente. Foi decidido utilizar a linguagem Python, juntamente com a biblioteca Web3 para mapear as funções dos contratos, pelo fato de proporcionarem aprendizado sobre o funcionamento do processo de realizar uma transação na rede *blockchain*.

Através destes recursos, podemos mapear as funções do contrato de NFTs, bem como do contrato *Marketplace*. Acerca do contrato de *tokens* não fungíveis, apenas a função de criar um novo *token* é viável ser mapeada, já que a outra funcionalidade deste contrato fica restringida para apenas o *Marketplace* gerenciador dos contratos de NFTs. Conforme a Listagem 2.24 sobre o mapeamento da função de criar um novo *token*, a biblioteca Web3 proporciona métodos que facilitam o processo de criar e enviar transações para a rede *blockchain*.

```
def criarNovoToken(tokenCID : str):
    nonce = web3.eth.get_transaction_count(public_key, 'latest')
    tx = contract.functions.criarNovoToken(tokenCID).buildTransaction({"nonce": nonce, "from": public_key})
    signed_tx = web3.eth.account.sign_transaction(tx, private_key)
    tx_hash = web3.eth.send_raw_transaction(signed_tx.rawTransaction)
    receipt = web3.eth.wait_for_transaction_receipt(tx_hash)
    token_id = Web3.toInt(receipt['logs'][0]['topics'][3])
    return token_id
```

Listagem 2.24. Função de criar um novo Token mapeada para a aplicação descentralizada.

Para uma transação ser validada na rede, é preciso primeiramente resgatar o *nonce* dado a chave pública de uma conta. Este valor nada mais é do que um número pseudo-aleatório, utilizado para meios de autenticação e criptografia durante o processo de mineração da transação. Em seguida, a transação pode ser modelada por meio deste valor gerado e a chave pública da conta que indica quem está realizando a transação. Além disso, a assinatura da função do contrato deve ser invocada, recebendo como parâmetro o CID do novo *token*. Posteriormente, a transação deve ser assinada com a chave privada da conta, sendo que em aplicações Web que utilizam as carteiras como o MetaMask[20], esta funcionalidade fica à cargo da carteira e confirmação do usuário. Ao final, a transação pode ser enviada para a rede e ter o seu *hash* retornado. Após o aguardo da conclusão deste processo, o evento emitido pela função no contrato pode ser consultado por meio dos logs da transação e, consequentemente, podemos armazenar o identificador do novo NFT criado.

[20]https://metamask.io

Analogamente, as funções do contrato *Marketplace* são mapeadas da mesma maneira, porém vale ressaltar o caso das transações que demandam o pagamento de *Ethers*. A Listagem 2.25 exemplifica este processo. No momento de modelar a transação, além dos parâmetros *nonce* e endereço público da conta de quem realiza a transação, são necessários outros valores, como o valor da transação convertidos em wei, isto é, a menor divisão de *ether* possível, o gás máximo que poderá ser consumido pela transação e o preço deste gás, que indica o quanto o usuário está disposto a pagar por cada unidade de gás, ou seja, para ter sua transação concluída de forma mais rápida é necessário um preço de gás mais alto. Novamente, este processo de gerar tais valores em uma aplicação Web fica à cargo da carteira utilizada pelo usuário na maioria dos casos.

```
1 def alugarItem(itemId: int, valor : float):
2     nonce = web3.eth.get_transaction_count(public_key, 'latest')
3     tx = contract.functions.alugarItem(contract_nft, itemId).buildTransaction({"nonce": nonce, "from": public_key,
          "value": web3.toWei(valor, 'ether'), "gas": 2000000, "gasPrice": web3.toWei('50', 'gwei')})
4     signed_tx = web3.eth.account.sign_transaction(tx, private_key)
5     tx_hash = web3.eth.send_raw_transaction(signed_tx.rawTransaction)
6     return tx_hash
```

Listagem 2.25. Função de alugar um *token* mapeada para a aplicação descentralizada.

2.6.7. Implantação e testes

Tendo em vista a construção de todo o fluxo de trabalho do sistema de aluguéis de NFTs, nesta subseção concluímos a prática apresentando o processo de implantar os contratos criados, assim como utilizá-los por meio da aplicação descentralizada desenvolvida. Com o auxílio do Remix IDE, há a possibilidade de compilar os contratos criados em diversas versões de compiladores, para logo em seguida obter dados como o ABI (*Application Binary Interface*) do contrato compilado e o *Bytecode*. O ABI nada mais é do que a interface utilizada para estabelecer a comunicação entre a aplicação descentralizada e o contrato implantado na rede *blockchain*. Já o *Bytecode* é em sua essência o contrato armazenado na rede.

Seguido da compilação de um contrato, o processo de implantação na rede *blockchain* pode ser realizado. Este processo inicia-se selecionando o ambiente em que o contrato será implantado. O Remix permite a utilização de máquinas virtuais para testes, assim como diversos provedores. No caso desta prática, podemos selecionar o Ganache diretamente pelas opções, ou utilizá-lo em conjunto com o MetaMask, sendo que desta última forma as contas fictícias geradas pelo Ganache são mapeadas na carteira MetaMask por meio das chaves privadas de cada conta.

Figura 2.11. Transação ao implantar um contrato na *blockchain*.

Através do contrato implantado, somos capazes de observar na interface do Gana-

che o registro e os dados da transação, conforme a Figura 2.11. Um dos dados importantes a serem analisados é o endereço em que o contrato foi criado na rede, pois por meio dele a *dApp* consegue buscar o contrato e fazer uso de suas funções, juntamente com o ABI. Em contraste, no Remix é possível utilizar as funções do contrato para teste por meio de uma interface previamente definida após a implantação, como na Figura 2.12.

Figura 2.12. Teste de implantação gerado pelo Remix.

De acordo com os processos anteriores, a aplicação descentralizada pode então ser inicializada por meio dos endereços dos contratos, em conjunto com os ABIs e endereços das contas que serão utilizadas. Adicionalmente, o provedor da rede deve ser especificado conforme a Listagem 2.26. Em suma, a execução da *dApp* é realizada via terminal, como mostra a Figura 2.13, local em que os dados como eventos emitidos e status de retorno podem ser conferidos.

```
1 public_key = "0x..."
2 private_key = "..."
3 web3 = Web3(Web3.HTTPProvider(ganache))
4 contract = self.web3.eth.contract(contract_address, contract_abi)
```

Listagem 2.26. Inicialização da aplicação descentralizada.

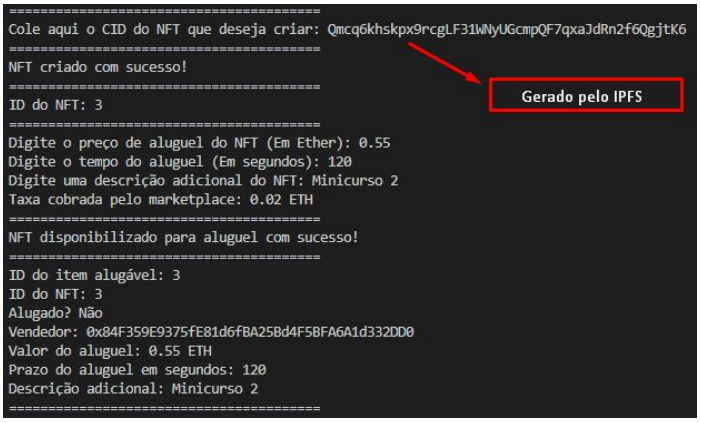

Figura 2.13. Exemplo de resultado obtido ao executar a dApp.

Em suma, a simulação prática objetivou associar a utilização dos contratos e a

comunicação entre *blockchain* e *dApp* com os *Marketplaces* existentes, como o OpenSea[21] e o Rarible[22]. Ademais, melhorias podem ser implementadas, bem como a expansão e teste da aplicação descentralizada em ambiente web, integrada com as carteiras de criptomoedas e *tokens*. Todo o código da seção prática é acessível pelo Github: `https://github.com/lesc-ufv/minicurso2-sbrc2022`.

2.7. Conclusão

Este capítulo discorreu sobre os conceitos, aplicações e desafios do padrão ERC-721 para *tokens* não fungíveis. Inicialmente foram definidos os fundamentos essenciais para um bom entendimento dos *tokens* passando pelas *blockchains*, contratos inteligentes, plataformas, carteiras, oráculos, aplicações e finanças descentralizadas, Daos e os cenários de aplicações. Em seguida, foram apresentados as estruturas e os padrões de *tokens* utilizados em rede *blockchain*. Padrões estes que deram origem aos NFTs e que tomou grande proporção no ano de 2021, com milhões de transações realizadas em *blockchains*. Este padrão, que representa objetos como artes, itens de jogos e colecionáveis, é negociado normalmente de forma online e com criptomoedas.

Foram elencados os principais aspectos de segurança relacionados ao desenvolvimento de contratos inteligentes para NFTs, bem como os desafios desta tecnologia emergente. Os padrões sugerem boas práticas que devem ser seguidas para manter a aplicação descentralizada segura e livre de vulnerabilidades que possam ser exploradas. Relacionamos a camada que provê a interface, que fica entre todas as tecnologias envolvidas na estrutura dos NFTs e os usuários, como a mais vulnerável. Portanto atenção a esta camada devem ser despendidas para que, sendo esta implementada normalmente de forma centralizada, não influencie de maneira negativa em uma estrutura descentralizada e segura.

Apresentamos como os NFTs podem fornecer a certificação de propriedade de arte digital, documentos, direitos de propriedade intelectual, licenças e marcas por meio de registros em *blockchain* permanentemente. Desta maneira vimos as funções de criação e comercialização de um NFT, podendo armazená-lo em uma carteira de ativos digitais e alugá-lo virtualmente. Como resultado destes registros podemos verificar a verdadeira propriedade e origem de ativos de forma confiável.

Abordamos na prática a criação de NFTs e como um *Marketplace* pode ser criado a partir dos recursos de contratos inteligentes para manipular os *tokens*. Observamos que é tecnicamente possível realizar a tokenização de um ativo físico e dividi-lo em partes como um NFT. Desta forma, pode-se permitir a concessão de direitos parciais a diferentes proprietários de NFTs representando parte de um mesmo ativo físico.

Por fim, este capítulo objetivou proporcionar uma base de conhecimento e visão do estado da arte para interessados na área de *blockchain* e NFTs, bem como possibilitar o aprendizado inicial destas tecnologias procedendo com implementações de práticas das soluções para NFTs.

[21] `https://opensea.io/`
[22] `https://rarible.com/`

Referências

[Androulaki et al. 2018] Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., et al. (2018). Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*, pages 1–15.

[Bao and Roubaud 2022] Bao, H. and Roubaud, D. (2022). Recent development in fintech: Non-fungible token. *FinTech*, 1(1):44–46.

[Bartoletti et al. 2020] Bartoletti, M., Carta, S., Cimoli, T., and Saia, R. (2020). Dissecting ponzi schemes on ethereum: identification, analysis, and impact. *Future Generation Computer Systems*, 102:259–277.

[Borkowski et al. 2019] Borkowski, M., Sigwart, M., Frauenthaler, P., Hukkinen, T., and Schulte, S. (2019). Dextt: Deterministic Cross-Blockchain Token Transfers. *IEEE Access*, 7:111030–111042.

[Buterin et al. 2014] Buterin, V. et al. (2014). Ethereum: A next-generation smart contract and decentralized application platform. *URL: https://github.com/ethereum/wiki/wiki/White-Paper*, 7.

[Cachin et al. 2016] Cachin, C. et al. (2016). Architecture of the hyperledger blockchain fabric. In *Workshop on distributed cryptocurrencies and consensus ledgers*. Chicago.

[Çağlayan Aksoy and Özkan Üner 2021] Çağlayan Aksoy, P. and Özkan Üner, Z. (2021). Nfts and copyright: challenges and opportunities. *Journal Of Intellectual Property Law and Practice*, 16(10):1115–1126.

[Casale-Brunet et al. 2021] Casale-Brunet, S., Ribeca, P., Doyle, P., and Mattavelli, M. (2021). Networks of ethereum non-fungible tokens: A graph-based analysis of the erc-721 ecosystem. In *2021 IEEE International Conference on Blockchain (Blockchain)*, pages 188–195. IEEE.

[Charles 2013] Charles, P. (2013). Openzeppelin. `https://github.com/OpenZeppelin`.

[Chohan 2021] Chohan, U. W. (2021). Non-fungible tokens: Blockchains, scarcity, and value. *Critical Blockchain Research Initiative (CBRI) Working Papers*.

[Dolgui et al. 2020] Dolgui, A., Ivanov, D., Potryasaev, S., Sokolov, B., Ivanova, M., and Werner, F. (2020). Blockchain-oriented dynamic modelling of smart contract design and execution in the supply chain. *International Journal of Production Research*, 58(7):2184–2199.

[El Faqir et al. 2020] El Faqir, Y., Arroyo, J., and Hassan, S. (2020). An overview of decentralized autonomous organizations on the blockchain. In *Proceedings of the 16th international symposium on open collaboration*, pages 1–8.

[Entriken et al. 2018] Entriken, W., Shirley, D., Evans, J., and Sachs, N. (2018). Eip-721: Non-fungible token standard,"ethereum improvement proposals, no. 721, january 2018. https://eips.ethereum.org/EIPS/eip-721.

[Fairfield 2021] Fairfield, J. (2021). Tokenized: The law of non-fungible tokens and unique digital property. *Indiana Law Journal, Forthcoming*.

[Gordon and Catalini 2018] Gordon, W. J. and Catalini, C. (2018). Blockchain technology for healthcare: Facilitating the transition to patient-driven interoperability. *Computational and Structural Biotechnology Journal*, 16:224 – 230.

[Hartel and Schumi 2020] Hartel, P. and Schumi, R. (2020). Mutation testing of smart contracts at scale. In *International Conference on Tests and Proofs*, pages 23–42. Springer.

[Harz and Knottenbelt 2018] Harz, D. and Knottenbelt, W. (2018). Towards safer smart contracts: A survey of languages and verification methods. *arXiv preprint arXiv:1809.09805*.

[Hasanova et al. 2019] Hasanova, H., jun Baek, U., gon Shin, M., Cho, K., and Kim, M. S. (2019). A survey on blockchain cybersecurity vulnerabilities and possible countermeasures. *International Journal of Network Management*, 29(2):1–36.

[Hewa et al. 2021a] Hewa, T., Ylianttila, M., and Liyanage, M. (2021a). Survey on blockchain based smart contracts: Applications, opportunities and challenges. *Journal of Network and Computer Applications*, 177:102857.

[Hewa et al. 2021b] Hewa, T. M., Hu, Y., Liyanage, M., Kanhare, S. S., and Ylianttila, M. (2021b). Survey on blockchain-based smart contracts: Technical aspects and future research. *IEEE Access*, 9:87643–87662.

[Jentzsch 2016] Jentzsch, C. (2016). Decentralized autonomous organization to automate governance. *White paper, November*.

[Johnson et al. 2019] Johnson, S., Robinson, P., and Brainard, J. (2019). Sidechains and interoperability.

[Kemmoe et al. 2020] Kemmoe, V. Y., Stone, W., Kim, J., Kim, D., and Son, J. (2020). Recent advances in smart contracts: A technical overview and state of the art. *IEEE Access*, 8:117782–117801.

[Luu et al. 2016] Luu, L., Chu, D.-H., Olickel, H., Saxena, P., and Hobor, A. (2016). Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 254–269.

[Monrat et al. 2020] Monrat, A. A., Schelen, O., and Andersson, K. (2020). Performance Evaluation of Permissioned Blockchain Platforms. *2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering, CSDE 2020*.

[Nadini et al. 2021] Nadini, M., Alessandretti, L., Di Giacinto, F., Martino, M., Aiello, L. M., and Baronchelli, A. (2021). Mapping the nft revolution: market trends, trade networks, and visual features. *Scientific reports*, 11(1):1–11.

[Nakamoto and Bitcoin 2008] Nakamoto, S. and Bitcoin, A. (2008). A peer-to-peer electronic cash system. *Bitcoin.–URL: https://bitcoin. org/bitcoin. pdf*, 4.

[Pal et al. 2021] Pal, O., Alam, B., Thakur, V., and Singh, S. (2021). Key management for blockchain technology. *ICT Express*, 7(1):76–80.

[Pillai et al. 2019] Pillai, A., Saraswat, V., and V. R., A. (2019). Smart wallets on blockchain—attacks and their costs. In Wang, G., El Saddik, A., Lai, X., Martinez Perez, G., and Choo, K.-K. R., editors, *Smart City and Informatization*, pages 649–660, Singapore. Springer Singapore.

[Pillai et al. 2017] Pillai, B., Muthukkumarasamy, V., and Biswas, K. (2017). Challenges in designing a blockchain platform.

[Radomski et al. 2018] Radomski, W., Cooke, A., Castonguay, P., Therien, J., Binet, E., and Sandford, R. (2018). Eip-1155: Multi token standard,"ethereum improvement proposals, no. 1155, june 2018. `https://eips.ethereum.org/EIPS/eip-1155`.

[Rogers et al. 2022] Rogers, I., Carter, D., Morgan, B., and Edgington, A. (2022). Diminishing dreams: The scoping down of the music nft. *M/C Journal*, 25(2).

[Rosenfeld 2012] Rosenfeld, M. (2012). Overview of colored coins. *White paper, bitcoil. co. il*, 41:94.

[Rouhani and Deters 2017] Rouhani, S. and Deters, R. (2017). Performance analysis of ethereum transactions in private blockchain. In *2017 8th IEEE(ICSESS)*, pages 70–74. IEEE.

[Rouhani and Deters 2019] Rouhani, S. and Deters, R. (2019). Security, performance, and applications of smart contracts: A systematic survey. *IEEE Access*, 7:50759–50779.

[Sayeed et al. 2020] Sayeed, S., Marco-Gisbert, H., and Caira, T. (2020). Smart contract: Attacks and protections. *IEEE Access*, 8:24416–24427.

[Schär 2020] Schär, F. (2020). Decentralized finance: On blockchain-and smart contract-based financial markets. *Available at SSRN 3571335*.

[Stradling and Voorhees 2018] Stradling, A. and Voorhees, E. (2018). System and method of providing a multi-validator oracle. US Patent App. 15/715,770.

[Valeonti et al. 2021] Valeonti, F., Bikakis, A., Terras, M., Speed, C., Hudson-Smith, A., and Chalkias, K. (2021). Crypto collectibles, museum funding and openglam: Challenges, opportunities and the potential of non-fungible tokens (nfts). *Applied Sciences*, 11(21).

[Vogelsteller and Buterin 2015] Vogelsteller, F. and Buterin, V. (2015). Eip-20: Token standard, ethereum improvement proposals. `https://eips.ethereum.org/EIPS/eip-20`.

[Wang et al. 2021] Wang, Q., Li, R., Wang, Q., and Chen, S. (2021). Non-fungible token (nft): Overview, evaluation, opportunities and challenges. *arXiv preprint arXiv:2105.07447*.

[Wang et al. 2018] Wang, X., Feng, Q., and Chai, J. (2018). The research of consortium blockchain dynamic consensus based on data transaction evaluation. In *2018 11th International Symposium on Computational Intelligence and Design (ISCID)*, volume 2, pages 214–217. IEEE.

[Wood 2014] Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32.

[Xia et al. 2020] Xia, P., Wang, H., Zhang, B., Ji, R., Gao, B., Wu, L., Luo, X., and Xu, G. (2020). Characterizing cryptocurrency exchange scams. *Computers & Security*, 98:101993.

[Xiao et al. 2020] Xiao, Y., Zhang, N., Lou, W., and Hou, Y. T. (2020). A survey of distributed consensus protocols for blockchain networks. *IEEE Communications Surveys & Tutorials*, 22(2):1432–1465.

[Zou et al. 2019] Zou, W., Lo, D., Kochhar, P. S., Le, X.-B. D., Xia, X., Feng, Y., Chen, Z., and Xu, B. (2019). Smart contract development: Challenges and opportunities. *IEEE Transactions on Software Engineering*.