Uma plataforma para gestão de finanças descentralizadas

Matheus Oliveira de Sousa Pereira[1], **Fábio Aiub Sperotto**[1]

[1] Instituto Federal de Santa Catarina (IFSC) - Campus Lages
R. Heitor Villa Lobos, 225 - Sao Francisco, 88506-400- Lages – SC – Brasil

matheus.op20@ifsc.edu.br, fabio.sperotto@ifsc.edu.br

Abstract. *The rise of Decentralized Finance (DeFi) represents a revolution in the way individuals interact with financial products and services. However, DeFi platforms pose significant barriers, especially for novice users, in terms of operational complexity, security, financial planning, and accessibility. This study aims to overcome these barriers by developing an integrated financial planning platform for DeFi users. The methodology includes descriptive research, qualitative analysis of market data, and the development of a user-friendly platform with robust security mechanisms. The expected outcome is a solution that simplifies DeFi access, enhances security, and improves investment experience.*

Resumo. *A ascensão das Finanças Descentralizadas (do inglês Decentrilized Finance ou DeFi) representa uma revolução na forma como os indivíduos interagem com produtos e serviços financeiros. No entanto, as plataformas DeFi apresentam barreiras significativas, especialmente para usuários novatos, em termos de complexidade operacional, segurança, planejamento financeiro e acessibilidade. Este estudo visa superar essas barreiras desenvolvendo uma plataforma integrada de planejamento financeiro para usuários de DeFi. A metodologia inclui pesquisa descritiva, análise qualitativa de dados de mercado e o desenvolvimento de uma plataforma amigável ao usuário com mecanismos robustos de segurança. O resultado esperado é uma solução que simplifica o acesso ao DeFi, aprimorando a segurança e a experiência de investimento.*

1. Introducão

No cenário financeiro contemporâneo, a ascensão das Finanças Descentralizadas (DeFi) representa uma revolução na forma como indivíduos interagem com produtos financeiros e serviços. DeFi constitui um ecossistema composto por plataformas, protocolos e aplicações que oferecem serviços financeiros, como empréstimos, trocas de ativos, poupança e seguros, de forma descentralizada. Diferentemente das finanças tradicionais, baseadas em bancos e intermediários centralizados, as soluções DeFi utilizam contratos inteligentes, programas autônomos que executam automaticamente transações pré-definidas para mediar operações financeiras de forma direta, transparente e acessível.

As principais características das plataformas DeFi incluem descentralização, que permite que transações sejam realizadas diretamente entre os usuários, sem a necessidade de intermediários e todas as operações são registradas em *blockchains* públicos e podem ser auditadas por qualquer pessoa. Além disso, a acessibilidade torna os serviços financeiros disponíveis para qualquer indivíduo com acesso à internet, independentemente de localização ou vínculos com instituições financeiras tradicionais. A automação proporcionada pelos contratos inteligentes garante que as transações sejam confiáveis, executadas automaticamente de acordo com condições previamente estabelecidas.

Entre as formas de utilização mais comuns no ecossistema DeFi estão as trocas descentralizadas, que facilitam a negociação direta de ativos digitais entre os participantes por meio de *pools* de liquidez, eliminando a necessidade de corretoras tradicionais. Estratégias como *staking* e *farming* de liquidez permitem que usuários depositem seus ativos em contratos inteligentes para ajudar a validar transações ou fornecer liquidez, recebendo recompensas financeiras em troca.

Dois indicadores destacam o crescimento acentuado que o DeFi já experimentou em um período relativamente curto de tempo Schueffel (2021): primeiro, a capitalização de mercado das moedas digitais em DeFi, que está em aproximadamente 109 bilhões de dólares americanos (CoinGecko, 2024; CoinMarketCap, 2024; Cap, 2024). De acordo com o site Llama (2024), o valor total dos ativos atualmente empregados nos sistemas DeFi é estimado em 96 bilhões de dólares americanos.

A ascensão das DeFi trouxe consigo um nível de complexidade operacional que pode ser intimidador para muitos usuários. A necessidade de interagir com contratos inteligentes, entender a funcionalidade de *farms* de liquidez e calcular recompensas e riscos são barreiras substantivas. *Farms* de liquidez são essenciais no ecossistema DeFi, pois fornecem a liquidez necessária para a execução de trocas de ativos sem a necessidade de intermediários tradicionais, facilitando um mercado mais eficiente e acessível. Estudos sugerem que a interface do usuário e a experiência de operar nessas plataformas podem ser determinantes para sua adoção em massa (Zetzsche et al., 2020; Schär, 2021). A complexidade não apenas limita a entrada de novos usuários, mas também aumenta o risco de erros operacionais que podem levar a perdas financeiras significativas.

A segurança é uma preocupação primária no espaço DeFi. O anonimato e a falta de regulação tornam o ambiente propício a atividades fraudulentas, incluindo a promoção de contratos inteligentes maliciosos Wang et al. (2020). A necessidade de uma plataforma que ofereça mecanismos de segurança robustos e transparentes fica clara ao analisarmos a ascendência dos casos de *hacks* e fraudes neste ecossistema. Propor uma solução que permita ao usuário verificar facilmente a segurança dos contratos inteligentes que ele interage se torna então crítico.

Nos sistemas centralizados, há intermediários que garantem a segurança e regulação das transações, oferecendo suporte em casos de falhas. Já no ecossistema descentralizado (DeFi), a ausência de intermediários aumenta a autonomia do usuário, mas também eleva os riscos, como ataques cibernéticos e fraudes, pois não há uma entidade responsável pela segurança. Assim, os investimentos são de inteira responsabilidade do usuário, não havendo garantias de reparação em caso de problemas com contratos ou perdas financeiras Wang et al. (2020). Apesar de algumas plataformas oferecerem seguros contra falhas ou ataques, a ausência de regulação limita a aplicação de leis de proteção ao consumidor.

Recentemente, um relatório da Immunefi (2024) revelou que no primeiro trimestre de 2024, o total de perdas no Web3 (a terceira geração da internet, caracterizada por ser descentralizada e baseada em *blockchain*) devido a *hacks* e golpes alcançou a cifra de US $336.3 milhões em 61 incidentes específicos. Destes incidentes, US $321.6 milhões foram resultantes de *hacks*, enquanto US $14.7 milhões foram atribuídos a fraudes Fadilpašić (2024). Esta soma representa uma redução de 23.1% em relação ao mesmo período do ano anterior, quando *hackers* e fraudadores conseguiram subtrair US $437.5 milhões. É importante notar que a maior parte dessas perdas ocorreu somente em janeiro, quando mais de US $133 milhões foram roubados.

O planejamento financeiro em ambientes DeFi é complexo devido à volatilidade dos

ativos e à diversidade de estratégias de investimento disponíveis, desde *staking*[1] até *farming* de liquidez[2] Gudgeon et al. (2020). Uma plataforma que ofereça ferramentas de planejamento financeiro adaptadas ao ecossistema DeFi, como simulações com valores fictícios, *dashboards* integrados e índices relevantes, pode mitigar parte dessa complexidade, tornando o investimento em DeFi mais acessível e seguro.

Os desafios da complexidade, segurança, planejamento financeiro e acessibilidade representam obstáculos significativos para os usuários no universo das Finanças Descentralizadas (DeFi). Este projeto tem como meta superar tais barreiras, procurando não somente simplificar o acesso para novatos no ecossistema DeFi, mas também aprimorar a segurança e enriquecer a experiência de investimento para a comunidade como um todo. Para alcançar este objetivo, são estabelecidos os seguintes objetivos específicos:

- Compreender os conceitos e processos que englobam o ecossistema DeFi e de suas integrações.
- Pesquisar sobre ferramentas de integração com o ecossistema DeFi com flexibilidade de uso pelo usuário.
- Compreender e implementar mecanismos de segurança que possam avaliar processos ou estruturas de investimentos na área.
- Identificar aspectos de interface que possam auxiliar e instruir as operações financeiras para o usuário.
- Projetar e implementar uma solução que tenha em sua composição os elementos que possam fornecer uma plataforma integrada de planejamento financeiro para os usuários.

O trabalho foi estruturado em quatro etapas principais. Inicialmente, foi conduzida uma pesquisa descritiva em publicações relevantes sobre DeFi, criptomoedas e modelos de sistemas integrados a investimentos, visando compreender os conceitos fundamentais. Em seguida, foram realizadas análises qualitativas dos dados de mercado disponíveis publicamente, buscando identificar padrões e práticas correntes. Posteriormente, foi dedicada uma etapa ao desenvolvimento de uma plataforma que integre funcionalidades relacionadas à segurança, facilidade de uso e planejamento financeiro no contexto DeFi. Finalmente, a avaliação dos resultados foi realizada com base em uma validação prática e detalhada da plataforma desenvolvida. Essa validação envolveu a comparação com uma *pool* de liquidez real, utilizando dados coletados ao longo de um período específico.

Sob o aspecto metodológico, em relação à natureza, o trabalho se classifica como uma pesquisa aplicada, pois busca gerar conhecimento para aplicação prática voltada para a solução de problemas específicos no campo das Finanças Descentralizadas (DeFi). No critério dos procedimentos, adota-se uma abordagem descritiva e exploratória, utilizando métodos qualitativos e quantitativos. A pesquisa descritiva visa descrever as características dos fenômenos estudados, enquanto a pesquisa exploratória tem como objetivo proporcionar maior familiaridade com o problema, com vistas a torná-lo mais explícito.

Em relação aos objetivos, a pesquisa é classificada como descritiva, exploratória e explicativa. A abordagem descritiva foi utilizada para identificar e detalhar os elementos que compõem o ecossistema DeFi, suas funcionalidades e desafios. A abordagem exploratória ajudou a compreender as percepções e experiências dos usuários com plataformas DeFi. Fi-

[1] *Staking* refere-se ao processo de bloquear ativos em uma *blockchain* para ajudar a validar transações e receber recompensas como contrapartida.

[2] *Farming* de liquidez é a prática de fornecer ativos a *pools* de liquidez em plataformas DeFi para facilitar trocas de *tokens* e receber recompensas, geralmente na forma de taxas de transação ou novos *tokens*.

nalmente, a pesquisa explicativa buscou identificar relações de causa e efeito entre as variáveis estudadas, particularmente em termos de usabilidade, segurança e planejamento financeiro.

Além desta seção introdutória, este artigo está organizado em quatro seções principais. Na Seção 2, são abordados os fundamentos teóricos das Finanças Descentralizadas, detalhando conceitos essenciais e tecnologias subjacentes. A Seção 3 foca no desenvolvimento da plataforma proposta, descrevendo sua arquitetura, funcionalidades e etapas de implementação. A Seção 4 apresenta os resultados da plataforma, com ênfase nas melhorias em segurança, usabilidade e planejamento financeiro. Por fim, a Seção 5 discute as conclusões do estudo e sugere direções para trabalhos futuros, destacando as contribuições da pesquisa e possíveis evoluções da plataforma.

2. Referencial Teórico

Esta seção apresenta temas que embasam a plataforma proposta, sendo dividida em cinco subseções. A subseção 2.1 aborda as Finanças Descentralizadas (DeFi), enquanto a subseção 2.2 é dedicada ao *Farming* e *Stakings* dentro deste ecossistema. Na subseção 2.3, são discutidos os aspectos relacionados à Garantia de Qualidade e Usabilidade de Sistemas. A subseção 2.4 apresenta uma análise de trabalhos relacionados ao tema.

2.1. Finanças Descentralizadas (DeFi)

Finanças descentralizadas é um termo amplo para todos os produtos e serviços financeiros construídos em cima de *blockchains* públicos de código aberto. Diversas definições de finanças descentralizadas, ou DeFi, surgiram na literatura. Em termos simples, finanças descentralizadas é qualquer infraestrutura financeira que utiliza a tecnologia *blockchain* para oferecer serviços financeiros sem envolver intermediários. Meegan (2020) e Caldarelli e Ellul (2021) define finanças descentralizadas como a transformação de produtos financeiros tradicionais em produtos que operam sem um intermediário por meio de contratos inteligentes em um *blockchain*. Popescu (2020) define finanças descentralizadas como um ecossistema de aplicações financeiras que são desenvolvidas e habilitadas usando a tecnologia *blockchain* e de registro distribuído.

Um mercado DeFi é um mercado onde produtos financeiros oferecidos por meio de aplicações descentralizadas são negociados. Nos mercados DeFi, todas as transações são transparentes, incluindo os termos, condições e obrigações do provedor de serviços financeiros ou do cliente. Vários elementos são necessários para um mercado DeFi, tais como: contratos inteligentes, protocolos de software DeFi, aplicações descentralizadas (dApps), plataformas de finanças descentralizadas, bolsas descentralizadas (DEXs) e plataformas de empréstimo de finanças descentralizadas (De Meijer, 2021). Contratos inteligentes são programas de computador executados em uma *blockchain* que controla ativos digitais e automatiza os termos de acordo entre compradores, vendedores ou emprestadores e mutuários[3] (De Meijer, 2021).

Protocolos de software de finanças descentralizadas em *blockchains* são padrões e regras escritos para governar tarefas ou atividades específicas na *blockchain*. Isso permite que compradores, vendedores, emprestadores e mutuários interajam entre si de forma direta (De Meijer, 2021). Aplicações descentralizadas (dApps) são aplicações que oferecem serviços simples focados no consumidor. Elas podem ser usadas para negociação de ativos cripto, empréstimos, tomada de empréstimos, poupança, pagamentos e negociação de derivativos.

[3]Mutuários são indivíduos ou entidades que recebem fundos emprestados sob determinadas condições e se comprometem a devolvê-los posteriormente.

Uma plataforma de finanças descentralizadas é uma interface financeira voltada para o consumidor que requer tecnologia *blockchain* e apoiadores de cripto para operar (De Meijer, 2021).

No modelo ilustrado pela Figura 1, adaptada de (Alamsyah et al., 2024), os protocolos DeFi operam com base na interação entre investidores, usuários, serviços financeiros e um tesouro central. Investidores comprometem recursos financeiros ao protocolo, recebendo rendimentos (*yields*) provenientes das taxas pagas pelos usuários e da valorização dos ativos geridos. Usuários acessam serviços financeiros como empréstimos e negociação de ativos, bloqueando garantias financeiras que permanecem vinculadas ao protocolo enquanto os serviços são utilizados. As taxas recolhidas pelos usuários são administradas pelo Tesouro DeFi, que também distribui os rendimentos aos investidores. Contratos inteligentes garantem a execução transparente e automatizada das operações.

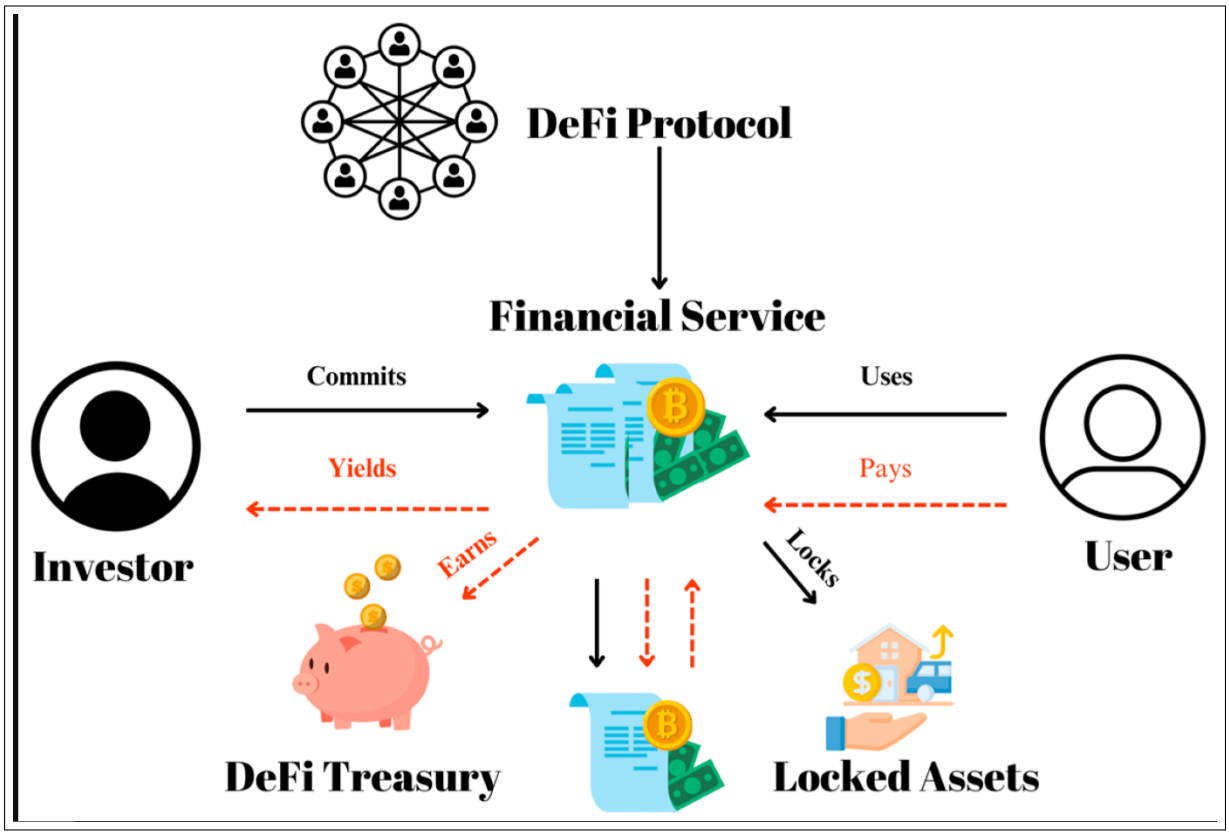

Figura 1. Modelo de funcionamento de um protocolo DeFi.

Bolsas descentralizadas (DEXs) são bolsas que permitem a negociação de ativos digitais sem nenhum controlador centralizado. As DEXs visam substituir as funções de formação de mercado e custódia das bolsas com um algoritmo que ajusta preços e executa negociações com base na liquidez disponível (De Meijer, 2021). Plataformas de empréstimo de finanças descentralizadas são plataformas que permitem aos detentores de criptomoedas emprestar uma grande quantidade de fundos instantaneamente e anonimamente para mutuários, desde que eles possam fornecer garantia suficiente para depositar em um contrato inteligente e liquidar o empréstimo dentro de um prazo acordado De Meijer (2021).

Existem produtos, plataformas e ferramentas diversas nos mercados DeFi. Os produtos proeminentes de gestão de liquidez DeFi incluem Yearn e Curve. As principais bolsas DeFi incluem Binance, dYdX, 1inch e Bancor. Alguns produtos e plataformas de gestão de ativos DeFi incluem Balancer, DeFi Saver, InstaDapp, Zerion e Zapper. Plataformas de seguro DeFi

incluem Nexus Mutual, Opyn e Opium Finance.

O financiamento descentralizado possui diversos benefícios, como a ampliação da inclusão financeira, o incentivo à inovação de forma irrestrita, a eliminação da necessidade de intermediários e a garantia da imutabilidade das transações. Além disso, ele resiste à censura, assegura regras iguais para todos os participantes, permite que as transações sejam auditadas por qualquer pessoa com acesso à internet, torna as transações transfronteiriças mais baratas e promove a intermediação financeira sem a necessidade de confiança (Ozcan, 2021; Macaskill, 2021; Chen e Bellavitis, 2020; Yavin e Reardon, 2021). No contexto do financiamento descentralizado, essa ausência de necessidade de confiança significa que os processos dependem exclusivamente da tecnologia, especialmente dos contratos inteligentes, que asseguram a execução de transações de forma automática e transparente. Dessa forma, os participantes não precisam confiar uns nos outros ou em intermediários centralizados, pois todas as regras e condições estão incorporadas e imutavelmente registradas na *blockchain*.

No que diz respeito à inclusão financeira, o financiamento pode proporcionar acesso ao crédito, permitindo que qualquer pessoa obtenha um empréstimo. Isso beneficia pequenas empresas e indivíduos sem histórico de crédito, pois não utiliza pontuação de crédito e não requer o cumprimento dos requisitos de conhecimento do cliente (Salami, 2020). O principal benefício é que este tipo de sistema oferece uma alternativa mais barata ao sistema financeiro tradicional, pois tudo é construído na *blockchain*, eliminando cobranças por serviços bancários (Salami, 2020).

O financiamento descentralizado vem com uma série de riscos. Em primeiro lugar, há o risco de execução em contratos inteligentes. Isso pode surgir de erros de codificação ao formar contratos financeiros inteligentes. Erros de codificação podem criar vulnerabilidades que permitem a um atacante tornar o código inutilizável ou que fundos incorporados sejam roubados de um contrato financeiro inteligente. A implicação é que o protocolo de financiamento descentralizado é tão seguro quanto os códigos subjacentes aos contratos inteligentes (Schär, 2021). Em segundo lugar, há um alto risco de responsabilidade legal em contratos inteligentes. Isso ocorre porque o usuário médio de tecnologia, que não consegue ler um código de contrato inteligente ou avaliar sua segurança, pode ser induzido a assinar um contrato inteligente comprometido, expondo-se a riscos e responsabilidades legais imprevistas.

Em terceiro lugar, há risco de roubo de dados. Muitos protocolos e aplicações de financiamento descentralizado usam chaves administrativas para atualizar o contrato inteligente e realizar desligamentos de emergência. Se os detentores das chaves não criarem ou armazenarem suas chaves de forma segura, partes maliciosas podem roubar as chaves e comprometer o contrato inteligente. Há também o risco de interconectividade ou risco de dependência. Isso surge porque os protocolos de financiamento descentralizado permitem que vários contratos inteligentes e aplicações *blockchain* descentralizadas interajam entre si para oferecer novos serviços baseados em uma combinação dos existentes. Essas interações introduzem dependências significativas de tal forma que, se houver um problema com um contrato inteligente, isso pode ter consequências de grande alcance para várias aplicações de financiamento descentralizado em todo o ecossistema de financiamento descentralizado (Schär, 2021).

Um outro risco é que muitas aplicações de financiamento descentralizado dependem de dados externos. Quando um contrato inteligente depende de dados que não estão disponíveis na *blockchain*, os dados serão fornecidos por fontes de dados externas. Consequentemente, aplicações *blockchain* estarão expostas a problemas de qualidade e disponibilidade de dados quando as aplicações forem fortemente dependentes destas fontes externas (Schär, 2021; Ku-

mar et al., 2020). Finalmente, há uma alta propensão para atividades ilícitas ao usar aplicações de financiamento descentralizado. A natureza descentralizada dos sistemas de financiamento os torna atraentes para atividades ilícitas, atores com intenções desonestas podem usar aplicações de financiamento para lavagem de dinheiro ou promover atividades ilegais, conforme discutido por Schär (2021).

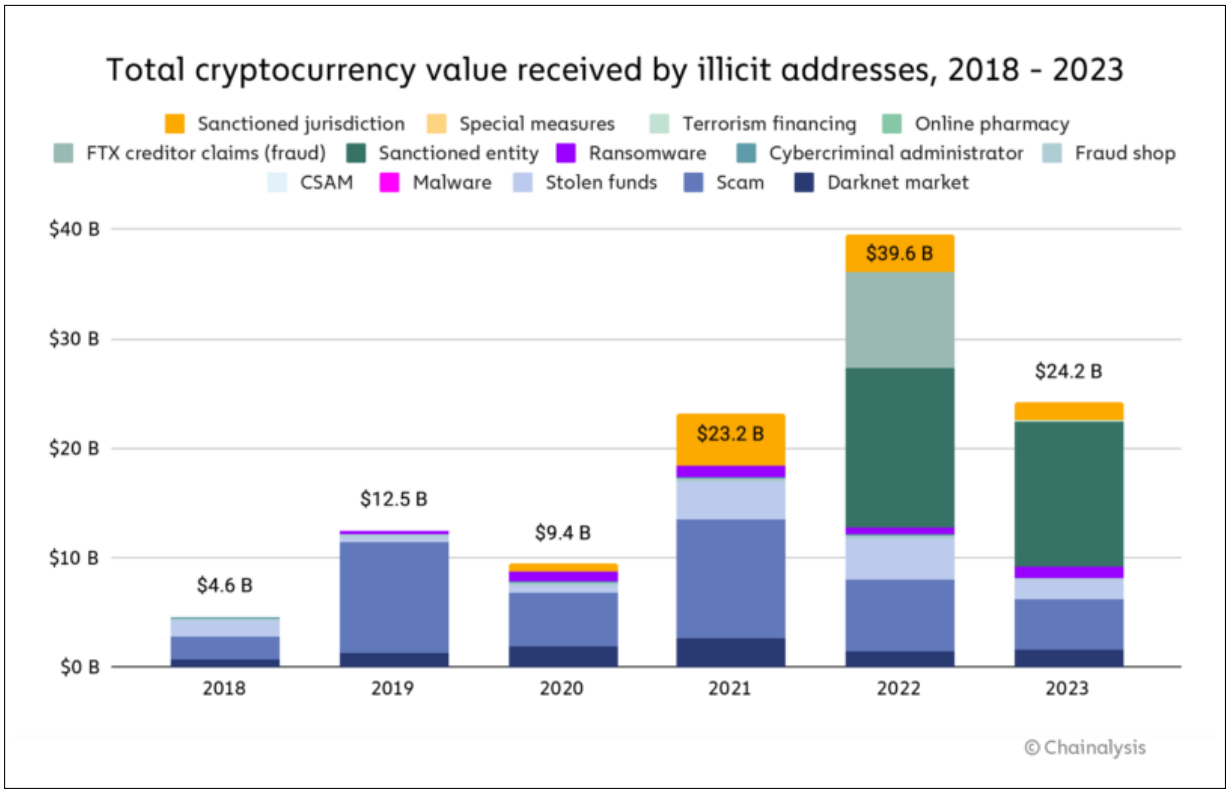

Figura 2. Valores totais recebidos por endereços ilícitos em criptomoedas entre 2018 e 2023, segmentados por categorias de atividades ilícitas (CoinDesk, 2023).

A figura 2 apresenta a evolução dos valores totais recebidos por endereços ilícitos em criptomoedas entre 2018 e 2023, segmentados por categorias de atividades ilegais. Observa-se um aumento acentuado no período, atingindo um pico de $39,6 bilhões de dólares em 2022, seguido por uma redução para $24,2 bilhões em 2023. Esses dados reforçam a necessidade de implementar medidas robustas de segurança para mitigar tais problemas no ecossistema DeFi (CoinDesk, 2023).

2.2. Farms e Stakes

Farms e *Stakes* são mecanismos fundamentais em Finanças Descentralizadas (DeFi), permitindo aos usuários maximizar seus rendimentos em criptomoedas. As *Farms*, ou fazendas de liquidez, envolvem a provisão de liquidez em *pools* em troca de recompensas em *tokens*. Já os *Stakes* referem-se ao bloqueio de criptomoedas em uma plataforma para apoiar operações da rede e ganhar juros. Esses processos são essenciais para a eficiência das plataformas DeFi, promovendo liquidez e segurança. A seguir, foi detalhado o funcionamento e as particularidades das *Farms* e *Stakes*, destacando seus benefícios e riscos.

As *Farms* na DeFi representam uma das principais inovações dentro do ecossistema *blockchain*, especialmente no que diz respeito à geração de rendimentos passivos. Essencialmente, *farms* são *pools* de liquidez onde os usuários podem depositar pares de *tokens*, como

ETH e USDT[4], para ganhar recompensas. Os provedores de liquidez (LPs) adicionam esses *tokens* em contratos inteligentes que facilitam as transações no mercado. Em troca, os LPs recebem uma parte das taxas de transação e, em muitos casos, *tokens* adicionais emitidos pelo protocolo. Esse processo é conhecido como "*yield farming*" [5] e tornou-se extremamente popular no mercado DeFi.

Um aspecto crítico do funcionamento dos *pools* de liquidez é o uso de Automated Market Makers (AMMs). Ao contrário das *exchanges* tradicionais que utilizam livros de ordens, os AMMs ajustam os preços dos ativos automaticamente com base na proporção de *tokens* nos *pools* (Tabtrader, 2023). Por exemplo, em um *pool* de ETH-DAI, se um usuário troca ETH por DAI, a quantidade de ETH no *pool* aumenta enquanto a quantidade de DAI diminui, alterando a proporção e, consequentemente, os preços. Conforme del Priore (2022), a constante responsável pelo mecanismos das AMMs é definida pela equação $Quantidade\ de\ ETH \times Quantidade\ de\ DAI = Constante$. Este mecanismo garante que cada operação de câmbio impacte o preço dos ativos dentro do *pool*.

Se um *pool* contém 10 ETH e 10.000 DAI ($10 \times 10.000 = 100.000$), e um usuário troca 1 ETH por DAI, o *pool* passa a ter 11 ETH. Para manter a constante 100.000, a nova quantidade de DAI será aproximadamente 9090,91 DAI ($11 \times 9090,91 = 100.000$). Isso ilustra como as trocas ajustam a proporção de *tokens* e, consequentemente, seus preços (del Priore, 2022).

Apesar dos atrativos rendimentos, as *Farms* na DeFi também apresentam riscos significativos que os investidores devem considerar. Um dos principais riscos é o de *impermanent loss*, que ocorre quando a proporção dos *tokens* depositados na *pool* de liquidez muda devido à volatilidade do mercado, resultando em perdas potenciais para os provedores de liquidez. Por exemplo, se o valor do ETH sobe significativamente no mercado externo, arbitradores trocarão DAI por ETH no *pool* a um preço mais baixo, causando uma diminuição na quantidade de ETH no *pool* e um aumento na quantidade de DAI. Como resultado, o valor dos ativos retirados pode ser menor do que se os *tokens* fossem simplesmente mantidos (*hold*) na carteira (del Priore, 2022).

Entender os AMMs e o *impermanent loss* é muito importante para quem deseja participar de *farms* de liquidez e maximizar seus rendimentos no ecossistema DeFi. Com as estratégias corretas, os investidores podem mitigar riscos e aproveitar as oportunidades oferecidas por essas inovações financeiras (del Priore, 2022).

Staking é uma maneira de colocar ativos de criptomoedas para trabalhar para ganhar rendimentos. O *staking* pode ser líquido, onde os ativos depositados podem ser retirados a qualquer momento, ou com *vesting*, onde o *staking* requer um bloqueio de tempo para ganhar os rendimentos da plataforma. Os usuários podem fazer *staking* de ativos nativos, como ETH ou MATIC,[6] para ganhar recompensas de *proof-of-stake* daquela *blockchain*.

2.3. Garantia de Qualidade e Usabilidade de Sistemas

A garantia de qualidade e a usabilidade de sistemas são pilares fundamentais no desenvolvimento de plataformas tecnológicas, especialmente em ambientes complexos como as Finanças

[4]ETH (Ethereum) é uma criptomoeda amplamente utilizada para transações e contratos inteligentes na *blockchain* Ethereum. USDT (*Tether*) é uma *stablecoin* atrelada ao valor do dólar americano, amplamente empregada em mercados de criptomoedas.

[5]O *yield farming* refere-se ao processo de utilizar *farms* para maximizar rendimentos passivos através de recompensas geradas pelo fornecimento de liquidez.

[6]MATIC é o token nativo da *blockchain* Polygon, uma solução de segunda camada que visa aumentar a escalabilidade e reduzir os custos de transação.

Descentralizadas (DeFi). Garantir que um sistema seja seguro e fácil de usar não só aumenta a satisfação dos usuários, mas também contribui significativamente para a redução de riscos e para a promoção da adoção em larga escala. Esta seção aborda a importância desses fatores, bem como os princípios e práticas recomendadas para alcançá-los.

A garantia de qualidade em sistemas de software refere-se ao conjunto de atividades planejadas e sistemáticas implementadas para assegurar que o sistema atenda aos requisitos especificados e funcione de maneira confiável e segura. No contexto das plataformas DeFi, a garantia de qualidade é muito importante e um diferencial, devido à natureza financeira e descentralizada dessas plataformas, erros e falhas podem levar a perdas financeiras significativas, comprometendo a confiança dos usuários (Galin, 2004).

A qualidade de um sistema é avaliada por meio de testes rigorosos, que incluem testes de funcionalidade, performance, segurança e usabilidade. A implementação de metodologias ágeis e práticas de DevOps também contribui para a melhoria contínua e a rápida resposta a problemas identificados. Adicionalmente, a auditoria de código e a verificação de contratos inteligentes são essenciais para prevenir vulnerabilidades e fraudes (Forsgren et al., 2018).

Usabilidade refere-se à eficácia, eficiência e satisfação com que os usuários podem alcançar objetivos específicos em um sistema particular. Segundo Krug (2014), a usabilidade deve se concentrar em tornar a interface do usuário intuitiva e fácil de navegar. Krug destaca a importância de reduzir a carga cognitiva dos usuários, permitindo que eles realizem tarefas com o mínimo de esforço mental.

Em Norman (2019) é identificado quatro princípios fundamentais do design centrado no usuário que são aplicáveis ao desenvolvimento de sistemas DeFi:

- Visibilidade: Os usuários devem sempre ser informados sobre o que está acontecendo no sistema através de *feedback* apropriado e dentro de um tempo razoável.
- Compatibilidade: O sistema deve falar a língua dos usuários, com palavras, frases e conceitos familiares, ao invés de termos técnicos.
- Controle do Usuário e Liberdade: Os usuários frequentemente escolhem funções do sistema por engano e precisarão de uma "saída de emergência" claramente marcada para sair do estado indesejado sem ter que passar por um diálogo extenso.
- Identificar aspectos de interface que possam auxiliar e instruir as operações financeiras para o usuário.
- Consistência e Padrões: Os usuários não devem ter que se perguntar se diferentes palavras, situações ou ações significam a mesma coisa. As convenções da plataforma devem ser seguidas.

A segurança de sistemas DeFi é uma preocupação primordial. A Open Web Application Security Project OWASP (2024) lista diversas vulnerabilidades comuns em aplicações web que também são relevantes para plataformas DeFi. Entre as mais críticas estão as injeções de código, falhas de autenticação e gerenciamento de sessões e exposições de dados sensíveis. Implementar boas práticas de desenvolvimento seguro, como a validação de entrada de dados e a criptografia de informações sensíveis, é essencial para mitigar esses riscos.

2.4. Trabalhos Relacionados

Nesta seção, serão apresentados os trabalhos ou sistemas existentes relacionados ao objetivo do presente estudo. A análise de trabalhos relacionados é essencial para compreender as principais abordagens já exploradas, seus sucessos e limitações. Esta compreensão fornece uma base sólida para a criação de uma solução inovadora e eficaz.

Os trabalhos relacionados foram selecionados com base nos critérios de gerenciamento de liquidez, projeções de investimento, monitoramento em tempo real dos lucros, integração com várias aplicações DeFi, guias, auto saque e dados detalhados do investimento, incluindo dados de segurança. Esses critérios foram estabelecidos com o propósito de avaliar como as soluções existentes enfrentam as necessidades das finanças descentralizadas, além de identificar possíveis lacunas e oportunidades para o desenvolvimento de inovações que agreguem valor ao setor.

Gerenciamento de Liquidez refere-se à forma como as plataformas permitem que os usuários gerenciem seus recursos em *pools* de liquidez, seja através de estratégias automáticas ou manuais. Projeções de investimento destacam a capacidade de prever retornos financeiros com base em métricas como o APY. Monitoramento em tempo real dos lucros refere-se à disponibilidade de *dashboards* que apresentam históricos detalhados de lucros diários, mensais e totais, permitindo uma visão clara do desempenho financeiro. Integração com várias aplicações DeFi analisa a flexibilidade das plataformas em trabalhar com diferentes protocolos e aplicações DeFi. Guias ou *Walkthroughs* são recursos que tornam a experiência mais acessível, especialmente para usuários novatos. Dados detalhados do investimento, consideram a profundidade das informações fornecidas sobre as moedas da *farm*, enquanto auto saque foca na automação do processo de retirada de fundos.

Metrix Finance é uma plataforma que oferece estratégias avançadas de gerenciamento de liquidez e automação para provedores institucionais e individuais. Ela combina ferramentas robustas de monitoramento e projeções detalhadas, permitindo aos usuários acessar dados em tempo real sobre o desempenho de seus ativos. No entanto, não apresenta *dashboards* detalhados com históricos de lucros ou funcionalidades como *walkthroughs* e auto saque, o que limita sua acessibilidade (Finance, 2024).

Orbit DeFi é voltada para automação de liquidez em protocolos como Uniswap v3. A plataforma oferece reequilíbrio automático e reinvestimento de taxas. Apesar disso, não fornece ferramentas para monitoramento em tempo real dos lucros ou integração com múltiplos protocolos DeFi, o que pode restringir sua aplicabilidade para usuários avançados (Orbit, 2024).

Teahouse Finance fornece estratégias de gerenciamento de ativos para investidores institucionais e individuais. Ela se diferencia por parcerias estratégicas e emissão de NFTs. Entretanto, não oferece funcionalidades como monitoramento em tempo real dos lucros, auto saque ou guias interativos, o que dificulta a análise de desempenho contínuo e a experiência de novos usuários (Teahouse, 2024).

Gecko Terminal é uma plataforma focada em fornecer monitoramento detalhado em tempo real de ativos financeiros e *pools* de liquidez. Embora ofereça dados valiosos para análise de mercado, ela não apresenta históricos de lucros nem funcionalidades como projeções de investimento ou auto saque, limitando seu uso para estratégias mais completas de investimento (Terminal, 2024).

Critério	Metrix Finance	Teahouse	Orbit DeFi	Gecko Terminal	Plataforma TCC
Gerenciamento de Liquidez	Sim	Sim	Sim	Não	Sim
Projeções de Investimento	Sim	Não	Não	Não	Sim
Monitoramento de lucros em tempo real	Sim	Não	Sim	Não	Sim
Integração com várias aplicações DeFi	Sim	Sim	Não	Sim	Sim
Walkthrough e Guias	Não	Não	Não	Não	Sim
Dados de segurança do investimento	Não	Não	Não	Sim	Sim
Auto Saque	Não	Não	Não	Não	Sim

Quadro 1. Comparação entre trabalhos relacionados e a plataforma do TCC.

A análise das plataformas existentes demonstra que, enquanto soluções como Metrix Finance e Gecko Terminal se destacam em monitoramento e dados em tempo real, há uma lacuna significativa em funcionalidades que melhorem a acessibilidade, automação e segurança para usuários com diferentes níveis de experiência. Esses pontos reforçam a necessidade de inovações que abordem essas limitações de maneira integrada.

3. Desenvolvimento

Esta seção é dedicada ao desenvolvimento do sistema e está dividida em seis partes. A subseção 3.1 descreve as tecnologias que foram utilizadas no projeto. A subseção 3.2 elabora os requisitos funcionais e não funcionais do sistema. A subseção 3.3 mostra a modelagem do banco de dados. A subseção 3.4 aborda os aspectos técnicos relacionados à programação, com foco na implementação de segurança para atender aos requisitos funcionais e não funcionais incluindo protótipos das interfaces. Por fim, a subseção 3.5 detalha a implementação do sistema, incluindo a organização do projeto e a arquitetura da aplicação.

3.1. Tecnologias

Esta seção apresenta as tecnologias utilizadas no desenvolvimento da aplicação, detalhando as características de cada uma e explicando como elas contribuem para a construção de um sistema robusto, escalável e eficiente. Além disso, serão discutidas as vantagens de cada tecnologia, demonstrando como suas funcionalidades específicas foram exploradas para atender aos requisitos do projeto e garantir a qualidade final do produto.

3.1.1. C#

Criada pela Microsoft em 2000, C# é uma linguagem de programação moderna e orientada a objetos, amplamente utilizada para o desenvolvimento de aplicações empresariais e web devido à sua robustez e versatilidade (Microsoft, 2024b). No contexto deste projeto, C# foi a principal linguagem utilizada para o desenvolvimento do backend do sistema, permitindo a implementação da lógica de negócios e garantindo a eficiência e a segurança nas operações.

3.1.2. Angular 15

Angular, lançado em 2010 e desenvolvido pela Google, é um framework JavaScript de código aberto utilizado para criar aplicações web dinâmicas e responsivas (Google, 2024). A versão Angular 15 foi utilizada neste projeto para o desenvolvimento do frontend, permitindo a construção de interfaces de usuário modernas e interativas, que garantam uma experiência de usuário fluida e eficiente.

3.1.3. Selenium

Selenium, lançado em 2004, é uma ferramenta de teste automatizado para aplicações web, amplamente utilizada para criar scripts de teste que simulam interações do usuário (Conservancy, 2024). No contexto deste trabalho, Selenium foi utilizado para a automação de testes de interface do usuário, garantindo que todas as funcionalidades web sejam testadas de maneira rigorosa antes de serem implementadas em produção.

3.1.4. SQL Server

SQL Server é um sistema de gerenciamento de banco de dados relacional desenvolvido pela Microsoft, conhecido por sua robustez, escalabilidade e segurança (Microsoft, 2024c). Neste projeto, SQL Server foi responsável pelo gerenciamento e armazenamento de dados, atuando como o banco de dados principal do sistema e garantindo a integridade e a eficiência nas operações de consulta e armazenamento.

3.1.5. Azure DevOps

Azure DevOps, lançado em 2018 pela Microsoft, é uma plataforma de gerenciamento de projetos que integra ferramentas de CI/CD, permitindo automação de testes, *deploy* e facilitando a colaboração entre equipes de desenvolvimento (Microsoft, 2024a). Neste trabalho, Azure DevOps foi utilizado para gerenciar o ciclo de vida do desenvolvimento do software, automatizando o processo de integração contínua e entrega contínua, além de coordenar as atividades entre os membros da equipe.

3.1.6. Inline Manual

O Inline Manual é uma plataforma especializada em guias interativos e *walkthroughs* que permite criar instruções passo a passo diretamente na interface do usuário. Essa tecnologia é amplamente utilizada para melhorar a experiência do usuário em sistemas complexos, fornecendo

orientação em tempo real sobre o funcionamento de funcionalidades e explicando conceitos específicos (Manua, 2024). No contexto deste projeto, o Inline Manual foi integrado ao sistema para oferecer guias e explicações interativas nas telas e modais, ajudando os usuários a navegar pelas funcionalidades e entender conceitos financeiros de forma intuitiva. A ferramenta permite personalização e segmentação dos guias com base nas ações e perfil do usuário, oferecendo uma experiência de aprendizado adaptativa e contextualizada.

3.2. Requisitos do Sistema

Nesta subseção são elaborados os requisitos funcionais e não funcionais do sistema a ser desenvolvido. Os requisitos são apresentados no formato de fichas, sendo que cada requisito possui um número único (RF1, RF2, etc.) e uma descrição detalhada do que o sistema deverá fazer e a saída esperada.

3.2.1. Requisitos Funcionais

Os requisitos funcionais foram obtidos por meio de uma análise dos trabalhos relacionados. Este levantamento inicial de requisitos foi realizado com base na identificação de funcionalidades e limitações de soluções existentes no campo das Finanças Descentralizadas (DeFi). Os estudos e plataformas disponíveis no mercado serviram de referência para definir as necessidades do sistema.

- **RF1 - Cadastro e Autenticação de Usuários:** O sistema deve permitir o cadastro de novos usuários e a autenticação dos registrados, usando nome, e-mail, senha e confirmação de senha. Após um login bem-sucedido, o usuário é redirecionado ao *dashboard*.
- **RF2 - Gerenciamento de Portfólios DeFi:** O sistema deve possibilitar que os usuários adicionem, editem e removam investimento em seus portfólios de finanças descentralizadas. Cada investimento disponibilizado deve exibir:
  - Nome dos ativos (exemplo: BTC, ADA),
  - Quantidade investida em moeda de pagamento,
  - Moeda gerada na *Farm*,
  - Valor total em dólares,
  - APY (Annual Percentage Yield),
  - Lucro (*Profit*),
  - Opção para auto saque.
- **RF3 - Visualização de Dados de Mercado:** O sistema deve fornecer dados de mercado em tempo real, incluindo histórico de preços, gráficos de desempenho por ativo, correlação entre ativos, informações básicas (como data de lançamento e dados de *supply*) e métricas de segurança dos investimentos. A saída esperada são gráficos interativos e tabelas atualizadas que apresentem as informações de forma clara e acessível, permitindo ao usuário tomar decisões mais informadas e seguras.
- **RF4 - Configuração e Listagem de Auto Saques:** O sistema deve permitir o cadastro e a visualização das configurações de auto saques. As configurações incluem:
  - Frequência de saque,
  - Nome da Configuração,
  - Data da Próxima Execução,
  - Moeda para Saque,
  - Tipo de Saque (Pix, transferência bancária, etc.),

  - Data do Último Saque.

  Os usuários devem poder adicionar uma nova configuração de Auto Withdraw e visualizar as existentes em uma tabela organizada.
- **RF5 - *Dashboard* de Desempenho Financeiro:** O sistema deve exibir um *dashboard* financeiro para visualização e projeção do desempenho dos ativos no portfólio. Campos e elementos obrigatórios:
  - Saldo Atual,
  - Gráficos de Desempenho Anual, Mensal e Diário,
  - Botões de Navegação.
- **RF6 - *Walkthrough* e Explicações de Conceitos:** O sistema deve oferecer um guia interativo em todas as telas, com explicações sobre conceitos essenciais. Funcionalidades incluem:
  - Apresentação inicial,
  - Guias de tela com explicações contextuais para conceitos como APY, *Stake*, lucro e auto saque,
  - Assistente de conceitos e ferramentas

3.2.2. Requisitos Não Funcionais

Os requisitos não funcionais foram definidos a partir de uma análise das necessidades específicas do sistema e limitações de soluções similares já disponíveis. Estabelecidos durante a fase de planejamento, esses requisitos seguem as melhores práticas de desenvolvimento de software e levam em consideração aspectos de desempenho, segurança e experiência do usuário.

- **RNF1 - Desempenho:** O sistema deve responder às ações dos usuários em até 2 segundos.
- **RNF2 - Segurança:** O sistema deve proteger os dados dos usuários e transações, utilizando criptografia e outros mecanismos de segurança..
- **RNF3 - Escalabilidade:** O sistema deve ser capaz de suportar um aumento de usuários e transações sem prejudicar o desempenho.

3.3. Modelagem do Banco de Dados

Nesta subseção, é apresentada a modelagem do banco de dados utilizada no sistema. O banco de dados foi estruturado para suportar todas as funcionalidades descritas nos requisitos funcionais da subseção 3.2. A Figura 3 mostra, respectivamente, o diagrama conceitual das principais tabelas relacionadas às transações financeiras e ao gerenciamento de usuários e permissões. A seguir, cada uma das tabelas e seus relacionamentos são brevemente explicados.

As entidades apresentadas na Figura 3 serão descritas em termos dos seus nomes e associações. A entidade *Defis* armazena informações sobre as plataformas DeFi, incluindo o nome da plataforma, a taxa de retirada aplicada e a URL de *staking*. Cada plataforma registrada nesta tabela pode estar associada a múltiplas *farms*, que são representados na tabela *Stakes*. Esta última tabela contém detalhes específicos sobre as *farms* oferecidas pelas plataformas DeFi, como o título do *stake*, a moeda utilizada para *staking*, a moeda de pagamento e o rendimento anual esperado. Dessa forma, a relação entre as tabelas *Defis* e *Stakes* é essencial para representar as diferentes opções de investimento oferecidas pelas plataformas DeFi.

A tabela *UserStakes* desempenha um papel importante ao armazenar as *farms* realizadas pelos usuários. Inclui detalhes como a quantidade em dolares investido, o lucro gerado

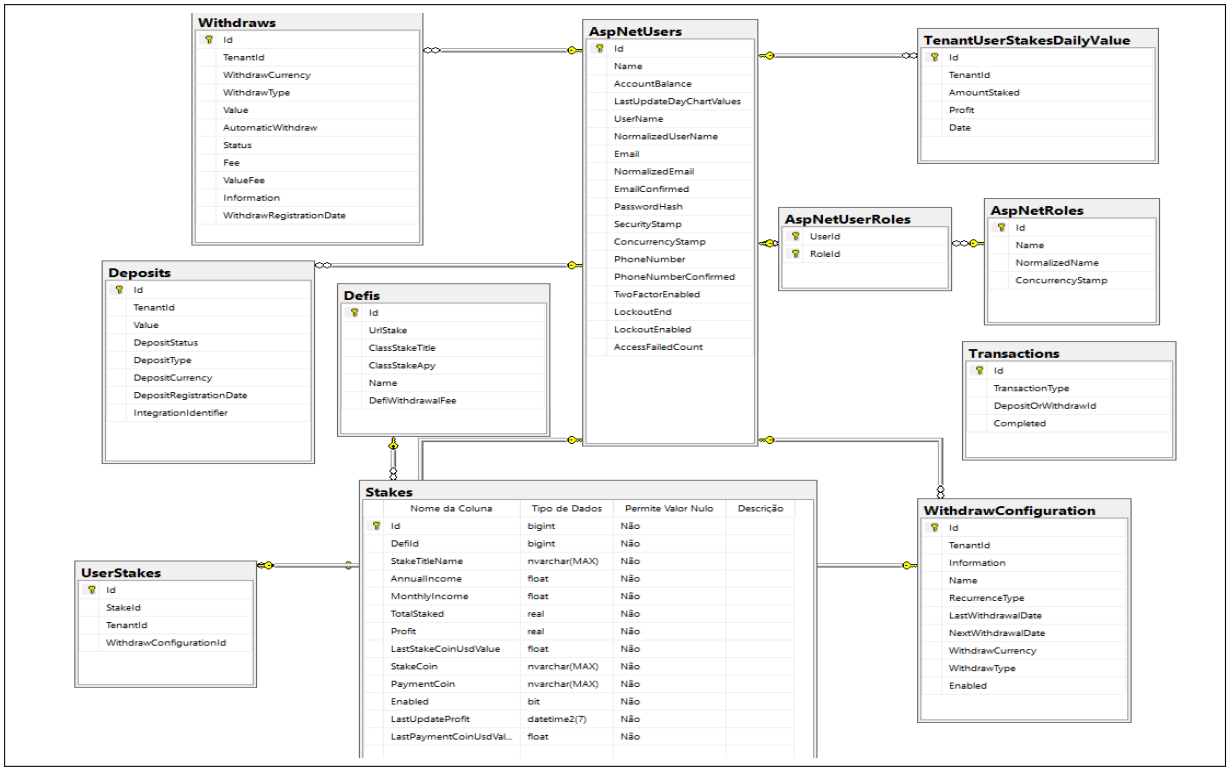

Figura 3. Modelo conceitual do banco de dados.

e as configurações de retirada automática. Essa associação é utilizada para acompanhar as atividades (adicionar saldo ou retirar saldo) de *farms* dos usuários dentro do sistema.

A tabela *Withdraws*, também presente na Figura 3, registra as retiradas de saldo realizadas pelos usuários, armazenando detalhes como o valor retirado, o tipo de retirada, a moeda envolvida e o status da transação. A configuração específica de retirada de cada usuário é armazenada na tabela *WithdrawConfiguration*, que define parâmetros como a recorrência das retiradas e as datas de última e próxima retirada. A tabela *WithdrawConfiguration* está relacionada à tabela *UserStakes*, o que permite a personalização das retiradas com base nas *farms* específicas dos usuários, assegurando que as operações de retirada sejam gerenciadas de forma eficaz e de acordo com as preferências individuais.

Além disso, os depósitos realizados pelos usuários são registrados na tabela *Deposits*, que armazena informações como o valor depositado, a moeda utilizada, o tipo de depósito e o status. A tabela *Transactions* documenta todas as transações financeiras realizadas no sistema, fornecendo detalhes sobre o tipo de transação, o ID do depósito ou retirada associado, e o status de conclusão.

A tabela TenantUserStakesDailyValue armazena os valores diários relacionados às *farms* dos usuários, incluindo a quantia em dolares investido e o lucro gerado, permitindo um monitoramento detalhado do desempenho dos investimentos ao longo do tempo.

A entidade AspNetUsers é a entidade central, armazenando as informações básicas dos usuários do sistema, como nome de usuário, email e senha. Esta tabela serve como referência principal para qualquer operação relacionada ao gerenciamento de usuários.

Associada à AspNetUsers, a tabela AspNetRoles armazena os diferentes papéis que um usuário pode ter dentro do sistema. Cada papel (*role*) é uma categoria que define diferentes

níveis de acesso e permissões, permitindo uma gestão eficaz dos direitos de acesso dos usuários.

A relação entre os usuários e seus papéis é fica na tabela AspNetUserRoles e estabelece um vínculo entre as tabelas AspNetUser e AspNetRoles, permitindo que um usuário seja associado a um ou mais papéis no sistema.

3.4. Implementação

Nesta subseção, é detalhada a implementação do sistema, incluindo a organização do projeto, a arquitetura da aplicação e a descrição das funcionalidades conforme os requisitos definidos na subseção 3.2.

A arquitetura adotada é a CQRS (*Command Query Responsibility Segregation*), que separa a lógica de leitura e escrita do sistema, atendendo aos requisitos não funcionais (RNF1, RNF3) (Betts et al., 2013). Os *Commands* são responsáveis pelas operações de escrita e atualização de dados, enquanto as *Queries* tratam das operações de leitura e consulta de dados, cada uma sendo manipulada por um *handler* específico. Essa estrutura melhora a eficiência e a segurança da aplicação, alinhando-se ao requisito de segurança (RNF2).

A separação entre operações de leitura e escrita permite o uso de bancos de dados diferentes, otimizados para cada tipo de operação, o que contribui significativamente para a escalabilidade do sistema. Por exemplo, um banco de dados transacional pode ser dedicado às operações de escrita, enquanto um banco de dados otimizado para consultas ou um banco NoSQL[7], pode ser utilizado para leitura, proporcionando respostas mais rápidas e maior capacidade de atender a múltiplas requisições simultâneas. Essa abordagem assegura que o sistema seja capaz de lidar com cargas crescentes de usuários e operações, atendendo ao RNF1 e RNF3.

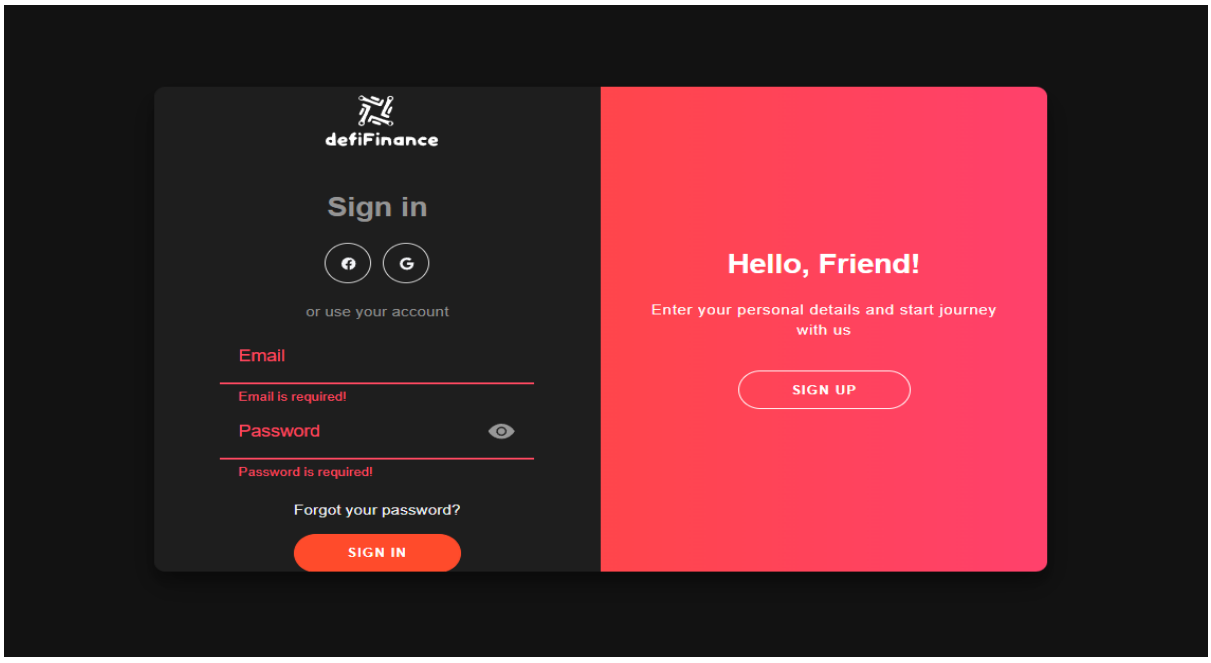

Figura 4. Wireframe da tela de login.

A interface de login visualizada na Figura 4 contém campos para e-mail e senha, e um botão de login. Ela inclui links para cadastro de novos usuários e recuperação de senha, aten-

[7]Bancos de dados NoSQL são sistemas de gerenciamento de dados não relacionais, projetados para lidar com grandes volumes de dados, alta velocidade de acesso e estrutura flexível, frequentemente usados em aplicações distribuídas.

dendo ao RF1. Os componentes desta interface incluem campos de texto para e-mail e senha, além de botões para login, um link para a tela de cadastro no sistema e outro para recuperação de senha (“Esqueci minha senha”). A interface foi projetada para ser funcional e acessível, garantindo que os usuários possam navegar facilmente entre as opções disponíveis. Validações foram implementadas nos campos de entrada para assegurar que os dados fornecidos estejam no formato correto, reduzindo erros e melhorando a experiência do usuário.

No *front-end*, a autenticação é gerida com componentes Angular e serviços que controlam *tokens* e protegem rotas. O principal componente é o AuthorizedGuard, que verifica a validade do *token* de autenticação e protege as rotas privadas. Caso o *token* seja inválido, o usuário é redirecionado para a página de login, garantindo acesso restrito conforme os requisitos de autenticação (RF1) e segurança (RNF2).

```
export class AuthorizedGuard implements CanActivate {
    constructor(private router: Router, private tokenService:
        TokenService) {}

    canActivate() {
        const token = this.tokenService.getToken();
        if (!this.tokenService.isValidToken(token)) {
            this.router.navigate(['/account/sign-in']);
            return false;
        }
        return true;
    }
}
```

Implementação 1. AuthorizedGuard: Exemplo de código para proteção de rotas no Angular.

No *back-end*, a autenticação é realizada por comandos e *handlers* que verificam as credenciais e geram *tokens* JWT. A implementação 2 ilustra o código do *handler* de autenticação:

```
var user = await _userManager.FindByEmailAsync(request.Email);
if (user == null || !await _signInManager.CheckPasswordSignInAsync(user,
    request.Password, false))
    return _result.Fail("Credenciais invalidas");

var roles = await _userManager.GetRolesAsync(user);
return _result.Ok(GenerateToken(user, roles.FirstOrDefault()));


private AuthorizedTenantModelResult GenerateToken(IdentityUser user,
    string role) {
    var claims = new List<Claim> {
        new Claim(JwtRegisteredClaimNames.Sub, user.UserName),
        new Claim(ClaimTypes.Role, role)
    };

    var token = new JwtSecurityToken(claims: claims, expires:
        DateTime.UtcNow.AddHours(2), signingCredentials:
        _signingCredentials);
    return new AuthorizedTenantModelResult { AccessToken = new
        JwtSecurityTokenHandler().WriteToken(token) };
}
```

Implementação 2. AuthorizeTenantCommandHandler: Exemplo de *handler* para autenticação e geração de tokens JWT.

O código da implementação 3 mostra a configuração no ASP.NET Core que define as políticas de autenticação e autorização usando JWT, fortalecendo a segurança:

```
public static void AddIdentity(this IServiceCollection services,
    IConfiguration configuration)
{
    services.AddIdentity<IdentityUser, IdentityRole>(options =>
    {
        options.User.RequireUniqueEmail = true;
        options.Password.RequiredLength = 5;
    })
    .AddEntityFrameworkStores<ApplicationDbContext>()
    .AddDefaultTokenProviders();

    services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
        .AddJwtBearer(options =>
        {
            options.TokenValidationParameters = new
                TokenValidationParameters
            {
                ValidateIssuer = true,
                ValidateAudience = true,
                ValidateLifetime = true,
                IssuerSigningKey = new SymmetricSecurityKey(
                Encoding.UTF8.GetBytes(configuration["Jwt:Key"]))
            };
        });

    services.AddAuthorization();
}
```

Implementação 3. Configuração do serviço Identity para definir políticas de autenticação e autorização baseadas em JWT.

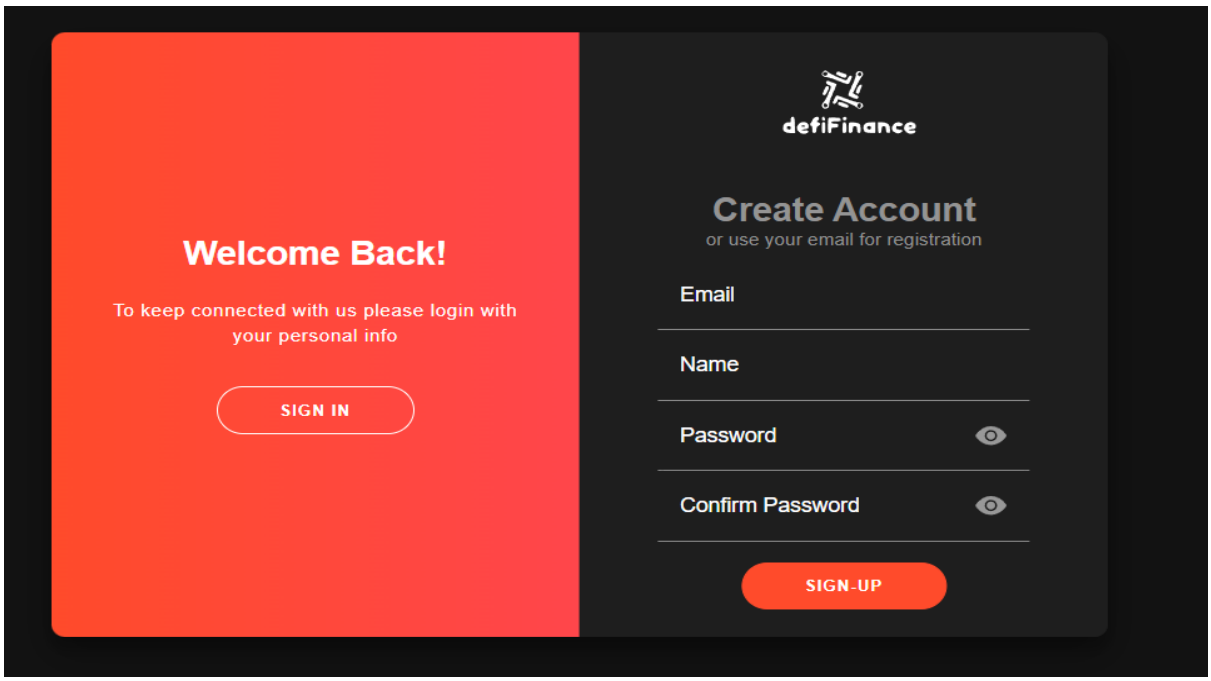

Figura 5. Wireframe da tela de cadastro.

A interface de cadastro visualizada na Figura 5 contém campos para e-mail, nome e senha, além de um botão de confirmação de cadastro. Ela inclui validações para assegurar que o usuário insira um nome com no mínimo 2 e no máximo 40 caracteres e confirme a senha, atendendo ao RF1. Os componentes desta interface incluem um campo de texto para e-mail, um campo de texto para nome e um componente de senha com confirmação, além de um botão para concluir o cadastro (“*Sign-Up*”).

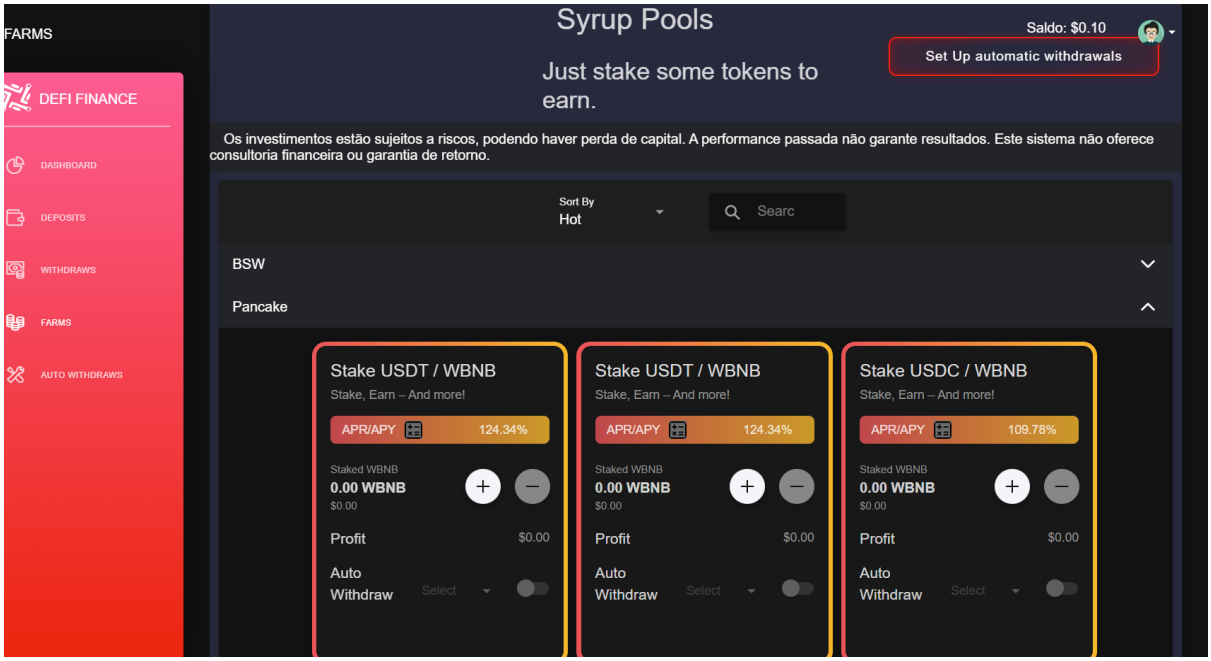

Figura 6. Wireframe da tela de cadastro de ativos.

A funcionalidade de Gerenciamento de Portfólios DeFi, visualizada na Figura 6, permite ao usuário adicionar, editar e remover ativos financeiros descentralizados, atendendo ao requisito RF2. Cada Farm no portfólio exibe dados, como moedas da Farm, quantidade em *Farm*, valor total em dólares, APY (*Annual Percentage Yield*), aviso sobre os investimentos exibidos na plataforma que não garantem resultados futuros. lucro acumulado e uma opção de auto-saque.

A funcionalidade de gerenciamento de portfólio DeFi (RF2) é implementada no *front-end* principalmente através do componente StakePage. Esta página inicializa dados essenciais, como informações de ativos DeFi (`getDefis()`), *Farms* dos usuários (`getUserStakes()`) e configurações de auto-saque (`getConfigurations()`), permitindo uma experiência de gerenciamento completa.

A página de *Farms* carrega informações das DeFi, *farms* do usuário e configurações de auto-withdraw. Os métodos `getDefis()`, `getUserStakes()` e `getConfigurations()` são executados no início para recuperar esses dados de maneira assíncrona.

```
ngOnInit() {
    this.getDefis();
    this.getUserStakes();
    this.getConfigurations();
}
```

Implementação 4. StakePage - Carga inicial dos dados.

O sistema exibe informações detalhadas de cada *farm*, incluindo o APR/APY que é a Taxa anual de retorno; Auto *withdraw* onde o usuário pode configurar saques automáticos selecionando uma opção no menu `ion-select`, com a configuração do `toggle` para habilitar ou desabilitar a funcionalidade; *Profit* com a Exibição do lucro acumulado em tempo real.

A estrutura com modais permite ao usuário adicionar ou remover saldo a um investimento disponível na página.

```
<ion-col class="col-5 ion-no-padding">
  <ion-fab-button size="small" id="open-modal"
      (click)="updateUserStakeStakedAmount(true)"
      [disabled]="accountBalance<=0">
       <ion-icon name="add"></ion-icon>
  </ion-fab-button>
  <ion-fab-button size="small"
      (click)="updateUserStakeStakedAmount(false)" [disabled]="userStake ?
      userStake.amountStaked<=0 : true" >
       <ion-icon name="remove"></ion-icon>
  </ion-fab-button>
</ion-col>
```

Implementação 5. CardStakeComponent - Interface de Farm.

Esse fluxo de criação e atualização de *farms* permite uma experiência integrada de gerenciamento de portfólio, onde o usuário pode visualizar e configurar facilmente seus investimentos, ajustando quantidades e definindo parâmetros de auto-saque conforme necessário. Esses elementos atendem ao requisito funcional RF2, permitindo uma gestão completa do portfólio de finanças descentralizadas e facilitando o acompanhamento dos retornos e lucros de cada ativo.

O AddUserStake (Implementação 6) implementa a lógica de ativação de uma nova *farm* quando o usuário não está com nenhum valor aplicado. O processo inicia com a validação de dados críticos, como o saldo da conta do usuário e a disponibilidade do ativo e, após, se o saldo do usuário for insuficiente para cobrir o valor da *farm* ou se o ativo estiver indisponível ou desativado, o sistema retorna mensagens de erro para cada caso de falha na validação (linhas 2 a 9 da implementação). Após passar pela validação, o *handler* atualiza o saldo do usuário, o valor total investido na *farm* e inicia uma transação de banco de dados para garantir que todas as mudanças sejam atômicas (a partir da linha 10 da implementação 6). Ao final, os dados são persistidos no banco e a transação é confirmada.

```
//Mapeamento e validacao do usuario e saldo
if (userStake.Tenant.AccountBalance < userStake.AmountStaked)
    return _result.Fail(BusinessErrors.AccountBalanceInsufficient
    .ToString());
//Verificacao de disponibilidade do stake
var stake = await _stakeRepository.GetBy(s => s.Id == userStake.StakeId,
    cancellationToken);
if (stake == null || !stake.Enabled)
    return _result.Fail(BusinessErrors.StakeUnavailable.ToString());
//Atualizacao do saldo e stake total
stake.TotalStaked += userStake.AmountStaked;
userStake.Tenant.AccountBalance -= userStake.AmountStaked;

await _base.UnitOfWork.BeginDatabaseTransactionAsync(cancellationToken);
await _base.UserManager.UpdateAsync(userStake.Tenant);
_stakeRepository.UpdateStake(stake);
await _base.UserStakeRepository.CreateUserStake(userStake,
    cancellationToken);
```

```
17
18  if (await
        _base.UnitOfWork.SaveChangesWithTransactionAsync(cancellationToken) ==
        0)
19      return _result.Fail(BusinessErrors.FailToCreateUserStake.ToString());
20
21  await _base.UnitOfWork.CommitDatabaseTransactionAsync(cancellationToken);
22  return _result.Ok(userStake);
```

Implementação 6. AddUserStakeCommandHandler - Criação de Stake.

O UpdateUserStakeConfiguration (Implementação 7) é responsável por atualizar a configuração de saque automático de uma *farm* já existente. Ele valida se o usuário e a *farm* existem e se o ativo está disponível. Caso positivo, o *handler* ajusta a configuração de saque automático, incluindo a ativação ou desativação da funcionalidade e atualiza a data de modificação da *farm*. Essas alterações são realizadas dentro de uma transação, assegurando que os dados estejam consistentes no banco de dados.

```
1   // Validacao do usuario e do stake
2   var tenant = await _base.UserManager.FindByNameAsync(request.Username);
3   if (tenant == null)
4       return _base.CommandResult.Fail("Erro ao atualizar UserStake de
            usuario, usuario nao encontrado.");
5
6   var databaseUserStake = await _base.UserStakeRepository.GetBy(
7       s => s.Id == request.UserStakeId && s.TenantId == tenant.Id,
            cancellationToken);
8
9   if (databaseUserStake == null)
10      return _base.CommandResult.Fail(BusinessErrors
11      .UserStakeNotFound.ToString());
12
13  // Atualizacao da configuracao de auto-saque
14  databaseUserStake.WithdrawConfigurationId =
        request.WithdrawConfigurationId;
15  databaseUserStake.AutoWithdrawEnabled = request.AutoWithdrawEnabled;
16  databaseUserStake.LastUpdate = DateTime.Now;
17
18  // Transacao segura para salvar a atualizacao
19  var result = await _base.UnitOfWork.ExecuteActionWithTransactionAsync(() =>
20  {
21      _base.UserStakeRepository.UpdateUserStake(databaseUserStake);
22      return Task.CompletedTask;
23  }, cancellationToken);
24
25  return result;
```

Implementação 7. UpdateUserStakeConfigurationCommandHandler - Atualização da configuracao de auto-saque da Farm.

Todas as alterações realizadas no sistema são envolvidas em transações para evitar inconsistências. Por exemplo, na atualização da configuração de auto-saque, conforme as linhas 19 a 23 no código da implementação 7. De maneira similar, durante a aplicação de investimento de um usuario a uma *farm*, na implementação 6 a transação é iniciada na linha 13, garantindo que todas as alterações, incluindo o saldo do usuário e o valor investido na *farm*, sejam realizadas de forma atômica. Caso a verificação na linha 18, não seja bem-sucedida, o

rollback implícito restaura o estado inicial do banco de dados, enquanto na linha 21 finaliza a transação quando todas as operações foram concluídas com sucesso.

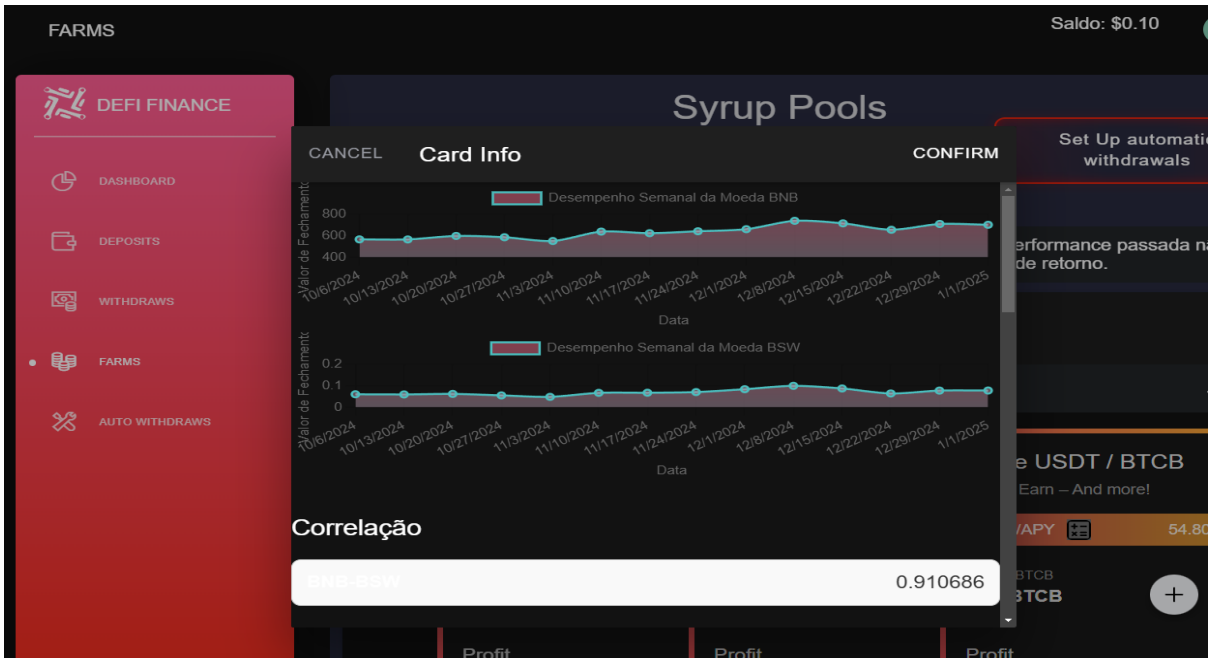

Figura 7. Wireframe da tela de visualização de dados de mercado.

A funcionalidade de Visualização de Dados de Mercado (Figura 7) permite acessar informações detalhadas sobre o desempenho de ativos de criptomoedas (RF3). Ao selecionar um ativo, um modal exibe gráficos interativos com histórico de preços, dados de *supply*, métricas de segurança e correlação entre preços, auxiliando na análise do portfólio.

A visualização de dados em tempo real (RF3) da *farm* é implementada no *front-end*, onde o usuário pode selecionar cada card de investimento para abrir uma modal com informações detalhadas sobre os ativos. Essa modal recebe uma lista de símbolos de criptomoedas do investimento e, ao ser inicializada, faz integrações com dois *endpoints* específicos da API da Bitget.[8] O *endpoint* `/spot/market/candles` fornece dados históricos de preços em intervalos configuráveis (como semanal) e o *endpoint* `/spot/market/tickers` retorna o preço atual de cada ativo.

Esses dados são processados para criar gráficos interativos, mostrando a variação de preço ao longo do tempo para cada ativo e facilitando o acompanhamento da performance individual. Além disso, a modal calcula a correlação entre os preços dos ativos exibidos, gerando uma matriz de correlação apresentada ao usuário.

Um exemplo do código que busca e organiza esses dados pode ser visto no método `fetchCandleData` (Implementação 8). Esse método acessa o *endpoint* `/spot/market/candles` para coletar o histórico de preços e, por meio dos parâmetros `granularity` e `startTime`, permite configurar o intervalo dos dados e o período inicial da série histórica. A granularidade ajusta a frequência dos dados, como diário ou semanal, enquanto o `startTime` especifica o período a ser analisado, aqui definido para os últimos três meses. Após coletar o histórico de preços, o método invoca `fetchCurrentPrice` para adicionar o preço atual, criando uma série temporal completa para cada ativo:

[8]Mais informações sobre as APIs da Bitget podem ser encontradas em `https://www.bitget.com/api-doc/common/intro`.

```
fetchCandleData(symbol: string, granularity: string) {
  const params = {
    symbol: symbol + "USDT",
    granularity: granularity,
    startTime: (Date.now() - 3 * 30 * 24 * 60 * 60 * 1000).toString(),
    endTime: Date.now().toString(),
  };

  this.http.get(this.apiUrl, { params }).subscribe(
    (response: any) => {
      let prices = response.data.map((candle: any) =>
          parseFloat(candle[1]));
      this.fetchCurrentPrice(symbol, (currentPrice) => {
        if (currentPrice) prices.push(currentPrice);
        this.updateChartData(symbol, prices);
      });
    }
  );
}
```

Implementação 8. Método para buscar dados históricos de velas (*candles*).

A Figura 8 apresenta o diagrama de sequência da funcionalidade da RF3, ilustrando o fluxo de interação entre os componentes e o usuário.

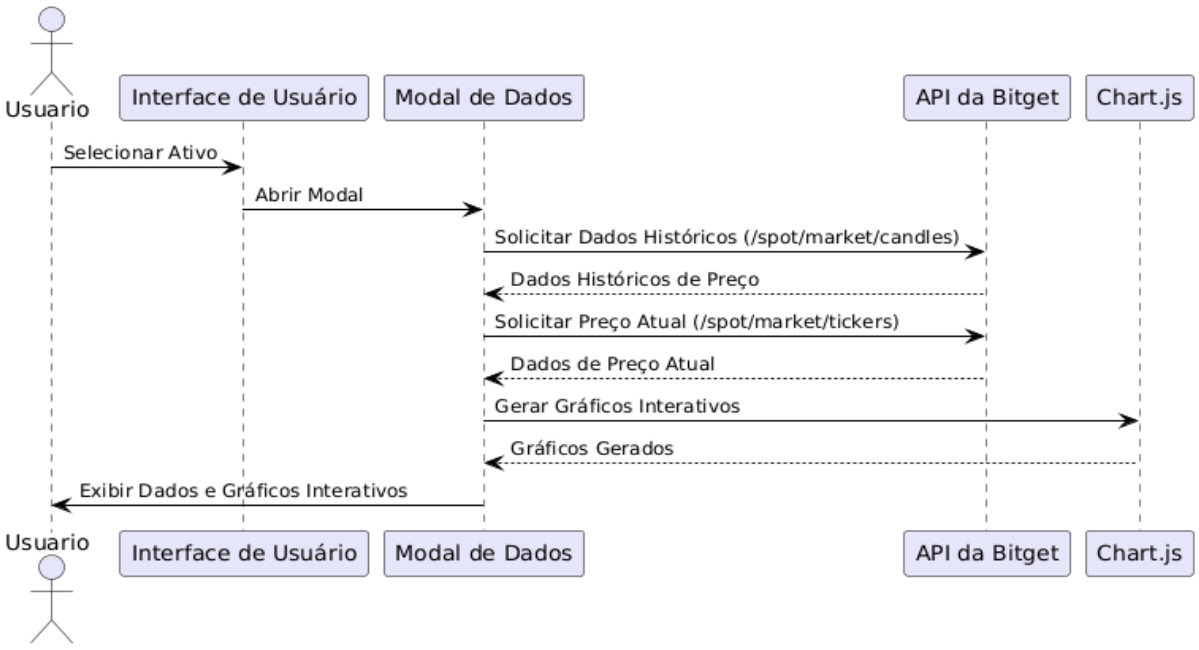

Figura 8. Diagrama de sequência da funcionalidade de visualização de dados em tempo real das *farms*(RF3).

A funcionalidade de Configuração e Listagem de Auto Saques (RF4) permite que o usuário cadastre e visualize configurações de saques automáticos com opções detalhadas. Na interface de saques automáticos, o usuário pode definir a frequência do saque (mensal, bimestral, semestral, ou anual), nomear a configuração, escolher a moeda, especificar o tipo de saque (Pix, transferência bancária, etc.), e ativar ou desativar a configuração. Esses dados são listados em uma tabela organizada, onde o usuário pode visualizar a data da próxima execução e a data do último saque realizado.

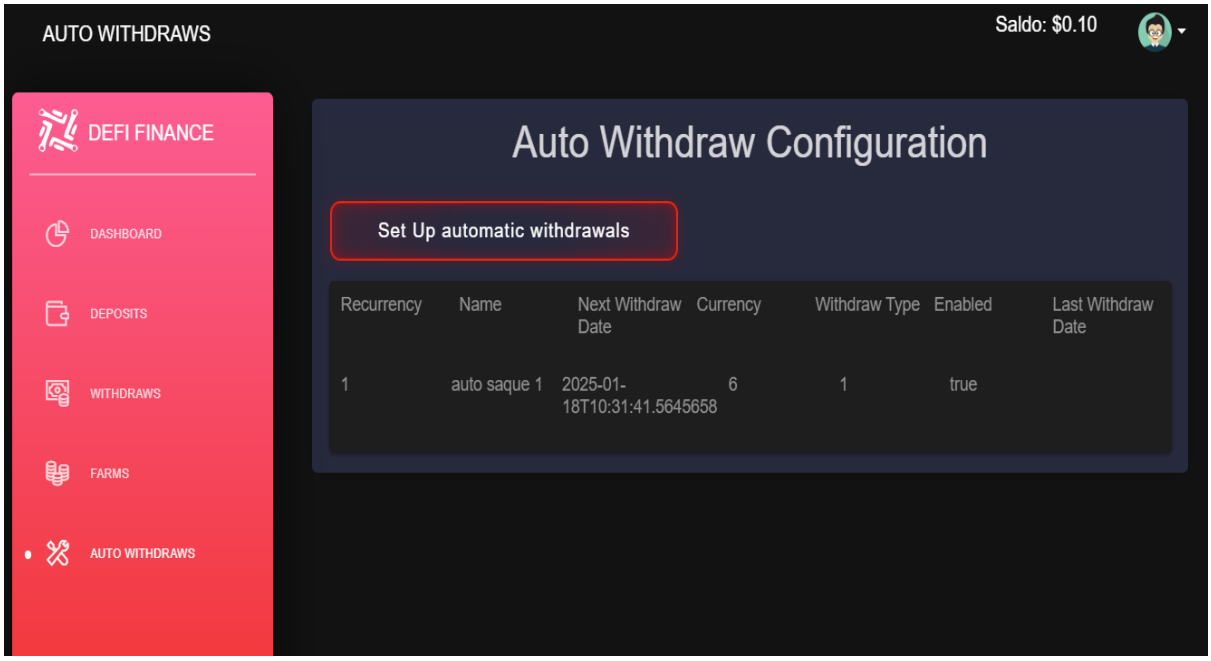

Figura 9. Wireframe da tela de configuração e listagem de auto saques.

A funcionalidade de Configuração e Listagem de Auto Saques (Figura 9) permite configurar e visualizar saques automáticos (RF4). A interface apresenta uma tabela com frequência, nome, moeda, tipo de saque, data do próximo saque e do último saque. O botão "Adicionar Auto Withdraw" abre um modal para definir os parâmetros do saque.

A tela de auto saques oferece o recurso "Adicionar Auto *Withdraw*" para configurar novos saques automáticos. Ao clicar, o sistema abre um modal para o cadastro de uma nova configuração, onde o usuário define os parâmetros do saque, como a frequência, o tipo e a moeda do saque.

Para implementar essa funcionalidade, o código utiliza o componente `AutoWithdrawPage` para exibir a lista de configurações e o componente `WithdrawConfigurationComponent` para abrir um modal que permite a criação de cada configuração de saque automático. O método `openWithdrawConfiguration` em `AutoWithdrawPage` exibe o modal de configuração (Implementação 9):

```
async openWithdrawConfiguration() {
  const modal = await this.modalCtrl.create({
    component: WithdrawConfigurationComponent,
  });
  await modal.present();

  const { data, role } = await modal.onWillDismiss();

  if (role === 'confirm' && data !== undefined) {
    this.configurations.push(data);
  }
}
```

Implementação 9. Método para abrir a configuração de auto saque.

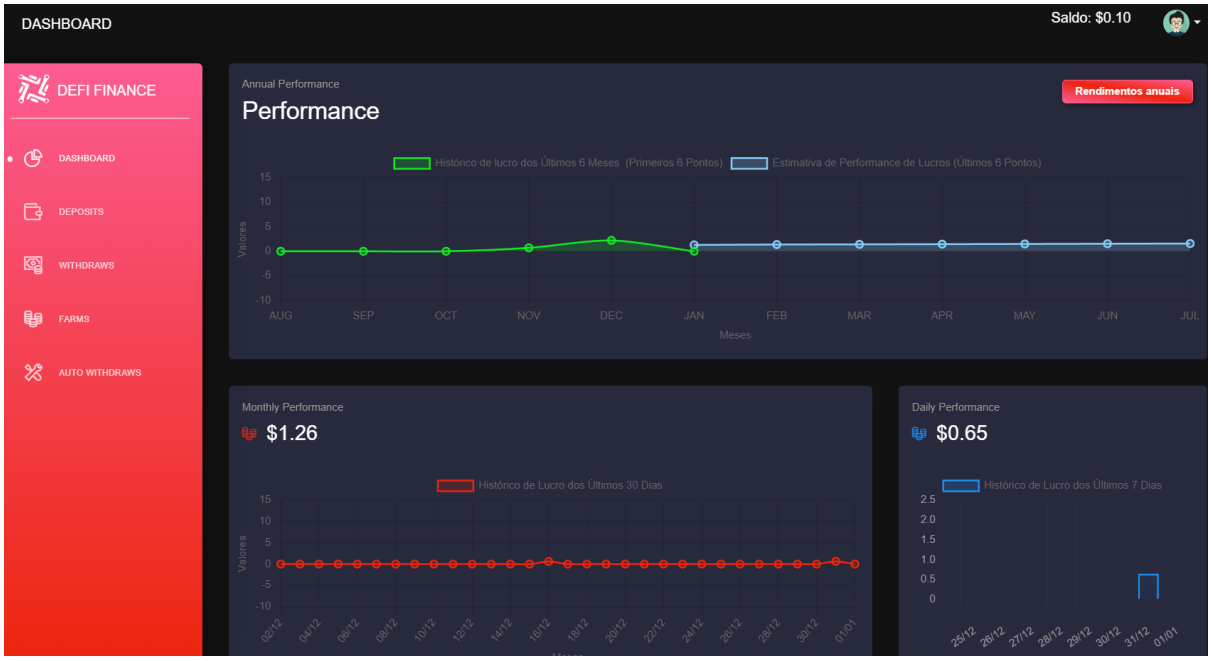

Figura 10. Wireframe do *dashboard* de desempenho financeiro.

A interface de *dashboard* visualizada na Figura 10 fornece uma visão abrangente do desempenho financeiro do usuário, atendendo ao requisito RF5. Ela exibe o saldo atual no canto superior direito, juntamente com gráficos de desempenho de lucros anual, mensal e semanal, projetando o crescimento dos ativos do portfólio do usuário. O gráfico de desempenho anual combina dados dos últimos seis meses (Linhas verde no gráfico anual) com uma projeção para os próximos seis (linha azul no gráfico anual), enquanto o gráfico mensal exibe o rendimento ao longo dos ultimos 30 dias.

A Figura 11 apresenta o diagrama de sequência da funcionalidade RF4 (Auto saque), ilustrando o fluxo de interação entre os componentes e o usuário.

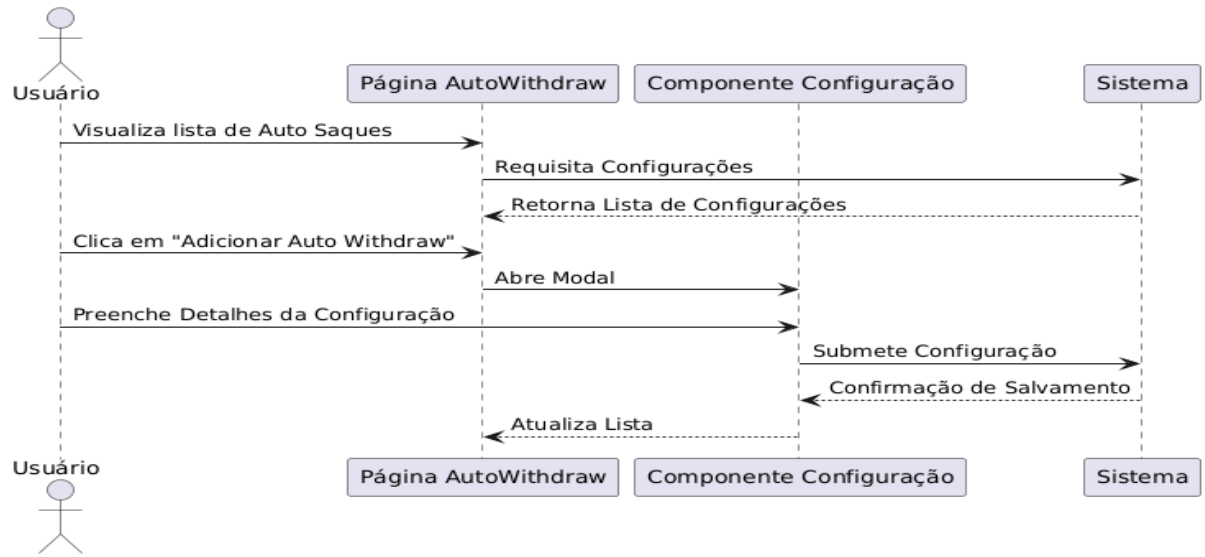

Figura 11. Diagrama de sequência da funcionalidade de configuração e listagem de auto saques (RF4).

A *dashboard* de desempenho financeiro (RF5) oferece ao usuário uma visão detalhada e projetada do desempenho dos investimentos, incluindo gráficos de desempenho anual, mensal e diário, que facilitam o entendimento do rendimento acumulado dos investimentos e a previsão de ganhos futuros.

O gráfico de desempenho anual combina dados dos últimos seis meses, exibindo o rendimento real até o mês atual e projetando o desempenho para os próximos seis meses com base nos investimentos feitos até o momento. No mês atual, o gráfico mostra o rendimento acumulado até o dia atual e projeta os ganhos para o restante do mês, considerando o APR/APY dos investimentos aplicados pelo usuário. Esse processo de projeção é implementado no método "generateStakesByYearChart" na implementação 10, onde o rendimento é ajustado de acordo com os ganhos reais e a previsão mensal. Assim, o usuário visualiza o crescimento potencial dos investimentos ao longo do ano.

O gráfico de desempenho mensal mostra o rendimento acumulado ao longo do mês atual, com cada ponto representando o rendimento diário. Além disso, o gráfico de desempenho semanal exibe o rendimento dos últimos sete dias em formato de barras, permitindo que o usuário acompanhe o desempenho recente de forma mais próxima.

Os gráficos exibidos no *dashboard* (Figura 10) são implementados com a biblioteca `Chart.js`[9], que oferece interatividade e permite uma visualização mais dinâmica do desempenho dos investimentos. Abaixo, um trecho do método `generateStakesByYearChart` (Implementação 10) ilustra a implementação do cálculo dos rendimentos reais e da projeção anual, com a visualização organizada em um gráfico.

```
1  public async generateStakesByYearChart() {
2    const { months, isFuture } = Helper.getPreviousAndNextSixMonths(new
       Date().getMonth());
3    let canvas: any = document.getElementById("stakesByYearChart");
4    let profitsByMonth: number[] = new Array(12).fill(0);
5    const currentMonthIndex = new Date().getMonth();
6    // Calculo dos rendimentos reais e projecao para os proximos meses
7    this.tenantUserStakesDailyValues.forEach((stake: UserStakesDailyValue)
       => {
8      const stakeDate = new Date(stake.date);
9      const stakeMonthIndex = stakeDate.getMonth();
10     const index = (stakeMonthIndex - currentMonthIndex + 12) % 12;
11     if (index === 0) {
12       const adjustedIndex = index + 5;
13       profitsByMonth[adjustedIndex] += stake.profit;
14     }
15   });
16   this.stakesPerYearChart = this.generateChartLineWithTooltipRed(months,
       canvas, "Income Performance", profitsByMonth);
17 }
```

Implementação 10. Geração do gráfico de desempenho anual.

[9]Mais informações sobre a biblioteca `Chart.js` estão disponíveis em: `https://www.chartjs.org`

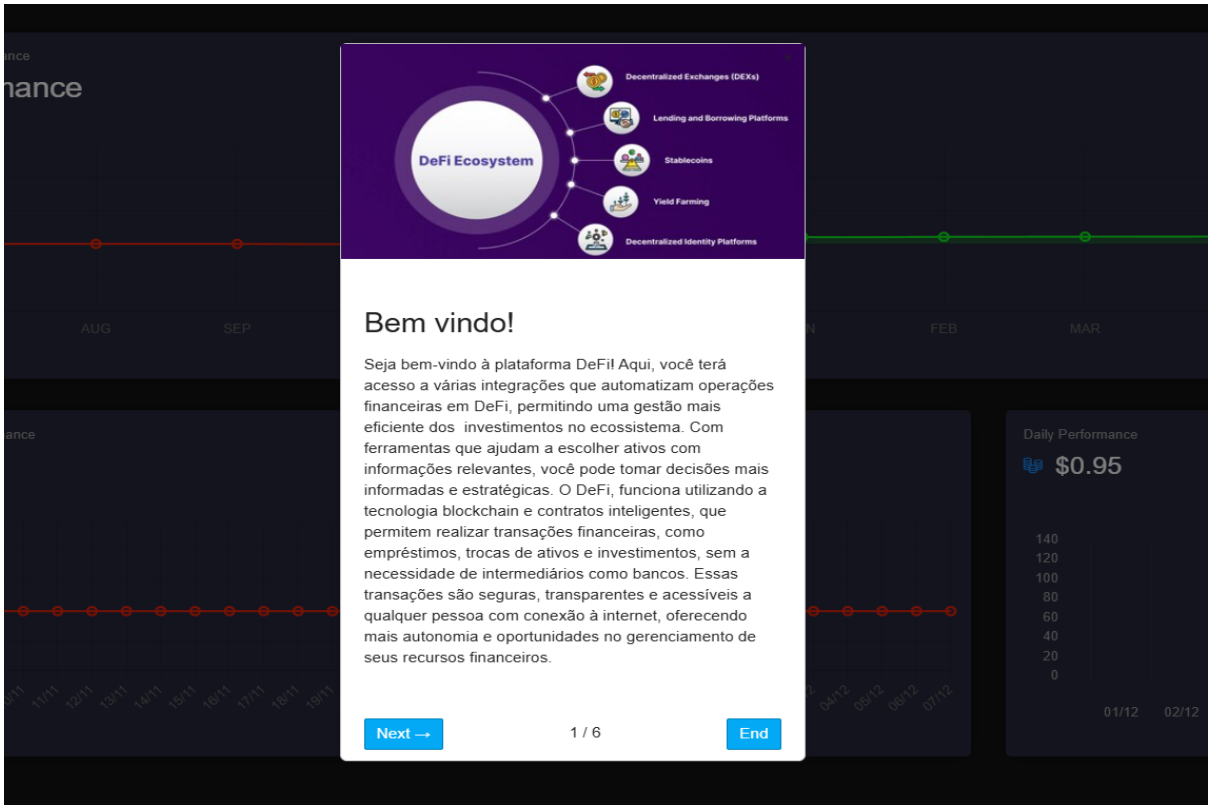

Figura 12. Exemplo do esquema de *walkthrough* implementado (RF6).

Para o requisito RF6, o sistema implementa um esquema de *walkthrough* que exibe um guia interativo sobre o uso das funcionalidades do sistema e os conceitos aplicados. Esse guia pode ser visualizado em todas as telas principais, conforme ilustrado na Figura 12.

Para implementar a RF6, que exige um guia interativo e explicações sobre os conceitos-chave do sistema, foi utilizada a ferramenta *InlineManual*, configurada para fornecer instruções passo a passo em todas as telas e modais do sistema. Esse guia interativo permite que o usuário receba orientações em tempo real sobre o funcionamento do sistema e dos conceitos aplicados nos investimentos. Além disso, o Inline Manual foi configurado para identificar e exibir um *walkthrough* inicial ao usuário, fornecendo também explicações contextuais sobre termos importantes, como “APY”, “Farm”, “Lucro” e “Auto saque”.

Para garantir que o Inline Manual seja carregado adequadamente para cada usuário, o código adiciona o script ao carregar o sistema, como exemplificado no método `addInlineManualScript` (Implementação 11). Esse método integra o guia ao sistema verificando a sessão do usuário por meio do *token* JWT e associando as instruções diretamente à identidade do usuário, garantindo que elas sejam exibidas de forma correta.

```
private addInlineManualScript(userId: string | undefined): void {
  const inlineManualScript = `
          var inlinemanual_app_key = "x";

          (function(e, n, a, t, i, r, s) {
            s.parentNode.insertBefore(r, s);
            r.onload = function() {
              console.log("Inline Manual script loaded successfully.");
              InlineManual("boot", { "uid": "${userId}" });
            };
```

```
            })(window, document, "script",
                "https://cdn.inlinemanual.com/inm/loader/loader." +
                inlinemanual_app_key + ".js", "InlineManual");
        `;

    const scriptElement = document.createElement("script");
    scriptElement.id = "inlinemanual";
    scriptElement.innerHTML = inlineManualScript;
    document.body.appendChild(scriptElement);
}
```

Implementação 11. Script para carregamento do Inline Manual.

Além disso, para facilitar a criação e personalização dos modais de orientação, foi utilizada a extensão *Inline Manual Authoring Tool*.[10] Essa extensão permite que os guias sejam editados e ajustados de forma intuitiva diretamente na interface do sistema, otimizando o processo de criação de tutoriais interativos e fornecendo um meio direto de adicionar explicações contextuais sobre o sistema e seus conceitos financeiros. A Figura 13 ilustra o uso da ferramenta para personalizar orientações na plataforma.

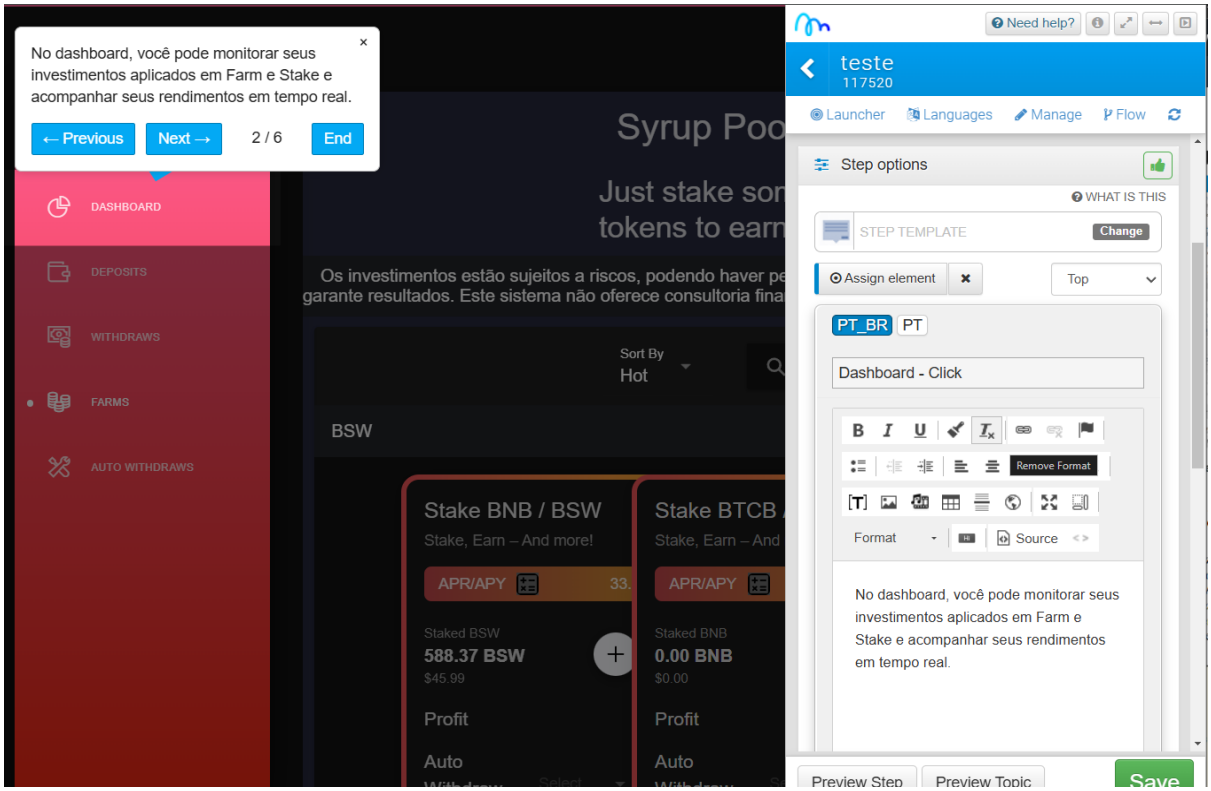

Figura 13. Exemplo do *Inline Manual Authoring Tool* sendo utilizado na plataforma.

Existe um conjunto de rotinas que realiza todo o processamento necessário para manter as informações corretas e prontas para os usuários. Nesse contexto, uma rotina é responsável por adicionar e atualizar automaticamente as *farms* utilizando *scraping*[11], outra cuida do cálculo

[10] Mais informações sobre a ferramenta estão disponíveis em `https://chromewebstore.google.com/detail/inline-manual-authoring-t/leiejllaoknebomjeebdjfeicdaaeijp?pli=1`.

[11] *Scraping* refere-se ao processo de coleta de dados diretamente das interfaces das plataformas de forma automatizada, garantindo que as informações das *farms* sejam constantemente atualizadas.

e atualização dos lucros (*profits*) dos usuários com base nos dados de mercado e APY obtidos pela rotina de atualização das *farms*; e uma terceira rotina consolida e atualiza os dados diários exibidos nos gráficos do *dashboard*.

A rotina de atualização das *farms* utiliza a tecnologia Selenium para realizar *web scraping* nas plataformas DeFi. Ela navega até as páginas, verifica se há validação humana e realiza múltiplos *scrolls* para garantir o carregamento de todo o conteúdo dinâmico. A seguir, são identificados os elementos correspondentes ao título das *farms* e ao APY (*Annual Percentage Yield*). Os dados obtidos são comparados com os já cadastrados no sistema, sendo atualizados ou adicionados conforme necessário. Abaixo está um trecho de código que exemplifica o carregamento dos dados com Selenium:

```
_wait.Until(d => js.ExecuteScript("return
    document.readyState").Equals("complete"));
ScrollToEndMultipleTimes(js, 15);
_wait.Until(d =>
{
    var stakeTitle = d.FindElements(By.CssSelector(stakeTitleClass));
    return stakeTitle.Count > 0;
});

var stakeTitle = _driver.FindElements(By.CssSelector(stakeTitleClass));
var stakeApy = _driver.FindElements(By.CssSelector(stakeApyClass));
for (var i = 0; i < stakeTitle.Count; i++)
{
    var stakeScrapingTitle = stakeTitle[i].Text;
    var stakeScrapingApy = stakeApy[i].Text.Replace("%", "").Trim();
    stakes.Add(new Stake
    {
        StakeTitleName = stakeScrapingTitle,
        AnnualIncome = Convert.ToDouble(stakeScrapingApy)
    });
}
```

Implementação 12. Carregamento de farms com Selenium.

A rotina que calcula os lucros dos usuários utiliza os dados de mercado obtidos pela rotina de *scraping* para recalcular os *profits* em intervalos regulares. Ela verifica o tempo decorrido desde a última atualização e ajusta os valores de acordo com as alterações no APY e nos valores das moedas da respectiva *farm*. Esses cálculos são fundamentais para a *dashboard* (RF5), além disso, ela complementa a RF2 ao garantir que as informações de lucro sejam constantemente atualizadas. O código a seguir demonstra o cálculo dos lucros baseados em períodos de cinco minutos, considerando que a rotina roda a cada 5 minutos:

```
var stakeProfitFiveMinutes = stake.AnnualIncome /
    qtdPeriodsOfFiveMinutesOnYear;
var profitForPeriods = (userStake.UserStakeStatus ==
    UserStakeStatus.Applied
     ? userStake.AmountStaked
     : userStake.LastAmountStakedByUser) * stakeProfitFiveMinutes *
        periodsOfFiveMinutes;

userStake.Profit += profitForPeriods;
userStake.AmountStaked += profitForPeriods;
```

Implementação 13. Cálculo dos lucros dos usuários.

A rotina de consolidação diária gera um resumo das operações para alimentar os gráficos do *dashboard*. Ela calcula os totais diários de lucros e quantidades investidas, e insere os dados no histórico na tabela TenantUserStakesDailyValue no banco de dados. Esses valores são posteriormente utilizados para exibir gráficos detalhados no *dashboard* (RF5). O trecho abaixo ilustra o cálculo e a consolidação dos valores diários:

```
var tenantUserStakesTotalAmountStaked = userStakeGrouped.Sum(userStake =>
{
    return userStake.UserStakeStatus switch
    {
        UserStakeStatus.Applied => userStake.AmountStaked,
        UserStakeStatus.UpdatedUnApplied =>
            userStake.LastAmountStakedByUser,
        _ => 0
    };
});

var command = new AddTenantUserStakesDailyValueCommand()
{
    TenantId = tenant?.Id,
    AmountStaked = tenantUserStakesTotalAmountStaked,
    Date = DateTime.Now,
    Profit = userStakeGrouped.Sum(userStake => userStake.Profit)
};
```

Implementação 14. Consolidação de dados diários.

Essas rotinas se complementam para garantir que o sistema esteja sempre atualizado e que os usuários tenham informações precisas e organizadas para tomar decisões no gerenciamento de seus portfólios DeFi.

3.4.1. Segurança Adicional

Para fortalecer a segurança geral (RNF2) do sistema, foram implementadas as seguintes medidas:

Foi adotado o algoritmo PBKDF2 (*Password-Based Key Derivation Function 2*) para armazenar senhas de maneira segura, devido à sua eficácia contra ataques de força bruta e dicionário. Ele utiliza múltiplas iterações de hashing, o que aumenta consideravelmente o tempo necessário para tentativas de quebra. Adicionalmente, cada senha é armazenada com um *salt* exclusivo e gerado de forma aleatória, garantindo que mesmo senhas idênticas resultem em hashes diferentes.[12]

Todas as entradas do usuário são validadas tanto no cliente quanto no servidor, prevenindo injeções de código e outros tipos de ataques. O código de validação pode ser visto na implementação 15.

```
public class AuthorizeTenantCommandValidator :
    AbstractValidator<AuthorizeTenantCommand>
{
    public AuthorizeTenantCommandValidator()
    {
        RuleFor(x => x.Email)
            .NotEmpty().WithMessage("O campo de email e obrigatorio.")
```

[12]Mais informações sobre o PBKDF2 podem ser encontradas em RFC 2898 - Seção 5.2.

```
            .EmailAddress().WithMessage("Formato de email invalido.");

        RuleFor(x => x.Password)
            .NotEmpty().WithMessage("O campo de senha e obrigatorio.");
    }
}
```

Implementação 15. Classe de validação de entrada.

A classe `AuthorizeTenantCommandValidator` (Implementação 15) valida as entradas de email e senha, garantindo que esses campos estejam preenchidos e que o formato de email seja válido, prevenindo injeções de código e outros ataques.

Foi implementada uma política de bloqueio de conta após múltiplas tentativas de login fracassadas em um curto período de tempo. A configuração desta política está ilustrada na implementação 16.

```
public static void AddIdentity(this IServiceCollection services)
{
    services.AddIdentity<Tenant, IdentityRole>(options =>
    {
        options.Lockout.MaxFailedAccessAttempts = 5; // Bloqueio apos 5
            tentativas
        options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(1);
            // Tempo de bloqueio
        options.User.RequireUniqueEmail = true;
    });
}
```

Implementação 16. Configuração de políticas de bloqueio.

A configuração na implementação 16, define um bloqueio temporário da conta após cinco tentativas falhas de login em um intervalo curto, prevenindo ataques de força bruta.

Todos os acessos e operações sensíveis são registrados em *logs* de auditoria, que são monitorados para detectar e responder rapidamente a comportamentos suspeitos. A implementação da auditoria está demonstrada na implementação 17.

```
public class LoggingBehavior<TRequest, TResponse> :
    IPipelineBehavior<TRequest, TResponse>
{
    public async Task<TResponse> Handle(TRequest request,
        RequestHandlerDelegate<TResponse> next,
        CancellationToken cancellationToken)
    {
        _logger.LogInformation("Requisicao recebida: {Request}",
            typeof(TRequest).Name);
        var response = await next();
        _logger.LogInformation("Resposta gerada: {Response}",
            typeof(TResponse).Name);
        return response;
    }
}
```

Implementação 17. Registro de *logs* para auditoria de acessos.

O código da implementação 17, implementa a auditoria de acessos, registrando em *logs* todas as requisições e respostas geradas, garantindo monitoramento constante das operações sensíveis.

3.4.2. Estrutura das Sessões de Usuário

Para a autenticação dos usuários no sistema, foi implementado o uso de JWT (JSON Web Token).[13] Este método permite a autenticação sem a necessidade de manter o estado do lado do servidor, essencial para sistemas escaláveis. Cada JWT contém informações codificadas do usuário, como nome de usuário, papéis (*roles*) e um tempo de expiração, garantindo que os *tokens* não possam ser reutilizados indefinidamente.

Para mitigar o risco de sequestro de sessão, o JWT é assinado digitalmente utilizando o algoritmo HMAC-SHA256 e uma chave secreta privada, assegurando que qualquer modificação no *token* seja detectada, invalidando-o. A geração do *token* JWT está detalhada na implementação 18.

```
1  private AuthorizedTenantModelResult GenerateToken(Domain.Entities.Tenant
       user, string role)
2  {
3      var claims = new List<Claim>
4      {
5          new Claim(JwtRegisteredClaimNames.UniqueName, user.UserName),
6          new Claim(JwtRegisteredClaimNames.Name, user.Name),
7          new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
8          new Claim(ClaimTypes.Role, role)
9      };
10
11     var key = new SymmetricSecurityKey(Encoding.UTF8
12         .GetBytes(_configuration["Jwt:key"]));
13     var signinCredentials = new SigningCredentials(key,
           SecurityAlgorithms.HmacSha256);
14
15     var expirationHours =
           double.Parse(_configuration["TokenConfiguration:ExpireHours"]);
16     var expiration = DateTime.UtcNow.AddHours(expirationHours);
17
18     var token = new JwtSecurityToken(
19         _configuration["TokenConfiguration:Issuer"],
20         _configuration["TokenConfiguration:Audience"],
21         claims,
22         expires: expiration,
23         signingCredentials: signinCredentials);
24
25     return new AuthorizedTenantModelResult
26     {
27         AccessToken = new JwtSecurityTokenHandler().WriteToken(token),
28         Expiration = expiration
29     };
30 }
```

Implementação 18. Geração de token JWT com criptografia.

Esse código (Implementação 18) gera um *token* JWT com criptografia HMAC-SHA256, garantindo sua segurança contra modificações. Nas linhas 4 a 7, são criadas as *claims* com informações do usuário, como nome de usuário, função e identificador único do *token*. Nas linhas 10 e 11, é gerada uma chave de segurança baseada em uma configuração secreta, usada para assinar digitalmente o *token*. A linha 13 define o tempo de expiração do token com base

[13] JWT é um padrão aberto (RFC 7519) que define um formato compacto e seguro para transmitir informações como um objeto JSON. Mais informações podem ser encontradas em `https://jwt.io/`.

nas configurações do sistema, e nas linhas 15 a 20, o *token* é construído e retornado, incluindo o *token* gerado e sua validade.

Cada JWT é configurado para expirar após um período específico. Além disso, as requisições autenticadas verificam constantemente o tempo de expiração para garantir que *tokens* expirados sejam rejeitados. A configuração da validação do *token* JWT está apresentada na implementação 19.

```
1  public static void AddJwtBearerAuthentication(this IServiceCollection
       services, IConfiguration configuration)
2  {
3      services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
4          .AddJwtBearer(options =>
5          {
6              options.TokenValidationParameters = new
                   TokenValidationParameters
7              {
8                  ValidateIssuer = true,
9                  ValidateAudience = true,
10                 ValidateLifetime = true,
11                 ValidAudience =
                       configuration["TokenConfiguration:Audience"],
12                 ValidIssuer = configuration["TokenConfiguration:Issuer"],
13                 IssuerSigningKey = new SymmetricSecurityKey(
14                     Encoding.UTF8.GetBytes(configuration["Jwt:key"])),
15                 ClockSkew = TimeSpan.Zero // Elimina tempo extra para
                       expiracao
16             };
17         });
18 }
```

Implementação 19. Configuração de validação de expiração do token JWT.

Foram implementadas medidas de proteção contra CSRF, garantindo que apenas requisições originadas de fontes confiáveis sejam aceitas. No *front-end* (camada de apresentação), um interceptor HTTP foi configurado para adicionar automaticamente o *token* JWT no cabeçalho de autorização de todas as requisições enviadas ao servidor. Isso assegura que o servidor receba apenas requisições autenticadas e reduz o risco de ataques CSRF. O código do interceptor está na implementação 20.

```
1
2  @Injectable()
3  export class RequestInterceptor implements HttpInterceptor {
4
5    constructor(private tokenService: TokenService) {}
6
7    intercept(req: HttpRequest<any>, next: HttpHandler):
         Observable<HttpSentEvent
8      | HttpHeaderResponse | HttpProgressEvent | HttpResponse<any> |
         HttpUserEvent<any>> {
9
10     const user = this.tokenService.getUser();
11
12     // Verifica se o usuario esta autenticado e se a URL pertence ao
         dominio seguro
13     if (user && user.accessToken &&
         req.url.includes(environment.apiEndpoint)) {
14
```

```
15        const token = user.accessToken;
16        const headers = req.headers.append('Authorization', 'Bearer ' +
              token);
17
18        // Clona a requisicao original e adiciona o cabecalho com o token JWT
19        req = req.clone({ headers: headers });
20      }
21
22      return next.handle(req);
23    }
24  }
```

Implementação 20. Interceptação HTTP para adicionar token de autenticação.

No código da implementação 20, o interceptor HTTP verifica se o usuário está autenticado e se a requisição está sendo enviada para o domínio configurado (`environment.apiEndpoint`). Se essas condições forem atendidas, o *token* JWT é recuperado do serviço `TokenService` e adicionado ao cabeçalho da requisição como `Authorization: Bearer <token>`. Esta abordagem assegura que apenas requisições autenticadas sejam processadas pelo servidor, contribuindo para a proteção contra CSRF.

4. Validação e Discussões do Projeto

Nesta seção, são apresentados os aspectos de validação e discussão das funcionalidades desenvolvidas para a plataforma de gestão DeFi. A validação do sistema foi baseada em uma abordagem prática, avaliando funcionalidades principais frente aos objetivos iniciais e requisitos estabelecidos. O foco esteve em garantir alinhamento das operações de gerenciamento de portfólios, auto saques, dados de mercado e segurança com as necessidades do usuário.

4.1. Comparação com Soluções Existentes

A plataforma desenvolvida no presente estudo tem alguns diferenciais em relação a outras soluções disponíveis no mercado, especialmente no que tange à segurança e à facilidade de uso. A segurança destaca-se pela disponibilização de informações detalhadas sobre as *farms*, como dados históricos, correlação entre valores das moedas da *farm*, métricas de segurança e informações de mercado em tempo real. Essas funcionalidades auxiliam os usuários na escolha de investimentos mais adequados ao seu perfil e com menor risco.

Adicionalmente, a integração de funcionalidades como saques automáticos configuráveis e projeções de desempenho financeiro demonstraram ser um diferencial importante, em contraste com outras plataformas que oferecem essas funções de forma mais fragmentada. A gestão centralizada de portfólios e a visibilidade de dados em tempo real proporcionam aos usuários uma visão ampla e acessível, facilitando decisões mais informadas.

No quesito segurança técnica, o uso de *tokens* JWT para autenticação e criptografia no armazenamento de senhas adicionam uma camada extra de proteção, destacando a plataforma em relação a concorrentes que podem não implementar práticas tão robustas.

O quadro 1 apresenta uma comparação detalhada entre a plataforma desenvolvida e soluções existentes, reforçando os diferenciais oferecidos. Essa análise evidencia como a plataforma atende a critérios considerados essenciais no cenário de finanças descentralizadas. Apesar de a plataforma atender a todos os critérios avaliados, alguns aspectos merecem destaque, promovendo uma solução mais completa em relação às opções já disponíveis no mercado.

A inclusão de funcionalidades como *walkthroughs*, dados detalhados de segurança e auto saque, ausentes nas demais soluções, tornam a plataforma uma solução versátil e eficiente para usuários de diferentes níveis de experiência. Essa abordagem não apenas atende às necessidades principais, mas também oferece uma experiência mais acessível e completa. A validação demonstrou que o sistema desenvolvido agrega valor ao setor, eliminando lacunas encontradas nas outras ferramentas avaliadas.

4.1.1. Validação

Para validar a funcionalidade e precisão da plataforma desenvolvida, foi realizada uma comparação com uma *pool* de liquidez real. Entre os dias 24/11/2024 e 07/12/2024, um valor inicial foi investido na *pool* e o saldo foi inserido na plataforma para acompanhar os resultados gerados.

Ao final do período, os valores finais apresentados foram os seguintes na plataforma desenvolvida, o saldo total final foi de **52.87 dólares**, com um valor de **1.197 dólares** em taxas recebidas. Na *pool* de liquidez real, o saldo final foi de **46.02 dólares**, com um valor de **1.210 dólares** em taxas recebidas.

Os dados apresentados nas Figuras 14 e 15 ilustram visualmente os resultados obtidos na *farm* real e na plataforma desenvolvida, respectivamente.

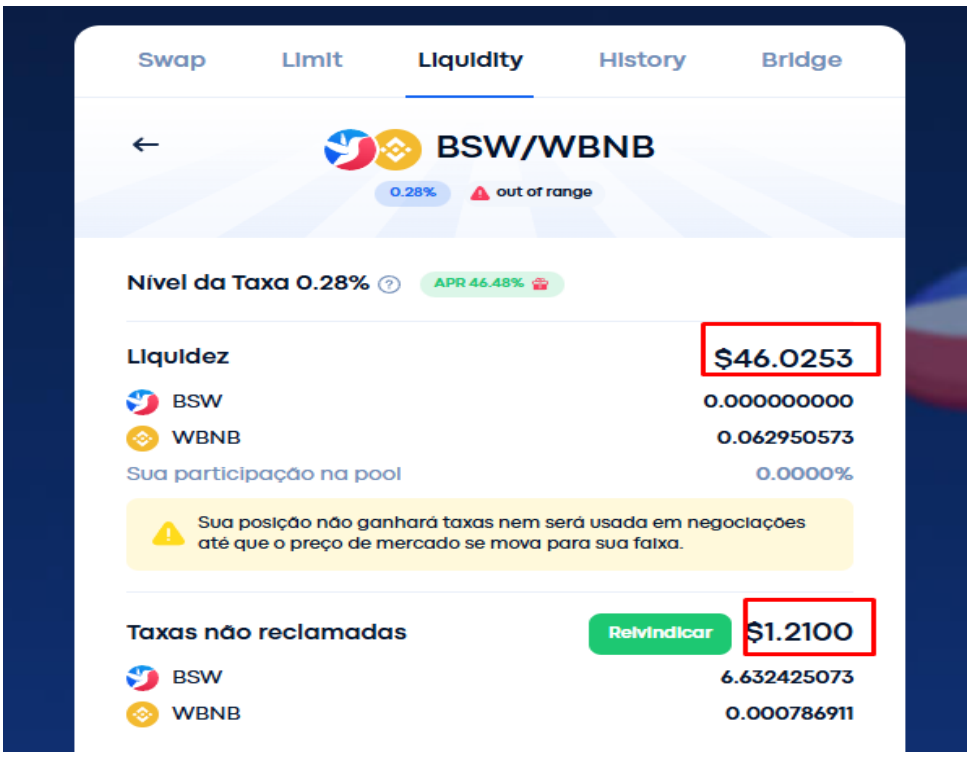

Figura 14. Resultado da *farm* real entre 24/11/2024 e 07/12/2024.

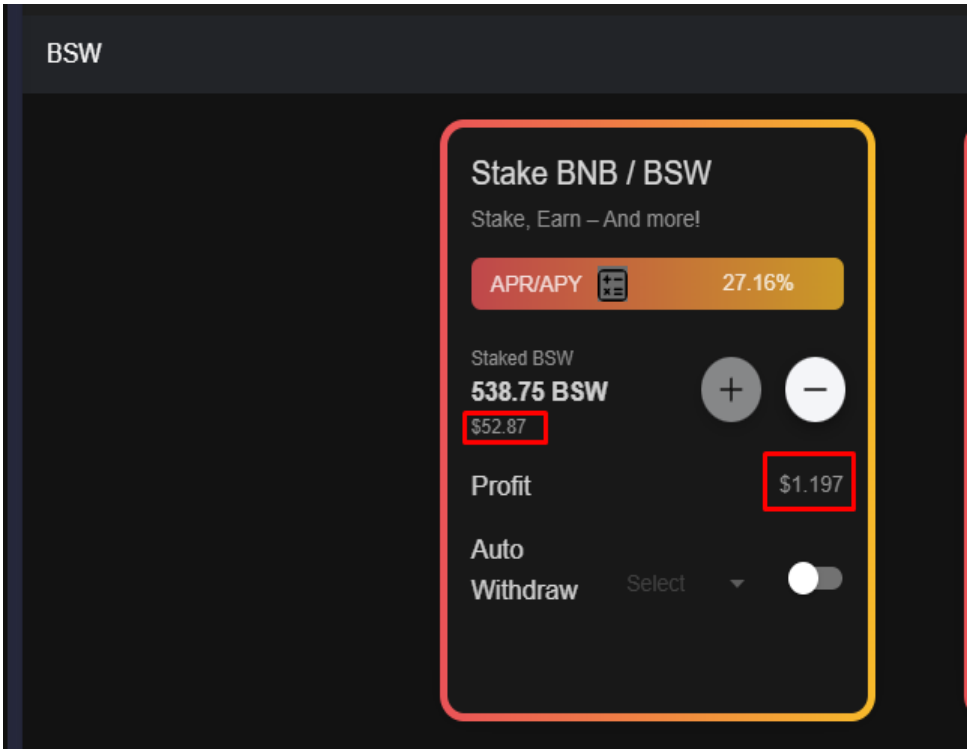

Figura 15. Resultado projetado pela plataforma desenvolvida entre 24/11/2024 e 07/12/2024.

Esses resultados destacam a capacidade da plataforma em fornecer projeções e cálculos de desempenho financeiros bastante próximos aos valores reais, ainda que variações possam ocorrer devido a pequenas diferenças nos métodos de cálculo e na sincronização de dados.

Dentre os desafios encontrados, a principal dificuldade está relacionada ao funcionamento das *pools* de liquidez, especialmente no que tange ao rebalanceamento automático que ocorre a cada transação. Este fator comprometeu a precisão nos cálculos de lucro, que atualmente se baseiam em uma proporção fixa de 50% para cada moeda na *farm*, conforme a fórmula do *impermanent loss*, essa abordagem pode levar a análises distorcidas quando há variações significativas no equilíbrio das moedas. Além disso, não foi possível contemplar *farms* mais avançadas que permitem configurações de preço dinâmicas ou ajustes de balanceamento, limitando a representatividade de cenários complexos.

As rotinas de automação também apresentaram limitações. Embora o *scraper* tenha desempenhado um papel central na coleta e atualização de dados, sua dependência de interfaces HTML sujeitas a mudanças aumenta a vulnerabilidade a falhas. A implementação de validações anti-*robo* em plataformas integradas também exigiu ajustes que comprometeram a eficiência.

5. Considerações Finais

Este estudo buscou solucionar desafios comuns enfrentados no ecossistema de Finanças Descentralizadas (DeFi), destacando-se a complexidade operacional associada à interação com contratos inteligentes. Esses mecanismos, embora essenciais para maximizar rendimentos, demandam um nível de entendimento técnico que pode ser uma barreira significativa para usuários iniciantes. A configuração e o acompanhamento de *pools* de liquidez exigem atenção constante, especialmente devido ao rebalanceamento automático que ocorre durante as transações, impactando a precisão dos cálculos de lucro. Propondo uma plataforma integrada para gestão de

portfólios e planejamento financeiro. Durante o desenvolvimento, a plataforma apresentou resultados positivos ao oferecer funcionalidades como projeções financeiras, monitoramento em tempo real de ativos e ferramentas de automação, destacando-se pela usabilidade e segurança implementadas. A integração inicial com as plataformas Biswap e PancakeSwap mostrou-se funcional, possibilitando o acesso a *farms* e liquidez, embora tenha revelado oportunidades para uma ampliação mais abrangente.

Para o futuro, há um potencial significativo para evoluir a plataforma. A ampliação da integração com mais protocolos e *exchanges* poderá aumentar a gama de ativos e estratégias disponíveis aos usuários. Essa evolução deve considerar a validação inicial de cada nova integração para mitigar problemas relacionados à apresentação de dados e validações específicas de cada plataforma. A introdução de inteligência artificial também representa um campo promissor, permitindo maior precisão na análise de padrões de mercado, além de oferecer recomendações personalizadas de *farms* e estratégias de acordo com o perfil do usuário.

É necessário investigar a integração com APIs que forneçam dados detalhados sobre as transações realizadas em *pools* de liquidez, possibilitando o acesso direto e em tempo real a informações como entradas, saídas e preços dos ativos. Na ausência dessas APIs, pode-se desenvolver rotinas de *scraping* para coletar dados diretamente das interfaces HTML das plataformas. Outra abordagem inclui utilizar a fórmula de *impermanent loss* e os preços dos ativos para estimar o balanço atual das *pools*. Por fim, ampliar funcionalidades, como notificações em tempo real sobre mudanças nas *pools* e explorar integrações adicionais com APIs são etapas importantes para tornar a plataforma mais eficiente e alinhada às demandas do ecossistema DeFi.

Outra melhoria possível seria explorar a integração com APIs proprietárias que forneçam dados diretamente, reduzindo a dependência de processos de *scraping*. Além disso, funcionalidades como notificações em tempo real sobre variações de preços e automação avançada de estratégias de investimento podem aumentar a utilidade da plataforma. Por fim, a replicação de estratégias de carteiras públicas surge como uma oportunidade de oferecer mais ferramentas para diversificação de investimentos, permitindo que usuários imitem modelos de sucesso.

Em síntese, a plataforma alcançou grande parte de seus objetivos iniciais e apresentou uma base sólida para evolução. Com a implementação das melhorias sugeridas, espera-se que ela se torne uma solução mais completa, eficiente e alinhada às demandas de um público diversificado no universo DeFi.

Referências

Alamsyah, A., Kusuma, G. N. W., e Ramadhani, D. P. (2024). A review on decentralized finance ecosystems. *Future Internet*, 16(3). Acesso em: 27 de dez. de 2024.

Betts, D., Dominguez, J., Melnik, G., Simonazzi, F., e Subramanian, M. (2013). Exploring cqrs and event sourcing: A journey into high scalability, availability, and maintainability with windows azure.

Caldarelli, G. e Ellul, J. (2021). The blockchain oracle problem in decentralized finance—a multivocal approach. *Applied Sciences*, 11(16):7572.

Cap, D. M. (2024). Defimarketcap. Disponível em: `https://defimarketcap.io/` . Acesso em: 2 de abr. de 2024.

Chen, Y. e Bellavitis, C. (2020). Blockchain disruption and decentralized finance: The rise of decentralized business models. *Journal of Business Venturing Insights*, 13:e00151.

CoinDesk (2023). Crypto crime hit all-time high of $20.6b in 2022: Chainalysis. Disponível em: https://www.coindesk.com/business/2023/02/27/crypto-crime-hit-all-time-high-of-206b-in-2022-chainalysis . Acesso em: 29 dez. 2024.

CoinGecko (2024). Coingecko. Disponível em: https://www.coingecko.com/en/categories/decentralized-finance-defi . Acesso em: 2 de abr. de 2024.

CoinMarketCap (2024). Coinmarketcap. Disponível em: https://coinmarketcap.com/view/defi/ . Acesso em: 2 de abr. de 2024.

Conservancy, S. F. (2024). Selenium. Disponível em: https://www.selenium.dev/ . Acesso em: 28 de jul. de 2024.

De Meijer, C. (2021). Defi and regulation: The european approach. Disponível em: Finextra Blog Article . Acesso em: 2 de abr. de 2024.

del Priore, M. (2022). Wiki lemon. Disponível em: https://wiki.lemon.me/pt-br/crypto-101/impermanent-loss/#:~:text=Impermanent%20loss . Acesso em: 17 de jun. de 2024.

Fadilpašić, S. (2024). Web3 losses cut by 23% in q1 2024, hackers may be eyeing $100 billion in locked funds – immunefi. Disponível em: https://cryptonews.com/news/web3-losses-23-q1-2024-hackers-eyeing-100-billion-in-locked-funds-immun htm . Acesso em: 28 de mar. de 2024.

Finance, M. (2024). Metrix finance. Disponível em: https://metrix.finance/ . Acesso em: 01 de dez. de 2024.

Forsgren, N., Humble, J., e Kim, G. (2018). *Accelerate: The Science of Lean Software and DevOps*. IT Revolution Press.

Galin, D. (2004). *Software Quality Assurance: From Theory to Implementation*. Addison-Wesley, Boston.

Google (2024). Angular. Disponível em: https://angular.dev/ . Acesso em: 28 de jul. de 2024.

Gudgeon, L., Werner, S., Perez, D., e Knottenbelt, W. J. (2020). Defi protocols for loanable funds: Interest rates, liquidity and market efficiency. Disponível em: Ethereum Foundation .

Immunefi (2024). Immunefi. Disponível em: https://immunefi.com/research/ . Acesso em: 7 de abr. de 2024.

Krug, S. (2014). *Don't Make Me Think: A Common Sense Approach to Web Usability*. New Riders.

Kumar, M., Nikhil, N., e Singh, R. (2020). Decentralising finance using decentralised blockchain oracles. In *2020 International Conference for Emerging Technology (INCET)*, pages 1–4. IEEE.

Llama, D. (2024). Defi llama. Disponível em: https://defillama.com/?volume=true&tvl=true . Acesso em: 2 de abr. de 2024.

Macaskill, S. (2021). Blockchain adoption in new zealand. Acesso em: 2 de abr. de 2024. Acesso em: 2 de abr. de 2024.

Manua, I. (2024). Inline manua. Disponível em: https://inlinemanual.com/ .

Meegan, X. (2020). Identifying key non-financial risks in decentralised finance on ethereum blockchain. *MIP Politecnico di Milano*.

Microsoft (2024a). Azure devops. Disponível em: https://azure.microsoft.com/pt-br/products/devops . Acesso em: 28 de jul. de 2024.

Microsoft (2024b). C#. Disponível em: https://dotnet.microsoft.com/pt-br/languages/csharp . Acesso em: 28 de jul. de 2024.

Microsoft (2024c). Sql server. Disponível em: https://learn.microsoft.com/en-us/sql/sql-server/what-is-sql-server . Acesso em: 28 de jul. de 2024.

Norman, D. (2019). The four fundamental principles of human-centered design and application. *jnd.org*. Acesso em: 2 de abr. de 2024.

Orbit (2024). Orbit. Disponível em: https://orbitdefi.finance/ . Acesso em: 12 de jun. de 2024.

OWASP (2024). Owasp vulnerabilities. Disponível em: https://owasp.org/www-community/vulnerabilities/ . Acesso em: 2 de jun. de 2024.

Ozcan, R. (2021). Decentralized finance. In *Financial Ecosystem and Strategy in the Digital Era: Global Approaches and New Opportunities*, pages 57–75. Springer.

Popescu, A.-D. (2020). Decentralized finance (defi)–the lego of finance. *Social Sciences and Education Research Review*, 7(1):321–349.

Salami, I. (2020). Decentralised finance: the case for a holistic approach to regulating the crypto industry. *Journal of International Banking and Financial Law*, 35(7):496–499.

Schueffel, P. (2021). Defi: Decentralized finance-an introduction and overview. *Journal of Innovation Management*, 9(3).

Schär, F. (2021). Decentralized finance: On blockchain- and smart contract-based financial markets. *Federal Reserve Bank of St. Louis Review*.

Tabtrader (2023). Tabtrader. Disponível em: https://tabtrader.com/pt/academy/articles/what-is-an-amm-automated-market-maker . Acesso em: 29 de jun. de 2024.

Teahouse (2024). Teahouse. Disponível em: https://teahouse.finance/ . Acesso em: 12 de jun. de 2024.

Terminal, G. (2024). Gecko terminal. Disponível em: https://geckoterminal.com/ . Acesso em: 01 de dez. de 2024.

Wang, Q., Li, R., Wang, Q., e Chen, S. (2020). Security issues in defi ecosystems. Disponível em: Cryptology ePrint Archive .

Yavin, O. e Reardon, A. (2021). What digital banks can learn from decentralised finance. *Journal of Digital Banking*, 5(3):255–263.

Zetzsche, D., Arner, D., e Buckley, R. (2020). Decentralized finance. *Journal of Financial Regulation*, 6:172–203.