



UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Conrado Zeitune Ruas

**DESENVOLVIMENTO DE UMA PLATAFORMA WEB3 PARA CRIAÇÃO E
COMERCIALIZAÇÃO DE NON-FUNGIBLE TOKENS EM BLOCKCHAIN**

Florianópolis
2026

Conrado Zeitune Ruas

**DESENVOLVIMENTO DE UMA PLATAFORMA WEB3 PARA CRIAÇÃO E
COMERCIALIZAÇÃO DE NON-FUNGIBLE TOKENS EM BLOCKCHAIN**

Trabalho de Conclusão de Curso submetido ao Curso de Graduação em Ciência da Computação do Centro Tecnológico da Universidade Federal de Santa Catarina para obtenção do título de Bacharel em Ciência da Computação.

Supervisor: Prof. Jean Everson Martina,
Dr.

Florianópolis
2026

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Ruas, Conrado Zeitune

Desenvolvimento de uma Plataforma Web3 para Criação e Comercialização de Non-Fungible Tokens em Blockchain : CryptoMint: um marketplace não custodiante com mint de aleatoriedade verificável e royalties automatizados na rede Ethereum / Conrado Zeitune Ruas ; orientador, Jean Everson Martina, coorientador, Liverson Paulo Furtado Severo, 2026.

96 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2026.

Inclui referências.

1. Ciências da Computação. 2. Web3. 3. Blockchain. 4. NFT. 5. Marketplace. I. Martina, Jean Everson. II. Severo, Liverson Paulo Furtado. III. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. IV. Título.

Conrado Zeitune Ruas

**DESENVOLVIMENTO DE UMA PLATAFORMA WEB3 PARA CRIAÇÃO E
COMERCIALIZAÇÃO DE NON-FUNGIBLE TOKENS EM BLOCKCHAIN**

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de Bacharel em Ciência da Computação, e aprovado em sua forma final pelo Curso de Graduação em Ciência da Computação.

Florianópolis, 26 de maio de 2026.

Profa. Lúcia Helena Martins Pacheco, Dra. Coordenadora do Curso

Banca Examinadora:

Prof. Jean Everson Martina, Dr.
Orientador

Liverson Paulo Furtado Severo
Coorientador

Alex Sandro Roschildt Pinto, Dr.
Avaliador

AGRADECIMENTOS

A realização deste trabalho representa não apenas a conclusão de uma etapa acadêmica, mas também o resultado do apoio e incentivo de pessoas fundamentais ao longo desta trajetória, às quais expresso minha sincera gratidão.

Agradeço, primeiramente, a Deus, por me conceder força, saúde, sabedoria e perseverança ao longo desta jornada. Sou grato por ter me permitido ser filho, irmão, namorado, amigo e aluno de pessoas extraordinárias, que fizeram e continuam fazendo a diferença em minha vida.

À minha mãe, dedico meu mais profundo agradecimento por todo o suporte oferecido ao longo da minha vida. Seu amor, dedicação, incentivo e apoio incondicional foram essenciais em cada etapa da minha caminhada, proporcionando a base necessária para enfrentar desafios e alcançar objetivos.

À minha irmã, agradeço por ter compartilhado grande parte desta caminhada comigo e por ter exercido papel fundamental na minha escolha pela área da Ciência da Computação. Seu incentivo e influência foram decisivos para que eu seguisse este caminho, sendo meu maior exemplo de dedicação aos estudos.

Ao meu pai, expresso minha profunda gratidão por todo o apoio e incentivo oferecidos ao longo desta jornada, fundamentais para minha formação pessoal e profissional. Além de pai, você sempre foi um grande amigo, oferecendo companheirismo, conselhos e ensinamentos fundamentais.

À minha namorada, agradeço pelo carinho, companheirismo, compreensão e paciência durante este período, especialmente pelas longas noites dedicadas ao desenvolvimento deste trabalho. Seu apoio constante tornou esta caminhada mais leve, mesmo nos momentos de maior dificuldade.

Aos meus amigos, agradeço pela amizade, pelo companheirismo e pelos inúmeros momentos compartilhados ao longo desta trajetória. O apoio, as conversas e os momentos de descontração foram fundamentais para tornar esta caminhada mais significativa.

Aos meus professores, expresso minha sincera gratidão pelo conhecimento compartilhado ao longo da graduação, essencial para minha formação acadêmica e profissional, bem como pelos ensinamentos que ultrapassam a sala de aula.

Por fim, agradeço à Universidade Federal de Santa Catarina, pela infraestrutura, pelo ambiente acadêmico e pelas oportunidades proporcionadas ao longo desta trajetória, fundamentais para meu desenvolvimento pessoal e profissional.

RESUMO

O crescimento do ecossistema Web3 e a consolidação dos Non-Fungible Tokens (NFTs) como mecanismo de representação de posse digital verificável em blockchain tornaram evidentes limitações estruturais dos marketplaces de NFTs existentes: custódia centralizada de ativos, falta de transparência na atribuição de raridades no processo de mint e evasão de royalties devidos a criadores em transações de revenda. O presente trabalho propõe e implementa o CryptoMint, uma plataforma Web3 descentralizada para criação e comercialização de NFTs construída sobre a blockchain Ethereum, que aborda esses problemas por meio de design de contratos inteligentes, padrões de segurança estabelecidos e arquitetura descentralizada. O sistema é composto por três camadas: contratos inteligentes em Solidity 0.8.20, implementados com a toolchain Foundry e as bibliotecas OpenZeppelin v5, que incluem os contratos NFTCollectionV2 — responsável pelo mint de NFTs com aleatoriedade verificável via esquema commit-reveal e algoritmo Fisher-Yates — e NFTMarketplace — responsável por listagens a preço fixo, sistema de ofertas com ETH em escrow e distribuição automática de royalties via padrão ERC-2981; um subgraph no protocolo The Graph, que indexa os eventos emitidos pelos contratos em um banco de dados GraphQL com treze entidades, viabilizando consultas complexas sobre o estado do marketplace; e uma aplicação frontend desenvolvida com Next.js 16, React 19 e TypeScript, integrada a carteiras Web3 via wagmi e viem, que adota uma política de dados subgraph-first com fallback autoritativo via RPC para operações financeiras. A validação do sistema foi realizada por meio de uma suíte de mais de quarenta casos de teste Foundry, incluindo testes de segurança com contratos atacantes de reentrada, análise estática com Slither e análise comparativa de custos de gas e postura de segurança com os protocolos OpenSea Seaport v1.6 e Rarible Protocol v2. Os resultados demonstram que o CryptoMint atende plenamente aos objetivos propostos, implementando com qualidade de produção os mecanismos de custódia não custodiante, aleatoriedade verificável e automação de royalties, e constituindo uma referência técnica para o desenvolvimento de aplicações descentralizadas no ecossistema Ethereum.

Palavras-chave: Non-Fungible Tokens. Blockchain. Ethereum. Contratos Inteligentes. Solidity. Web3. IPFS. The Graph. Next.js.

ABSTRACT

The growth of the Web3 ecosystem and the consolidation of Non-Fungible Tokens (NFTs) as a mechanism for verifiable digital ownership on blockchain have exposed structural limitations in existing NFT marketplaces: centralized asset custody, lack of transparency in the rarity assignment process during minting, and evasion of royalties owed to creators in secondary sales. This work proposes and implements CryptoMint, a decentralized Web3 platform for the creation and trading of NFTs built on the Ethereum blockchain, addressing these problems through smart contract design, established security patterns, and a decentralized architecture. The system comprises three layers: Solidity 0.8.20 smart contracts, developed with the Foundry toolchain and OpenZeppelin v5 libraries, including the NFTCollectionV2 contract — responsible for minting NFTs with verifiable randomness via a commit-reveal scheme and Fisher-Yates algorithm — and the NFTMarketplace contract — responsible for fixed-price listings, an ETH-escrowed offer system, and automatic royalty distribution via the ERC-2981 standard; a subgraph on The Graph Protocol, which indexes contract-emitted events into a GraphQL database with thirteen entities, enabling complex marketplace state queries; and a frontend application built with Next.js 16, React 19, and TypeScript, integrated with Web3 wallets via wagmi and viem, adopting a subgraph-first data policy with authoritative RPC fallback for financial operations. System validation was conducted through a suite of over forty Foundry test cases, including security tests with reentrancy attacker contracts, static analysis with Slither, and a comparative analysis of gas costs and security posture against the OpenSea Seaport v1.6 and Rarible Protocol v2. Results demonstrate that CryptoMint fully meets the proposed objectives, implementing production-quality mechanisms for non-custodial asset management, verifiable randomness, and royalty automation, constituting a technical reference for decentralized application development in the Ethereum ecosystem.

Keywords: Non-Fungible Tokens. Blockchain. Ethereum. Smart Contracts. Solidity. Web3. IPFS. The Graph. Next.js.

LISTA DE SIGLAS E ABREVIACÕES

ABNT	Associação Brasileira de Normas Técnicas
API	Application Programming Interface
CEI	Checks-Effects-Interactions
CID	Content Identifier
DAG	Directed Acyclic Graph
DApp	Decentralized Application
DHT	Distributed Hash Table
EIP	Ethereum Improvement Proposal
ERC	Ethereum Request for Comments
ETH	Ether
EVM	Ethereum Virtual Machine
IPFS	InterPlanetary File System
NFT	Non-Fungible Token
PoS	Proof of Stake
PoW	Proof of Work
RPC	Remote Procedure Call
SSG	Static Site Generation
SSR	Server-Side Rendering
URI	Uniform Resource Identifier
WASM	WebAssembly
Web3	World Wide Web, terceira geração

SUMÁRIO

1	INTRODUÇÃO	14
1.1	Contextualização e Motivação	14
1.2	Problematização	15
1.3	Objetivos	16
1.3.1	Objetivo Geral	16
1.3.2	Objetivos Específicos	16
1.4	Organização do Trabalho	17
2	FUNDAMENTAÇÃO TEÓRICA	19
2.1	Web3 e a Terceira Geração da Internet	19
2.2	Aplicações Descentralizadas (DApps)	20
2.3	Tecnologia Blockchain	22
2.4	Ethereum e Criptomoedas	24
2.5	Contratos Inteligentes e Solidity	25
2.6	Aleatoriedade Verificável em Contratos Inteligentes	27
2.6.1	Esquemas de Compromisso e o Padrão Commit-Reveal	28
2.6.2	O Algoritmo de Embaralhamento de Fisher-Yates	28
2.7	Non-Fungible Tokens (NFTs)	29
2.7.1	Padrão ERC-721	29
2.7.2	Padrão ERC-2981 — Royalties	30
2.8	Protocolo IPFS	31
2.9	The Graph e Subgraphs	33
2.10	Tecnologias de Frontend: Next.js, React e TypeScript	34
2.10.1	React	35
2.10.2	Next.js	35
2.10.3	TypeScript	36
2.11	Trabalhos Relacionados	37
2.11.1	OpenSea e o Modelo Custodial de Primeira Geração	37
2.11.2	OpenSea Seaport v1.6	38
2.11.3	Rarible Protocol v2	38
2.11.4	Zora Protocol	38
2.11.5	Foundation	39
2.11.6	Posicionamento do CryptoMint em Relação aos Trabalhos Relacionados	39
3	METODOLOGIA E PROJETO	40

3.1	Classificação Metodológica da Pesquisa	40
3.2	<i>Design Science Research</i> como Abordagem de Desenvolvimento	41
3.3	Processo de Desenvolvimento e Ferramentas	42
3.4	Concepção e Modelagem do Sistema	43
3.5	Requisitos Funcionais	44
3.6	Requisitos Não Funcionais	45
4	DESENVOLVIMENTO DO SISTEMA	47
4.1	Visão Geral da Arquitetura	47
4.2	Contratos Inteligentes	53
4.2.1	NFTCollectionFactory e NFTCollectionV2	53
4.2.2	NFTMarketplace	56
4.2.3	Padrões de Segurança Aplicados	57
4.3	Indexação com The Graph (Subgraph)	59
4.3.1	Motivação e Filosofia de Design Subgraph-First	59
4.3.2	Estrutura do Subgraph	59
4.3.3	Circuit-Breaker e Degradação Graciosa	61
4.4	Frontend	62
4.4.1	Arquitetura Next.js e Proxies Server-Side	62
4.4.2	Integração Web3: wagmi, viem e ConnectKit	63
4.4.3	Apollo Client e Padrão Subgraph-First	64
4.4.4	Armazenamento Descentralizado via Pinata IPFS	65
5	RESULTADOS E AVALIAÇÃO	66
5.1	Funcionalidades Implementadas	66
5.2	Testes e Validação	68
5.2.1	Testes de Contratos Inteligentes (Foundry/Forge)	68
5.2.2	Testes de Frontend (Vitest)	69
5.3	Análise de Gas e Desempenho	69
5.3.1	Custo de Gas por Operação	69
5.3.2	Comparação com Marketplaces de Produção	70
5.4	Análise de Segurança	73
5.4.1	Análise Estática com Slither	73
5.4.2	Vulnerabilidades Mitigadas	73
5.4.3	Limitações de Segurança Identificadas	73
6	APRESENTAÇÃO E DEMONSTRAÇÃO DA PLATAFORMA CRYPTOMINT	75
6.1	Visão Geral da Interface e Navegação	75
6.2	Página Inicial	77

6.3	Criação de Coleções	77
6.4	Cunhagem de NFTs	79
6.5	Exploração e Descoberta de Ativos	80
6.6	Página da Coleção	82
6.7	Página do Ativo e Operações de Comercialização	82
6.8	Perfil do Usuário	83
6.9	Feed de Atividades	85
6.10	Síntese das Funcionalidades	86
7	CONCLUSÃO	88
7.1	Considerações Finais	88
7.2	Contribuições Técnicas	89
7.3	Limitações Reconhecidas	90
7.4	Trabalhos Futuros	91
	REFERÊNCIAS BIBLIOGRÁFICAS	93
	APÊNDICE A – ARTIGO NO FORMATO SBC	97
	APÊNDICE B – REPOSITÓRIO DO CÓDIGO-FONTE DO PROJETO	105

LISTA DE ILUSTRAÇÕES

Figura 1 – Diagrama de Contexto C4 — CryptoMint NFT Marketplace	49
Figura 2 – Diagrama de Contêineres C4 — CryptoMint NFT Marketplace . . .	51
Figura 3 – Diagrama de Componentes C4 — CryptoMint NFT Marketplace . .	52
Figura 4 – Elementos interativos da barra de navegação nos temas escuro e claro	76
Figura 5 – Página inicial da plataforma CryptoMint	77
Figura 6 – Assistente de criação de coleção	79
Figura 7 – Tela de cunhagem (<i>mint</i>) de NFTs	80
Figura 8 – Tela de exploração de ativos com painel de filtros	81
Figura 9 – Página de coleções com <i>ranking</i> por desempenho	81
Figura 10 – Página de uma coleção	82
Figura 11 – Página de um ativo com as operações de comercialização	83
Figura 12 – Página de perfil do usuário	84
Figura 13 – Formulário de edição de perfil	85
Figura 14 – Página de perfil antes e depois da edição das informações	85
Figura 15 – Feed de atividades do <i>marketplace</i>	86

LISTA DE QUADROS

Quadro 1 – Aderência das etapas do trabalho ao método de <i>Design Science Research</i>	42
Quadro 2 – Ferramentas e tecnologias por etapa de desenvolvimento	43
Quadro 3 – Categorias de testes de contratos inteligentes	69

LISTA DE TABELAS

Tabela 1 – Requisitos funcionais do CryptoMint	45
Tabela 2 – Requisitos não funcionais do CryptoMint	46
Tabela 3 – Contratos implantados na rede de testes Ethereum Sepolia	53
Tabela 4 – Funcionalidades implementadas no CryptoMint	67
Tabela 5 – Custo de gas das operações do NFTMarketplace	70
Tabela 6 – Comparativo de custo de gas por operação: CryptoMint vs. Seaport v1.6	71
Tabela 7 – Comparativo arquitetural: CryptoMint, Seaport v1.6 e Rarible v2 . .	72
Tabela 8 – Vulnerabilidades analisadas e mitigações implementadas	74
Tabela 9 – Mapeamento entre telas e requisitos funcionais atendidos	87

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO E MOTIVAÇÃO

A última década testemunhou uma transformação profunda no ecossistema da internet, impulsionada pelo surgimento das tecnologias blockchain e pela crescente adoção do paradigma conhecido como Web3. Diferentemente da Web 2.0, dominada por plataformas centralizadas que intermediam e monetizam as interações dos usuários sem lhes conferir controle ou propriedade sobre seus dados e ativos digitais, a Web3 propõe um modelo radicalmente distinto: um ambiente computacional distribuído, no qual as regras são expressas em código imutável verificável publicamente, a propriedade digital é conferida por criptografia de chave pública e a confiança é estabelecida por protocolo, não por reputação corporativa (WOOD, 2014; NAKAMOTO, 2008).

Nesse contexto, os **Non-Fungible Tokens** (NFTs) emergiram como um dos casos de uso mais expressivos da tecnologia blockchain, ao oferecer uma representação digital de posse verificável e não replicável de ativos únicos — como obras de arte digital, itens de jogos, credenciais e domínios — registrada de forma permanente e auditável em uma blockchain pública (ENTRIKEN et al., 2018; WANG et al., 2021). O volume de transações no mercado de NFTs atingiu cifras significativas ao longo de 2021 e 2022, atraindo a atenção de artistas, colecionadores, pesquisadores e desenvolvedores de todo o mundo, e consolidando-se como uma área de pesquisa e desenvolvimento de crescente relevância acadêmica e prática (NADINI et al., 2021).

Paralelamente ao crescimento do mercado, tornaram-se evidentes limitações estruturais nos principais *marketplaces* de NFTs existentes. Plataformas de primeira geração, como as primeiras versões do OpenSea, adotavam modelos de custódia centralizada — nos quais os ativos dos usuários eram transferidos para contratos controlados pela plataforma —, e seus mecanismos de distribuição de royalties dependiam

de acordos informais sem garantias *on-chain*. A concentração de poder nessas plataformas expõe os usuários a riscos de perda de ativos por decisões unilaterais dos operadores, incapacidade de auditar as regras de negócio aplicadas e ausência de garantias sobre o pagamento de royalties a criadores em transações de revenda.

A crescente sofisticação técnica do ecossistema Ethereum — com o amadurecimento de padrões como ERC-721, ERC-2981, e a consolidação de ferramentas de desenvolvimento como Solidity, Foundry e The Graph — abre a possibilidade de desenvolver plataformas de *marketplace* que endereçam essas limitações de forma técnica e verificável, sem depender da boa-fé de um operador centralizado. O presente trabalho propõe e implementa o **CryptoMint**, uma plataforma Web3 para criação e comercialização de NFTs que aborda diretamente essas questões por meio de design de contratos inteligentes, padrões de segurança estabelecidos e arquitetura descentralizada.

1.2 PROBLEMATIZAÇÃO

A concepção do CryptoMint foi motivada pela identificação de três problemas estruturais recorrentes nos *marketplaces* de NFTs existentes, documentados tanto na literatura técnica quanto na experiência prática do ecossistema.

O primeiro problema é a **custódia centralizada de ativos**. Plataformas que custodiavam os NFTs dos usuários durante o período de listagem apresentavam o risco inerente de que uma falha técnica, decisão regulatória ou comprometimento de segurança da plataforma resultasse em perda de acesso aos ativos. O CryptoMint adota o modelo de *marketplace* não custodiante (*non-custodial*): os NFTs permanecem na carteira do proprietário durante toda a listagem, e o contrato de *marketplace* apenas recebe uma aprovação `setApprovalForAll` — uma delegação limitada de autoridade para executar a transferência exclusivamente no momento de uma venda efetivada.

O segundo problema é a **falta de transparência e verificabilidade na alea-**

toriedade do *mint*. Coleções generativas — que atribuem atributos de raridade aleatoriamente a cada *token* durante o processo de *mint* — frequentemente realizavam esse sorteio em servidores centralizados, impossibilitando a auditoria pública do processo. Essa opacidade abre espaço para manipulação da distribuição de raridades em benefício de compradores privilegiados ou do próprio criador — prática conhecida no ecossistema como *rug pull* de raridades. O CryptoMint implementa um mecanismo de *commit-reveal* combinado com o algoritmo Fisher-Yates diretamente no contrato inteligente, tornando o processo de atribuição de raridades publicamente auditável e matematicamente verificável por qualquer participante.

O terceiro problema é a **evasão de pagamento de royalties a criadores**. O padrão ERC-721 original não inclui mecanismos de royalties, e contratos de *marketplace* que não implementam o padrão ERC-2981 podem simplesmente não pagar royalties ao criador em transações de revenda — uma prática que se tornou polêmica quando grandes plataformas tornaram os royalties opcionais para os compradores. O NFTMarketplace do CryptoMint implementa o ERC-2981 e executa o pagamento de royalties de forma automática e obrigatória em cada transação de venda mediada pelo contrato, com um teto de 10% sobre o valor da venda para proteger compradores de valores abusivos.

1.3 OBJETIVOS

1.3.1 Objetivo Geral

Projetar, implementar e avaliar uma plataforma Web3 descentralizada para criação e comercialização de Non-Fungible Tokens na *blockchain* Ethereum, com ênfase em custódia não custodiante, transparência no processo de *mint* e automação de royalties.

1.3.2 Objetivos Específicos

Para alcançar o objetivo geral, foram estabelecidos os seguintes objetivos específicos:

1. implementar contratos inteligentes em Solidity para a criação de coleções NFT, com mecanismo de *mint* de aleatoriedade verificável — por meio do esquema *commit-reveal* combinado com o algoritmo Fisher-Yates — e suporte ao padrão de royalties ERC-2981;
2. desenvolver um contrato de *marketplace* não custodiante, com listagens a preço fixo, sistema de ofertas com ETH em *escrow* e distribuição automática de royalties e taxas de plataforma, garantindo que nenhum ativo seja transferido para a custódia do contrato durante o período de listagem;
3. implementar um *subgraph* no protocolo The Graph para a indexação eficiente dos eventos *on-chain*, habilitando consultas GraphQL complexas sobre o estado do *marketplace* sem sobrecarregar o provedor RPC com dezenas de chamadas individuais;
4. desenvolver uma interface de usuário com Next.js 16 que integre os contratos inteligentes e o *subgraph*, oferecendo uma experiência de uso completa do *marketplace* com suporte a carteiras Web3;
5. validar a implementação por meio de testes unitários, de integração e de segurança, avaliando os custos de *gas* e a postura de segurança do sistema em comparação com protocolos de produção.

1.4 ORGANIZAÇÃO DO TRABALHO

O presente trabalho está organizado em sete capítulos, cujos conteúdos são descritos a seguir.

O **Capítulo 2 — Fundamentação Teórica** apresenta os conceitos e tecnologias necessários para a compreensão do sistema desenvolvido, abrangendo Web3 e aplicações descentralizadas, a tecnologia *blockchain* e a plataforma Ethereum, contratos inteligentes e a linguagem Solidity, o padrão de *tokens* não fungíveis ERC-721 e o padrão de royalties ERC-2981, o protocolo de armazenamento descentralizado IPFS,

o protocolo de indexação The Graph e, por fim, as tecnologias de *frontend* empregadas: Next.js, React e TypeScript.

O **Capítulo 3 — Metodologia e Projeto** apresenta a classificação metodológica da pesquisa, a abordagem de *Design Science Research* que orientou a construção do artefato, o processo de desenvolvimento e as ferramentas empregadas, a concepção do sistema e a especificação dos requisitos funcionais e não funcionais.

O **Capítulo 4 — Desenvolvimento do Sistema** descreve a arquitetura geral do CryptoMint e detalha a implementação de cada uma das três camadas do sistema: os contratos inteligentes (NFTCollectionV2, NFTCollectionFactoryV2 e NFTMarketplace), o *subgraph* com seus *handlers* e *schema* GraphQL, e o *frontend* com sua integração Web3 e fluxos de IPFS.

O **Capítulo 5 — Resultados e Avaliação** apresenta as funcionalidades implementadas, os resultados dos testes de contratos inteligentes e *frontend*, a análise de custos de *gas* e a comparação arquitetural com protocolos de produção, e a análise de segurança conduzida com ferramentas estáticas e testes de contratos atacantes.

O **Capítulo 6 — Apresentação e Demonstração da Plataforma CryptoMint** apresenta o sistema sob a perspectiva do usuário final, percorrendo suas principais telas e demonstrando como as funcionalidades especificadas se materializam na interface.

O **Capítulo 7 — Conclusão** retoma os objetivos do trabalho, avalia o grau de alcance de cada objetivo específico, discute as contribuições técnicas do sistema e aponta as limitações reconhecidas e as direções para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 WEB3 E A TERCEIRA GERAÇÃO DA INTERNET

A trajetória da *World Wide Web* pode ser compreendida em fases evolutivas que refletem transformações profundas na relação entre usuários, dados e infraestrutura tecnológica. A primeira geração da *web*, convencionalmente denominada Web 1.0, caracterizou-se pela natureza estática e unidirecional das páginas: os usuários consumiam conteúdo publicado por um número restrito de produtores, sem capacidade de interação ou contribuição direta (BERNERS-LEE; FISCHETTI, 1999). O modelo predominante era o de documentos hiperlinkados servidos por servidores centralizados, acessíveis por meio de navegadores, sem mecanismos de autenticação sofisticados ou personalização dinâmica.

A segunda geração, a Web 2.0, emergiu no início dos anos 2000 e trouxe consigo a participação ativa dos usuários, a geração colaborativa de conteúdo e o surgimento de plataformas como redes sociais, motores de busca e serviços em nuvem (O'REILLY, 2007). Embora esse paradigma tenha democratizado a produção de conteúdo e viabilizado a economia digital, ele também concentrou o controle de dados e infraestrutura nas mãos de um número reduzido de grandes corporações tecnológicas. Nesse modelo, os dados dos usuários passaram a ser o principal ativo explorado por essas plataformas, sem que os próprios usuários detivessem controle efetivo sobre suas informações ou sobre as regras que governam os serviços que utilizam.

Em resposta a essa concentração de poder e à erosão da autonomia do usuário, Gavin Wood, cofundador do projeto Ethereum, cunhou o termo **Web3** em 2014 para descrever uma nova geração de aplicações construídas sobre infraestruturas descentralizadas e protocolos abertos (WOOD, 2014). A proposta central do paradigma Web3 é substituir os intermediários centralizados por mecanismos criptográficos e redes de consenso distribuídas, de forma que a confiança deixe de ser depositada em

uma entidade corporativa e passe a residir no próprio protocolo verificável publicamente. Nesse contexto, a propriedade digital, a identidade e a soberania sobre dados pertencem ao usuário, representado por chaves criptográficas, e não a plataformas intermediárias.

Do ponto de vista técnico, o ecossistema Web3 apoia-se em três pilares fundamentais. O primeiro é a infraestrutura de *blockchain*, que provê um registro imutável, auditável e distribuído de transações sem a necessidade de uma autoridade central (NAKAMOTO, 2008; ZHENG et al., 2017). O segundo é a criptografia de chave pública, que garante autenticidade e controle exclusivo sobre ativos digitais por meio de pares de chaves criptográficas. O terceiro são os protocolos de armazenamento descentralizado, como o IPFS, que eliminam a dependência de servidores centralizados para hospedagem de dados (BENET, 2014). A plataforma Ethereum, em particular, tornou-se a principal infraestrutura da Web3 ao introduzir contratos inteligentes — programas de computador executados deterministicamente na *blockchain* —, permitindo que regras de negócio complexas sejam implementadas sem a intervenção de terceiros (BUTERIN, 2013).

No contexto do presente trabalho, o paradigma Web3 constitui a fundação sobre a qual o CryptoMint foi projetado. Ao delegar a custódia de ativos digitais a contratos inteligentes e ao utilizar o protocolo IPFS para armazenamento de metadados, o sistema elimina a dependência de servidores centralizados e garante que as regras do *marketplace* sejam aplicadas de forma transparente e auditável por qualquer participante da rede.

2.2 APLICAÇÕES DESCENTRALIZADAS (DAPPS)

As aplicações descentralizadas, comumente denominadas **DApps** (*Decentralized Applications*), representam uma categoria de *software* cujo *backend* opera sobre redes de *blockchain* ou outros sistemas de computação distribuída, em contraste com as aplicações tradicionais que dependem de servidores centralizados administrados por uma organização específica. Johnston et al. (2014) propuseram uma das primei-

ras definições formais para DApps, estabelecendo quatro critérios fundamentais: o código-fonte deve ser de código aberto (*open source*) e operar de forma autônoma; os dados e registros de operação devem ser armazenados em uma *blockchain* descentralizada; a aplicação deve utilizar *tokens* criptográficos como mecanismo de acesso e recompensa; e os *tokens* devem ser gerados por meio de um algoritmo de consenso padronizado.

A principal distinção entre DApps e aplicações Web 2.0 convencionais reside no modelo de confiança. Em uma aplicação centralizada, o usuário deposita confiança na organização que controla o servidor: confia que seus dados serão protegidos, que as regras serão aplicadas de forma justa e que o serviço estará disponível continuamente. Em uma DApp, essa confiança é transferida para o código do contrato inteligente, que é imutável após o *deploy*, publicamente auditável e executado deterministicamente por toda a rede de validadores (ANTONPOULOS; WOOD, 2018). Dessa forma, as regras de negócio tornam-se transparentes e não modificáveis unilateralmente por qualquer entidade.

Sob o aspecto arquitetural, uma DApp típica é composta por duas camadas principais. A camada de *smart contracts*, implantada na *blockchain*, encapsula a lógica de negócio, o armazenamento persistente de estado e a movimentação de ativos. A camada de interface do usuário, frequentemente uma aplicação *web* ou *mobile*, interage com os contratos por meio de chamadas RPC (*Remote Procedure Call*) assinadas criptograficamente pela carteira do usuário (DANNEN, 2017). Essa separação permite que múltiplos *frontends* sejam desenvolvidos sobre os mesmos contratos, favorecendo a interoperabilidade e a resiliência do sistema.

É importante reconhecer que as DApps enfrentam desafios específicos que as diferenciam das aplicações tradicionais. A escalabilidade das redes de *blockchain*, os custos de *gas* associados a cada operação *on-chain* e a experiência do usuário — que requer gerenciamento de carteiras e assinatura de transações — representam barreiras de adoção não triviais (ZHENG et al., 2017). No entanto, para casos de uso que demandam alta transparência, eliminação de intermediários e execução confiável

de regras de negócio, as DApps oferecem vantagens estruturais significativas. O mercado de NFTs é um desses casos: a propriedade de ativos digitais únicos, a execução automática de royalties e a custódia dos *tokens* pelo próprio titular são garantias que apenas uma arquitetura descentralizada pode prover de forma verificável.

O CryptoMint segue o modelo de DApp ao manter toda a lógica de negócio — listagem, compra, ofertas, royalties e taxas — em contratos inteligentes imutáveis implantados na Ethereum Sepolia. O *frontend* Next.js atua como interface de usuário, sem armazenar estado crítico ou intermediar transações financeiras; todas as operações monetárias são executadas diretamente na *blockchain*, com os NFTs permanecendo sob custódia exclusiva do titular em sua carteira.

2.3 TECNOLOGIA BLOCKCHAIN

A tecnologia *blockchain* foi introduzida em 2008 por Satoshi Nakamoto no artigo seminal que descreve o protocolo Bitcoin, concebido como um sistema de dinheiro eletrônico *peer-to-peer* capaz de processar transações sem a necessidade de uma instituição financeira intermediária (NAKAMOTO, 2008). O mecanismo central do protocolo é uma estrutura de dados encadeada por funções criptográficas de *hash*, na qual cada bloco contém um conjunto de transações validadas, um *timestamp*, metadados e o *hash* do bloco precedente, formando uma cadeia historicamente verificável e praticamente imutável.

Do ponto de vista estrutural, uma *blockchain* pode ser entendida como um banco de dados distribuído e replicado entre todos os participantes (*nós*) de uma rede *peer-to-peer*. Cada nó mantém uma cópia completa do histórico de transações e participa do processo de validação de novos blocos por meio de um protocolo de consenso. A integridade do registro é garantida por dois mecanismos complementares: a criptografia de *hash*, que torna computacionalmente inviável alterar um bloco sem refazer toda a cadeia subsequente, e o consenso distribuído, que exige concordância da maioria dos nós para que um novo bloco seja aceito (ZHENG et al., 2017). O resultado é um registro que, na prática, é imutável e auditável por qualquer participante da rede.

Cada bloco é composto, essencialmente, por um cabeçalho e um corpo. O cabeçalho contém o *hash* do bloco anterior (*parent hash*), o *hash* raiz de uma Árvore de Merkle que resume todas as transações do bloco, um *nonce* utilizado no processo de mineração e metadados como *timestamp* e dificuldade de mineração. O corpo contém a lista de transações validadas. A Árvore de Merkle (*Merkle Tree*), proposta por Ralph Merkle em 1987, permite verificar de forma eficiente a inclusão de uma transação específica no bloco sem a necessidade de baixar todo o seu conteúdo, o que é especialmente relevante para clientes leves (*light clients*) que não armazenam a *blockchain* completa (SWAN, 2015).

Os mecanismos de consenso determinam como os nós da rede chegam a um acordo sobre qual versão da *blockchain* é a legítima. O protocolo original do Bitcoin utiliza o mecanismo de Prova de Trabalho (*Proof of Work*, PoW), no qual os mineradores competem para resolver um problema computacional custoso, e o vencedor tem o direito de propor o próximo bloco e receber uma recompensa em criptomoeda (NAKAMOTO, 2008). Embora o PoW seja robusto contra ataques de duplo gasto, ele apresenta alto consumo energético e baixa escalabilidade em termos de taxa de transações por segundo. Em resposta, a Ethereum concluiu em 2022 a transição para o mecanismo de Prova de Participação (*Proof of Stake*, PoS), denominado *The Merge*, no qual os validadores bloqueiam (*stake*) *ether* como garantia colateral para propor e atestar blocos, reduzindo o consumo energético em aproximadamente 99,95% sem sacrificar as propriedades de segurança e descentralização da rede (Ethereum Foundation, 2022).

No contexto do presente trabalho, a *blockchain* da Ethereum Sepolia serve como substrato computacional para todos os contratos inteligentes do CryptoMint. A imutabilidade da *blockchain* garante que o histórico de criações de coleções, listagens, vendas, ofertas e pagamentos de royalties seja auditável por qualquer participante, sem que qualquer entidade — incluindo os desenvolvedores do sistema — possa modificar retroativamente esses registros. Essa propriedade é fundamental para estabelecer a confiança dos usuários em um *marketplace* que opera sem custodiante

centralizado.

2.4 ETHEREUM E CRIPTOMOEDAS

A Ethereum é uma plataforma de *blockchain* de propósito geral, lançada em 2015 por Vitalik Buterin e um grupo de colaboradores, projetada para suportar a execução de contratos inteligentes e aplicações descentralizadas. Diferentemente do Bitcoin, cuja *blockchain* foi concebida primariamente como sistema de transferência de valor, a Ethereum introduziu uma máquina virtual de estados completa de Turing — a **Ethereum Virtual Machine** (EVM) —, capaz de executar programas arbitrários de forma determinística em todos os nós da rede (WOOD, 2014). Essa generalidade transformou a Ethereum na principal plataforma de desenvolvimento de DApps e no substrato sobre o qual a maior parte do ecossistema de NFTs foi construída.

O modelo de contas da Ethereum distingue dois tipos de entidades. As **Contas de Propriedade Externa** (*Externally Owned Accounts*, EOA) são controladas por chaves privadas criptográficas e representam usuários e carteiras externas; elas podem iniciar transações, mas não possuem código executável associado. As **Contas de Contrato** (*Contract Accounts*) são controladas pelo código de um contrato inteligente armazenado na *blockchain*; elas respondem a chamadas externas e internas executando sua lógica de forma determinística, sem intervenção humana (ANTONPOULOS; WOOD, 2018). Essa distinção é fundamental para compreender como o *marketplace* CryptoMint opera: os usuários interagem com a plataforma por meio de suas EOAs, enquanto as regras de negócio residem nas contas de contrato dos *smart contracts* implantados.

Cada operação executada na EVM consome uma quantidade de recursos computacionais mensurada em unidades chamadas **gas**. O *gas* é o mecanismo que previne a execução indefinida de programas (o problema da parada) e compensa os validadores pelo esforço computacional. O custo em *gas* de cada operação é denominado em *wei* (a menor unidade da criptomoeda Ether, sendo 1 Ether equivalente a 10^{18} *wei*), e o custo total de uma transação é calculado como o produto entre o *gas* utilizado e

o preço do *gas*, definido pelo usuário em unidades de *gwei* (10^9 *wei*) (WOOD, 2014). A otimização do consumo de *gas* é, portanto, uma preocupação central no *design* de contratos inteligentes destinados a produção, pois custos excessivos impactam diretamente a experiência do usuário.

O **Ether** (ETH) é a criptomoeda nativa da Ethereum, desempenhando dois papéis essenciais no ecossistema: serve como unidade de pagamento das taxas de *gas* para execução de transações e como meio de troca para transferência de valor entre contas. Criptomoedas, em sentido amplo, são ativos digitais baseados em criptografia que operam sobre redes descentralizadas, sem emissão ou controle por parte de governos ou bancos centrais (NAKAMOTO, 2008). No contexto do CryptoMint, o Ether é utilizado como moeda de troca em todas as operações comerciais: compras de NFTs, pagamento de taxas de *mint*, pagamentos de royalties e taxas de plataforma são todos denominados e liquidados em ETH, sem a necessidade de intermediação financeira.

Para fins de desenvolvimento e testes acadêmicos, o projeto CryptoMint foi implantado na rede de testes **Sepolia**, uma rede de *blockchain* mantida pela Ethereum Foundation que replica o comportamento da *mainnet* sem envolver ativos financeiros reais. O uso de uma *testnet* permite validar o comportamento dos contratos e da interface em condições reais de execução sem expor usuários a riscos financeiros, sendo uma prática padrão no desenvolvimento de aplicações *blockchain* (Ethereum Foundation, 2022).

2.5 CONTRATOS INTELIGENTES E SOLIDITY

O conceito de **contratos inteligentes** (*smart contracts*) foi proposto pelo jurista e cientista da computação Nick Szabo em 1994, antes mesmo da existência das *blockchains* modernas. Szabo (1994) definiu contratos inteligentes como protocolos de transação computadorizados que executam os termos de um contrato, com o objetivo de satisfazer condições contratuais comuns, minimizar exceções — acidentais e maliciosas — e reduzir a necessidade de intermediários. A ideia permaneceu teórica por duas décadas, até que a Ethereum materializou o conceito ao fornecer um ambiente

de execução distribuído, verificável e resistente à censura.

Na plataforma Ethereum, um contrato inteligente é um programa armazenado de forma permanente na *blockchain*, cuja execução é desencadeada por transações ou chamadas de outras contas. Uma vez implantado, o código do contrato é imutável: ele não pode ser alterado ou removido por nenhuma entidade, incluindo o desenvolvedor original, exceto se o próprio contrato implementar mecanismos explícitos de atualização ou autodestruição (ANTONPOULOS; WOOD, 2018). Essa imutabilidade é ao mesmo tempo a principal garantia de confiança — as regras são fixas e auditáveis — e um dos maiores desafios do desenvolvimento, pois qualquer vulnerabilidade no código se torna permanente após o *deploy*.

A linguagem de programação dominante para a escrita de contratos inteligentes na Ethereum é o **Solidity**, criada por Gavin Wood e colaboradores e lançada em 2014. Solidity é uma linguagem estaticamente tipada, de alto nível, com sintaxe inspirada em JavaScript e C++, compilada para *bytecode* da EVM (DANNEN, 2017). Entre suas características relevantes estão a definição explícita de visibilidade de funções (*public*, *private*, *internal*, *external*), o controle de modificadores de estado (*view*, *pure*, *payable*), os eventos (*events*) para emissão de *logs* indexáveis, e os modificadores (*modifiers*) para reuso de lógica de controle de acesso. O ecossistema Solidity é enriquecido por bibliotecas auditadas e amplamente utilizadas, como as da **OpenZeppelin**, que fornecem implementações seguras de padrões de *tokens* e controles de acesso (OpenZeppelin, 2023). A análise empírica conduzida por Bartoletti e Pompianu (2017) sobre milhares de contratos implantados evidenciou que o desenvolvimento de contratos inteligentes converge para um conjunto recorrente de padrões de projeto — como os de *token*, autorização e *pull-payment* —, vários dos quais são adotados no presente trabalho e detalhados no Capítulo 4.

O processo de desenvolvimento de contratos inteligentes envolve, além da escrita em Solidity, a compilação para *bytecode* da EVM, a escrita de testes abrangentes e a análise estática de segurança antes do *deploy*. No presente trabalho, adotou-se o **Foundry** como *toolchain* principal, uma suíte de ferramentas de desenvolvimento para

Solidity implementada em Rust, que oferece compilação rápida, um *framework* de testes nativo (Forge), um cliente RPC local (Anvil) e uma ferramenta de *deploy* por linha de comando (Cast) (Paradigm, 2024). O Foundry foi selecionado por sua performance superior, pela capacidade de escrever testes em Solidity com acesso nativo a *cheatcodes* de manipulação de estado da EVM, e pela integração natural com ferramentas de análise estática como o Slither.

No contexto do CryptoMint, os contratos inteligentes representam o único componente verdadeiramente descentralizado e imutável do sistema. Toda a lógica de criação de coleções, *mint* de NFTs, listagem, compra, ofertas e distribuição de royalties e taxas está encapsulada em contratos Solidity auditáveis. Essa escolha garante que as regras do *marketplace* sejam transparentes, não possam ser alteradas unilateralmente após o *deploy* e sejam aplicadas de forma automática a cada transação, sem necessidade de intervenção de um operador humano.

2.6 ALEATORIEDADE VERIFICÁVEL EM CONTRATOS INTELIGENTES

A geração de números aleatórios em uma *blockchain* é um problema notoriamente difícil. A EVM é determinística por necessidade — todos os nós da rede precisam chegar exatamente ao mesmo resultado ao reexecutar uma transação —, de modo que não dispõe de uma fonte de entropia segura nativa e qualquer aleatoriedade utilizada deve ser reproduzível. As fontes ingênuas frequentemente empregadas, como `block.timestamp`, `blockhash` ou `block.prevrandao`, podem ser observadas ou, em alguma medida, influenciadas pelos validadores responsáveis por propor blocos, o que abre espaço para manipulação do resultado em benefício próprio (ATZEI; BARTOLLETTI; CIMOLI, 2017). Em coleções generativas de NFTs, nas quais a aleatoriedade determina a distribuição de raridades, essa fragilidade é especialmente sensível. Esta seção apresenta os dois fundamentos que, combinados, sustentam o mecanismo de aleatoriedade verificável adotado neste trabalho: o esquema *commit-reveal* e o algoritmo de embaralhamento de Fisher-Yates.

2.6.1 Esquemas de Compromisso e o Padrão Commit-Reveal

Uma técnica criptográfica clássica para mitigar a manipulação de valores é o **esquema de compromisso** (*commitment scheme*), formalizado por [Blum \(1983\)](#) no contexto do protocolo de “lançamento de moeda por telefone” (*coin flipping by telephone*). Um esquema de compromisso opera em duas fases. Na fase de **compromisso** (*commit*), uma parte publica um valor que a vincula a um segredo previamente escolhido — tipicamente o *hash* criptográfico desse segredo — sem, contudo, revelá-lo. Na fase de **revelação** (*reveal*), a parte divulga o segredo, e qualquer observador pode verificar que ele corresponde ao compromisso publicado anteriormente.

O esquema apoia-se em duas propriedades fundamentais: o **sigilo** (*hiding*), pois o compromisso publicado nada revela sobre o segredo; e a **vinculação** (*binding*), pois, após o *commit*, a parte não consegue apresentar um segredo diferente do comprometido sem ser detectada. Aplicado à geração de aleatoriedade *on-chain*, o padrão *commit-reveal* combina o valor comprometido com uma fonte de entropia que só se torna conhecida *após* o compromisso — como o *hash* de um bloco futuro. Uma vez que nenhuma das partes consegue prever essa entropia no momento do *commit* nem alterar o valor comprometido posteriormente, o resultado é uma aleatoriedade que não pode ser manipulada unilateralmente e que, com a posterior revelação do segredo, torna-se publicamente auditável.

2.6.2 O Algoritmo de Embaralhamento de Fisher-Yates

Definida a fonte de aleatoriedade, resta o problema de utilizá-la para produzir uma **permutação não enviesada** de um conjunto finito de elementos — no caso deste trabalho, a associação aleatória dos metadados (URIs) aos *tokens* cunhados. O **algoritmo de embaralhamento de Fisher-Yates**, originalmente proposto por [Fisher e Yates \(1938\)](#) e popularizado em sua forma computacional moderna por [Knuth \(1997\)](#), resolve esse problema de maneira ótima.

O algoritmo percorre a sequência e, para cada posição, troca o elemento corrente por um elemento escolhido uniformemente entre aqueles ainda não fixados. Sua

principal garantia é a **uniformidade**: cada uma das $n!$ permutações possíveis de uma sequência de n elementos tem exatamente a mesma probabilidade de ocorrer, desde que a fonte de aleatoriedade subjacente seja uniforme. Adicionalmente, o algoritmo executa em tempo linear $O(n)$ e opera *in-place*, no próprio espaço da sequência, o que o torna particularmente adequado a ambientes com forte restrição de recursos como a EVM, na qual cada operação tem custo de *gas*.

Essa combinação de ausência de viés e eficiência distingue o Fisher-Yates de abordagens ingênuas — como ordenar os elementos por uma chave aleatória ou sortear índices repetidamente, descartando repetições —, que introduzem viés estatístico ou apresentam custo computacional superior. A aplicação conjunta do padrão *commit-reveal* com o algoritmo de Fisher-Yates, detalhada no Capítulo 4, constitui o mecanismo de aleatoriedade verificável que fundamenta a atribuição transparente de raridades no CryptoMint.

2.7 NON-FUNGIBLE TOKENS (NFTS)

Non-Fungible Tokens (NFTs), em português *tokens* não fungíveis, são representações digitais de ativos únicos e indivisíveis registradas em uma *blockchain*. O conceito de fungibilidade, proveniente da economia, diferencia bens substituíveis por outros de mesma espécie e qualidade — como moedas — daqueles que possuem características únicas e não intercambiáveis (WANG et al., 2021). No contexto de ativos digitais, o Ether é um *token* fungível: qualquer unidade é equivalente a qualquer outra. Em contraste, um NFT possui um identificador único (*tokenId*) que o distingue de todos os outros *tokens*, mesmo que estes compartilhem o mesmo contrato de origem.

2.7.1 Padrão ERC-721

O padrão técnico que rege a implementação de NFTs na Ethereum foi formalizado no **ERC-721** (*Ethereum Request for Comments 721*), proposto por William Entriken, Dieter Shirley, Jacob Evans e Nastassia Sachs em 2018 (ENTRIKEN et al., 2018). O ERC-721 define uma interface mínima que todo contrato de *token* não fun-

gível deve implementar, incluindo funções essenciais como `balanceOf(address)`, que retorna a quantidade de *tokens* detidos por um endereço; `ownerOf(uint256 tokenId)`, que retorna o proprietário de um *token* específico; e `safeTransferFrom(address from, address to, uint256 tokenId)`, que executa a transferência de um *token* garantindo que o destinatário seja capaz de receber NFTs (no caso de contratos, verificando a implementação da interface `IERC721Receiver`). O padrão também especifica os mecanismos de delegação de aprovação — `approve` e `setApprovalForAll` —, que permitem que terceiros (como *marketplaces*) executem transferências em nome do proprietário sem custodiar o ativo.

A associação entre um NFT e seus metadados — como nome, descrição, imagem e atributos — é estabelecida pela função `tokenURI(uint256 tokenId)`, que retorna um URI (*Uniform Resource Identifier*) apontando para um documento JSON conforme o padrão de metadados do ERC-721. Em projetos que valorizam a descentralização, esse URI aponta para o **IPFS** (*InterPlanetary File System*), garantindo que os metadados sejam imutáveis e não possam ser substituídos unilateralmente pelo desenvolvedor após o *mint*. O CryptoMint adota essa abordagem: os metadados de cada *token* são fixados no IPFS no momento da criação da coleção, e o URI correspondente é gravado no contrato durante o processo de *mint* (BENET, 2014).

2.7.2 Padrão ERC-2981 — Royalties

A questão dos royalties para criadores em revendas de NFTs representa um desafio técnico significativo, pois o padrão ERC-721 original não define mecanismos para sua distribuição automática. O **ERC-2981**, proposto em 2020, padroniza uma interface de consulta de royalties que permite que *marketplaces* recuperem, de forma programática, o receptor e o percentual de royalty para qualquer *token* de uma coleção (BURKS et al., 2020). A função `royaltyInfo(uint256 tokenId, uint256 salePrice)` retorna o endereço do receptor e o valor a ser pago, cabendo ao *marketplace* incorporar esse pagamento na lógica de venda. O NFTMarketplace do CryptoMint implementa integralmente o ERC-2981: em cada venda, o contrato consulta as informações

de royalty via `staticcall` limitado a 30.000 *gas* (para prevenir ataques de DoS por meio de royalties maliciosos), calcula o valor correspondente e executa o pagamento automático ao receptor designado pelo criador, com um teto obrigatório de 10% sobre o valor da venda.

Desde o mercado especulativo de 2021, os NFTs tornaram-se objeto crescente de estudo acadêmico. [Nadini et al. \(2021\)](#), em análise abrangente de 6,1 milhões de transações em seis mercados de NFTs, identificaram padrões de redes de comércio, características visuais correlacionadas com valor e a estrutura de mercado dominada por um pequeno número de coleções de alto valor. [Wang et al. \(2021\)](#) oferecem uma visão técnica detalhada dos mecanismos de NFTs, suas aplicações em arte digital, jogos e metaverso, e os desafios ainda não resolvidos, como escalabilidade, sustentabilidade ambiental e questões de direito autoral. Esses estudos evidenciam a relevância e a complexidade do domínio abordado pelo CryptoMint, que busca contribuir com uma implementação de referência que endereça questões técnicas específicas — como transparência no *mint* e automação de royalties — identificadas como problemas em aberto na literatura.

2.8 PROTOCOLO IPFS

O **IPFS** (*InterPlanetary File System*) é um protocolo de hipermídia distribuído e um sistema de arquivos *peer-to-peer* desenvolvido por Juan Benet e lançado em 2015. O IPFS foi concebido para substituir o modelo de localização de conteúdo baseado em URL (*location-based addressing*), predominante na *web* atual, por um modelo de endereçamento baseado em conteúdo (*content-based addressing*), no qual cada arquivo é identificado por um *hash* criptográfico de seus dados, denominado **CID** (*Content Identifier*) ([BENET, 2014](#)). Essa mudança de paradigma tem consequências profundas: ao invés de perguntar “*onde está o arquivo X?*”, o protocolo pergunta “*qual arquivo corresponde ao hash H?*”, garantindo que o conteúdo entregue seja exatamente o solicitado, independentemente de qual nó da rede o serviu.

O IPFS combina conceitos de diversas tecnologias distribuídas maduras. A es-

estrutura de dados do sistema — a **DAG Merkle** (*Merkle Directed Acyclic Graph*) — é uma extensão das Árvore de Merkle que permite representar arquivos arbitrariamente grandes como grafos de blocos endereçáveis, onde cada bloco é identificado pelo *hash* de seu conteúdo (BENET, 2014). O protocolo de roteamento e descoberta de nós é baseado em uma **Kademlia DHT** (*Distributed Hash Table*), que permite localizar quais nós da rede possuem um dado bloco de forma eficiente. A transferência de dados entre nós utiliza o protocolo **Bitswap**, inspirado no BitTorrent, que incentiva os pares a compartilhar blocos reciprocamente. Finalmente, a descoberta de nós locais é facilitada pelo protocolo **mDNS**, e as conexões entre nós são estabelecidas via **libp2p**, uma biblioteca modular de protocolos de rede *peer-to-peer*.

A propriedade de imutabilidade do endereçamento por conteúdo é particularmente valiosa no contexto de NFTs. Uma vez que os metadados de um *token* — incluindo a imagem e os atributos — são fixados (*pinned*) no IPFS, o CID que os identifica é determinístico e permanente: qualquer alteração nos metadados produziria um CID completamente diferente. Ao armazenar o CID no contrato inteligente como `tokenURI`, garante-se que a associação entre o *token* e seus metadados seja imutável e verificável por qualquer pessoa com acesso à *blockchain* (WANG et al., 2021). Isso elimina uma vulnerabilidade comum em projetos de NFT que armazenam metadados em servidores *web* centralizados: a possibilidade de o desenvolvedor substituir a imagem associada a um *token* após a compra — prática conhecida como *rug pull* de metadados.

Para que um arquivo permaneça acessível no IPFS, é necessário que ao menos um nó da rede o mantenha *pinned* — isto é, o inclua em seu conjunto de dados armazenados persistentemente. Serviços de *pinning* gerenciado, como o **Pinata**, oferecem infraestrutura dedicada para esse fim, garantindo alta disponibilidade dos arquivos mesmo que o nó do desenvolvedor fique *offline*. O CryptoMint utiliza o Pinata como serviço de *pinning* para todas as imagens e arquivos de metadados JSON das coleções NFT criadas na plataforma, integrando-o via sua API REST por meio de *Route Handlers* do Next.js no lado do servidor, de forma que as chaves de API nunca

sejam expostas ao *bundle* do cliente.

2.9 THE GRAPH E SUBGRAPHS

O **The Graph** é um protocolo descentralizado de indexação e consulta de dados de *blockchain*, lançado em 2020. Sua principal finalidade é resolver um problema estrutural das *blockchains*: embora elas ofereçam um registro imutável e auditável de todas as transações, as interfaces de consulta nativas são extremamente limitadas. As chamadas RPC padrão (`eth_call`, `eth_getLogs`) permitem consultar o estado atual de variáveis de armazenamento de contratos ou filtrar *logs* de eventos por endereço e tópico, mas não suportam operações complexas como agregações, junções entre contratos, ordenação por múltiplos campos ou consultas paginadas eficientes ([The Graph Foundation, 2021](#)). Para uma interface de *marketplace* que precisa exibir todos os NFTs de uma coleção com seus preços de listagem, histórico de ofertas e estatísticas de volume, esse modelo seria proibitivamente caro e lento se implementado exclusivamente via RPC.

O The Graph resolve esse problema por meio do conceito de **subgraph**, uma unidade de indexação que declara quais contratos monitorar, quais eventos processar e como transformar esses eventos em entidades estruturadas de um banco de dados GraphQL. Um *subgraph* é definido por três componentes principais: o **manifest** (`<subgraph.yaml>`), que especifica as fontes de dados (contratos, redes, blocos de início), os eventos mapeados e os *handlers* correspondentes; o **schema** (`<schema.graphql>`), que define os tipos de entidades e suas relações no modelo de dados GraphQL; e os **handlers** `AssemblyScript` (`<src/*.ts>`), funções que são executadas a cada vez que um evento monitorado é emitido, lendo dados do evento e atualizando as entidades no banco de dados ([The Graph Foundation, 2021](#)).

Do ponto de vista técnico, a execução dos *handlers* ocorre em um ambiente de *sandbox* WebAssembly (WASM), compilado a partir de código **AssemblyScript** — um subconjunto de TypeScript compilável para WASM. Essa abordagem garante determinismo na execução dos *handlers* entre diferentes nós do protocolo descen-

tralizado, que é fundamental para o consenso sobre o estado indexado. O banco de dados subjacente é um PostgreSQL gerenciado pelo *graph-node*, o *software* de referência que coordena a sincronização com a *blockchain* e a execução dos *handlers*. O serviço **Graph Studio**, utilizado no CryptoMint, é uma versão hospedada gerenciada pela Graph Foundation que elimina a necessidade de operar infraestrutura própria de *graph-node* e PostgreSQL.

Um recurso avançado do protocolo é a criação de **templates dinâmicos** (*Data Source Templates*), que permitem iniciar a indexação de contratos cujos endereços não são conhecidos em tempo de implantação do *subgraph*. No CryptoMint, isso é essencial: cada vez que um usuário cria uma nova coleção NFT via `NFTCollectionFactoryV2`, um novo contrato `NFTCollectionV2` é implantado em um endereço único. O *handler* do evento `CollectionCreated` captura esse endereço e invoca `<NFTCollection.create(event.params.contractAddress)>`, instruindo o *graph-node* a começar a indexar eventos do novo contrato a partir daquele bloco. Essa abordagem permite que o sistema escale horizontalmente para N coleções sem qualquer alteração no código do *subgraph* ou do contrato de *factory*.

O *frontend* do CryptoMint adota uma filosofia **subgraph-first**: todas as consultas de leitura não críticas (listagens, exploração de coleções, *feed* de atividade, estatísticas) são realizadas via GraphQL ao *subgraph*, que fornece dados desnormalizados e altamente consultáveis com baixa latência. Consultas autoritativas — como o preço exato de uma listagem imediatamente antes de uma compra — e todas as operações de escrita (transações) são realizadas diretamente via RPC, garantindo dados atualizados ao momento da ação.

2.10 TECNOLOGIAS DE FRONTEND: NEXT.JS, REACT E TYPESCRIPT

O desenvolvimento da interface do CryptoMint empregou um conjunto de tecnologias modernas de *frontend* cujo entendimento é necessário para compreender as decisões arquiteturais da camada de apresentação. A seguir, apresenta-se uma fundamentação dos três pilares tecnológicos: o *framework* Next.js, a biblioteca React e a

linguagem TypeScript.

2.10.1 React

O **React** é uma biblioteca JavaScript de código aberto desenvolvida pelo Facebook (atualmente Meta) e lançada em 2013, voltada para a construção de interfaces de usuário por meio de uma abordagem baseada em componentes ([Meta Platforms, 2013](#)). No modelo de programação do React, a interface é decomposta em componentes — funções JavaScript que recebem propriedades (*props*) e retornam elementos de interface — que podem ser reutilizados, compostos e gerenciados de forma independente. O mecanismo central de eficiência do React é o **Virtual DOM** (*Document Object Model* virtual): ao invés de manipular diretamente o DOM real do navegador a cada atualização de estado, o React mantém uma representação em memória da interface, compara (*diff*) o estado atual com o estado desejado e aplica ao DOM apenas as mudanças mínimas necessárias, resultando em maior desempenho em interfaces com atualizações frequentes ([GACKENHEIMER, 2015](#)).

A introdução dos **hooks** na versão 16.8 do React transformou o modelo de desenvolvimento de componentes, substituindo as classes como mecanismo primário para gerenciamento de estado e efeitos colaterais. *Hooks* como `useState`, `useEffect`, `useContext` e `useReducer` permitem adicionar estado e lógica de ciclo de vida a componentes funcionais, enquanto *hooks* customizados possibilitam a extração e o reuso de lógica *stateful* entre componentes. No CryptoMint, os *hooks* customizados encapsulam casos de uso completos — como `useListNFT`, `useBuyNFT` e `useAcceptOffer` —, abstraindo toda a complexidade de interação com a *blockchain* em interfaces simples e testáveis.

2.10.2 Next.js

O **Next.js** é um *framework* de desenvolvimento *web* baseado em React, desenvolvido pela Vercel e lançado em 2016, que adiciona ao React funcionalidades essenciais para aplicações *web* de produção, como renderização no lado do servidor

(*Server-Side Rendering*, SSR), geração estática de sites (*Static Site Generation*, SSG), roteamento baseado em sistema de arquivos, otimização automática de imagens e suporte a *Route Handlers* (API Routes) no lado do servidor (Vercel, 2024). O CryptoMint utiliza o **App Router**, o sistema de roteamento moderno introduzido no Next.js 13, que é baseado em componentes de servidor React (*React Server Components*) e organizado em torno do sistema de arquivos: diretórios dentro de `app/` mapeiam automaticamente para rotas, e arquivos especiais como `layout.tsx`, `page.tsx` e `loading.tsx` controlam o comportamento de cada segmento de rota.

Uma característica particularmente relevante do Next.js para aplicações *blockchain* é a capacidade de criar **Route Handlers** — *endpoints* de API executados exclusivamente no servidor. No CryptoMint, esse mecanismo é utilizado para criar *proxies* de segurança: as chaves de API do Alchemy (provedor RPC) e do Pinata (IPFS) são injetadas pelo servidor nas requisições, sem jamais serem incluídas no *bundle* JavaScript enviado ao navegador do cliente. Dessa forma, credenciais sensíveis permanecem protegidas mesmo em uma aplicação com forte componente de *frontend*, sem a necessidade de um *backend* separado.

2.10.3 TypeScript

O **TypeScript** é uma linguagem de programação de código aberto desenvolvida pela Microsoft, lançada em 2012, que estende o JavaScript com um sistema de tipos estáticos opcional e progressivo (Microsoft, 2012). A tipagem estática do TypeScript permite que o compilador detecte erros de tipo em tempo de compilação — como passar um número onde uma *string* é esperada, ou acessar uma propriedade inexistente em um objeto —, reduzindo significativamente a classe de erros que chegam ao ambiente de produção. Adicionalmente, a tipagem explícita melhora a legibilidade e a manutenibilidade do código, funciona como documentação viva das interfaces entre módulos e habilita recursos avançados de ferramentas de desenvolvimento, como autocompletar, refatoração automatizada e navegação de código.

No contexto do CryptoMint, o TypeScript é utilizado em modo `strict` com con-

figurações adicionais como `noImplicitOverride`, `noUnusedLocals` e `noUnusedParameters`, que eliminam categorias adicionais de erros sutis. A integração do TypeScript com as ABIs dos contratos Solidity — por meio de bibliotecas como a `viem`, que gera tipos TypeScript automaticamente a partir das ABIs — garante que as chamadas ao *blockchain* sejam tipadas de ponta a ponta: se a assinatura de uma função do contrato mudar, o compilador TypeScript sinalizará todos os pontos de chamada afetados no *frontend*. Essa rastreabilidade de tipos entre a camada de *blockchain* e a camada de apresentação é uma contribuição técnica significativa do *design* do sistema para a robustez e manutenibilidade do código.

2.11 TRABALHOS RELACIONADOS

O desenvolvimento de plataformas descentralizadas para criação e comercialização de *Non-Fungible Tokens* tem sido objeto crescente de atenção tanto na indústria quanto na pesquisa acadêmica, especialmente após o pico de mercado registrado entre 2021 e 2022 (NADINI et al., 2021). Esta seção examina os principais trabalhos e protocolos relacionados ao CryptoMint, identificando convergências, divergências e as contribuições específicas deste trabalho em relação ao estado da arte.

2.11.1 OpenSea e o Modelo Custodial de Primeira Geração

O OpenSea consolidou-se como o maior *marketplace* de NFTs em volume de transações, operando inicialmente sob um modelo de custódia centralizada de listagens no qual as ordens de venda eram armazenadas em servidores proprietários e as transferências de NFTs dependiam de contratos controlados pela própria plataforma (WANG et al., 2021). Embora eficiente em termos operacionais, esse modelo sujeitava os usuários a riscos de perda de acesso a ativos por falhas ou decisões unilaterais do operador, além de tornar opacas as regras de negócio aplicadas — limitações documentadas por Wang et al. (2021) e que constituem parte da motivação do presente trabalho, conforme descrito na Seção 1.2.

2.11.2 OpenSea Seaport v1.6

O protocolo Seaport, desenvolvido pela OpenSea e lançado em 2022, representa um avanço significativo em relação ao modelo custodial, introduzindo ordens assinadas criptograficamente por meio do padrão EIP-712 e armazenadas *off-chain*. O Seaport suporta condições de troca compostas (*consideration items*), execução em lote de múltiplas transações e custo de criação de listagem essencialmente nulo em *gas*. O CryptoMint diferencia-se do Seaport ao manter o estado de todas as listagens e ofertas diretamente *on-chain*, privilegiando a verificabilidade total sem dependência de infraestrutura *off-chain* proprietária, em detrimento do menor custo de *gas* por listagem. A análise comparativa quantitativa de desempenho e postura de segurança é apresentada no Capítulo 5.

2.11.3 Rarible Protocol v2

O Rarible Protocol é um protocolo aberto de *marketplace* com suporte a múltiplos padrões de *tokens* — incluindo ERC-721 e ERC-1155 (RADOMSKI et al., 2018) — e a múltiplas redes de *blockchain*. À semelhança do Seaport, o Rarible utiliza ordens assinadas *off-chain* para minimizar o custo de *gas* de criação de listagens, porém com uma arquitetura multi-contrato mais modular que viabiliza a extensibilidade entre diferentes redes. A comparação de postura de segurança e custos de *gas* com o Rarible Protocol é detalhada na Seção 5.3.2.

2.11.4 Zora Protocol

O Zora Protocol (Zora Labs, 2022) é um protocolo *on-chain* aberto que estendeu o modelo de *marketplace* para incluir suporte nativo ao padrão ERC-1155 (RADOMSKI et al., 2018) — que permite a criação de edições fungíveis de obras de arte digital — e mecanismos de leilão com descoberta de preço automatizada, incluindo leilões holandeses. Essa abordagem amplia significativamente os mecanismos de monetização disponíveis para criadores em relação ao modelo de preço fixo com ofertas adotado pelo CryptoMint. A ausência de suporte a ERC-1155 e a leilões *on-chain* no

presente trabalho é reconhecida como uma limitação de escopo e constitui uma das direções de trabalho futuro identificadas no Capítulo 7.

2.11.5 Foundation

A plataforma Foundation ([Foundation Labs, 2021](#)) especializou-se em leilões à inglesa para arte digital de alta curadoria, implementando um mecanismo de extensão automática do prazo quando um lance é registrado próximo ao encerramento — estratégia que previne a prática de *bid sniping* observada em leilões *on-chain* com prazo fixo. Esse mecanismo ilustra a sofisticação possível em *marketplaces* orientados à descoberta de preço por leilão, em contraste com o modelo de ofertas com expiração fixa de sete dias adotado pelo NFTMarketplace do CryptoMint.

2.11.6 Posicionamento do CryptoMint em Relação aos Trabalhos Relacionados

Em relação aos trabalhos e plataformas relacionados, o CryptoMint diferencia-se pela combinação de três características implementadas e documentadas academicamente: (i) *marketplace* não custodiante com lógica de negócio integralmente *on-chain* e auditável por qualquer participante da rede; (ii) mecanismo de aleatoriedade verificável no processo de *mint* por meio do esquema *commit-reveal* combinado com o algoritmo Fisher-Yates, sem custo de oráculo externo pago; e (iii) automação obrigatória e auditável de royalties segundo o padrão ERC-2981 ([BURKS et al., 2020](#)), com teto de proteção ao comprador. Nenhuma das plataformas relacionadas apresenta simultaneamente essas três características acompanhadas de documentação técnica detalhada e cobertura abrangente de testes formais.

3 METODOLOGIA E PROJETO

Este capítulo apresenta a metodologia adotada para o desenvolvimento do CryptoMint e descreve a concepção do sistema. Inicialmente, classifica-se a pesquisa quanto à sua natureza, objetivos, abordagem e procedimentos técnicos. Em seguida, descreve-se a abordagem de *Design Science Research* (DSR) que orientou a construção do artefato de *software*, bem como o processo de desenvolvimento iterativo-incremental e as ferramentas empregadas. Por fim, apresenta-se a concepção do sistema — com seus atores e principais fluxos — e a especificação dos requisitos funcionais e não funcionais que nortearam o projeto.

3.1 CLASSIFICAÇÃO METODOLÓGICA DA PESQUISA

A caracterização metodológica deste trabalho segue as taxonomias consolidadas na literatura de metodologia científica (GIL, 2002) e, em particular, as orientações específicas para a pesquisa em Ciência da Computação propostas por Wazlawick (2014), que reconhecem o desenvolvimento de artefatos de *software* como produto legítimo de uma investigação acadêmica.

Quanto à sua **natureza**, esta pesquisa classifica-se como **aplicada**, pois objetiva gerar conhecimento dirigido à solução de um problema concreto e específico — as limitações estruturais dos *marketplaces* de NFTs descritas no Capítulo 1 — por meio da construção de um sistema funcional. Quanto aos seus **objetivos**, a pesquisa é simultaneamente **exploratória**, ao investigar a aplicação de tecnologias emergentes do ecossistema Ethereum a um domínio ainda em consolidação, e **descritiva**, ao documentar de forma sistemática a arquitetura e os mecanismos do artefato desenvolvido. Quanto à **abordagem**, adota-se uma perspectiva predominantemente **qualitativa**, complementada por elementos quantitativos na fase de avaliação — notadamente a mensuração de custos de *gas* e a contagem de casos de teste, apresentadas no Capítulo 5. Por fim, quanto aos **procedimentos técnicos**, o trabalho combina a

pesquisa bibliográfica — base da fundamentação teórica e do levantamento do estado da arte — com a **pesquisa experimental e de desenvolvimento**, materializada na implementação, na implantação e na validação empírica do sistema.

3.2 DESIGN SCIENCE RESEARCH COMO ABORDAGEM DE DESENVOLVIMENTO

O eixo metodológico que estrutura o desenvolvimento do CryptoMint é a *Design Science Research* (DSR), paradigma de pesquisa orientado à criação e à avaliação de artefatos de tecnologia da informação projetados para resolver problemas organizacionais ou técnicos identificados (HEVNER et al., 2004). Diferentemente das ciências naturais, que buscam descrever e explicar fenômenos existentes, a DSR é fundamentalmente uma ciência do projeto: seu produto é um artefato inédito — neste caso, uma plataforma descentralizada — cuja construção e cuja avaliação constituem, em si, a contribuição de conhecimento. A escolha da DSR justifica-se pela natureza essencialmente construtiva deste trabalho, cujo cerne é a concepção de uma solução técnica para um problema bem delimitado.

Segundo Hevner et al. (2004), a pesquisa em *design science* articula-se em torno de três ciclos complementares, todos contemplados neste trabalho. O **ciclo de relevância** (*relevance cycle*) conecta o ambiente do problema à atividade de projeto: as limitações dos *marketplaces* existentes — custódia centralizada, opacidade do *mint* e evasão de royalties — estabelecem os requisitos e os critérios de aceitação do artefato. O **ciclo de rigor** (*rigor cycle*) conecta a atividade de projeto à base de conhecimento existente: padrões consagrados (ERC-721, ERC-2981, *Checks-Effects-Interactions*, *pull-payment*), bibliotecas auditadas (OpenZeppelin) e técnicas estabelecidas (esquema *commit-reveal*, algoritmo Fisher-Yates) fundamentam as decisões de projeto e asseguram seu rigor técnico. O **ciclo de projeto** (*design cycle*) itera entre a construção do artefato e sua avaliação, refinando sucessivamente a solução. Essa estruturação dialoga diretamente com a metodologia de DSR para sistemas de informação e engenharia de *software* sistematizada por Wieringa (2014), que organiza a

investigação em torno do ciclo de *engenharia* — investigação do problema, projeto do tratamento e validação do tratamento.

O Quadro 1 relaciona as atividades do método de DSR às etapas concretas deste trabalho e aos capítulos em que cada uma é documentada, evidenciando a aderência da pesquisa ao paradigma adotado.

Quadro 1 – Aderência das etapas do trabalho ao método de *Design Science Research*

Atividade da DSR	Etapas correspondente no trabalho	Capítulo
Identificação do problema e motivação	Levantamento das limitações dos <i>marketplaces</i> existentes	1
Definição dos objetivos da solução	Estabelecimento dos objetivos geral e específicos	1
Aporte da base de conhecimento	Fundamentação teórica e levantamento de padrões e tecnologias	2
Projeto e desenvolvimento	Especificação de requisitos, modelagem da arquitetura e implementação das três camadas	3, 4
Demonstração	Implantação dos contratos na rede Ethereum Sepolia e da aplicação <i>web</i>	4
Avaliação	Suíte de testes, análise estática, comparação de <i>gas</i> e de segurança	5
Comunicação	Redação da presente monografia	—

Fonte: elaborado pelo autor (2026), com base em [Hevner et al. \(2004\)](#).

3.3 PROCESSO DE DESENVOLVIMENTO E FERRAMENTAS

A construção do artefato seguiu um **processo de desenvolvimento iterativo e incremental** ([SOMMERVILLE, 2011](#)), no qual o sistema foi edificado em incrementos sucessivos, cada um agregando funcionalidades a uma base já validada. Essa escolha é coerente com o contexto de um trabalho conduzido por um único desenvolvedor e com a natureza experimental da DSR, em que o artefato amadurece por meio de ciclos de projeto-avaliação. O desenvolvimento partiu da camada mais crítica e imutável — os contratos inteligentes — e progrediu, de forma ascendente, para a camada de indexação e, em seguida, para a camada de apresentação, refletindo a relação de dependência entre as três camadas descritas no Capítulo 4.

Uma prática central adotada na camada de contratos inteligentes foi o **desenvolvimento orientado a testes** (*Test-Driven Development*, TDD), no qual os casos de teste são escritos previamente ou concomitantemente à implementação, guiando o projeto do código e estabelecendo uma rede de proteção contra regressões (BECK, 2003). Essa prática mostrou-se especialmente apropriada ao domínio de contratos inteligentes, em que falhas são irreversíveis após a implantação e o custo de uma vulnerabilidade é elevado; a suíte de mais de quarenta casos de teste Foundry, apresentada no Capítulo 5, é resultado direto dessa abordagem. A integridade do código ao longo das iterações foi mantida por meio do sistema de controle de versão Git, com o histórico hospedado no GitHub.

O Quadro 2 sintetiza as principais ferramentas e tecnologias empregadas em cada etapa do desenvolvimento, cujos fundamentos teóricos foram apresentados no Capítulo 2.

Quadro 2 – Ferramentas e tecnologias por etapa de desenvolvimento

Etapa / Camada	Ferramentas e tecnologias
Contratos inteligentes	Solidity 0.8.20, Foundry, OpenZeppelin v5
Análise de segurança	Slither, contratos atacantes de reentrada (testes Foundry)
Indexação	The Graph (<i>subgraph</i>), AssemblyScript, GraphQL
Armazenamento descentralizado	IPFS via Pinata
Frontend	Next.js 16, React 19, TypeScript, wagmi, viem, ConnectKit, Apollo Client
Infraestrutura de rede	Ethereum Sepolia, Alchemy RPC
Modelagem da arquitetura	Modelo C4
Controle de versão	Git e GitHub

Fonte: elaborado pelo autor (2026).

3.4 CONCEPÇÃO E MODELAGEM DO SISTEMA

A concepção do CryptoMint partiu da identificação dos **atores** do sistema e de suas necessidades. Dois perfis de usuário foram definidos: o **Criador**, responsável por lançar coleções e cunhar NFTs, e o **Colecionador**, que adquire, oferta e revende ati-

vos. Cabe destacar que os dois perfis não são mutuamente exclusivos — um mesmo endereço de carteira pode atuar simultaneamente como criador e colecionador —, refletindo a fluidez característica das economias de criadores no ecossistema Web3. Como ator externo não humano, a **rede Ethereum** medeia e registra de forma autoritativa todas as operações de escrita.

A partir dos atores e dos problemas a serem resolvidos, derivaram-se os principais fluxos de uso do sistema: a criação de uma coleção e o subsequente *mint* de *tokens* com aleatoriedade verificável; a listagem de um NFT a preço fixo; a compra de um *token* listado; a realização, a aceitação e a recuperação de ofertas com ETH em *escrow*; e o saque de valores pendentes. Esses fluxos foram traduzidos nos requisitos funcionais especificados na Seção 3.5 e materializados nas funções dos contratos inteligentes detalhadas no Capítulo 4.

A modelagem arquitetural do sistema adotou o **modelo C4** (BROWN, 2018) como notação para descrever a estrutura do *software* em níveis progressivos de abstração — contexto, contêineres e componentes. A decisão de organizar o sistema em três camadas desacopladas (contratos inteligentes, indexação e apresentação) foi tomada nesta fase de concepção, com vistas a satisfazer os requisitos não funcionais de manutenibilidade, testabilidade e segurança. Os diagramas C4 resultantes, por documentarem diretamente a implementação, são apresentados em conjunto com ela no Capítulo 4.

3.5 REQUISITOS FUNCIONAIS

Os requisitos funcionais (RF) especificam os serviços que o sistema deve prestar, descrevendo seu comportamento observável diante das entradas dos usuários (SOMMERVILLE, 2011). A Tabela 1 apresenta os requisitos funcionais do CryptoMint, levantados a partir dos objetivos do trabalho e dos fluxos de uso descritos na seção anterior.

Tabela 1 – Requisitos funcionais do CryptoMint

ID	Descrição
RF01	O sistema deve permitir que um criador lance uma nova coleção NFT, definindo nome, símbolo, preço de <i>mint</i> e oferta máxima.
RF02	O sistema deve permitir o <i>upload</i> de imagens e a fixação (<i>pin</i>) dos metadados associados no IPFS.
RF03	O sistema deve permitir o <i>mint</i> de NFTs com atribuição de raridades por meio de aleatoriedade verificável (esquema <i>commit-reveal</i> e algoritmo Fisher-Yates).
RF04	O sistema deve permitir que o proprietário liste um NFT para venda a preço fixo, sem transferir a custódia do ativo.
RF05	O sistema deve permitir que um colecionador compre um NFT listado mediante pagamento em ETH.
RF06	O sistema deve permitir que um colecionador realize ofertas em ETH, mantidas em <i>escrow</i> pelo contrato.
RF07	O sistema deve permitir que o proprietário de um NFT aceite uma oferta recebida.
RF08	O sistema deve permitir o cancelamento de uma listagem ativa pelo seu autor.
RF09	O sistema deve permitir a recuperação de ofertas expiradas, com recompensa (<i>bounty</i>) a quem efetuar a operação.
RF10	O sistema deve permitir que os destinatários de pagamentos saquem seus saldos pendentes (<i>pull-payment</i>).
RF11	O sistema deve distribuir royalties ao criador de forma automática e obrigatória em cada revenda, conforme o padrão ERC-2981.
RF12	O sistema deve cobrar e contabilizar a taxa de plataforma sobre cada venda mediada pelo <i>marketplace</i> .
RF13	O sistema deve permitir a conexão de carteiras Web3 (MetaMask e compatíveis) para autenticação e assinatura de transações.
RF14	O sistema deve permitir a exploração e a visualização de coleções, NFTs, listagens e ofertas, a partir dos dados indexados pelo <i>subgraph</i> .
RF15	O sistema deve apresentar o histórico de atividades e estatísticas agregadas do <i>marketplace</i> e das coleções.

Fonte: elaborado pelo autor (2026).

3.6 REQUISITOS NÃO FUNCIONAIS

Os requisitos não funcionais (RNF) expressam restrições e atributos de qualidade que o sistema deve satisfazer, não relacionados diretamente a funcionalidades específicas, mas determinantes para a aceitação do produto (SOMMERVILLE, 2011). Os requisitos não funcionais do CryptoMint foram organizados segundo as características de qualidade definidas pela norma **ISO/IEC 25010** (International Organization for

Standardization, 2011), que estabelece um modelo de qualidade de produto de *software*. A Tabela 2 relaciona cada requisito à respectiva característica de qualidade e indica o mecanismo de projeto adotado para satisfazê-lo, detalhado no Capítulo 4.

Tabela 2 – Requisitos não funcionais do CryptoMint

ID	Característica (ISO/IEC 25010)	Descrição
RNF01	Segurança	Os contratos devem ser resistentes a ataques de reentrada, mediante o modificador <i>nonReentrant</i> e a aplicação rigorosa do padrão <i>Checks-Effects-Interactions</i> .
RNF02	Segurança	O <i>marketplace</i> deve ser não custodiante, mantendo os NFTs na carteira do proprietário durante toda a listagem.
RNF03	Segurança	As chaves de API de serviços de terceiros não devem ser expostas no cliente, sendo intermediadas por <i>proxies server-side</i> .
RNF04	Adequação funcional	O processo de atribuição de raridades deve ser publicamente auditável e verificável <i>on-chain</i> .
RNF05	Eficiência de desempenho	O consumo de <i>gas</i> deve ser otimizado por meio de empacotamento de <i>storage</i> e erros customizados; as leituras devem usar o <i>subgraph</i> , evitando múltiplas chamadas RPC.
RNF06	Confiabilidade	O sistema deve degradar de forma graciosa diante de indisponibilidade do serviço de indexação, mediante mecanismo de <i>circuit-breaker</i> e <i>cache</i> .
RNF07	Usabilidade	A interface deve oferecer <i>feedback</i> claro durante operações de múltiplas etapas (aprovação e execução), por meio de uma máquina de estados.
RNF08	Manutenibilidade	O sistema deve adotar separação em camadas desacopladas e segurança de tipos de ponta a ponta, com tipos gerados automaticamente a partir das ABIs e do <i>schema GraphQL</i> .
RNF09	Compatibilidade	Os contratos devem aderir aos padrões ERC-721 e ERC-2981, garantindo interoperabilidade com carteiras e <i>marketplaces</i> de terceiros.
RNF10	Eficiência de desempenho	A arquitetura deve escalar para um número arbitrário de coleções sem alteração de código, mediante o padrão <i>Factory</i> e <i>templates</i> dinâmicos de <i>subgraph</i> .

Fonte: elaborado pelo autor (2026).

4 DESENVOLVIMENTO DO SISTEMA

4.1 VISÃO GERAL DA ARQUITETURA

O CryptoMint é uma plataforma descentralizada para criação e comercialização de NFTs organizada em três camadas interdependentes, cada qual com responsabilidades claramente delimitadas e interfaces bem definidas. A camada de contratos inteligentes, implementada em Solidity sobre a *blockchain* Ethereum Sepolia, constitui o núcleo imutável do sistema, encapsulando toda a lógica de negócio, a custódia de ETH em *escrow* e as regras de transferência de ativos. A camada de indexação, implementada como um *subgraph* no protocolo The Graph, transforma os eventos emitidos pelos contratos em um banco de dados GraphQL estruturado e eficientemente consultável, tornando viáveis as operações de leitura complexas que a interface de usuário demanda. A camada de apresentação, implementada como uma aplicação Next.js 16, integra as duas camadas anteriores, oferecendo ao usuário uma interface intuitiva para todas as operações do *marketplace*, com integração nativa a carteiras Web3.

A separação em camadas foi uma decisão arquitetural deliberada que favorece a testabilidade individual de cada componente, a substituíbilidade da interface de usuário sem alteração das regras de negócio, e a auditabilidade dos contratos de forma independente da camada de apresentação. Para documentar essa arquitetura de forma rigorosa, o presente trabalho adota o **modelo C4** (*Context, Containers, Components, Code*), proposto por [Brown \(2018\)](#) como uma abordagem hierárquica para visualização de arquiteturas de *software*. O modelo organiza a documentação em quatro níveis de abstração progressivos, dos quais os três primeiros são empregados neste capítulo: o Diagrama de Contexto apresenta o sistema em relação a seus usuários e dependências externas; o Diagrama de Contêineres detalha as grandes peças implantáveis que compõem o sistema; e o Diagrama de Componentes expõe os elementos internos de cada contêiner e suas inter-relações.

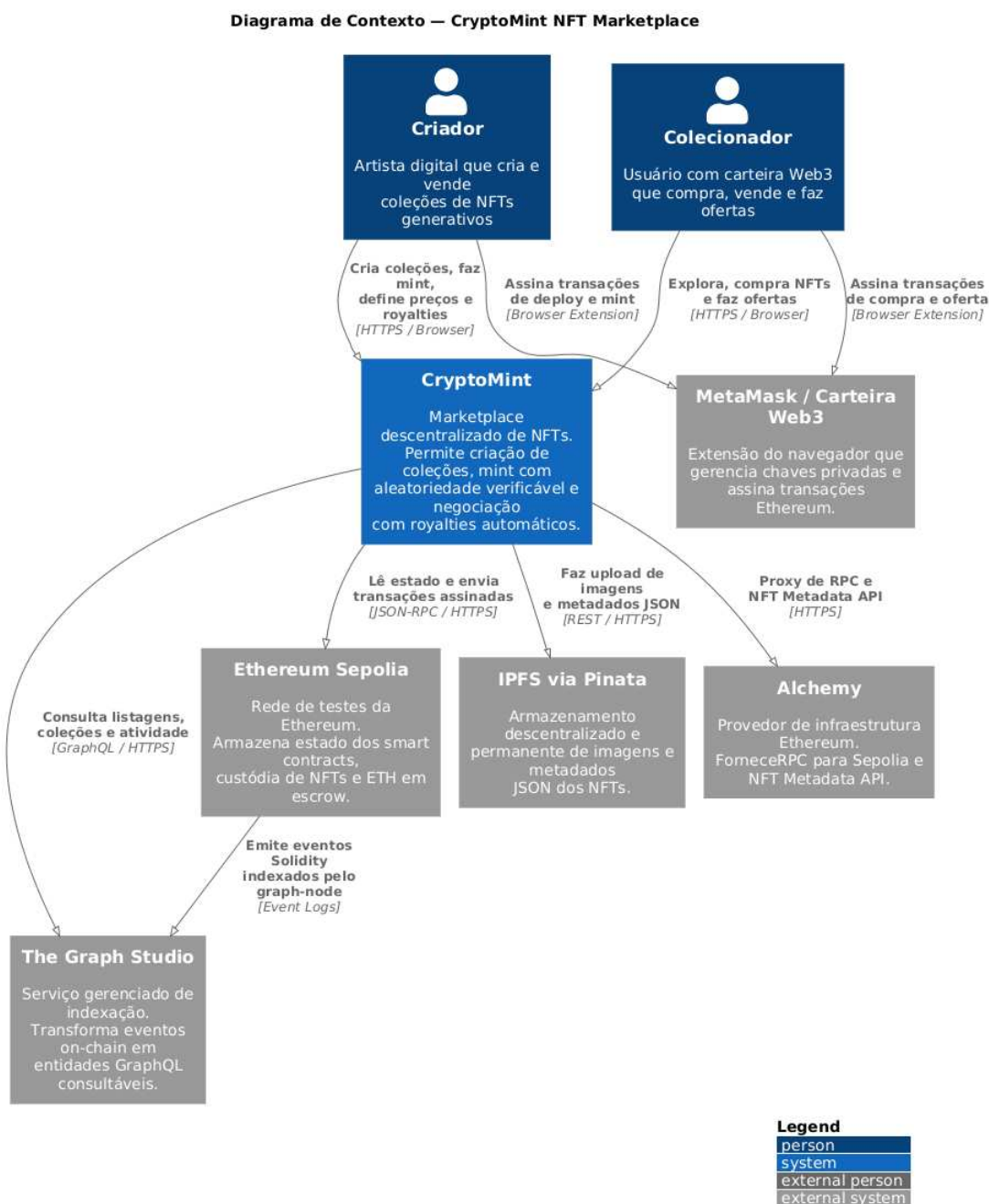
O Diagrama de Contexto (Figura 1) situa o CryptoMint em seu ecossistema: dois perfis de usuário — o *Criador* e o *Colecionador* — interagem com o sistema por meio de um navegador *web*, enquanto o sistema, por sua vez, comunica-se com quatro sistemas externos: a rede Ethereum Sepolia (acessada via Alchemy RPC), o serviço de armazenamento descentralizado IPFS por meio da plataforma Pinata, o serviço gerenciado de indexação The Graph Studio e a carteira digital MetaMask, utilizada para assinatura de transações. Essa visão de alto nível evidencia a natureza descentralizada do sistema, que não possui um *backend* centralizado próprio.

O Diagrama de Contêineres (Figura 2) aprofunda a visão para o interior do sistema, revelando os três contêineres que o compõem: o **Frontend** (aplicação Next.js 16 executada no navegador e no servidor), os **Contratos Inteligentes** (implantados de forma permanente na Ethereum Sepolia) e o **Subgraph** (implantado no Graph Studio). O diagrama torna explícitos os protocolos de comunicação entre contêineres: o Frontend consulta o Subgraph por meio do protocolo GraphQL via HTTP, lê estado autoritativo e submete transações aos contratos via JSON-RPC por meio de um *proxy* servidor (`/api/rpc`), e envia mídias ao IPFS via chamada REST à API do Pinata. Os contratos inteligentes, por sua vez, emitem eventos que o *graph-node* do The Graph captura e persiste como entidades GraphQL, fechando o ciclo de dados do sistema.

O Diagrama de Componentes (Figura 3) decompõe os três contêineres em seus elementos internos. No Frontend, destacam-se o *App Router* com as páginas React, o Apollo Client responsável pelas consultas GraphQL, o par `wagmi 3 + viem 2` que gerencia a comunicação com a EVM, os *hooks* customizados que encapsulam as mutações e os três *proxies* servidor que protegem as chaves de API (`/api/rpc`, `/api/alchemy/*` e `/api/upload*`). Na camada *blockchain*, os três contratos — *NFTMarketplace*, *NFTCollectionFactoryV2* e as instâncias dinâmicas de *NFTCollectionV2* — comunicam-se com seus respectivos *handlers* *AssemblyScript* do Subgraph por meio de eventos Solidity. O arquivo `helpers.ts` centraliza a lógica compartilhada de agregação, e a *GraphQL API* expõe ao Frontend as treze entidades persistidas.

O fluxo principal de interação do usuário segue o seguinte percurso: o usuário

Figura 1 – Diagrama de Contexto C4 — CryptoMint NFT Marketplace

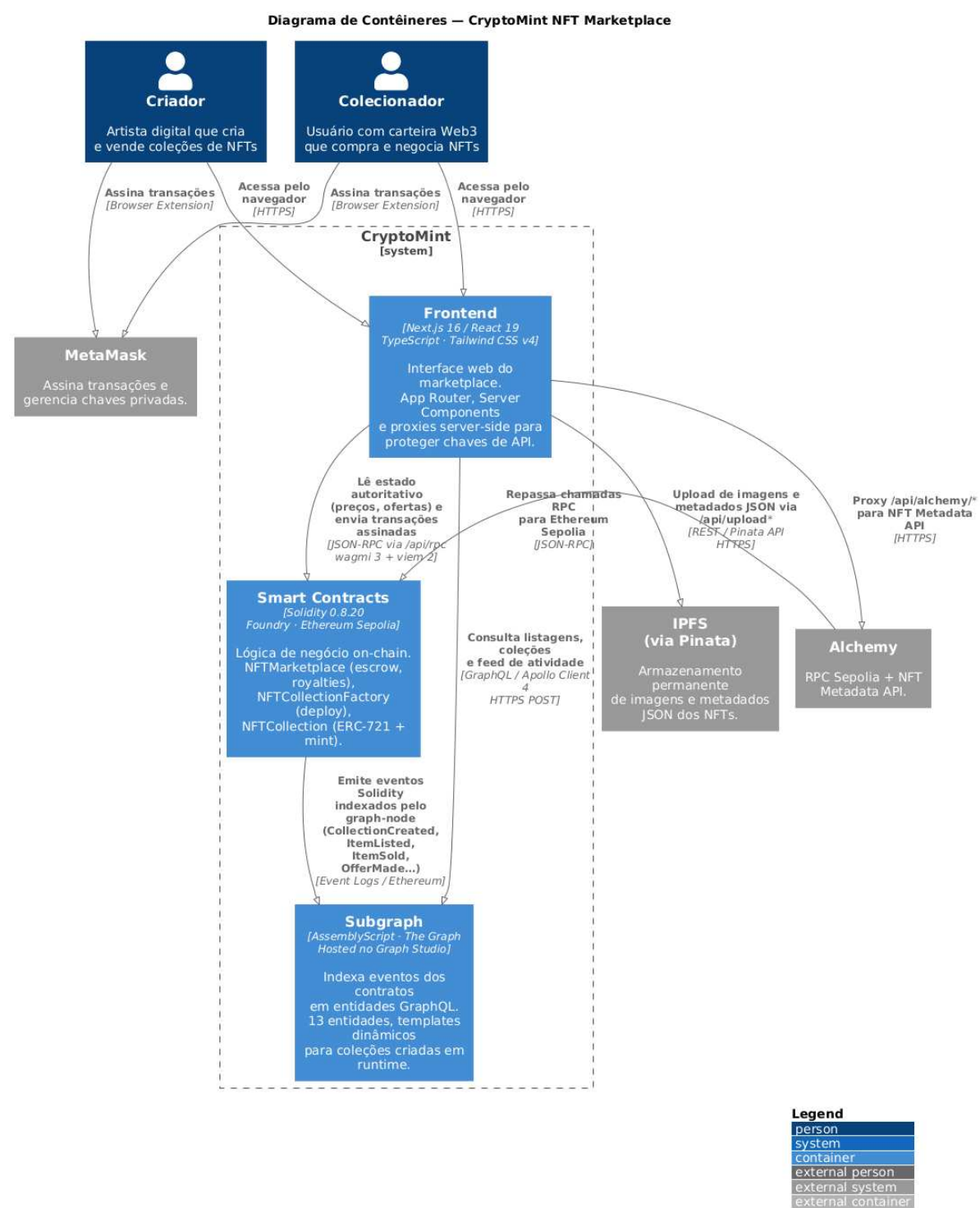


Fonte: elaborado pelo autor (2026)

acessa a interface *web* e conecta sua carteira MetaMask (ou compatível) via Connect-Kit; operações de leitura, como explorar coleções e visualizar listagens, são satisfeitas por consultas GraphQL ao *subgraph*; operações de escrita, como criar uma coleção, mintar um NFT ou comprar um *token* listado, são traduzidas em transações Ethereum

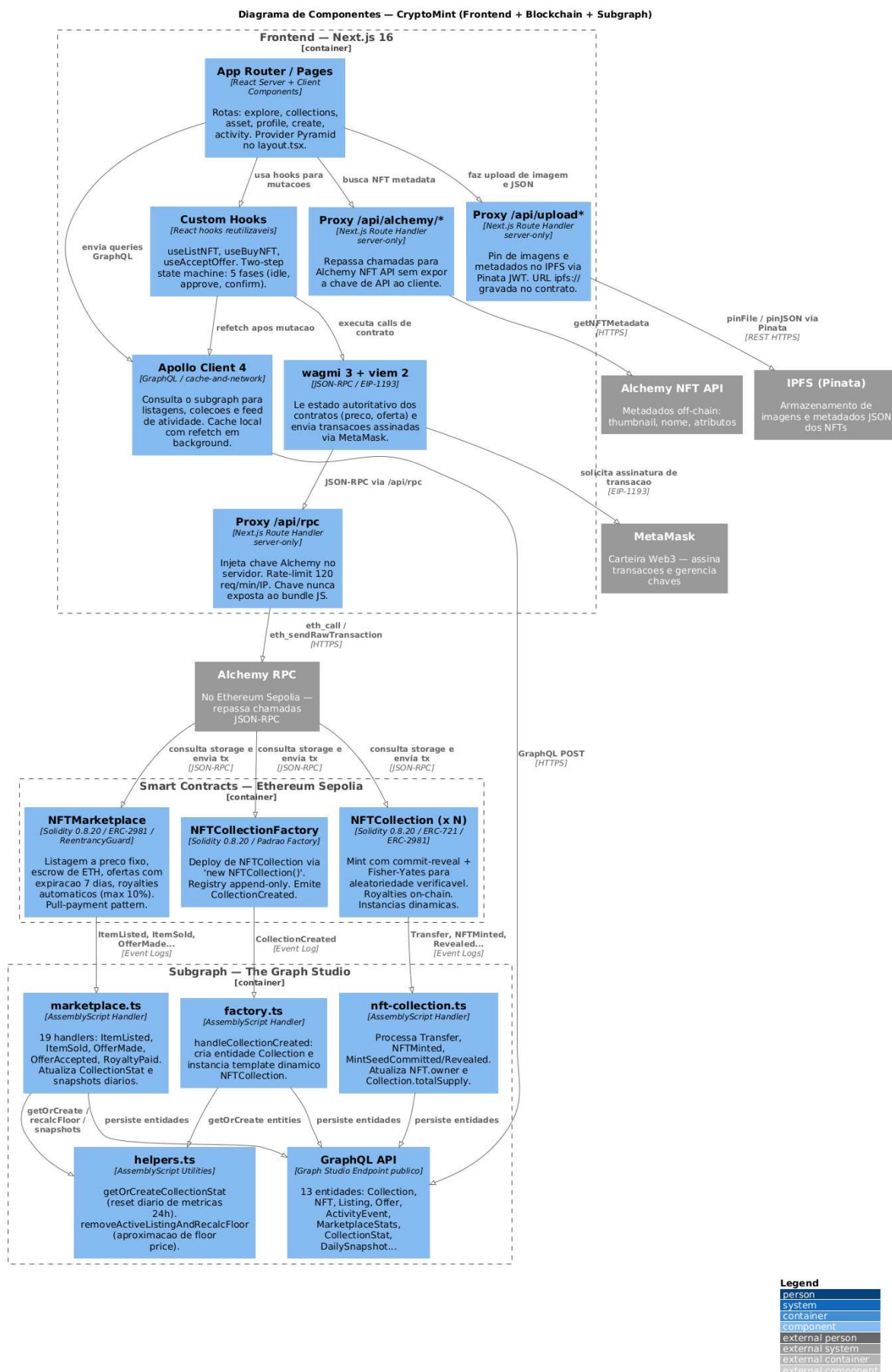
assinadas pela carteira do usuário e submetidas à rede via *proxy* RPC no servidor do Next.js; após a confirmação, os eventos emitidos pelos contratos são capturados pelo *graph-node* e atualizados no banco de dados GraphQL, de forma que a interface reflita o novo estado na próxima consulta.

Figura 2 – Diagrama de Contêineres C4 — CryptoMint NFT Marketplace



Fonte: elaborado pelo autor (2026)

Figura 3 – Diagrama de Componentes C4 — CryptoMint NFT Marketplace



4.2 CONTRATOS INTELIGENTES

Os contratos inteligentes do CryptoMint foram escritos em Solidity versão 0.8.20 e desenvolvidos com a *toolchain* Foundry. A suíte de contratos é composta por cinco contratos organizados em dois grupos funcionais: os contratos de **coleção**, responsáveis pela criação e *mint* de NFTs, e o **marketplace**, responsável pela comercialização. A Tabela 3 apresenta os contratos implantados e seus endereços na Ethereum Sepolia.

Tabela 3 – Contratos implantados na rede de testes Ethereum Sepolia

Contrato	Endereço (Sepolia)	Propósito
NFTMarketplace	0x32286F...1C74	Listagem, compra, ofertas, royalties, taxas
NFTCollectionFactoryV2	0xf17F...6c3	<i>Factory</i> para criar coleções NFT
NFTCollectionV2	(endereços dinâmicos)	Coleção ERC-721 individual por criador

Fonte: elaborado pelo autor (2026)

4.2.1 NFTCollectionFactory e NFTCollectionV2

O NFTCollectionFactoryV2 implementa o padrão de projeto **Factory**, no qual um contrato central é responsável por instanciar outros contratos e manter um registro (*registry*) das instâncias criadas. Ao invocar a função `createCollection()`, o usuário fornece os parâmetros da nova coleção — nome, símbolo, preço de *mint*, oferta máxima (*max supply*), endereço do *marketplace* autorizado e URI do contrato — e o *factory* instancia um novo contrato NFTCollectionV2 em um endereço único, emite o evento `CollectionCreated` com todos os parâmetros relevantes e registra o endereço do novo contrato em um *array* de coleções. Esse padrão é fundamental para a escalabilidade horizontal do sistema: cada criador pode lançar quantas coleções desejar sem que seja necessário implantar manualmente novos contratos ou alterar o código do *factory*.

O NFTCollectionV2 implementa os padrões ERC-721 e ERC-2981 da Open-

Zeppelin v5, adicionando lógica específica para o processo de *mint* com aleatoriedade verificável. A criação de uma coleção segue um fluxo de quatro etapas: (1) o criador invoca `loadTokenURIs()` para carregar no contrato os URIs de metadados IPFS de cada *token* da coleção; (2) o criador invoca `commitMintSeed(bytes32 seedHash)` para publicar um *commitment* criptográfico — o *hash* Keccak-256 de um segredo escolhido —, registrando-se na *blockchain* a comprometer-se com esse valor sem revelá-lo; (3) os usuários realizam o *mint* de *tokens* via `mint(address to)`, pagando o preço definido — e, a cada *mint*, uma URI ainda não atribuída é selecionada pseudoaleatoriamente do conjunto restante por meio de um passo do algoritmo **Fisher-Yates shuffle** (Seção 2.6.2), conforme detalhado adiante; (4) após o encerramento do período de *mint*, o criador invoca `revealMintSeed(bytes32 secret)` para publicar o segredo previamente comprometido, permitindo que qualquer participante audite e verifique que a atribuição de raridades respeitou o compromisso original.

O mecanismo de *commit-reveal* foi adotado para garantir a **imparcialidade verificável** no processo de atribuição de raridades. O criador não pode selecionar quais raridades atribuir a quais compradores, pois a semente só pode ser revelada após o *mint*, e qualquer revelação diferente do compromisso previamente publicado é rejeitada pelo contrato. Adicionalmente, o *blockhash* de um bloco passado — que o criador não poderia prever no momento do *commit* — é incorporado à aleatoriedade, impedindo que mesmo um *block proposer* malicioso manipule o resultado. Essa abordagem é superior ao uso isolado de `block.prevrandao` ou `block.timestamp` como fontes de entropia, que podem ser observados ou influenciados por validadores na Ethereum PoS (WOOD, 2014).

A atribuição propriamente dita dos metadados ocorre de forma **incremental**, a cada operação de *mint*, por meio de um único passo do algoritmo de Fisher-Yates (Seção 2.6.2) sobre o vetor de URIs ainda disponíveis. Em vez de embaralhar todo o vetor de uma única vez, o contrato realiza uma **seleção sem reposição**: calcula um índice pseudoaleatório a partir do compromisso da semente, do *hash* do bloco anterior e de dados do *token*; seleciona a URI naquela posição; move o último elemento do

vetor para a posição liberada; e reduz o vetor em uma unidade (*swap-and-pop*). O Código 4.1 apresenta esse passo em pseudocódigo.

Listing 4.1 – Passo incremental de Fisher-Yates na seleção de URI durante o mint

```

1 // Executado a cada mint -- selecao de uma URI sem reposicao
2 remaining = availableURIs.length;
3 bhash      = blockhash(block.number - 1);    // entropia
           imprevisivel no commit
4 index      = uint256(
5     keccak256(abi.encodePacked(commitment, bhash, to, tokenId))
6 ) % remaining;                               // indice
           pseudoaleatorio uniforme
7
8 chosenUri   = availableURIs[index];           // URI
           sorteada
9 availableURIs[index] = availableURIs[remaining - 1]; // troca
           com o ultimo
10 availableURIs.pop();                         // remove o
           ultimo

```

Essa estratégia oferece quatro garantias relevantes. Primeiro, a **unicidade**: como cada URI selecionada é removida do conjunto disponível, nenhum metadado é atribuído a mais de um *token*. Segundo, a **uniformidade** estatística herdada do Fisher-Yates, que assegura que todas as permutações sejam equiprováveis e impede a concentração tendenciosa de raridades. Terceiro, a **eficiência**: cada *mint* executa um único passo de custo $O(1)$, diluindo o custo total de $O(n)$ ao longo das cunhagens em vez de concentrá-lo em uma única transação custosa de embaralhamento. Por fim, a **verificabilidade**: como o índice deriva do compromisso publicado antes do *mint* e do *hash* de um bloco que o criador não poderia prever, a seleção não pode ser manipulada; a posterior revelação do segredo via `revealMintSeed` permite que qualquer participante recompute as atribuições e audite a integridade de todo o processo.

O padrão **ERC-2981** de royalties é configurado no construtor do `NFTCollecti-`

onV2 com uma taxa padrão de 5% sobre o valor de cada revenda, pagável ao endereço do criador da coleção. O padrão foi escolhido por ser o único mecanismo padronizado de royalties suportado amplamente por *marketplaces* de terceiros, como OpenSea e Blur (BURKS et al., 2020). A interface `royaltyInfo(uint256 tokenId, uint256 salePrice)` é consultada pelo NFTMarketplace em cada transação de venda, conforme descrito na seção seguinte.

4.2.2 NFTMarketplace

O NFTMarketplace é o contrato central do sistema, responsável pela mediação de todas as operações comerciais. Ele implementa um modelo de *marketplace não custodiante* (*non-custodial*): os NFTs nunca são transferidos para o contrato do *marketplace* durante uma listagem; em vez disso, o vendedor concede ao *marketplace* uma aprovação via `setApprovalForAll`, que autoriza o contrato a executar a transferência apenas no momento em que uma venda ou oferta for aceita. Esse modelo elimina um vetor de risco significativo presente em *marketplaces* custodiais de primeira geração: a perda de acesso aos ativos em caso de falha ou comprometimento do contrato do *marketplace*.

O estado de uma listagem é encapsulado na *struct* `Listing`, cuidadosamente empacotada em dois *slots* de *storage* de 256 bits para minimizar o consumo de *gas* em operações de leitura e escrita. O campo `seller` (160 bits) e o campo booleano `active` (1 bit) ocupam o primeiro *slot*, enquanto o campo `price` do tipo `uint128` ocupa o segundo *slot*, permitindo preços de até aproximadamente 340 undecilhões de *wei* sem desperdício de espaço de armazenamento:

Listing 4.2 – Struct Listing com empacotamento de slots

```
1 struct Listing {
2     address seller;    // slot 0 [bits 0..159]
3     bool active;       // slot 0 [bit 160]
4     uint128 price;     // slot 1 [bits 0..127]
5 }
```

O fluxo de compra de um NFT listado, executado pela função `buyItem(address nftContract, uint256 tokenId)`, ilustra a aplicação rigorosa do padrão **Checks-Effects-Interactions** (CEI): primeiramente, todas as pré-condições são verificadas (listagem ativa, valor enviado igual ao preço, comprador diferente do vendedor); em seguida, os efeitos de estado são aplicados (a listagem é deletada e as taxas são contabilizadas) antes de qualquer interação com contratos externos; finalmente, as interações com contratos externos são realizadas na seguinte ordem: pagamento ao vendedor, pagamento de royalties, e transferência do NFT via `safeTransferFrom`. A inversão dessa ordem exporia o contrato a ataques de reentrada (*reentrancy attacks*), no qual um contrato malicioso poderia reentrar na função `buyItem` durante o recebimento de ETH e adquirir o NFT sem pagar.

O sistema de ofertas permite que compradores depositem ETH em *escrow* no contrato para manifestar interesse em NFTs que não estejam listados ou para fazer ofertas acima do preço de listagem. Cada oferta possui validade de sete dias (`OFFER_DURATION = 7 days`), após a qual o ETH pode ser recuperado pelo ofertante ou por qualquer terceiro que invoque `reclaimExpiredOffer()`. Esse terceiro recebe uma recompensa de 0,5% (*bounty*) do valor da oferta expirada como incentivo para manter o contrato limpo de ofertas obsoletas. O mecanismo de pagamento de royalties e taxas para o vendedor e receptor de royalty utiliza um **padrão pull-payment**: ao invés de transferir ETH diretamente — o que exporia o contrato a ataques de DoS por destinatários maliciosos que rejeitam ETH —, os valores são creditados em um mapeamento `pendingWithdrawals`, e cada receptor pode invocar `withdrawPending()` para sacar seu saldo a qualquer momento.

4.2.3 Padrões de Segurança Aplicados

A segurança dos contratos inteligentes é uma preocupação fundamental, uma vez que falhas de segurança em contratos implantados são irreversíveis e frequentemente exploradas com consequências financeiras imediatas ([ATZEI; BARTOLETTI; CIMOLI, 2017](#)). O CryptoMint incorpora múltiplas camadas de defesa, cujos principais

padrões são descritos a seguir.

O modificador `nonReentrant` da OpenZeppelin é aplicado a todas as funções que movimentam ETH ou NFTs, prevenindo ataques de reentrada por meio de uma trava de reentrância no *storage* do contrato. O padrão CEI (*Checks-Effects-Interactions*) é aplicado rigorosamente em todas as funções `payable`, garantindo que o estado do contrato esteja sempre consistente antes de qualquer chamada a contratos externos. Os erros de contrato são implementados como **erros customizados** (*custom errors*, introduzidos no Solidity 0.8.4), que consomem significativamente menos *gas* que as *strings* de `require()` equivalentes — uma economia diretamente repassada ao usuário em cada transação revertida.

A consulta ao `royaltyInfo` do ERC-2981 é realizada via `staticcall` com um limite explícito de 30.000 *gas* e verificação de que o retorno seja de exatamente 64 bytes, prevenindo um vetor de ataque conhecido: um contrato malicioso que declare suporte ao ERC-2981 mas implemente uma função `royaltyInfo` que consuma todo o *gas* disponível da transação compradora, tornando as compras impossíveis. O teto de 10% para royalties (`MAX_ROYALTY_BPS = 1000`) protege compradores de coleções que retornem valores de royalty abusivos ou fabricados.

O Slither, ferramenta de análise estática de código Solidity desenvolvida pela Trail of Bits e formalizada academicamente por [Feist, Grieco e Groce \(2019\)](#) ([Trail of Bits, 2018](#)), foi executado sobre todo o código dos contratos para identificar vulnerabilidades potenciais. As anotações `// slither-disable-next-line reentrancy-benign` presentes em pontos específicos do código são falsos positivos documentados: o estado relevante (listagem ou oferta) já foi deletado antes da chamada externa marcada, de forma que uma eventual reentrada encontraria o estado já limpo e reverteria sem efeitos.

4.3 INDEXAÇÃO COM THE GRAPH (SUBGRAPH)

4.3.1 Motivação e Filosofia de Design Subgraph-First

O *design* do CryptoMint adota uma filosofia denominada **subgraph-first**: toda informação que o *frontend* precisa para suas operações de leitura não críticas é derivada de eventos *on-chain* e disponibilizada pelo *subgraph* como entidades GraphQL, em vez de ser consultada individualmente via chamadas RPC ao estado dos contratos. A motivação para essa escolha é objetiva: uma consulta típica na página de exploração do *marketplace* — “todos os NFTs listados de uma determinada coleção, com preços, histórico de ofertas e informações do criador” — exigiria dezenas de chamadas `eth_call` independentes ao RPC da Sepolia caso o estado fosse consultado diretamente nos contratos. Além de lenta, essa abordagem sobrecarregaria o provedor RPC e resultaria em uma interface com visível latência de carregamento para o usuário final.

O *subgraph* resolve esse problema ao transformar os eventos emitidos pelos contratos em um banco de dados relacional GraphQL mantido pelo *graph-node*. Cada evento *on-chain* — como `ItemListed`, `ItemSold`, `OfferMade` ou `NFTMinted` — dispara um *handler* `AssemblyScript` que cria ou atualiza entidades no banco de dados. O resultado é um estado derivado, desnormalizado e otimizado para leitura, acessível via uma única *query* GraphQL com *joins* implícitos, filtros, ordenação e paginação — funcionalidades que o protocolo RPC nativo simplesmente não oferece.

4.3.2 Estrutura do Subgraph

O *subgraph* do CryptoMint é definido por três componentes. O **manifest** (`<subgraph.yaml>`) declara as fontes de dados: dois contratos estáticos — `NFT-Marketplace` (implantado no bloco 10.647.296) e `NFTCollectionFactoryV2` (bloco 10.647.287) — e um *template* dinâmico denominado `NFTCollection`, que não possui endereço fixo no momento da implantação do *subgraph*. O *template* dinâmico é o mecanismo que permite ao sistema escalar para N coleções sem alteração do

código: cada vez que o *handler* do evento `CollectionCreated` é executado, ele invoca `<NFTCollection.create(event.params.contractAddress)>`, instruindo o *graph-node* a iniciar a indexação de eventos do novo contrato de coleção a partir daquele bloco.

O **schema** GraphQL (`schema.graphql`) define treze entidades organizadas em dois grupos. As entidades mutáveis representam o estado atual do sistema e são atualizadas a cada evento relevante: `Collection`, `NFT`, `Listing`, `Offer`, `ActivityEvent`, `MarketplaceStats`, `CollectionStat`, `DailyCollectionSnapshot` e `PendingBalance`. As entidades imutáveis, anotadas com `@entity(immutable: true)`, formam um *log* de auditoria permanente de eventos significativos: `FeeUpdate`, `AdminWithdrawal`, `RoyaltyPayment`, `PendingWithdrawalEvent` e `CollectionWithdrawal`. A distinção entre os dois grupos é relevante tanto para a semântica dos dados quanto para a *performance* do *graph-node*, que pode otimizar o armazenamento e a indexação de entidades declaradas como imutáveis.

Um desafio de modelagem específico do The Graph motivou a adoção de um **padrão de dual-ID** para as ofertas. Como o protocolo não oferece deleção de entidades, representar simultaneamente o estado atual de uma oferta e o seu histórico completo exige uma estratégia explícita. A solução adotada mantém, para cada par (NFT, comprador), uma entidade `Offer` canônica de identificador estável — composto pelos identificadores do *token* e do comprador —, que funciona como ponteiro mutável para a oferta corrente; e, a cada nova oferta, cria uma segunda entidade `Offer` de identificador único por transação, que constitui um registro imutável daquele evento. O campo `latestOfferId` liga o ponteiro canônico ao registro histórico mais recente. Dessa forma, as consultas ao estado atual resolvem-se pelo identificador canônico, enquanto o histórico íntegro de todas as ofertas de um comprador sobre um *token* permanece reconstituível, contornando a ausência de deleção sem sacrificar a auditabilidade.

Os **handlers** `AssemblyScript`, localizados em `<src/factory.ts>`, `<src/marketplace.ts>` e `<src/collection.ts>`, implementam a lógica de transformação de eventos em entidades. Cada *handler* recebe o evento tipado — gerado automa-

ticamente pelo compilador do *subgraph* a partir das ABIs dos contratos — e realiza operações de criação, leitura e atualização de entidades no banco de dados. O Código 4.3 apresenta um trecho representativo do *handler* do evento `ItemSold`:

Listing 4.3 – Handler `handleItemSold` (trecho simplificado — `AssemblyScript`)

```
1 // src/marketplace.ts -- handler handleItemSold
2 export function handleItemSold(event: ItemSoldEvent): void {
3     let listingId = event.params.nftContract.toHexString()
4         + "-" + event.params.tokenId.toString();
5     let listing = Listing.load(listingId);
6     if (listing) {
7         listing.active = false;
8         listing.soldAt = event.block.timestamp;
9         listing.save();
10    }
11    // Atualiza estatísticas globais e da coleção...
12 }
```

4.3.3 Circuit-Breaker e Degradação Graciosa

O *frontend* implementa um `SubgraphProxy` — um *Route Handler* do `Next.js` que intermedia todas as *queries* GraphQL do `Apollo Client` ao `Graph Studio` — com um mecanismo de ***circuit-breaker*** para o código de status HTTP 429 (*Too Many Requests*). Quando o `Graph Studio` retorna um erro 429 indicando que o limite de requisições por minuto foi atingido, o *proxy* abre o *circuit* e passa a servir dados em *cache* da resposta mais recente bem-sucedida, enquanto registra o estado de degradação em memória. O *frontend* detecta esse estado por meio de um campo `_rateLimited` nas respostas do *proxy* e exibe um *banner* informativo ao usuário, mantendo a navegação funcional com dados eventualmente consistentes. Após um período de *cooldown* configurável, o *circuit* se fecha automaticamente e as *queries* voltam a ser encaminhadas ao `Graph Studio`. Esse padrão de resiliência, inspirado no *Circuit Breaker Pattern* descrito por [Nygard \(2007\)](#), é especialmente relevante para o ambiente acadêmico, no qual os limites

de uso do serviço gerenciado são mais restritivos do que em ambientes de produção pagos.

O estado de degradação precisa ser observável por componentes React dispersos pela interface — como o *banner* informativo — sem ser propagado por meio de uma cadeia de propriedades ou de múltiplos *Context Providers*. Para isso, o *frontend* mantém esse estado em um *store* externo a React e o expõe aos componentes por meio do *hook* `useSyncExternalStore`, introduzido no React 18 justamente para a integração segura com fontes de estado externas ao ciclo de renderização. O mesmo mecanismo é empregado para outros estados globais de alta frequência de atualização, como o relógio sincronizado que alimenta as contagens regressivas de expiração das ofertas. Essa abordagem evita a proliferação de *Context Providers* e as re-renderizações desnecessárias que decorreriam de propagar tais valores pela árvore de componentes, mantendo localizadas as atualizações apenas nos componentes efetivamente inscritos no *store*.

4.4 FRONTEND

4.4.1 Arquitetura Next.js e Proxies Server-Side

A camada de apresentação do CryptoMint é implementada como uma aplicação Next.js 16 com o paradigma **App Router**, no qual a estrutura de roteamento é definida diretamente pela hierarquia de diretórios dentro da pasta `app/`. Cada rota de página é representada por um arquivo `page.tsx`, e comportamentos compartilhados entre rotas — como *layout*, carregamento e tratamento de erros — são expressos por arquivos convencionais como `layout.tsx`, `loading.tsx` e `error.tsx`. O App Router do Next.js 16 adota por padrão os **React Server Components** (RSC), que são renderizados exclusivamente no servidor e não incluem código JavaScript no *bundle* do cliente, contribuindo para tempos de carregamento iniciais reduzidos e para a separação entre lógica de servidor e lógica de cliente.

Um aspecto arquitetural central do *frontend* é a criação de **Route Handlers** —

endpoints de API executados no servidor Next.js — para intermediar chamadas a serviços externos que requerem autenticação por chave de API. O *proxy* `/api/rpc` recebe todas as requisições JSON-RPC do cliente, injeta a chave de API do Alchemy no servidor e encaminha a requisição ao *endpoint* RPC da Sepolia, garantindo que a chave nunca seja exposta no *bundle* JavaScript entregue ao navegador. Analogamente, os *endpoints* `/api/upload`, `/api/upload-auth` e `/api/upload-metadata` intermediam as operações de *upload* ao Pinata IPFS. Adicionalmente, o *middleware* de taxa de requisições (*rate limiting*) — implementado como uma janela deslizante em memória por IP e por endereço de carteira — é aplicado a esses *endpoints* para prevenir uso abusivo da infraestrutura de terceiros contratada.

4.4.2 Integração Web3: wagmi, viem e ConnectKit

A integração entre o *frontend* e a *blockchain* da Ethereum Sepolia é realizada por meio de três bibliotecas complementares. O **wagmi** é uma biblioteca de *hooks* React para interação com a Ethereum, que abstrai a complexidade de gerenciamento de conexões de carteira, assinatura de transações e leitura de contratos em uma API baseada em *hooks* do React (Wevm, 2024). O **viem** é uma biblioteca TypeScript de baixo nível para interação com a EVM, tipada com base nas ABIs dos contratos e utilizada pelo wagmi como transporte de dados; sua principal contribuição é a geração automática de tipos TypeScript a partir das ABIs, garantindo segurança de tipos de ponta a ponta nas chamadas aos contratos. O **ConnectKit** é um componente de interface de usuário que gerencia o fluxo de conexão de carteiras Web3 — como MetaMask, WalletConnect e Coinbase Wallet — fornecendo um modal de conexão pronto para uso com suporte a múltiplos provedores.

A configuração do cliente wagmi restringe o sistema exclusivamente à rede Sepolia (`chains: [sepolia]`) e roteia todo o tráfego RPC pelo *proxy server-side* (`transport: http("/api/rpc")`). Um componente `useWrongNetwork` detecta se a carteira conectada está em uma rede diferente de Sepolia e exibe um alerta interativo na barra de navegação com a opção de troca automática de rede via `useSwitchChain`.

O *hook* customizado `useTwoStepContractMutation` encapsula o padrão de interação de dois passos necessário para operações de venda e listagem no *marketplace*: antes de listar ou transferir um NFT, o *marketplace* precisa receber a aprovação `setApprovalForAll` do usuário. Esse *hook* implementa uma **máquina de estados** com cinco fases — `idle` → `approve-wallet` → `approve-confirm` → `exec-wallet` → `exec-confirm` —, cada qual com estados de UX distintos (textos de botão, indicadores de carregamento), garantindo que o usuário compreenda em qual etapa do processo de dois passos se encontra e que nenhuma etapa seja iniciada sem a conclusão bem-sucedida da anterior. Um `inFlightRef` atômico previne submissões duplicadas por duplo-clique.

4.4.3 Apollo Client e Padrão Subgraph-First

As consultas GraphQL ao *subgraph* são gerenciadas pelo **Apollo Client 4**, uma biblioteca de gerenciamento de estado para GraphQL em aplicações React. O Apollo Client mantém um *cache* normalizado em memória que armazena os resultados das *queries* indexados por tipo e identificador de entidade, permitindo atualizações otimizadas e renderização imediata a partir do *cache* enquanto revalida os dados em *background*. A política `cache-and-network` é adotada para todas as *queries* de listagem: o cliente renderiza imediatamente os dados do *cache* local (se disponíveis) e simultaneamente envia uma nova requisição ao *subgraph*, atualizando a interface quando os dados frescos chegam.

A centralização de todas as *queries* GraphQL em um único arquivo (`<src/lib/graphql/queries.ts>`), com tipos gerados automaticamente pelo GraphQL Code Generator, garante consistência entre as entidades do *schema* do *subgraph* e os tipos TypeScript consumidos pelos *hooks* do *frontend*. Qualquer alteração no *schema* do *subgraph* que quebre uma *query* existente é detectada em tempo de compilação, antes de atingir o ambiente de produção.

4.4.4 Armazenamento Descentralizado via Pinata IPFS

O *upload* de imagens e metadados ao IPFS é realizado em duas etapas para garantir segurança e integridade. Na primeira etapa, o *endpoint* `/api/upload-auth` gera um *token* de autenticação assinado com validade de 5 minutos para que o cliente possa realizar o *upload* diretamente ao Pinata sem expor a chave de API permanente. Na segunda etapa, o cliente realiza o *upload* da imagem ao Pinata e, em seguida, invoca `/api/upload-metadata` para fixar (*pin*) o JSON de metadados — que inclui o CID da imagem, o nome do *token*, a descrição e os atributos de raridade —, recebendo o CID definitivo que será gravado no contrato no momento do *mint*. Esse fluxo garante que tanto a imagem quanto os metadados estejam permanentemente fixados no IPFS antes que qualquer transação seja submetida à *blockchain*, eliminando o risco de *mint* de *tokens* com URIs de metadados não resolvíveis.

5 RESULTADOS E AVALIAÇÃO

5.1 FUNCIONALIDADES IMPLEMENTADAS

O sistema CryptoMint foi desenvolvido com o objetivo de prover uma plataforma completa para criação e comercialização de NFTs, e o escopo de funcionalidades implementadas cobre os principais fluxos de um *marketplace* descentralizado. A Tabela 4 apresenta as funcionalidades implementadas organizadas por módulo do sistema.

O CryptoMint opera integralmente na rede de testes Ethereum Sepolia, com os contratos implantados nos endereços permanentes descritos na Tabela 3 (Seção 4.2). O sistema não suporta leilões *on-chain*, listagens de *tokens* ERC-1155 ou ofertas denominadas em *tokens* ERC-20 — limitações reconhecidas e documentadas, decorrentes da escolha de escopo que priorizou a qualidade técnica e a cobertura de testes sobre a amplitude de funcionalidades.

Tabela 4 – Funcionalidades implementadas no CryptoMint

Módulo	Funcionalidade	Status
Coleções	Criação de coleção ERC-721 com metadados IPFS	✓
Coleções	<i>Upload</i> de URIs de <i>tokens</i> ao IPFS via Pinata	✓
Coleções	<i>Mint</i> de NFTs com aleatoriedade verificável (<i>commit-reveal</i> + Fisher-Yates)	✓
Coleções	Schema de atributos por coleção (<i>contractURI</i> e <i>trait_schema</i>)	✓
Coleções	Saque de receitas de <i>mint</i> pelo criador	✓
Marketplace	Listagem de NFT a preço fixo	✓
Marketplace	Atualização de preço de listagem	✓
Marketplace	Compra de NFT listado	✓
Marketplace	Cancelamento de listagem	✓
Marketplace	Submissão de oferta com ETH em <i>escrow</i>	✓
Marketplace	Aceitação de oferta pelo proprietário do NFT	✓
Marketplace	Cancelamento de oferta e devolução de ETH	✓
Marketplace	Reclamação de oferta expirada com <i>bounty</i>	✓
Marketplace	Pagamento automático de royalties ERC-2981 (<i>cap</i> de 10%)	✓
Marketplace	Cobrança e saque de taxas de plataforma (2,5% padrão)	✓
Frontend	Interface de exploração do <i>marketplace</i> com filtros	✓
Frontend	Detalhe de NFT com painel de compra/oferta/listagem	✓
Frontend	Página de perfil (NFTs coletados, criados, favoritos, atividade)	✓
Frontend	<i>Feed</i> de atividade <i>on-chain</i>	✓
Frontend	<i>Wizard</i> multi-step de criação de coleção	✓
Frontend	Integração com carteiras Web3 (MetaMask, WalletConnect)	✓
Subgraph	Indexação de 13 entidades de estado e auditoria	✓
Subgraph	<i>Circuit-breaker</i> com degradação graciosa em caso de <i>rate-limit</i>	✓

Fonte: elaborado pelo autor (2026)

5.2 TESTES E VALIDAÇÃO

5.2.1 Testes de Contratos Inteligentes (Foundry/Forge)

A suíte de testes dos contratos inteligentes foi implementada com o *framework* Foundry/Forge (Paradigm, 2024), utilizando a linguagem Solidity nativa para a escrita dos casos de teste. A escolha do Foundry como *framework* de testes foi motivada pela sua capacidade de manipular diretamente o estado da EVM por meio de *cheatcodes* — funções especiais do contrato `vm` do Forge Standard Library que permitem, por exemplo, simular passagem de tempo (`vm.warp`), simular a identidade do remetente (`vm.prank`), manipular saldos (`vm.deal`) e verificar emissão de eventos (`vm.expectEmit`) de forma precisa e declarativa.

A suíte de testes abrange os seguintes contratos e cenários:

NFTMarketplace (`test/NFTMarketplace.t.sol` — 1.237 linhas): O arquivo de testes do *marketplace* inclui mais de 40 casos de teste cobrindo os fluxos felizes e os caminhos de erro de todas as funções externas do contrato. Os testes de segurança incluem dois contratos atacantes (`ReentrancyAttacker` e `ReentrancyCancelAttacker`) que tentam explorar ataques de reentrada durante o recebimento de ETH — ambos são verificados como revertendo com `ReentrancyGuardReentrantCall` e `OfferNotActive`, respectivamente, confirmando a eficácia das proteções implementadas. Os testes de *fuzz* (geração pseudoaleatória de *inputs*) são aplicados às funções de cálculo de taxas e royalties, verificando invariantes matemáticos como a conservação do valor total da venda entre *marketplace*, vendedor e receptor de royalty.

NFTCollectionV2 (`test/NFTCollectionV2.t.sol`): A suíte de testes da coleção V2 inclui verificações específicas para o mecanismo de *commit-reveal* e o algoritmo Fisher-Yates, confirmando que todos os *tokens* de uma coleção recebem URIs únicas após a revelação da semente e que a função `contractURI()` é imutável após o *deploy*.

O Quadro 3 resume os principais casos de teste por categoria funcional:

Quadro 3 – Categorias de testes de contratos inteligentes

Categoria	Casos de Teste	Verificações-Chave
Listagem	<code>test_listItem_success</code> , <code>test_listItem_revertsIfNotOwner</code>	Evento <code>ItemListed</code> , validação de propriedade
Compra	<code>test_buyItem_success</code> , <code>test_buyItem_revertsIfWrongPrice</code>	Transferência de NFT, distribuição de pagamentos
Ofertas	<code>test_makeOffer_success</code> , <code>test_acceptOffer_distributes_correctly</code>	<i>Escrow</i> , pagamento de royalties
Segurança	<code>test_reentrancy_attack_fails</code> , <code>test_reentrancy_cancel_fails</code>	Proteção contra reentrada
Aleatoriedade	<code>test_mint_allTokensUnique</code> , <code>test_revealSeed_verifiable</code>	Unicidade de URIs, <i>commit-reveal</i>
Royalties	<code>test_royalty_defaultFivePercent</code> , <code>test_royalty_cappedAtTenPercent</code>	Cálculo e distribuição
Fuzz	<code>testFuzz_feeCalculation_conserveValue</code>	Invariante de conservação de valor

Fonte: elaborado pelo autor (2026)

5.2.2 Testes de Frontend (Vitest)

Os testes do *frontend* foram implementados com o Vitest — *framework* de testes compatível com Vite e Jest —, com a biblioteca Testing Library para testes de componentes React em ambiente jsdom. O escopo de testes do *frontend* cobre os *hooks* customizados de maior criticidade, como o SubgraphProxy (com cenários de *circuit-breaker* e *rate-limit*), os *helpers* de tratamento de erros de transação e os *schemas* de validação Zod utilizados nos formulários de criação de coleção.

5.3 ANÁLISE DE GAS E DESEMPENHO

5.3.1 Custo de Gas por Operação

A análise de custos de *gas* foi conduzida com base em medições das operações do NFTMarketplace executadas no ambiente de testes do Foundry, com o otimizador Solidity configurado para 200 *runs* — o padrão do projeto. A Tabela 5 apresenta os custos estimados por operação principal do *marketplace*.

Tabela 5 – Custo de gas das operações do NFTMarketplace

Operação	Gas Estimado	Custo Aprox. (ETH a 30 gwei)
<code>listItem()</code>	≈50.000 gas	≈0,0015 ETH
<code>updateListingPrice()</code>	≈5.000 gas	≈0,00015 ETH
<code>cancelListing()</code>	≈10.000 gas	≈0,0003 ETH
<code>buyItem()</code>	≈80.000–120.000 gas	≈0,0024–0,0036 ETH
<code>makeOffer()</code>	≈50.000 gas	≈0,0015 ETH
<code>acceptOffer()</code>	≈120.000 gas	≈0,0036 ETH
<code>cancelOffer()</code>	≈20.000 gas	≈0,0006 ETH

Fonte: elaborado pelo autor (2026)

A otimização de *gas* mais impactante implementada no CryptoMint é o **empacotamento de *structs***: a *struct* `Listing` ocupa exatamente dois *slots* de 32 bytes, combinando o endereço do vendedor (20 bytes) e o booleano `active` (1 byte) no primeiro *slot*, e o preço `uint128` (16 bytes) no segundo, evitando um *slot* de *storage* desperdiçado que custaria 20.000 *gas* adicionais por operação de escrita a frio. A *struct* `Offer` utiliza o mesmo princípio, comprimindo quatro campos em dois *slots*.

5.3.2 Comparação com Marketplaces de Produção

A análise comparativa com o OpenSea Seaport v1.6 e o Rarible Protocol v2 — dois dos maiores protocolos de *marketplace* da Ethereum — revela diferenças arquiteturais significativas que se traduzem em custos de *gas* distintos. A comparação é apresentada em dois níveis: a Tabela 6 detalha o custo por operação em relação ao Seaport, tomado como protocolo de referência por dispor de medições públicas precisas; e a Tabela 7 sintetiza as diferenças arquiteturais entre os três protocolos, evidenciando que tanto o Seaport quanto o Rarible adotam o modelo de livro de ordens *off-chain*.

O principal fator de diferença é a estratégia de armazenamento de listagens: o CryptoMint armazena cada listagem *on-chain* como uma *struct* no *storage* do contrato (custo de ≈50.000 *gas*), enquanto o Seaport utiliza ordens assinadas criptograficamente (EIP-712) armazenadas *off-chain*, com custo de criação de listagem essencial-

Tabela 6 – Comparativo de custo de gas por operação: CryptoMint vs. Seaport v1.6

Operação	CryptoMint	Seaport v1.6	Delta
Criar listagem	≈50.000 gas	≈0 gas (assinatura <i>off-chain</i>)	+50.000
Compra simples	≈100.000 gas	≈70.000–90.000 gas	+10.000–30.000
Compra em lote (5 NFTs)	≈500.000 gas	≈200.000–250.000 gas	+250.000
Fazer oferta	≈50.000 gas	≈0 gas (WETH sem <i>escrow</i>)	+50.000

Fonte: elaborado pelo autor (2026)

mente zero — o armazenamento é delegado à infraestrutura *off-chain* da plataforma. A vantagem do modelo *on-chain* do CryptoMint é que o estado das listagens pode ser consultado diretamente via RPC sem depender de uma infraestrutura *off-chain* proprietária, o que é coerente com os objetivos acadêmicos de transparência total e verificabilidade *on-chain* do sistema.

Para situar o CryptoMint diante de um espectro mais amplo de decisões de projeto, a Tabela 7 confronta os três protocolos em dimensões arquiteturais-chave. A comparação evidencia que o custo de *gas* de listagem do CryptoMint não decorre de uma escolha isolada, mas de uma postura coerente de manter todo o estado *on-chain* e de adotar uma arquitetura monolítica e imutável.

O Rarible Protocol v2 compartilha com o Seaport o modelo de ordens assinadas *off-chain*, o que reduz a praticamente zero o custo de *gas* de criação de listagens e ofertas. Sua arquitetura, contudo, distingue-se pela modularidade — com os contratos *Exchange*, *AssetMatcher* e *RoyaltiesProvider* separados — e pela adoção de um *proxy* atualizável (UUPS) no contrato *Exchange*, que permite registrar novos tipos de ativos e corrigir o protocolo sem nova implantação. Em contrapartida, o CryptoMint adota deliberadamente uma arquitetura monolítica e imutável: essa escolha sacrifica a extensibilidade e a possibilidade de atualização em favor de uma superfície de ataque mínima e de auditabilidade integral, prioridades coerentes com o caráter acadêmico do trabalho. Cabe ainda destacar que, enquanto o Rarible impõe a dedução de royalties no nível do *Exchange*, o CryptoMint a realiza por meio do padrão ERC-2981 com

Tabela 7 – Comparativo arquitetural: CryptoMint, Seaport v1.6 e Rarible v2

Dimensão	CryptoMint	Seaport v1.6	Rarible v2
Armazenamento de ordens	Mapeamentos <i>on-chain</i>	<i>Off-chain (nonce on-chain)</i>	<i>Off-chain</i> (preenchimento <i>on-chain</i>)
Gas de listagem	≈50k (2 SSTOREs)	≈0 (ordem assinada)	≈0 (ordem assinada)
Atualização	Imutável	Núcleo imutável + Conduit	<i>Proxy</i> UUPS no Exchange
Padrões de <i>token</i>	ERC-721	ERC-721/1155/20	ERC-721/1155/ <i>lazy</i>
Liquidação em lote	Não suportada	<code>fulfillAvailableOrders</code>	<code>matchOrders</code> (2 ordens)
Royalties	ERC-2981 voluntário, teto 10%	Por <i>zone (opt-in)</i>	Dedução obrigatória no Exchange
Cancelamento	Por item <i>on-chain</i>	<code>incrementCounter</code> (em massa)	<i>Soft cancel / nonce</i>

Fonte: elaborado pelo autor (2026)

teto de proteção ao comprador, conforme discutido no Capítulo 4.

5.4 ANÁLISE DE SEGURANÇA

A análise de segurança do CryptoMint foi conduzida em três frentes complementares: revisão manual de código com foco nos padrões de vulnerabilidade conhecidos em contratos Solidity, análise estática com a ferramenta Slither, e testes de segurança com contratos atacantes.

5.4.1 Análise Estática com Slither

O Slither ([FEIST; GRIECO; GROCE, 2019](#); [Trail of Bits, 2018](#)), desenvolvido pela Trail of Bits e amplamente utilizado em auditorias de contratos inteligentes, foi executado sobre todos os contratos do diretório `src/` com configurações documentadas em `slither.config.json`. Os alertas identificados pelo Slither foram analisados e classificados em falsos positivos documentados — marcados com anotações `// slither-disable-next-line reentrancy-benign` — e melhorias de menor impacto. Nenhuma vulnerabilidade crítica não mitigada foi identificada pela análise estática.

5.4.2 Vulnerabilidades Mitigadas

A Tabela 8 apresenta as principais vulnerabilidades do ecossistema Ethereum Smart Contracts ([ATZEI; BARTOLETTI; CIMOLI, 2017](#)) e o status de mitigação no CryptoMint.

5.4.3 Limitações de Segurança Identificadas

A análise comparativa com protocolos de produção identificou três limitações de segurança que não representam vulnerabilidades críticas no contexto acadêmico, mas que seriam endereçadas em uma implantação de produção.

A ausência de expiração de listagens significa que uma listagem criada a baixo preço permanece válida indefinidamente, ainda que o vendedor não possa mais ser surpreendido por uma venda indesejada desde que revogue a aprovação `setApprovalForAll` — contudo, essa ação requer uma transação adicional com custo de *gas*. O limite de *gas* não especificado no reembolso de excesso de pagamento na função

Tabela 8 – Vulnerabilidades analisadas e mitigações implementadas

Vulnerabilidade	Mitigação Implementada
Reentrada (<i>reentrancy</i>)	<code>nonReentrant</code> em todas as funções que movem ETH/NFT + padrão CEI
<i>Overflow/underflow</i> aritmético	Solidity $\geq 0.8.x$ com aritmética verificada por padrão
<i>Gas bomb</i> via <code>royaltyInfo</code>	<code>staticcall</code> com limite de 30.000 <i>gas</i> + verificação de 64 bytes de retorno
<i>Return bomb</i> via <code>royaltyInfo</code>	<i>Cap</i> de 64 bytes no <code>returndatasize</code> do <code>staticcall</code>
DoS por receptor ETH malicioso	Padrão <i>pull-payment</i> para vendedor, royalty e reembolsos
<i>Front-running</i> em atualização de preço	<code>updateListingPrice</code> é atômico <i>on-chain</i>
ETH bloqueado em <i>escrow</i>	Mecanismo de expiração de ofertas e função <code>reclaimExpiredOffer</code>

Fonte: elaborado pelo autor (2026)

`mint()` poderia, teoricamente, permitir que um contrato receptor executasse lógica custosa durante o recebimento — mitigável com a adição de um limite explícito de 2.300 *gas* na chamada. O pagamento de *bounty* para limpeza de ofertas expiradas não possui *fallback* de *pull-payment*, inconsistência em relação à filosofia geral do contrato que poderia ser corrigida com o crédito do valor em `pendingWithdrawals` em caso de falha na transferência direta.

6 APRESENTAÇÃO E DEMONSTRAÇÃO DA PLATAFORMA CRYPTOMINT

Este capítulo apresenta a plataforma CryptoMint do ponto de vista do usuário final, percorrendo suas principais telas e demonstrando, de forma narrativa, como cada funcionalidade especificada nos requisitos do Capítulo 3 se materializa na interface. Enquanto o Capítulo 4 descreveu a implementação interna das três camadas do sistema, o presente capítulo adota a perspectiva da experiência de uso, ilustrando o percurso do usuário desde a página inicial até as operações de criação, cunhagem e comercialização de NFTs. As capturas de tela apresentadas referem-se à versão da aplicação implantada sobre a rede de testes Ethereum Sepolia, publicamente acessível¹.

6.1 VISÃO GERAL DA INTERFACE E NAVEGAÇÃO

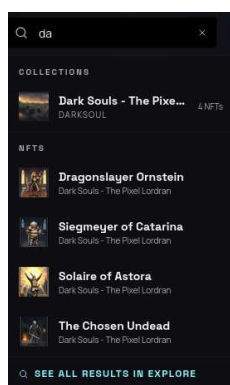
A interface do CryptoMint foi concebida segundo os princípios do *design system* **Material Design 3**, com uma paleta de cores organizada em torno de papéis semânticos — superfícies, cores primárias, secundárias e terciárias — e a tipografia composta pelas famílias *Space Grotesk*, para títulos, e *Manrope*, para o corpo de texto. A plataforma oferece dois temas visuais alternáveis pelo usuário — o tema escuro (*dark mode*) e o tema claro (*light mode*) —, e, ao longo deste capítulo, cada tela é apresentada lado a lado em ambos os temas, para fins de comparação. Essa escolha visual busca conferir à plataforma uma identidade contemporânea, coerente com o ecossistema Web3, ao mesmo tempo em que privilegia a legibilidade e a hierarquia visual da informação.

A navegação principal é mediada por uma barra de navegação (*navbar*) persistente no topo de todas as páginas, que dá acesso às cinco áreas centrais da aplicação: **Explore**, **Collections**, **Mint**, **Activity** e, quando há uma carteira conectada, **Profile**. A

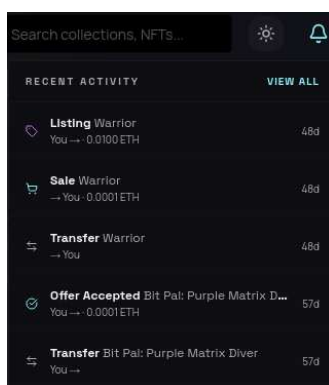
¹ Endereço público do sistema: <<https://cryptomint.com.br>>. Acesso em: jun. 2026.

barra também hospeda três elementos interativos, ilustrados na Figura 4 nos temas escuro e claro: uma **barra de busca global** (*search bar*), que permite localizar coleções e *tokens* por nome; um **menu de notificações** (*bell*), que informa o usuário sobre eventos recentes relevantes envolvendo seus ativos, com indicação de itens não vistos; e o **botão de conexão de carteira** (*wallet*) — responsável pela autenticação Web3 via ConnectKit (RF13) —, que, uma vez conectado, expõe um menu com as informações da conta e o atalho para o perfil. As operações de leitura disponíveis nessas áreas são satisfeitas pelas consultas ao *subgraph*, ao passo que as operações de escrita exigem a assinatura de uma transação pela carteira do usuário, conforme descrito no Capítulo 4.

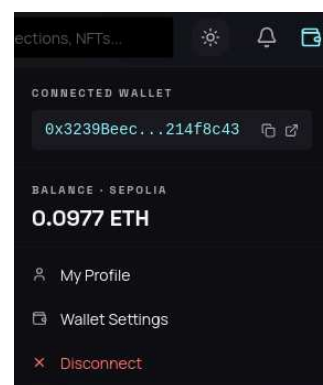
Figura 4 – Elementos interativos da barra de navegação nos temas escuro e claro



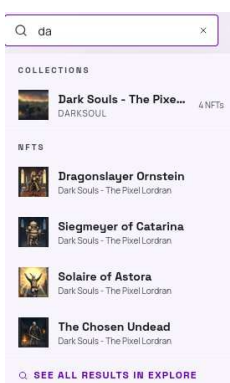
(a) Busca — escuro



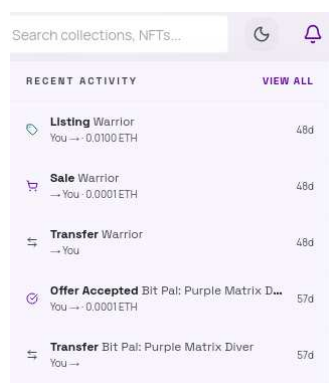
(b) Notificações — escuro



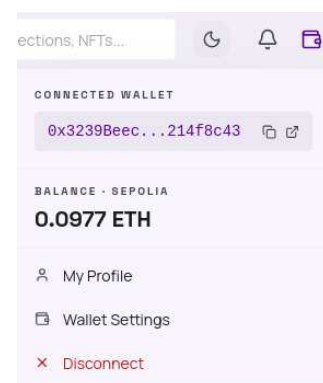
(c) Carteira — escuro



(d) Busca — claro



(e) Notificações — claro



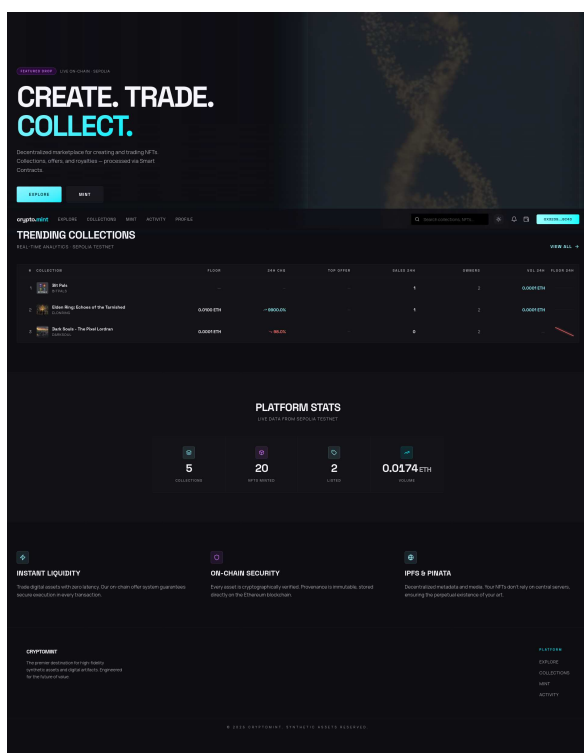
(f) Carteira — claro

Fonte: Captura de tela da aplicação desenvolvida pelo autor (2026)

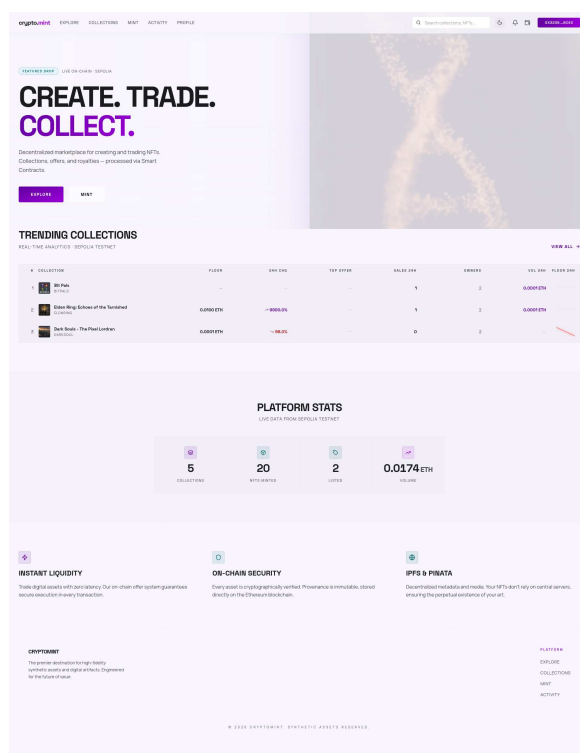
6.2 PÁGINA INICIAL

A página inicial (*landing page*), ilustrada na Figura 5, é o ponto de entrada da plataforma. Ela é composta por uma seção de destaque (*hero*) que comunica a proposta de valor do *marketplace* e convida o usuário a explorar ou criar coleções; por uma seção de estatísticas agregadas do *marketplace* — volume negociado, número de coleções e de itens —, derivada da entidade `MarketplaceStats` do *subgraph*; e por uma seção de coleções em destaque (*trending*), que apresenta as coleções de maior atividade recente. Essa página cumpre o papel de oferecer, já no primeiro acesso, uma visão sintética do estado do *marketplace* (RF15) e atalhos para os fluxos principais.

Figura 5 – Página inicial da plataforma CryptoMint



(a) Tema escuro



(b) Tema claro

Fonte: Captura de tela da aplicação desenvolvida pelo autor (2026)

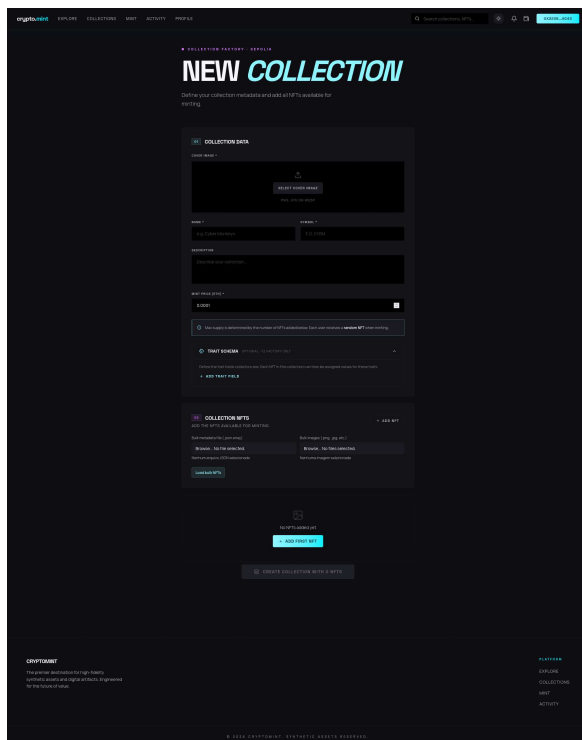
6.3 CRIAÇÃO DE COLEÇÕES

A criação de uma coleção, acessível pela rota `/collections/create`, é um dos fluxos mais elaborados da plataforma, por integrar o *upload* ao IPFS, a implantação

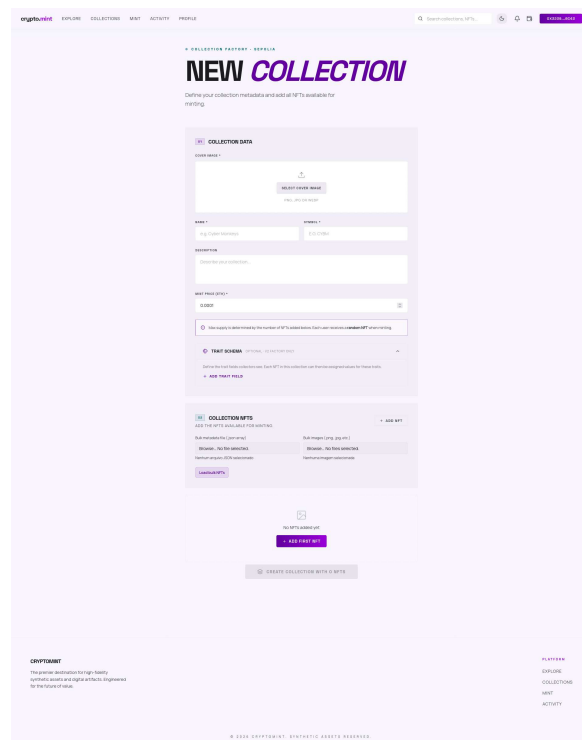
de um novo contrato `NFTCollectionV2` via *factory* e o esquema de *commit-reveal*. A interface guia o criador por um assistente de múltiplas etapas, ilustrado na Figura 6. Na primeira etapa, o criador preenche os metadados da coleção — nome, símbolo, preço de *mint* e oferta máxima — e submete as imagens dos *tokens* (RF01, RF02). Na sequência, a aplicação realiza a fixação (*pin*) das mídias e dos metadados no IPFS via Pinata e carrega os respectivos URIs no contrato. Por fim, o criador publica o *commitment* criptográfico da semente de aleatoriedade (*seed*), conforme o mecanismo de *commit-reveal* descrito no Capítulo 4, dando origem a uma coleção apta a receber operações de *mint* com atribuição de raridades verificável (RF03).

A interface fornece *feedback* explícito sobre a etapa corrente do processo — assinatura na carteira, confirmação na *blockchain* e conclusão —, em conformidade com o requisito de usabilidade RNF07. Esse comportamento é orquestrado pela máquina de estados encapsulada nos *hooks* de mutação de dois passos descritos no Capítulo 4.

Figura 6 – Assistente de criação de coleção



(a) Tema escuro

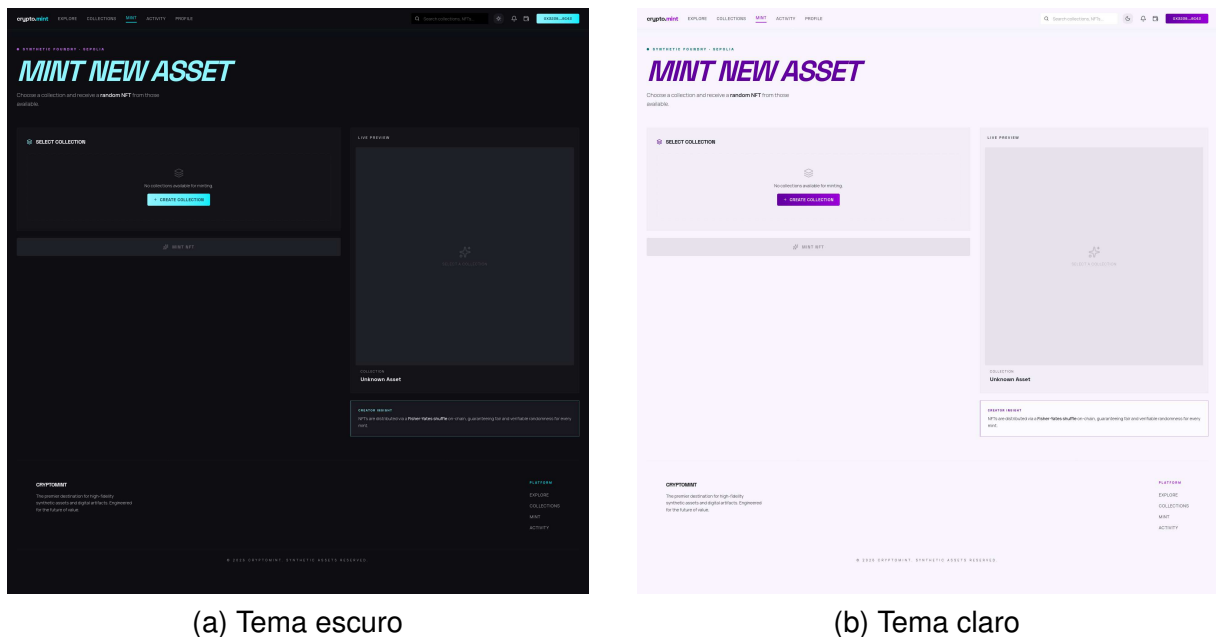


(b) Tema claro

Fonte: Captura de tela da aplicação desenvolvida pelo autor (2026)

6.4 CUNHAGEM DE NFTS

A cunhagem (*mint*) de NFTs é oferecida na rota `/create` e também por meio de um *modal* acessível diretamente na página de cada coleção. Conforme ilustrado na Figura 7, o usuário seleciona uma coleção disponível, visualiza informações como o preço de *mint*, a oferta máxima e a quantidade já cunhada, e confirma a operação mediante o pagamento em ETH (RF03). Após a confirmação da transação, a interface exibe o resultado da cunhagem e disponibiliza um *link* para o *token* recém-criado e para o registro da transação no explorador de blocos Etherscan, reforçando a transparência e a auditabilidade que caracterizam o sistema (RNF04).

Figura 7 – Tela de cunhagem (*mint*) de NFTs

(a) Tema escuro

(b) Tema claro

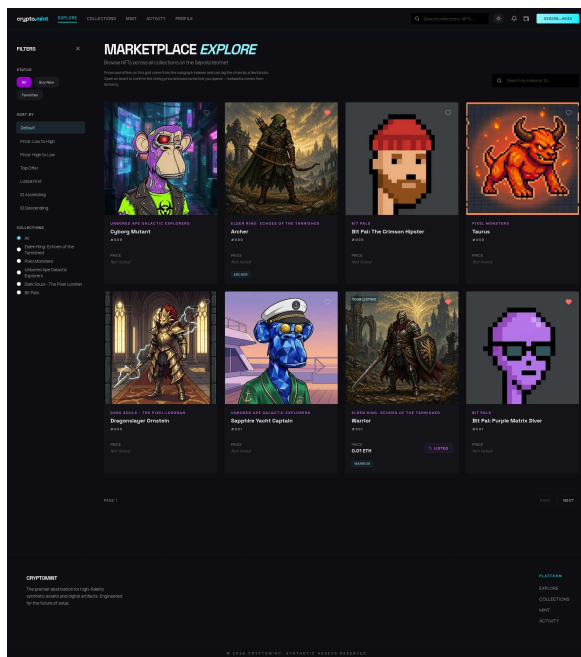
Fonte: Captura de tela da aplicação desenvolvida pelo autor (2026)

6.5 EXPLORAÇÃO E DESCOBERTA DE ATIVOS

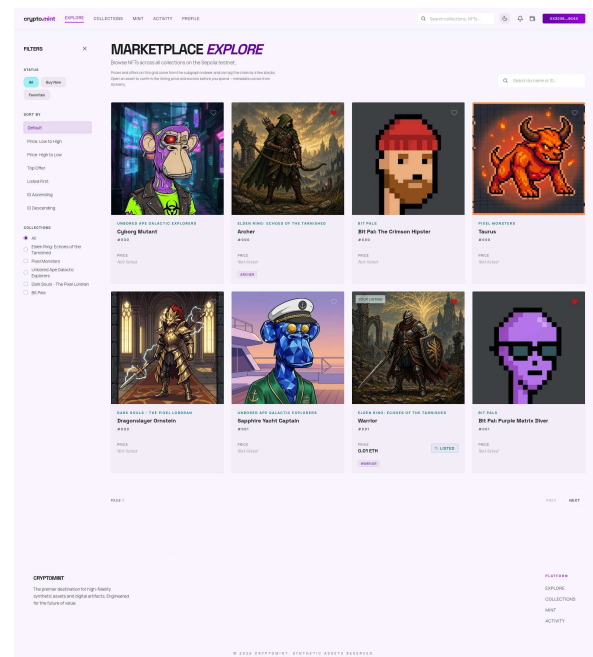
A descoberta de ativos é viabilizada por duas telas complementares. A página **Explore** (rota `/explore`), apresentada na Figura 8, lista os NFTs de todas as coleções em uma grade paginada de cartões, acompanhada de um painel lateral de filtros que permite ao usuário pesquisar por texto, ordenar os resultados, restringir a visualização apenas aos itens listados para venda, exibir somente os favoritos e filtrar por atributos de raridade (*traits*) específicos da coleção selecionada (RF14). A possibilidade de marcar ativos como favoritos confere à navegação um caráter personalizado.

A página **Collections** (rota `/collections`), apresentada na Figura 9, oferece uma visão complementar, organizando as coleções em um *ranking* ordenado por métricas de desempenho extraídas das entidades de estatística do *subgraph*, tais como volume negociado e número de itens (RF15). Ambas as telas dependem exclusivamente do *subgraph* para suas consultas, o que evidencia, na prática, os benefícios de desempenho da filosofia *subgraph-first* discutida no Capítulo 4 (RNF05).

Figura 8 – Tela de exploração de ativos com painel de filtros

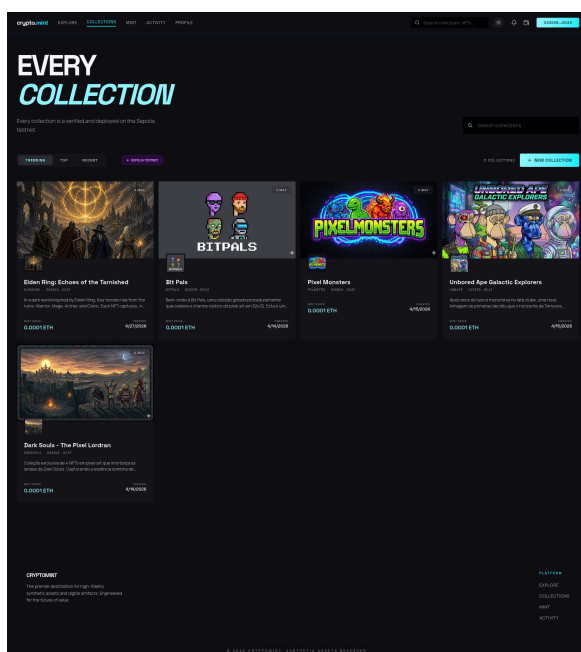


(a) Tema escuro

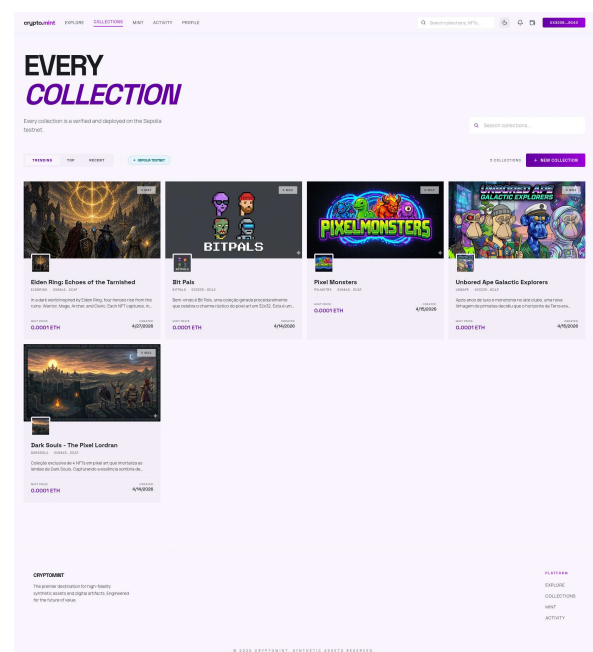


(b) Tema claro

Fonte: Captura de tela da aplicação desenvolvida pelo autor (2026)

Figura 9 – Página de coleções com *ranking* por desempenho

(a) Tema escuro



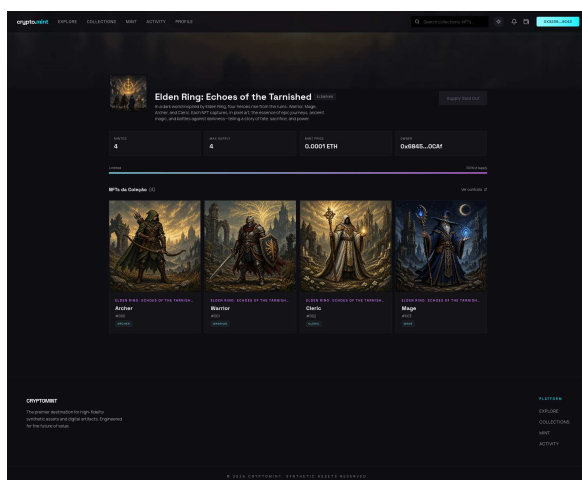
(b) Tema claro

Fonte: Captura de tela da aplicação desenvolvida pelo autor (2026)

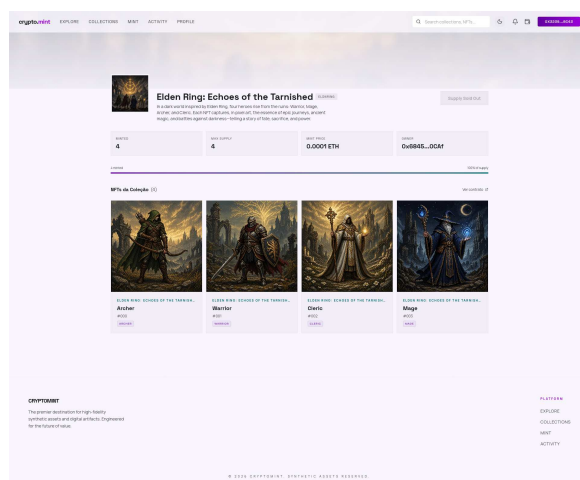
6.6 PÁGINA DA COLEÇÃO

Ao selecionar uma coleção específica, o usuário é conduzido à respectiva página (rota `/collections/{endereço}`), ilustrada na Figura 10. Essa tela exibe a imagem de capa, a descrição e as estatísticas da coleção, seguidas pela grade de *tokens* que a compõem, cada um com seu *status* de listagem. A partir dela, o usuário pode acionar o *modal* de cunhagem (quando ainda há *tokens* disponíveis) ou navegar para a página de um ativo individual. Quando o usuário conectado é o criador da coleção, painéis administrativos adicionais são exibidos, permitindo operações reservadas ao proprietário, como a revelação da semente de aleatoriedade e o saque de valores acumulados.

Figura 10 – Página de uma coleção



(a) Tema escuro



(b) Tema claro

Fonte: Captura de tela da aplicação desenvolvida pelo autor (2026)

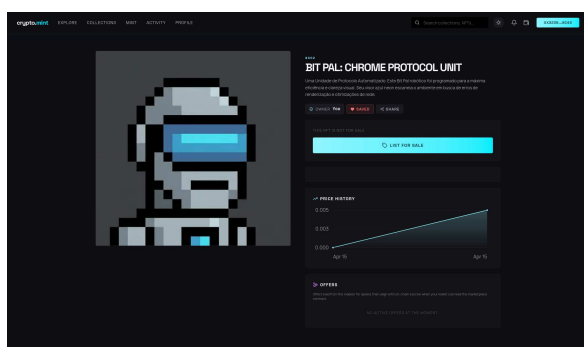
6.7 PÁGINA DO ATIVO E OPERAÇÕES DE COMERCIALIZAÇÃO

A página de um ativo individual (rota `/asset/{id}`), apresentada na Figura 11, concentra as operações comerciais do *marketplace* e constitui o núcleo da experiência de negociação. Nela são exibidos a imagem do NFT, seus metadados e atributos de raridade, o histórico de propriedade e a informação de royalties (RF11). As ações disponíveis variam conforme a relação entre o usuário conectado e o ativo:

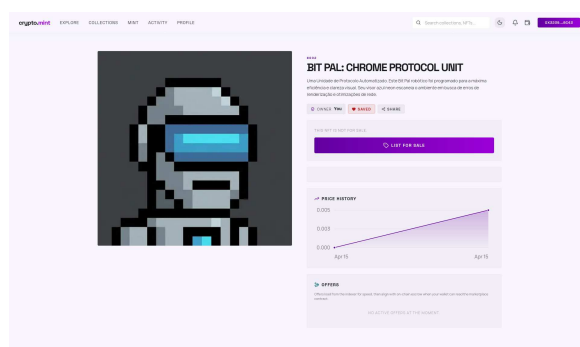
- Quando o usuário é o **proprietário** do *token*, a interface oferece as ações de **listar** o NFT a um preço fixo (RF04) e de **cancelar** uma listagem ativa (RF08), além de permitir a aceitação de ofertas recebidas (RF07).
- Quando o usuário **não é o proprietário**, a interface oferece as ações de **comprar** o NFT, caso ele esteja listado (RF05), e de **realizar uma oferta** em ETH, que fica retida em *escrow* pelo contrato (RF06).

A operação de listagem ilustra o fluxo de duas etapas característico do *marketplace* não custodiante: o usuário primeiro concede ao contrato a aprovação `setApprovalForAll` e, em seguida, registra a listagem — processo conduzido pela máquina de estados que mantém o usuário informado sobre cada etapa (RNF07). As ofertas recebidas são exibidas em uma lista, com indicação de validade; ofertas expiradas podem ser recuperadas, devolvendo o ETH em *escrow* ao ofertante (RF09). Os valores resultantes de vendas, royalties e recompensas são creditados em saldos pendentes que o usuário resgata por meio da ação de saque (RF10), em conformidade com o padrão *pull-payment* adotado nos contratos.

Figura 11 – Página de um ativo com as operações de comercialização



(a) Tema escuro



(b) Tema claro

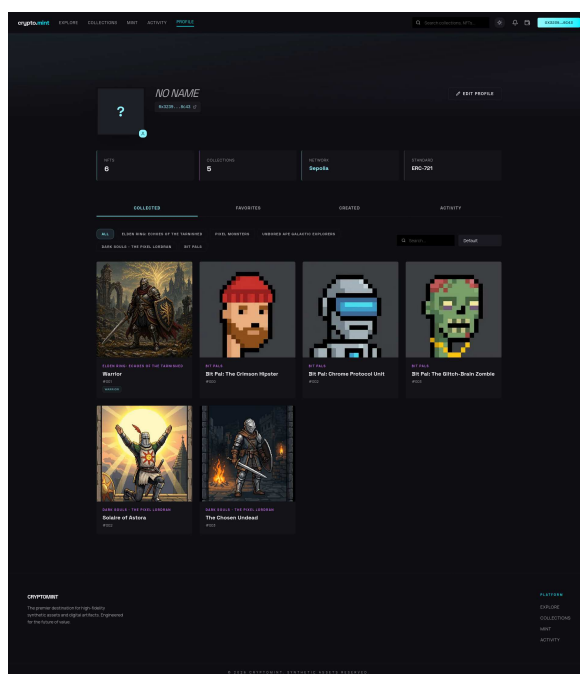
Fonte: Captura de tela da aplicação desenvolvida pelo autor (2026)

6.8 PERFIL DO USUÁRIO

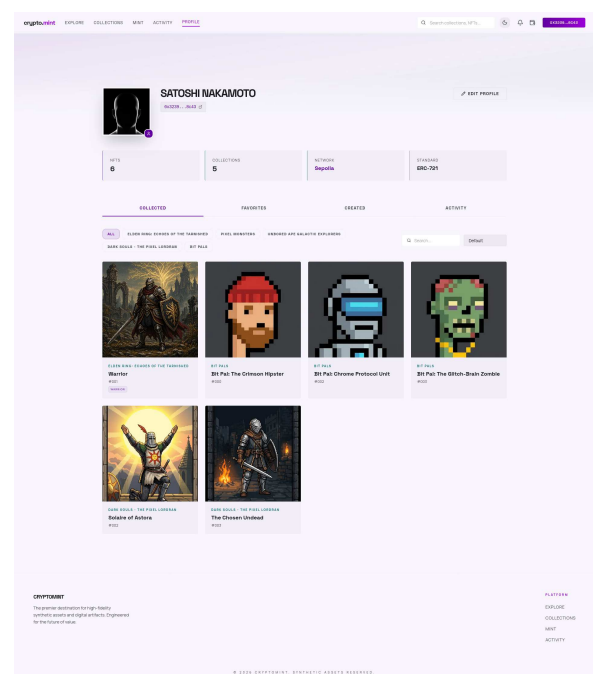
A página de perfil (rota `/profile`), ilustrada na Figura 12, consolida a atividade do usuário conectado. Ela organiza, em abas, os NFTs de sua propriedade, os *tokens*

por ele criados, os ativos marcados como favoritos e o histórico de suas atividades no *marketplace*. A partir do perfil, o usuário também acessa os saldos pendentes de saque (RF10) e pode editar suas informações públicas por meio da rota `/profile/edit`, ilustrada na Figura 13; o perfil antes e depois da edição é comparado na Figura 14. Essa tela reúne, em um único ponto, a visão consolidada do portfólio do usuário, agregando dados provenientes tanto do *subgraph* quanto do serviço de metadados de NFTs.

Figura 12 – Página de perfil do usuário



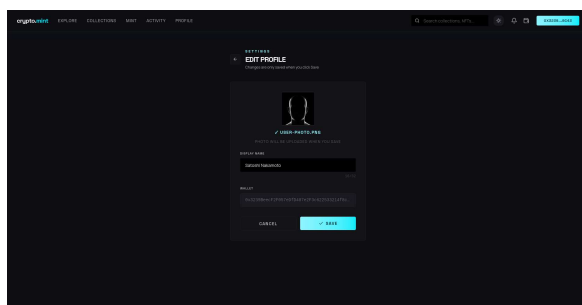
(a) Tema escuro



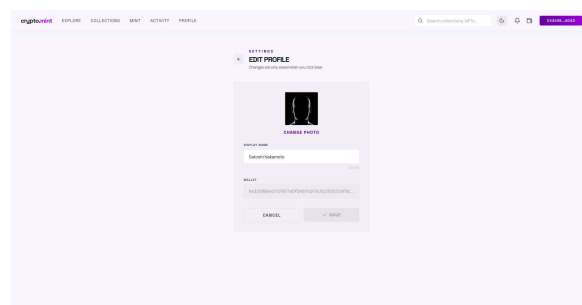
(b) Tema claro

Fonte: Captura de tela da aplicação desenvolvida pelo autor (2026)

Figura 13 – Formulário de edição de perfil



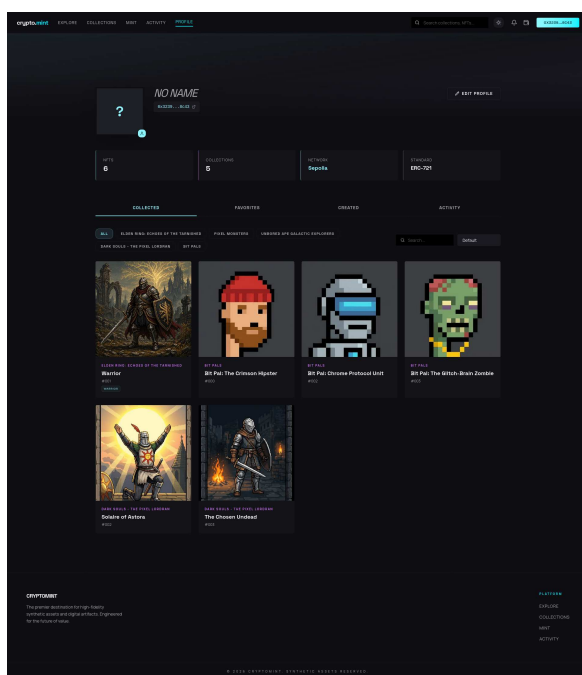
(a) Tema escuro



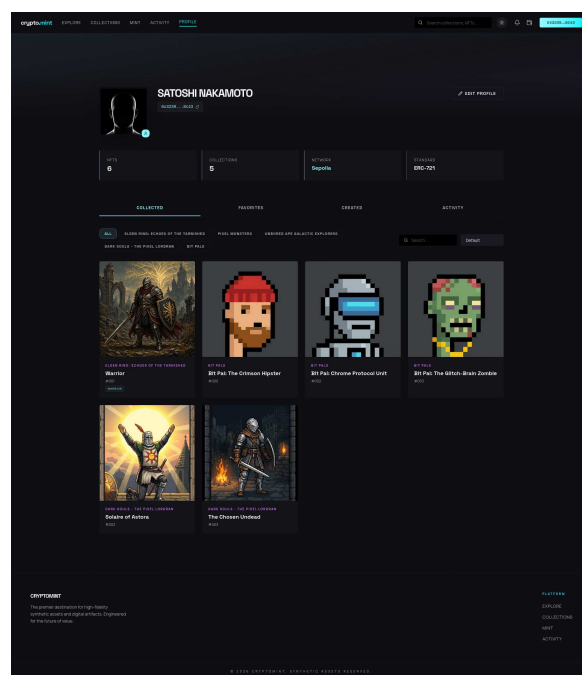
(b) Tema claro

Fonte: Captura de tela da aplicação desenvolvida pelo autor (2026)

Figura 14 – Página de perfil antes e depois da edição das informações



(a) Antes da edição



(b) Depois da edição

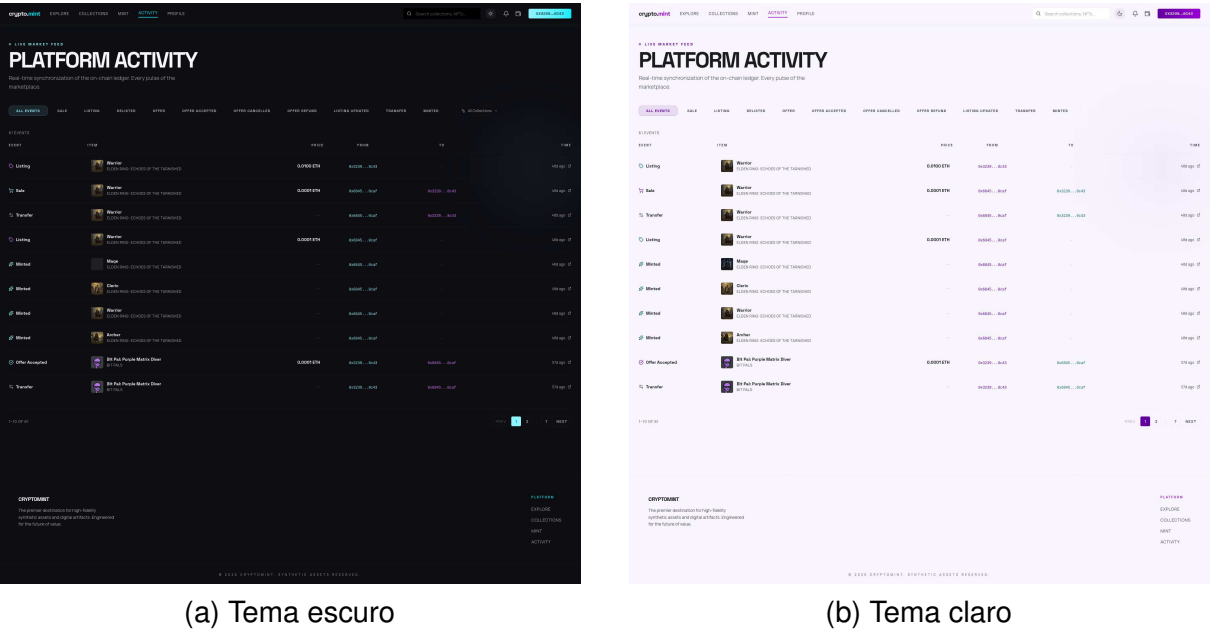
Fonte: Captura de tela da aplicação desenvolvida pelo autor (2026)

6.9 FEED DE ATIVIDADES

A página de atividades (rota `/activity`), apresentada na Figura 15, oferece um *feed* cronológico de todos os eventos significativos ocorridos no *marketplace* — cunhagens, listagens, vendas, ofertas e cancelamentos —, alimentado pela entidade

ActivityEvent do *subgraph* (RF15). O usuário pode filtrar o *feed* por tipo de evento e por coleção, e cada registro disponibiliza um *link* para a respectiva transação no Etherscan, reforçando o compromisso da plataforma com a transparência e a auditabilidade *on-chain*.

Figura 15 – Feed de atividades do *marketplace*



Fonte: Captura de tela da aplicação desenvolvida pelo autor (2026)

6.10 SÍNTESE DAS FUNCIONALIDADES

A demonstração conduzida ao longo deste capítulo evidencia que a interface da plataforma CryptoMint contempla a totalidade dos requisitos funcionais especificados no Capítulo 3. A Tabela 9 sintetiza o mapeamento entre as principais telas da aplicação e os requisitos funcionais por elas atendidos, demonstrando a rastreabilidade entre a especificação e a solução entregue.

Tabela 9 – Mapeamento entre telas e requisitos funcionais atendidos

Tela	Requisitos funcionais atendidos
Página inicial	RF15
Criação de coleções	RF01, RF02, RF03
Cunhagem (<i>mint</i>)	RF03
Exploração (<i>Explore</i>)	RF14
Coleções (<i>ranking</i>)	RF14, RF15
Página da coleção	RF03, RF14
Página do ativo	RF04, RF05, RF06, RF07, RF08, RF09, RF10, RF11, RF12
Perfil do usuário	RF10, RF14, RF15
Feed de atividades	RF15
Barra de navegação	RF13

Fonte: elaborado pelo autor (2026)

7 CONCLUSÃO

7.1 CONSIDERAÇÕES FINAIS

O presente trabalho propôs e implementou o **CryptoMint**, uma plataforma Web3 para criação e comercialização de Non-Fungible Tokens na *blockchain* Ethereum, com o objetivo de demonstrar que é possível construir um *marketplace* descentralizado que endereça, por meios técnicos e verificáveis, três problemas estruturais identificados nos sistemas existentes: a custódia centralizada de ativos, a falta de transparência na atribuição de raridades no processo de *mint* e a evasão de royalties devidos a criadores em transações de revenda.

O sistema desenvolvido atende integralmente ao objetivo geral proposto, provendo uma plataforma funcional de ponta a ponta, desde a criação e *mint* de coleções NFT até a comercialização em *marketplace*, com interface de usuário conectada a carteiras Web3 e dados indexados em tempo real pelo protocolo The Graph. Os contratos inteligentes do sistema foram implantados na rede de testes Ethereum Sepolia e validados por uma suíte de testes abrangente implementada com o Foundry. A seguir, avalia-se o alcance de cada objetivo específico enunciado na Seção 1.3.2.

O primeiro objetivo específico — implementar contratos inteligentes com mecanismo de *mint* com aleatoriedade verificável e suporte ao padrão ERC-2981 — foi plenamente atendido. O contrato `NFTCollectionV2` implementa o esquema de *commit-reveal* combinado com o algoritmo Fisher-Yates para a atribuição de URIs de metadados durante o *mint*, garantindo imparcialidade auditável publicamente. O padrão ERC-2981 está implementado com taxa padrão de 5% e mecanismo de *pull-payment* para proteção contra receptores maliciosos.

O segundo objetivo específico — desenvolver um contrato de *marketplace* não custodiante — foi atendido pelo contrato `NFTMarketplace`, que implementa listagens a preço fixo, sistema de ofertas com ETH em *escrow* e expiração de sete dias, dis-

tribuição automática de royalties com teto de 10%, e cobrança de taxa de plataforma configurável (padrão de 2,5%). O modelo não custodiante é garantido pelo fato de os NFTs permanecerem sob propriedade exclusiva do vendedor durante toda a listagem.

O terceiro objetivo específico — implementar um *subgraph* para indexação de eventos *on-chain* — foi atendido por um *subgraph* com treze entidades, *handlers* AssemblyScript para todos os eventos relevantes dos contratos e suporte a *templates* dinâmicos para indexação de coleções criadas em *runtime*. O *frontend* implementa adicionalmente um mecanismo de *circuit-breaker* que garante degradação graciosa em caso de limitação de taxa pelo Graph Studio.

O quarto objetivo específico — desenvolver uma interface de usuário com Next.js 16 — foi atendido pela aplicação *frontend*, que oferece navegação completa do *marketplace*, integração com carteiras MetaMask e WalletConnect via ConnectKit, fluxo de criação de coleções com *upload* ao IPFS e interfaces de compra, oferta e listagem de NFTs.

O quinto objetivo específico — validar a implementação por meio de testes e análise comparativa — foi atendido pela suíte de mais de 40 casos de teste Foundry, pelos testes Vitest do *frontend*, pela análise estática com Slither e pela análise comparativa com o OpenSea Seaport v1.6 e o Rarible Protocol v2 documentada no Capítulo 5.

7.2 CONTRIBUIÇÕES TÉCNICAS

O CryptoMint oferece contribuições técnicas em dois âmbitos: o *design* e implementação de cada componente individualmente, e a integração coesa das três camadas em um sistema que demonstra padrões arquiteturais aplicáveis ao desenvolvimento de DApps.

No âmbito dos contratos inteligentes, destacam-se: a implementação do mecanismo de *commit-reveal* com Fisher-Yates sem custo de oráculo pago (como seria necessário com Chainlink VRF), provendo aleatoriedade verificável a custo zero de

gas adicional para o usuário; a aplicação disciplinada do padrão CEI combinada com `nonReentrant` como defesas complementares — não alternativas — contra ataques de reentrada; e o uso de `staticcall` com limite de *gas* e limite de tamanho de retorno para a consulta ao `royaltyInfo`, prevenindo dois vetores de ataque distintos (*gas bomb* e *return bomb*) que contratos de *marketplace* ingênuos poderiam ignorar.

No âmbito do *subgraph*, destaca-se o padrão de *dual-ID* para entidades de oferta — que resolve a limitação do The Graph de não suportar deleção de entidades, permitindo simultaneamente o estado atual mutável e um histórico imutável de todas as ofertas —, e a integração de *circuit-breaker* no *proxy* do *frontend* para degradação graciosa em cenários de *rate-limiting* do serviço gerenciado.

No âmbito do *frontend*, destacam-se: a política explícita de fontes de dados (*subgraph-first* com RPC autoritativo para transações) implementada como um conjunto de decisões arquiteturais documentadas e aplicadas nos *hooks*; o uso de `useSyncExternalStore` para compartilhamento de estado global de alta frequência sem proliferação de `Context` e sem re-renders desnecessários; e a separação rigorosa de segredos de API em *Route Handlers* do servidor, com *rate-limiting* por IP e por endereço de carteira.

7.3 LIMITAÇÕES RECONHECIDAS

O presente trabalho reconhece um conjunto de limitações que, embora aceitáveis no contexto acadêmico, seriam endereçadas em uma implantação de produção.

A utilização da rede de testes Ethereum Sepolia implica que o sistema não foi exposto à pressão econômica e à diversidade de comportamentos maliciosos presentes na *mainnet*. A entropia fornecida pelo `blockhash` na Sepolia é menos robusta do que na *mainnet*, reduzindo a garantia do mecanismo de *commit-reveal* contra manipulação por parte de validadores em cenários de alto valor econômico. Para produção em *mainnet* com coleções de alto valor, seria recomendável substituir o *commit-reveal* pelo Chainlink VRF v2.5, que fornece entropia criptograficamente verificável por um

oráculo externo descentralizado.

Os contratos implantados não utilizam padrões de *upgradeability* (como *proxies* UUPS ou Transparent), o que significa que qualquer correção de *bug* ou melhoria de funcionalidade exigiria a implantação de novos contratos e a migração manual dos dados históricos. Essa decisão foi deliberada para maximizar a auditabilidade e minimizar a superfície de ataque no contexto acadêmico, mas representa um *trade-off* relevante para sistemas de produção com longa expectativa de vida.

O *marketplace* suporta apenas o padrão ERC-721 e ofertas denominadas em ETH nativo. Suporte a *tokens* ERC-1155 ([RADOMSKI et al., 2018](#)) (que permitem edições fungíveis de arte) e a ofertas em ERC-20 (como WETH ou *stablecoins*) ampliariam significativamente o alcance da plataforma. A ausência de listagens com expiração automática é também uma limitação de segurança identificada na análise comparativa, pois permite que listagens antigas a preços desatualizados permaneçam válidas indefinidamente enquanto a aprovação do vendedor não for revogada.

Por fim, a dependência do Graph Studio como serviço gerenciado para o *subgraph* introduz um ponto de falha externo: se o serviço ficar indisponível, o *circuit-breaker* do *frontend* degrada para leituras RPC, mas operações de leitura complexas (como filtros por atributos de raridade) não têm *fallback* equivalente via RPC.

7.4 TRABALHOS FUTUROS

As direções para trabalhos futuros foram identificadas a partir das limitações reconhecidas e das lacunas em relação a protocolos de produção levantadas na análise comparativa do Capítulo 5.

No curto prazo, a adição de expiração de listagens — pela inclusão de um campo `uint64 expiresAt` na *struct* `Listing` — é a melhoria de segurança de maior impacto e menor complexidade de implementação. A cobertura de testes do *subgraph* com o *framework* `Matchstick`, específico para *handlers* `AssemblyScript`, completaria a suíte de testes *end-to-end* do sistema. A substituição do *store* em memória do *rate-*

limiter do *frontend* por Redis tornaria a arquitetura adequada para implantações com múltiplas réplicas do servidor Next.js.

No médio prazo, a adoção do padrão de listagens por ordens assinadas (EIP-712), análogo ao utilizado pelo Seaport, eliminaria o custo de *gas* de criação de listagem (≈ 50.000 *gas* por listagem) ao transferir o armazenamento das ordens para infraestrutura *off-chain*, mantendo a liquidação *on-chain*. A extensão do *marketplace* para suporte ao padrão ERC-1155 (RADOMSKI et al., 2018) ampliaria o universo de ativos negociáveis. A adição de suporte a redes de camada 2 (L2) da Ethereum — como Base, Arbitrum ou Polygon — com custos de *gas* significativamente menores tornaria o sistema viável para um universo mais amplo de usuários.

No longo prazo, a implementação de mecanismos de leilão *on-chain* (à inglesa ou holandesa) — conforme demonstrado pelos protocolos Zora (Zora Labs, 2022) e Foundation (Foundation Labs, 2021) — ampliaria os mecanismos de descoberta de preço disponíveis, e a integração com o Ethereum Name Service (ENS) para resolução de identidades de usuários substituiria o armazenamento de nomes em `localStorage` por um mecanismo descentralizado e portátil entre plataformas.

REFERÊNCIAS BIBLIOGRÁFICAS

ANTONPOULOS, A. M.; WOOD, G. *Mastering Ethereum: Building Smart Contracts and DApps*. Sebastopol: O'Reilly Media, 2018. ISBN 978-1491971949. Citado 3 vezes nas páginas 21, 24 e 26.

ATZEI, N.; BARTOLETTI, M.; CIMOLI, T. A survey of attacks on ethereum smart contracts (SoK). In: *Principles of Security and Trust (POST)*. [S.l.]: Springer, 2017. (Lecture Notes in Computer Science, v. 10204), p. 164–186. Citado 3 vezes nas páginas 27, 57 e 73.

BARTOLETTI, M.; POMPIANU, L. An empirical analysis of smart contracts: Platforms, applications, and design patterns. In: *Financial Cryptography and Data Security*. [S.l.]: Springer, 2017. (Lecture Notes in Computer Science, v. 10323), p. 494–509. Citado na página 26.

BECK, K. *Test-Driven Development: By Example*. Boston: Addison-Wesley, 2003. ISBN 978-0321146533. Citado na página 43.

BENET, J. IPFS – content addressed, versioned, P2P file system. *arXiv preprint arXiv:1407.3561*, 2014. Acesso em: jan. 2026. Disponível em: <<https://arxiv.org/abs/1407.3561>>. Citado 4 vezes nas páginas 20, 30, 31 e 32.

BERNERS-LEE, T.; FISCHETTI, M. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web*. New York: HarperCollins, 1999. ISBN 978-0062515865. Citado na página 19.

BLUM, M. Coin flipping by telephone a protocol for solving impossible problems. *ACM SIGACT News*, v. 15, n. 1, p. 23–27, 1983. Citado na página 28.

BROWN, S. *The C4 model for visualising software architecture*. 2018. Acesso em: maio 2026. Disponível em: <<https://c4model.com>>. Citado 2 vezes nas páginas 44 e 47.

BURKS, Z. et al. *ERC-2981: NFT Royalty Standard*. 2020. Ethereum Improvement Proposal 2981. Acesso em: jan. 2026. Disponível em: <<https://eips.ethereum.org/EIPS/eip-2981>>. Citado 3 vezes nas páginas 30, 39 e 56.

BUTERIN, V. *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform*. 2013. Whitepaper. Acesso em: jan. 2026. Disponível em: <<https://ethereum.org/en/whitepaper/>>. Citado na página 20.

DANNEN, C. *Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners*. Berkeley: Apress, 2017. ISBN 978-1484225349. Citado 2 vezes nas páginas 21 e 26.

ENTRIKEN, W. et al. *ERC-721: Non-Fungible Token Standard*. 2018. Ethereum Improvement Proposal 721. Acesso em: jan. 2026. Disponível em: <<https://eips.ethereum.org/EIPS/eip-721>>. Citado 2 vezes nas páginas 14 e 29.

Ethereum Foundation. *The Merge*. 2022. Acesso em: jan. 2026. Disponível em: <<https://ethereum.org/en/upgrades/merge/>>. Citado 2 vezes nas páginas 23 e 25.

FEIST, J.; GRIECO, G.; GROCE, A. Slither: A static analysis framework for smart contracts. In: *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. [S.l.]: IEEE, 2019. p. 8–15. Citado 2 vezes nas páginas 58 e 73.

FISHER, R. A.; YATES, F. *Statistical Tables for Biological, Agricultural and Medical Research*. Edinburgh: Oliver and Boyd, 1938. Citado na página 28.

Foundation Labs. *Foundation: A New Creative Economy*. 2021. Acesso em: maio 2026. Disponível em: <<https://foundation.app/>>. Citado 2 vezes nas páginas 39 e 92.

GACKENHEIMER, C. *Introduction to React*. Berkeley: Apress, 2015. ISBN 978-1484212462. Citado na página 35.

GIL, A. C. *Como Elaborar Projetos de Pesquisa*. 4. ed. São Paulo: Atlas, 2002. ISBN 978-8522431694. Citado na página 40.

HEVNER, A. R. et al. Design science in information systems research. *MIS Quarterly*, Management Information Systems Research Center, v. 28, n. 1, p. 75–105, 2004. Citado 2 vezes nas páginas 41 e 42.

International Organization for Standardization. *ISO/IEC 25010: Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — System and Software Quality Models*. 2011. ISO/IEC 25010:2011. Acesso em: maio 2026. Disponível em: <<https://www.iso.org/standard/35733.html>>. Citado na página 46.

JOHNSTON, D. et al. *The General Theory of Decentralized Applications, Dapps*. 2014. Acesso em: jan. 2026. Disponível em: <<https://github.com/DavidJohnstonCEO/DecentralizedApplications>>. Citado na página 20.

KNUTH, D. E. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. 3. ed. Reading: Addison-Wesley, 1997. ISBN 978-0201896848. Citado na página 28.

Meta Platforms. *React: A JavaScript Library for Building User Interfaces*. 2013. Acesso em: jan. 2026. Disponível em: <<https://react.dev/>>. Citado na página 35.

Microsoft. *TypeScript: JavaScript That Scales*. 2012. Acesso em: jan. 2026. Disponível em: <<https://www.typescriptlang.org/>>. Citado na página 36.

NADINI, M. et al. Mapping the NFT revolution: Market trends, trade networks, and visual features. *Scientific Reports*, Nature Publishing Group, v. 11, n. 1, p. 20902, 2021. Citado 3 vezes nas páginas 14, 31 e 37.

NAKAMOTO, S. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008. Whitepaper. Acesso em: jan. 2026. Disponível em: <<https://bitcoin.org/bitcoin.pdf>>. Citado 5 vezes nas páginas 14, 20, 22, 23 e 25.

NYGARD, M. T. *Release It!: Design and Deploy Production-Ready Software*. Raleigh: Pragmatic Bookshelf, 2007. ISBN 978-0978739218. Citado na página 61.

OpenZeppelin. *OpenZeppelin Contracts Documentation*. 2023. Acesso em: jan. 2026. Disponível em: <<https://docs.openzeppelin.com/contracts/>>. Citado na página 26.

O'REILLY, T. What is web 2.0: Design patterns and business models for the next generation of software. *Communications & Strategies*, n. 65, p. 17–37, 2007. Citado na página 19.

Paradigm. *Foundry Book*. 2024. Acesso em: jan. 2026. Disponível em: <<https://book.getfoundry.sh/>>. Citado 2 vezes nas páginas 27 e 68.

RADOMSKI, W. et al. *ERC-1155: Multi Token Standard*. 2018. Ethereum Improvement Proposal 1155. Acesso em: jan. 2026. Disponível em: <<https://eips.ethereum.org/EIPS/eip-1155>>. Citado 3 vezes nas páginas 38, 91 e 92.

SOMMERVILLE, I. *Engenharia de Software*. 9. ed. São Paulo: Pearson Prentice Hall, 2011. ISBN 978-8579361081. Citado 3 vezes nas páginas 42, 44 e 45.

SWAN, M. *Blockchain: Blueprint for a New Economy*. Sebastopol: O'Reilly Media, 2015. ISBN 978-1491920497. Citado na página 23.

SZABO, N. *Smart Contracts*. 1994. Manuscrito não publicado. Acesso em: jan. 2026. Disponível em: <<https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html>>. Citado na página 25.

The Graph Foundation. *The Graph Protocol Documentation*. 2021. Acesso em: jan. 2026. Disponível em: <<https://thegraph.com/docs/>>. Citado na página 33.

Trail of Bits. *Slither: Static Analyzer for Solidity*. 2018. Acesso em: jan. 2026. Disponível em: <<https://github.com/crytic/slither>>. Citado 2 vezes nas páginas 58 e 73.

Vercel. *Next.js Documentation*. 2024. Acesso em: jan. 2026. Disponível em: <<https://nextjs.org/docs>>. Citado na página 36.

WANG, Q. et al. Non-fungible token (NFT): Overview, evaluation, opportunities and challenges. *arXiv preprint arXiv:2105.07447*, 2021. Acesso em: jan. 2026. Disponível em: <<https://arxiv.org/abs/2105.07447>>. Citado 5 vezes nas páginas 14, 29, 31, 32 e 37.

WAZLAWICK, R. S. *Metodologia de Pesquisa para Ciência da Computação*. 2. ed. Rio de Janeiro: Elsevier, 2014. ISBN 978-8535271782. Citado na página 40.

Wevm. *wagmi: React Hooks for Ethereum*. 2024. Acesso em: jan. 2026. Disponível em: <<https://wagmi.sh/>>. Citado na página 63.

WIERINGA, R. J. *Design Science Methodology for Information Systems and Software Engineering*. Berlin, Heidelberg: Springer, 2014. ISBN 978-3662438381. Citado na página 41.

WOOD, G. *Ethereum: A Secure Decentralised Generalised Transaction Ledger*. [S.l.], 2014. Ethereum Yellow Paper. Acesso em: jan. 2026. Disponível em: <<https://ethereum.github.io/yellowpaper/paper.pdf>>. Citado 5 vezes nas páginas 14, 19, 24, 25 e 54.

ZHENG, Z. et al. An overview of blockchain technology: Architecture, consensus, and future trends. In: *IEEE International Congress on Big Data (BigData Congress)*. [S.l.]: IEEE, 2017. p. 557–564. Citado 3 vezes nas páginas 20, 21 e 22.

Zora Labs. *Zora Protocol: A Market for NFTs Built on Ethereum*. 2022. Acesso em: maio 2026. Disponível em: <<https://docs.zora.co/>>. Citado 2 vezes nas páginas 38 e 92.

APÊNDICE A – ARTIGO NO FORMATO SBC

Este apêndice reproduz, na íntegra, o artigo científico elaborado a partir deste Trabalho de Conclusão de Curso, redigido conforme as *Instruções para autores de conferências da SBC* (Sociedade Brasileira de Computação). O artigo sintetiza a motivação, a arquitetura, a implementação e os resultados de validação da plataforma CryptoMint em formato condensado, adequado à submissão em anais de conferências e eventos científicos da área. As páginas a seguir são apresentadas em sua diagramação original SBC.

Desenvolvimento de uma Plataforma Web3 para Criação e Comercialização de *Non-Fungible Tokens* em *Blockchain*

Conrado Zeitune Ruas

Departamento de Informática e Estatística (INE)
Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88.040-900 – Florianópolis – SC – Brasil

conrado.ruas@grad.ufsc.br

Abstract. The growth of the Web3 ecosystem and the consolidation of Non-Fungible Tokens (NFTs) as a mechanism for verifiable digital ownership have exposed structural limitations in existing NFT marketplaces: centralized asset custody, opaque rarity assignment during minting, and evasion of creator royalties in secondary sales. This paper presents CryptoMint, a decentralized Web3 platform for creating and trading NFTs on Ethereum. The system comprises Solidity smart contracts (non-custodial marketplace, verifiable-randomness minting via commit-reveal and Fisher-Yates, and automatic ERC-2981 royalties), a subgraph on The Graph for event indexing, and a Next.js frontend. Validation combined a Foundry test suite, static analysis with Slither, and a comparative gas and security analysis against OpenSea Seaport v1.6 and Rarible Protocol v2.

Resumo. O crescimento do ecossistema Web3 e a consolidação dos *Non-Fungible Tokens* (NFTs) como mecanismo de posse digital verificável evidenciaram limitações estruturais dos *marketplaces* de NFTs existentes: custódia centralizada de ativos, opacidade na atribuição de raridades no *mint* e evasão de royalties de criadores em revendas. Este artigo apresenta o CryptoMint, uma plataforma Web3 descentralizada para criação e comercialização de NFTs na Ethereum. O sistema reúne contratos inteligentes em Solidity (*marketplace* não custodiante, *mint* com aleatoriedade verificável via *commit-reveal* e Fisher-Yates, e royalties ERC-2981 automáticos), um *subgraph* no The Graph para indexação de eventos e um *frontend* em Next.js. A validação combinou uma suíte de testes Foundry, análise estática com Slither e análise comparativa de *gas* e segurança com o OpenSea Seaport v1.6 e o Rarible Protocol v2.

1. Introdução

A terceira geração da web — referida como Web3 — propõe uma internet descentralizada, na qual o controle sobre dados e ativos é devolvido aos usuários por meio de redes *blockchain* públicas [O'Reilly 2007, Swan 2015]. Nesse contexto, os *Non-Fungible Tokens* (NFTs), padronizados na Ethereum pelo ERC-721

[Entriiken et al. 2018], consolidaram-se como o principal mecanismo para representar a posse de ativos digitais únicos de forma verificável *on-chain* [Nadini et al. 2021, Wang et al. 2021]. A comercialização desses ativos ocorre, majoritariamente, por meio de *marketplaces* — plataformas que intermedeiam a listagem, a compra e a oferta de NFTs.

Apesar da maturidade do ecossistema, os *marketplaces* existentes apresentam três limitações estruturais recorrentes. A primeira é a **custódia centralizada de ativos**: plataformas que retêm os NFTs dos usuários durante a listagem expõem esses ativos ao risco de falhas técnicas, decisões regulatórias ou comprometimentos de segurança. A segunda é a **falta de transparência na atribuição de raridades** durante o *mint* de coleções generativas, cujo sorteio, quando realizado em servidores centralizados, é impossível de auditar publicamente, abrindo espaço para manipulação. A terceira é a **evasão de royalties**: como o ERC-721 não define pagamentos ao criador, *marketplaces* que ignoram o padrão ERC-2981 [Saito e Riley 2020] podem simplesmente não repassar royalties em vendas.

Este trabalho tem por objetivo desenvolver o **CryptoMint**, uma plataforma Web3 descentralizada para criação e comercialização de NFTs na *blockchain* Ethereum que enfrenta diretamente esses três problemas, por meio de: (i) um *marketplace* não custodiante, no qual os ativos permanecem sob controle do proprietário; (ii) um mecanismo de aleatoriedade verificável *on-chain* para o *mint*; e (iii) o pagamento automático e obrigatório de royalties. O restante do artigo está organizado como segue: a Seção 2 revisa os fundamentos técnicos e os trabalhos relacionados; a Seção 3 descreve a arquitetura e a implementação do sistema; a Seção 4 apresenta os resultados de validação; e a Seção 5 conclui o trabalho.

2. Fundamentação e Trabalhos Relacionados

A tecnologia *blockchain* foi introduzida por [Nakamoto 2008] como um registro distribuído, imutável e resistente à censura, mantido por consenso entre nós de uma rede peer-to-peer. A Ethereum [Wood 2014] generalizou o conceito ao introduzir uma máquina virtual (EVM) capaz de executar **contratos inteligentes** — programas determinísticos armazenados *on-chain* cujo código, uma vez implantado, é publicamente auditável e executado sem intermediários [Szabo 1994, Antonopoulos e Wood 2018]. Contratos inteligentes, contudo, manipulam ativos de valor e constituem uma superfície de ataque sensível, tornando padrões de segurança como *Checks-Effects-Interactions* e proteção contra reentrada requisitos essenciais [Atzei et al. 2017, Bartoletti e Pompianu 2017].

O padrão ERC-721 [Entriiken et al. 2018] define a interface mínima para *tokens* não fungíveis, enquanto o ERC-2981 [Saito e Riley 2020] padroniza a sinalização de royalties. O armazenamento dos metadados e das imagens associadas aos NFTs apoia-se tipicamente no InterPlanetary File System (IPFS) [Benet 2014], um sistema de arquivos distribuído e endereçado por conteúdo. Para viabilizar consultas eficientes sobre o histórico de eventos emitidos pelos contratos, o protocolo The Graph [The Graph 2021] permite indexar tais eventos em *subgraphs* consultáveis via GraphQL. No lado do cliente, aplicações descentralizadas (DApps) integram-se às *blockchains* por meio de bibliotecas Web3 e de *frameworks* web modernos como o React [React 2013] e o Next.js [Next.js 2024].

Entre os *marketplaces* consolidados, o **OpenSea Seaport** e o **Rarible Protocol** são referências. Ambos adotam o modelo de *ordens assinadas off-chain* (EIP-712), no qual a criação de uma listagem tem custo de *gas* praticamente nulo, ao preço de delegar o armazenamento das ordens a uma infraestrutura *off-chain* proprietária. O CryptoMint diferencia-se ao manter todo o estado das listagens *on-chain*, priorizando a transparência e a verificabilidade integral sobre a economia de *gas* — uma escolha coerente com o caráter acadêmico do trabalho, discutida na Seção 4.

3. Desenvolvimento do Sistema

O CryptoMint adota uma arquitetura em três camadas: (1) contratos inteligentes em Solidity 0.8.20, desenvolvidos com a *toolchain* Foundry [Foundry 2024] e as bibliotecas OpenZeppelin v5 [OpenZeppelin 2023]; (2) um *subgraph* no The Graph que indexa os eventos emitidos pelos contratos; e (3) uma aplicação *frontend* em Next.js 16, React 19 e TypeScript, integrada a carteiras Web3 via *wagmi* e *viem*.

3.1. Camada de Contratos Inteligentes

A camada *on-chain* é composta por três contratos principais. O NFT CollectionFactoryV2 atua como fábrica que implanta novas coleções a partir de parâmetros fornecidos pelo criador (nome, símbolo, descrição, imagem, *supply* máximo, preço de *mint* e *contractURI*). O NFTCollectionV2, derivado do ERC-721 da OpenZeppelin, é responsável pelo *mint* de NFTs com **aleatoriedade verificável**. O NFTMarketplace implementa a lógica de comercialização.

O **mecanismo de aleatoriedade** combina um esquema *commit-reveal* com o algoritmo de embaralhamento de Fisher-Yates [Fisher e Yates 1938, Knuth 1997]. No momento do *deploy* da coleção, o criador registra o *hash* de uma semente secreta (*mintSeedCommitment*). A atribuição de identificadores a cada *token* é derivada, no instante do *mint*, de fontes de entropia *on-chain* — notadamente o *blockhash* do bloco anterior —, o que impede que o resultado seja previsto ou manipulado pelo criador. Posteriormente, a função *revealMintSeed* permite tornar pública a semente originalmente comprometida, viabilizando a **auditoria** de que o compromisso registrado correspondia, de fato, ao valor revelado. Dessa forma, o processo de distribuição de raridades torna-se publicamente verificável, mitigando o problema de opacidade descrito na Introdução.

O NFTMarketplace adota o modelo **não custodiante**: os NFTs permanecem na carteira do proprietário durante toda a listagem, e o contrato opera apenas com uma aprovação *setApprovalForAll*, executando a transferência exclusivamente no momento de uma venda efetivada. O contrato oferece listagens a preço fixo e um sistema de ofertas com ETH mantido em *escrow*, e distribui automaticamente os **royalties** ao criador via ERC-2981 em toda venda mediada, aplicando um teto de 10% para proteger o comprador de valores abusivos. A segurança apoia-se no padrão *Checks-Effects-Interactions* e no modificador *nonReentrant* da OpenZeppelin. Como otimização de *gas*, as *structs* *Listing* e *Offer* foram **empacotadas** de modo a ocupar exatamente dois *slots* de armazenamento cada, evitando escritas a frio adicionais de 20.000 *gas*.

3.2. Camadas de Indexação e Frontend

O *subgraph* escuta os eventos emitidos pelos contratos (criação de coleções, *mints*, listagens, vendas e ofertas) e os materializa em um esquema GraphQL, viabilizando consultas complexas sobre o estado do *marketplace* sem a necessidade de varrer a *blockchain* a cada requisição. O *frontend* adota uma política de dados ***subgraph-first***: as leituras de catálogo e histórico são servidas pelo *subgraph*, enquanto operações financeiras críticas recorrem a um *fallback* autoritativo via RPC diretamente ao contrato, garantindo consistência no momento da confirmação de transações. As imagens e os metadados das coleções são armazenados no IPFS, com o processo de *upload* intermediado pelo servidor da aplicação, que realiza a fixação (*pin*) do conteúdo.

4. Resultados e Validação

A validação do sistema foi conduzida em três frentes: testes automatizados, análise estática de segurança e análise comparativa de *gas* e arquitetura.

4.1. Testes e Análise de Segurança

A suíte de testes dos contratos foi implementada em Solidity nativo com o *framework* Foundry/Forge [Foundry 2024], explorando *cheatcodes* da EVM para simular passagem de tempo, identidade de remetente e saldos. O contrato `NFTMarketplace` concentra mais de 40 casos de teste cobrindo os fluxos de sucesso e os caminhos de erro de todas as funções externas. Os testes de **segurança** incluem dois contratos atacantes (`ReentrancyAttacker` e `ReentrancyCancelAttacker`) que tentam explorar reentrada durante o recebimento de ETH; ambos são verificados como revertendo corretamente, confirmando a eficácia das proteções. Testes de *fuzz* aplicados ao cálculo de taxas e royalties verificam o invariante de conservação do valor total da venda entre *marketplace*, vendedor e receptor de royalty. A suíte do `NFTCollectionV2` confirma a unicidade das URIs atribuídas e a imutabilidade da `contractURI` após o *deploy*. A camada de *frontend* foi validada com o Vitest, cobrindo os *hooks* críticos, como o *proxy* do *subgraph* com *circuit-breaker* e tratamento de *rate-limit*.

Complementarmente, o código dos contratos foi submetido à análise estática com a ferramenta **Slither** [Slither 2018, Feist et al. 2019], utilizada para detectar padrões de vulnerabilidade conhecidos, e a revisão manual de código orientada pelos padrões de segurança de contratos Solidity [Atzei et al. 2017].

4.2. Análise Comparativa de Gas e Arquitetura

A Tabela 4.2 apresenta a comparação de custo de *gas* entre o CryptoMint e o OpenSea Seaport v1.6, tomado como referência por dispor de medições públicas. A diferença mais expressiva está na criação de listagens: o CryptoMint armazena cada listagem *on-chain* (≈ 50.000 *gas*, duas operações `SSTORE`), enquanto o Seaport utiliza ordens assinadas *off-chain* (EIP-712), com custo de listagem essencialmente nulo. Trata-se de uma escolha de projeto deliberada: manter o estado *on-chain* torna o sistema consultável diretamente via RPC, sem dependência de infraestrutura *off-chain* proprietária, em coerência com os objetivos de transparência e verificabilidade integral do trabalho.

Tabela 1. Comparativo de custo de gas por operação: CryptoMint vs. Seaport v1.6

Operação	CryptoMint	Seaport v1.6
Criar listagem	≈50.000 gas	≈0 (assinatura <i>off-chain</i>)
Compra simples	≈100.000 gas	≈70.000–90.000 gas
Fazer oferta	≈50.000 gas	≈0 (WETH sem <i>escrow</i>)

Em uma perspectiva arquitetural mais ampla, tanto o Seaport quanto o Rarible Protocol v2 adotam o modelo de ordens assinadas *off-chain*; o Rarible distingue-se ainda pela modularidade (contratos `Exchange`, `AssetMatcher` e `RoyaltiesProvider` separados) e por um *proxy* atualizável (UUPS). Em contraste, o CryptoMint adota deliberadamente uma arquitetura **monolítica e imutável**, sacrificando extensibilidade e atualização em favor de uma superfície de ataque mínima e de auditabilidade integral. Enquanto o Rarible impõe a dedução de royalties no nível do `Exchange`, o CryptoMint a realiza via ERC-2981 com teto de proteção ao comprador.

4.3. Alcance dos Objetivos

Os resultados demonstram que o CryptoMint atende aos objetivos propostos, implementando com qualidade de produção os três mecanismos centrais: custódia não custodiante, aleatoriedade verificável e automação de royalties. O sistema opera integralmente na rede de testes Ethereum Sepolia. As limitações reconhecidas — ausência de leilões *on-chain*, de listagens ERC-1155 e de ofertas em *tokens* ERC-20 — decorrem de uma escolha de escopo que priorizou qualidade técnica e cobertura de testes sobre amplitude de funcionalidades.

5. Conclusão

Este trabalho apresentou o CryptoMint, uma plataforma Web3 descentralizada para criação e comercialização de NFTs na Ethereum, concebida para enfrentar três problemas estruturais dos *marketplaces* existentes: a custódia centralizada, a opacidade da aleatoriedade no *mint* e a evasão de royalties. A solução articula contratos inteligentes seguros, um *subgraph* de indexação e um *frontend* moderno, validados por uma suíte de testes abrangente, por análise estática de segurança e por uma análise comparativa com protocolos consolidados do mercado. Como contribuição, o sistema constitui uma referência técnica de arquitetura descentralizada, transparente e auditável para o ecossistema Ethereum. Entre os trabalhos futuros, destacam-se a incorporação de leilões *on-chain*, o suporte a *tokens* ERC-1155 e ERC-20, e a integração de uma fonte de aleatoriedade externa (*oracle*) para reforçar as garantias de imprevisibilidade no *mint*. O código-fonte completo do projeto está disponível publicamente no repositório indicado no Apêndice correspondente desta monografia.

Referências

- [Atzei et al. 2017] Atzei, N., Bartoletti, M. and Cimoli, T. (2017). A Survey of Attacks on Ethereum Smart Contracts (SoK). In *Principles of Security and Trust*, pages 164–186. Springer.

- [Antonopoulos e Wood 2018] Antonopoulos, A. M. and Wood, G. (2018). *Mastering Ethereum: Building Smart Contracts and DApps*. O'Reilly Media.
- [Bartoletti e Pompianu 2017] Bartoletti, M. and Pompianu, L. (2017). An Empirical Analysis of Smart Contracts: Platforms, Applications, and Design Patterns. In *Financial Cryptography and Data Security*, pages 494–509. Springer.
- [Benet 2014] Benet, J. (2014). IPFS – Content Addressed, Versioned, P2P File System. *arXiv preprint arXiv:1407.3561*.
- [Entriken et al. 2018] Entriken, W., Shirley, D., Evans, J. and Sachs, N. (2018). EIP-721: Non-Fungible Token Standard. Ethereum Improvement Proposals. <https://eips.ethereum.org/EIPS/eip-721>.
- [Feist et al. 2019] Feist, J., Grieco, G. and Groce, A. (2019). Slither: A Static Analysis Framework for Smart Contracts. In *IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, pages 8–15.
- [Fisher e Yates 1938] Fisher, R. A. and Yates, F. (1938). *Statistical Tables for Biological, Agricultural and Medical Research*. Oliver and Boyd, London.
- [Foundry 2024] Foundry (2024). Foundry Book: A Portable and Modular Toolkit for Ethereum Application Development. <https://book.getfoundry.sh/>.
- [Knuth 1997] Knuth, D. E. (1997). *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley, 3rd edition.
- [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):20902.
- [Nakamoto 2008] Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. <https://bitcoin.org/bitcoin.pdf>.
- [Next.js 2024] Vercel (2024). Next.js Documentation. <https://nextjs.org/docs>.
- [OpenZeppelin 2023] OpenZeppelin (2023). OpenZeppelin Contracts: A Library for Secure Smart Contract Development. <https://docs.openzeppelin.com/contracts/>.
- [O'Reilly 2007] O'Reilly, T. (2007). What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. *Communications & Strategies*, (65):17–37.
- [React 2013] Meta (2013). React: A JavaScript Library for Building User Interfaces. <https://react.dev/>.
- [Saito e Riley 2020] Saito, Z. and Riley, N. (2020). EIP-2981: NFT Royalty Standard. Ethereum Improvement Proposals. <https://eips.ethereum.org/EIPS/eip-2981>.

- [Slither 2018] Trail of Bits (2018). Slither: The Solidity Source Analyzer. <https://github.com/crytic/slither>.
- [Swan 2015] Swan, M. (2015). *Blockchain: Blueprint for a New Economy*. O'Reilly Media.
- [Szabo 1994] Szabo, N. (1994). Smart Contracts. <https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html>.
- [The Graph 2021] The Graph (2021). The Graph: A Decentralized Protocol for Indexing and Querying Blockchain Data. <https://thegraph.com/docs/>.
- [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*.
- [Wood 2014] Wood, G. (2014). Ethereum: A Secure Decentralised Generalised Transaction Ledger. *Ethereum Project Yellow Paper*.

APÊNDICE B – REPOSITÓRIO DO CÓDIGO-FONTE DO PROJETO

O código-fonte completo da plataforma CryptoMint — abrangendo os contratos inteligentes em Solidity (com a respectiva suíte de testes Foundry), o *subgraph* de indexação para o protocolo The Graph e a aplicação *frontend* em Next.js — está disponível publicamente no repositório Git institucional da Universidade Federal de Santa Catarina, hospedado na plataforma GitLab do Departamento de Informática e Estatística (INE), no seguinte endereço:

[<https://codigos.ufsc.br/c.z.ruas/crypto-mint-tcc>](https://codigos.ufsc.br/c.z.ruas/crypto-mint-tcc)

O repositório reúne a estrutura de diretórios do projeto, os arquivos de configuração de *build* e *deploy*, a documentação técnica complementar e o histórico de *commits* do desenvolvimento, permitindo a reprodução integral do ambiente e a auditoria do código descrito ao longo desta monografia.