UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA (INE)
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Luiz Gustavo Abou Hatem de Liz

Sistema de Assistência Estudantil e Real Digital: Usando “Blockchain” para melhorar o financiamento de assistências sociais

Florianópolis
2025

Luiz Gustavo Abou Hatem de Liz

Sistema de Assistência Estudantil e Real Digital: Usando “Blockchain” para melhorar o financiamento de assistências sociais

Trabalho de Conclusão de Curso submetido ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal de Santa Catarina para a obtenção do título de Bacharel em Sistemas de Informação.
Orientador: Prof. Dr. Jean Everson Martina
Coorientador: Me. Lucas Palma

Florianópolis
2025

Luiz Gustavo Abou Hatem de Liz

Sistema de Assistência Estudantil e Real Digital: Usando "Blockchain" para melhorar o financiamento de assistências sociais

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "Bacharel em Sistemas de Informação" e aprovado em sua forma final pelo Curso de Graduação em Sistemas de Informação.

Florianópolis, 18 de dezembro de 2025.

Banca Examinadora:

Prof. Dr. Jean Everson Martina
Orientador
UFSC

Me. Johann Westphall
Membro da Banca
UFSC

Me. Gabriel Estevam de Oliveira
Membro da Banca
UFSC

Dedico este trabalho à minha vó, Wilma Machado Carrilho, que acreditou e investiu na minha educação, mas mais do que isso, foi minha referência de disciplina e resiliência. À minha esposa e eterna parceira, Marina Rocha Guimarães, exemplo de esforço, sempre me apoiou, me ajudou, e me fez voltar a acreditar em mim quando eu queria ter desistido. Dedico à minha mãe, Rosa Abou Hatem, que me deu muito amor, teve pulso firme sempre que precisava, e me ensinou pelo exemplo que a vida é feliz quando fazemos o que amamos. Aos meus irmãos mais velhos, João Bruno e Edson Augusto, duas espécimes de inteligência invejável, a quem eu admiro e orgulhosamente segui os passos enquanto encontrava meu próprio caminho. A minha segunda mãe, Ivone de Fátima Teixeira, que me acolheu e me mostrou um amor incondicional. Ao meu pai, João Macedo de Liz, que é meu exemplo de calma e paz. Ao meu irmão mais novo, Geder, que se tornou um homem de família respeitável, e do qual tenho muito orgulho. E por último, agradeço à Universidade Federal de Santa Catarina, estendendo a todos os amigos que fiz, na qual estive por doze anos, em três cursos diferentes, aprendi muitas coisas, conheci muitas pessoas, fui muito feliz e me tornei um adulto funcional nesse lugar, pra sempre lembrarei com muito entusiasmo dessa fase da minha vida.

"Hesitação é derrota." — Napoleão Bonaparte

Resumo

Este trabalho apresenta uma proposta de implementação de um sistema de assistência estudantil utilizando a tecnologia blockchain, especificamente o DREX (Real Digital), para melhorar o financiamento e a gestão de programas de assistência social em universidades. O estudo explora como a tokenização de moedas e contratos inteligentes pode proporcionar maior transparência, rastreabilidade e eficiência na distribuição de recursos para estudantes em situação de vulnerabilidade socioeconômica. A metodologia adotada inclui o desenvolvimento de um protótipo funcional baseado em Hyperledger Besu, que simula o ambiente do DREX, permitindo a emissão e transferência de tokens representando o Real Digital. Como resultado, demonstra-se a viabilidade técnica da solução proposta e discutem-se os benefícios potenciais para as instituições de ensino e para os beneficiários dos programas de assistência estudantil. Conclui-se que a tecnologia blockchain pode contribuir significativamente para a modernização dos sistemas de assistência social, reduzindo custos operacionais, minimizando fraudes e ampliando a inclusão financeira dos estudantes.

Palavras-chave: DREX; Blockchain; Assistência Estudantil; Inclusão Financeira; Real Digital

Abstract

This work presents a proposal for implementing a student assistance system using blockchain technology, specifically DREX (Digital Real), to improve the financing and management of social assistance programs in universities. The study explores how currency tokenization and smart contracts can provide greater transparency, traceability, and efficiency in the distribution of resources to students in socioeconomically vulnerable situations. The methodology adopted includes the development of a functional prototype based on Hyperledger Besu, which simulates the DREX environment, enabling the issuance and transfer of tokens representing the Digital Real. As a result, the technical feasibility of the proposed solution is demonstrated, and the potential benefits for educational institutions and beneficiaries of student assistance programs are discussed. It is concluded that blockchain technology can significantly contribute to the modernization of social assistance systems, reducing operational costs, minimizing fraud, and expanding financial inclusion for students.

Keywords: DREX; Blockchain; Student Assistance; Financial Inclusion; Digital Real

Lista de figuras

Lista de tabelas

Sumário

1 Introdução

Desde a implementação do Programa Nacional de Assistência Estudantil (PNAES) em 2010, milhares de estudantes brasileiros têm encontrado nas políticas de permanência universitária o suporte necessário para concluir seus cursos superiores (República, 2010). Contudo, quem já teve contato com esses sistemas—seja como beneficiário, gestor ou pesquisador—reconhece que ainda há muito a ser melhorado.

Existem alguns desafios evidentes, sendo um dos maiores a falta de transparência na execução do programa já constatada pelo Tribunal de Contas da União. A fiscalização identificou que muitas universidades não divulgam informações detalhadas sobre os beneficiários e os resultados das ações de assistência estudantil. A ausência de relatórios de avaliação e dados abertos dificulta o acompanhamento da execução do programa, comprometendo a capacidade de monitoramento e controle social sobre os recursos públicos utilizados (Tribunal de Contas da União, 2018). Enquanto isso, assistimos ao surgimento de tecnologias que prometem revolucionar a forma como lidamos com transações financeiras e gestão de recursos públicos.

É nesse contexto que surge uma oportunidade concreta de transformação. O projeto DREX (Digital Real Experimental), liderado pelo Banco Central do Brasil, representou uma iniciativa de modernização do sistema financeiro nacional através da implementação de uma moeda digital de banco central (CBDC). Embora o piloto DREX tenha sido descontinuado pelo Banco Central em 2025 (Econômico, 2025), a infraestrutura tecnológica desenvolvida durante o projeto viabilizou o desenvolvimento de soluções inovadoras para a gestão de recursos públicos, especialmente em áreas sensíveis como a assistência estudantil (Brasil, 2023).

Este trabalho apresenta o desenvolvimento de um sistema de assistência estudantil que integra a tecnologia blockchain com os contratos inteligentes e a arquitetura desenvolvidos no contexto do piloto DREX, resultando em uma solução prática e funcional que simplifica o processo de concessão de benefícios e aumenta a transparência na aplicação dos recursos públicos, demonstrando a viabilidade técnica dessa abordagem no contexto brasileiro mesmo após a descontinuação do projeto oficial.

1.1 Contextualização

Para compreender o potencial desta proposta, é necessário primeiro entender como funcionam atualmente os dois sistemas que pretendemos conectar: o sistema de assistência estudantil brasileiro e o piloto DREX.

O sistema de assistência estudantil brasileiro funciona através de um processo que inicia com o edital de inscrição, onde os estudantes submetem documentação comprobatória de sua situação socioeconômica. As Pró-Reitorias de Assuntos Estudantis das universidades federais realizam a análise dos documentos, classificam os candidatos e definem os valores dos auxílios. Após a aprovação, os repasses são realizados através de transferências bancárias tradicionais, geralmente mensais, para contas correntes dos estudantes. Este fluxo, embora estabelecido,

apresenta limitações em termos de transparência no rastreamento dos recursos desde a origem até o uso final, além de depender de processos manuais de gestão e conferência.

O piloto DREX, por sua vez, foi desenvolvido sobre uma infraestrutura blockchain permissionada onde apenas instituições autorizadas pelo Banco Central podem participar como validadores da rede. O sistema utilizava contratos inteligentes programáveis que permitem a criação de regras automatizadas para transferências e gestão de ativos digitais. Cada transação realizada na rede fica registrada de forma permanente e auditável, permitindo rastreabilidade completa. Os contratos oficiais desenvolvidos incluíam funcionalidades para emissão de moeda digital (RealDigital), tokenização de ativos (RealTokenizado), descoberta de endereços (AddressDiscovery) e operações de liquidação (STR) (Araújo; Barbosa, 2023). Mesmo com a descontinuação do projeto oficial, essa infraestrutura tecnológica permanece relevante para aplicações em contextos como a assistência estudantil, onde transparência e rastreabilidade são essenciais.

1.2 Justificativa

A relevância desta pesquisa fundamenta-se em três pilares principais. Primeiro, existe uma necessidade prática evidente: os sistemas atuais de assistência estudantil, embora cumpram seu papel social, apresentam melhorias visíveis a serem implementadas relacionadas principalmente à transparência do gasto de dinheiro público. Segundo, o momento é oportuno: o Brasil desenvolveu, através do piloto DREX, uma infraestrutura tecnológica de moeda digital que, mesmo descontinuada oficialmente, criou conhecimento e ferramentas que permitem experimentação e inovação. Terceiro, o potencial de impacto é significativo: uma solução bem-sucedida poderia servir como modelo para outros programas de assistência social no país, demonstrando que a tecnologia desenvolvida durante o piloto DREX possui aplicabilidade além de seu propósito original.

1.3 Problema de Pesquisa

Este trabalho busca responder à seguinte questão de pesquisa: Como implementar um sistema de assistência estudantil utilizando a arquitetura e os contratos inteligentes desenvolvidos no contexto do piloto DREX para automatizar processos de distribuição de benefícios e aumentar a transparência na aplicação dos recursos?

1.4 Objetivos

1.4.1 Objetivo Geral

Implementar e validar tecnicamente um sistema de assistência estudantil baseado nos contratos inteligentes e arquitetura desenvolvidos no contexto do piloto DREX, utilizando tec-

nologia blockchain para automatizar a distribuição de benefícios e proporcionar maior transparência e rastreabilidade na gestão dos recursos.

1.4.2 Objetivos Específicos

- Implementar a infraestrutura blockchain utilizando os contratos oficiais do piloto DREX disponibilizados pela Wire Labs em ambiente de desenvolvimento;
- Desenvolver um contrato inteligente (StudentAssistanceVault) que implemente funcionalidades específicas para gestão de assistência estudantil;
- Criar um sistema de indexação blockchain para monitoramento e análise das transações realizadas no sistema;
- Implementar uma API completa para integração com sistemas universitários existentes;
- Desenvolver um processo de implantação automatizado que facilite a replicação e teste do sistema;
- Validar tecnicamente a solução através de testes integrados e demonstração funcional.

1.5 Escopo do Trabalho

A implementação desenvolvida inclui:

- **Infraestrutura Blockchain Completa**: Implantação funcional dos contratos oficiais do piloto DREX (RealDigital, AddressDiscovery, KeyDictionary, STR) disponibilizados pela Wire Labs;
- **Smart Contract Principal**: StudentAssistanceVault com funcionalidades de registro de estudantes, distribuição em lotes, transferências e controles de acesso;
- **Sistema de Indexação**: Indexador blockchain implementado com Ponder para monitoramento de eventos em tempo real;
- **API Backend Completa**: Servidor com múltiplos endpoints para gestão de estudantes, beneficiários, tipos de despesa e autenticação;
- **Interface Web Parcial**: Dashboard administrativo com funcionalidades básicas implementadas;
- **Implantação Automatizada**: Roteiros e configuração Docker para implantação completa do sistema;
- **Suite de Testes**: Testes unitários e de integração validando as funcionalidades críticas do sistema.

1.6 Contribuições Técnicas

Este trabalho apresenta contribuições práticas relevantes para a área:

- **Aplicação Prática da Arquitetura do Piloto DREX**: Primeira implementação conhecida de um sistema de assistência social utilizando os contratos e a arquitetura desenvolvidos durante o piloto DREX;
- **Arquitetura Vault**: Desenvolvimento de uma arquitetura de custódia (vault) para gestão segura de recursos estudantis;
- **Solução de Escalabilidade**: Implementação de distribuição em lotes para otimizar custos de *gas* (taxa de processamento na rede blockchain) em operações com múltiplos beneficiários;
- **Indexação Especializada**: Sistema de monitoramento específico para transações de assistência estudantil;
- **Implantação Reproduzível**: Ambiente completamente automatizado que facilita replicação e experimentação por outros pesquisadores.

O código-fonte completo do projeto está disponível publicamente, contribuindo para a transparência da pesquisa e facilitando trabalhos futuros na área.

1.7 Estrutura do Trabalho

Este trabalho está organizado da seguinte forma:

- O Capítulo 2 apresenta a fundamentação teórica, abordando conceitos essenciais sobre blockchain, moedas digitais de bancos centrais (CBDCs), o piloto DREX e o funcionamento do Programa Nacional de Assistência Estudantil (PNAES);
- O Capítulo 3 descreve a metodologia adotada para o desenvolvimento do trabalho, incluindo a abordagem de pesquisa, as ferramentas utilizadas e os procedimentos metodológicos;
- O Capítulo 4 detalha a implementação da infraestrutura blockchain baseada nos contratos do piloto DREX, explicando a configuração do ambiente de desenvolvimento e a integração com os contratos oficiais;
- O Capítulo 5 apresenta o sistema de assistência estudantil desenvolvido, incluindo a arquitetura do StudentAssistanceVault, o sistema de indexação e as APIs implementadas;
- O Capítulo 6 expõe os resultados obtidos, demonstrando as funcionalidades implementadas, métricas de desempenho e validações técnicas realizadas;

- O Capítulo 7 apresenta as conclusões, resumindo as contribuições técnicas alcançadas, limitações identificadas e direções para trabalhos futuros.

2 Fundamentação Teórica

Para compreender como um sistema de assistência estudantil pode ser melhorado pela tecnologia blockchain, é necessário primeiro estabelecer uma base sólida sobre os conceitos que fundamentam esta proposta. Este capítulo constrói essa base de forma progressiva, conectando tecnologias emergentes com desafios práticos da assistência estudantil brasileira.

2.1 Programa Nacional de Assistência Estudantil (PNAES)

De acordo com o Decreto nº 7.234/2010, o PNAES é quem regulamenta e define diretrizes para ações de assistência estudantil nas instituições federais de ensino superior (República, 2010). Dentre seus principais objetivos, estão a democratização das condições de permanência dos jovens na educação superior pública federal, a minimização dos efeitos das desigualdades sociais e regionais, a redução das taxas de retenção e evasão e a contribuição para a promoção da inclusão social pela educação.

Vale ressaltar que o PNAES atua em dez áreas prioritárias, sendo elas:

1. Moradia estudantil;
2. Alimentação;
3. Transporte;
4. Atenção à saúde;
5. Inclusão digital;
6. Cultura;
7. Esporte;
8. Creche;
9. Apoio pedagógico;
10. Acesso, participação e aprendizagem de estudantes com deficiência.

2.2 Tecnologia Blockchain

Quando Satoshi Nakamoto publicou o *white paper* do Bitcoin em 2008, poucos poderiam prever que a tecnologia subjacente—o blockchain—transcenderia o universo das criptomoedas para transformar setores inteiros da economia (Nakamoto, 2008). Hoje, mais de uma década depois, essa tecnologia está sendo considerada para aplicações que vão desde cadeias de suprimentos até programas governamentais de assistência social, como mostra este trabalho.

Em essência, blockchain é um registro distribuído que funciona como um livro-razão digital compartilhado, onde cada "página"(bloco) contém um conjunto de transações e está criptograficamente ligada à página anterior. Esta estrutura aparentemente simples carrega consigo propriedades notáveis que a tornam adequada para cenários onde a confiança é um fator crítico.

2.2.1 Características Fundamentais

As principais características que tornam o blockchain uma tecnologia disruptiva incluem (Nakamoto, 2008):

- **Descentralização**: Não há uma autoridade central controlando a rede, sendo mantida por múltiplos participantes (servidores validadores);
- **Imutabilidade**: Uma vez que uma transação é confirmada e adicionada ao blockchain, ela se torna praticamente impossível de ser alterada devido à necessidade de modificar todos os blocos subsequentes e obter consenso da maioria dos validadores da rede;
- **Transparência**: Todas as transações são visíveis para todos os participantes da rede;
- **Consenso**: Mecanismos que garantem que todos os nós concordem sobre o estado atual da rede;
- **Criptografia**: Utilização de funções hash criptográficas para garantir a integridade dos dados.

2.2.2 Tipos de Blockchain

Existem diferentes tipos de blockchain, classificados principalmente pela forma de acesso e controle (Antonopoulos; Wood, 2018):

- **Blockchain Público**: Aberto para qualquer pessoa participar, como Bitcoin e Ethereum;
- **Blockchain Privado**: Restrito a um grupo específico de participantes;
- **Blockchain Consórcio**: Controlado por um grupo pré-selecionado de organizações;
- **Blockchain Híbrido**: Combina elementos de blockchains públicos e privados.

2.3 Contratos Inteligentes

Os contratos inteligentes (*smart contracts*) são programas de computador que executam automaticamente os termos de um acordo quando condições predefinidas são atendidas. O conceito foi originalmente proposto por Nick Szabo em 1994 (Szabo, 1997), mas ganhou implementação prática com o advento da plataforma Ethereum (Buterin, 2014).

2.3.1 Características dos Contratos Inteligentes

- **Autonomia**: Executam automaticamente sem necessidade de intermediários;
- **Precisão**: Eliminam ambiguidades através de código determinístico;
- **Transparência**: O código é visível e auditável por todos os participantes;
- **Eficiência**: Reduzem custos e tempo de execução de contratos;
- **Confiabilidade**: Impossibilidade de alteração após *deployment* (implantação). Em condições normais, uma vez implantado, o contrato não pode ser modificado, garantindo que as regras estabelecidas permaneçam inalteradas. Em situações excepcionais, como vulnerabilidades críticas de segurança, pode ser necessário substituir o contrato por uma nova versão.

2.3.2 Plataforma Ethereum

O Ethereum, proposto por Vitalik Buterin em 2014, é uma plataforma blockchain que permite a criação e execução de contratos inteligentes. Utiliza uma máquina virtual (Ethereum Virtual Machine - EVM) que executa código em uma linguagem de programação específica, principalmente Solidity (Buterin, 2014).

2.3.3 Gas: Medida de Custo Computacional

O *gas* é uma unidade de medida fundamental em blockchains compatíveis com a Ethereum Virtual Machine (EVM), incluindo a infraestrutura do DREX. Cada operação executada na EVM—seja uma operação aritmética simples, leitura de dados ou execução de um contrato inteligente—consome uma quantidade específica de *gas* (Antonopoulos; Wood, 2018).

O conceito de *gas* serve a três propósitos principais:

- **Prevenção de ataques**: Impede que programas maliciosos ou ineficientes consumam recursos computacionais excessivos da rede. Como cada operação tem um custo, loops infinitos ou operações excessivamente complexas tornam-se economicamente inviáveis;
- **Medida de complexidade**: Fornece uma métrica objetiva e determinística da complexidade computacional de uma operação, independente do hardware específico que a executa. Uma operação que consome 100.000 *gas* tem o mesmo custo computacional relativo em qualquer nó da rede;
- **Alocação de recursos**: Permite que a rede priorize transações com base em sua urgência e importância, garantindo uso eficiente dos recursos computacionais distribuídos.

Importante destacar: *gas* não mede tempo de execução ou poder de processamento (como FLOPS—operações de ponto flutuante por segundo). Uma operação que consome 50.000

gas representa uma quantidade fixa de trabalho computacional que pode ser executada rapidamente em hardware potente ou mais lentamente em hardware modesto, mas o custo em *gas* permanece constante.

Em redes públicas como Ethereum, usuários pagam por *gas* usando a criptomoeda nativa da rede. No entanto, em redes permissionadas como a infraestrutura do DREX, o modelo de custos é diferente: os participantes autorizados da rede (instituições financeiras e Banco Central) mantêm a infraestrutura sem cobrança direta de *gas* aos usuários finais. Neste contexto, *gas* serve primariamente como métrica de eficiência e consumo de recursos computacionais, não como custo financeiro direto.

2.4 Moedas Digitais de Bancos Centrais (CBDCs)

Uma *Central Bank Digital Currency* (CBDC) é uma forma digital da moeda de um país, emitida e regulada pelo banco central nacional. Diferentemente das criptomoedas descentralizadas, as CBDCs mantêm o controle centralizado da autoridade monetária (Auer; Haene; Holden, 2021), oferecendo:

- **Curso legal**: Status oficial como moeda nacional;
- **Controle centralizado**: Emissão e política monetária controladas pelo banco central;
- **Programabilidade**: Capacidade de implementar políticas através de código;
- **Rastreabilidade**: Possibilidade de monitoramento de transações para fins regulatórios;
- **Inclusão financeira**: Acesso facilitado a serviços financeiros digitais.

2.5 DREX - Real Digital Brasileiro

O DREX (Digital Real EXperimental) é o projeto do Banco Central do Brasil para desenvolvimento de uma moeda digital nacional. Lançado oficialmente em 2023, o projeto visa criar uma infraestrutura para a tokenização de ativos e implementação de uma CBDC brasileira (Brasil, 2023). Os principais objetivos do projeto DREX incluem a modernização do sistema financeiro nacional, a redução dos custos de transações, o aumento da eficiência do sistema de pagamentos, a facilitação do processo de implementação de políticas públicas e o aumento da inclusão financeira.

2.5.1 Arquitetura Técnica

O DREX utiliza tecnologia de registro distribuído (DLT) baseada em uma rede blockchain permissionada, ou seja, seu acesso é controlado por instituições autorizadas pelo Banco Central. Suas principais características técnicas incluem:

- **Contratos Inteligentes**: Suporte para programabilidade e automação de processos;
- **Privacidade**: Implementação de tecnologias que preservam a privacidade das transações;
- **Interoperabilidade**: Capacidade de integração com sistemas existentes;
- **Escalabilidade**: Arquitetura preparada para alto volume de transações.

2.5.2 Fases de Desenvolvimento

O projeto DREX está sendo desenvolvido em fases:

1. **Fase Piloto**: Testes com instituições financeiras selecionadas;
2. **Fase de Expansão**: Ampliação para mais participantes e casos de uso;
3. **Implementação Completa**: Lançamento para o público geral.

2.6 Arquiteturas de Distribuição em Blockchain

A implementação de sistemas de distribuição de recursos em blockchain envolve decisões arquiteturais fundamentais que impactam diretamente a escalabilidade, segurança e custos operacionais do sistema (Santos, 2023).

2.6.1 Arquitetura Vault para Controle de Custódia

Este trabalho adota uma arquitetura de custódia (vault) onde os fundos permanecem sob controle de um contrato inteligente, que gerencia saldos virtuais para os beneficiários. Nesta abordagem, os recursos não são transferidos diretamente para carteiras individuais dos estudantes, mas sim mantidos em custódia, com o contrato registrando os saldos de cada beneficiário.

A escolha por esta arquitetura fundamenta-se na necessidade de manter controle administrativo completo sobre os recursos, permitindo:

- **Gestão de políticas de uso**: Implementação de regras sobre como e onde os recursos podem ser utilizados, com lista de receptores autorizados;
- **Ajustes administrativos**: Capacidade de corrigir benefícios, cancelar distribuições ou ajustar saldos sem necessidade de cooperação dos beneficiários;
- **Auditoria centralizada**: Todos os saldos e transações permanecem em um único contrato, facilitando a geração de relatórios e prestação de contas;
- **Controle de acesso**: Possibilidade de suspender ou revogar acesso aos fundos em casos específicos, mantendo a conformidade com políticas institucionais.

Esta abordagem é particularmente adequada para programas de assistência estudantil, onde há necessidade de garantir que os recursos sejam utilizados conforme as finalidades previstas (alimentação, moradia, transporte, etc.) e onde a instituição mantém responsabilidade sobre a correta aplicação dos recursos públicos.

2.7 Tecnologias de Indexação Blockchain

A monitoração e análise de eventos blockchain requerem infraestrutura especializada para captura, processamento e armazenamento de dados.

2.7.1 Necessidade de Indexação

Blockchains armazenam dados de forma otimizada para consenso e verificação, não para consultas complexas. Para aplicações que necessitam de análise de dados históricos, relatórios ou dashboards em tempo real, é essencial implementar sistemas de indexação que:

- Capturam eventos emitidos pelos contratos inteligentes;
- Processam e estruturam os dados para consultas eficientes;
- Mantêm sincronização com o estado atual da blockchain;
- Oferecem APIs para acesso aos dados processados.

Esta fundamentação técnica estabelece os conceitos necessários para compreender as decisões arquiteturais e tecnológicas apresentadas nos próximos capítulos.

3 Metodologia

Este capítulo detalha as escolhas metodológicas realizadas e justifica as decisões tomadas durante o desenvolvimento da pesquisa.

3.1 Caracterização da Pesquisa

Esta pesquisa caracteriza-se como aplicada, de natureza exploratória e abordagem quali-quantitativa. A escolha por uma pesquisa aplicada fundamenta-se no objetivo de desenvolver uma solução prática para problemas reais enfrentados pelos sistemas de assistência estudantil. O caráter exploratório justifica-se pela relativa novidade da aplicação de tecnologias blockchain em programas de assistência social no contexto brasileiro.

A abordagem quali-quantitativa foi adotada por permitir tanto a análise de aspectos técnicos mensuráveis (como custos de transação, tempos de processamento e *throughput* do sistema) quanto a validação técnica da solução através de testes integrados e demonstração funcional.

Do ponto de vista dos procedimentos técnicos, a pesquisa combina elementos de pesquisa bibliográfica, estudo de caso e desenvolvimento experimental. A pesquisa bibliográfica forneceu a fundamentação teórica necessária sobre blockchain, CBDCs e assistência estudantil. O desenvolvimento experimental resultou na implementação do protótipo funcional apresentado neste trabalho.

3.2 Procedimentos Metodológicos

O desenvolvimento deste trabalho seguiu uma sequência estruturada de oito etapas, cada uma com objetivos específicos e critérios de avaliação.

A primeira etapa consistiu na análise dos sistemas atuais de assistência estudantil e estudo aprofundado da arquitetura DREX, incluindo a análise dos códigos-fonte disponibilizados pela Wire Labs e a compreensão dos contratos oficiais do projeto. Esta análise baseou-se tanto em fontes bibliográficas quanto na observação empírica dos processos existentes.

A segunda etapa envolveu o desenvolvimento da infraestrutura blockchain, incluindo a configuração do ambiente de rede permissionada, implantação dos contratos base do DREX e configuração do ambiente de desenvolvimento containerizado.

A terceira etapa consistiu no desenvolvimento do contrato inteligente principal do sistema, incluindo a implementação de funcionalidades de registro de estudantes, distribuição de benefícios e transferências de recursos.

A quarta etapa envolveu o desenvolvimento da camada de backend, incluindo a implementação de API RESTful completa, sistema de autenticação e integração com a blockchain.

A quinta etapa compreendeu a implementação do sistema de indexação blockchain para monitoramento de eventos e sincronização com banco de dados relacional.

A sexta etapa consistiu no desenvolvimento da interface web, com foco nas funcionalidades administrativas e de visualização para gestão do sistema.

A sétima etapa envolveu a implementação de testes automatizados abrangentes em três níveis: testes unitários para contratos inteligentes cobrindo todas as funcionalidades críticas, testes de integração para validação dos endpoints da API, e testes de fluxos completos do sistema incluindo autenticação, gestão de estudantes e receptores.

3.3 Ambiente de Desenvolvimento do Protótipo

O ambiente de desenvolvimento foi estruturado para simular as condições de operação do DREX considerando as limitações de não ter acesso à rede oficial durante o período de desenvolvimento. Foi utilizada uma rede blockchain local configurada para desenvolvimento, com sistema de indexação para persistência de dados e consultas eficientes. O ambiente foi containerizado para garantir reprodutibilidade em diferentes sistemas.

3.4 Coleta e Análise de Dados

A coleta de dados deste trabalho abrangeu tanto fontes primárias quanto secundárias. As fontes primárias incluíram dados gerados pelo próprio protótipo durante os testes, logs de execução dos contratos inteligentes e observações qualitativas sobre o comportamento do sistema.

As fontes secundárias compreenderam documentação oficial do projeto DREX disponibilizada pela Wire Labs, literatura acadêmica sobre blockchain e CBDCs, relatórios governamentais sobre assistência estudantil e casos de uso internacionais de blockchain em programas sociais.

A análise dos dados técnicos focou na verificação da funcionalidade dos contratos inteligentes, confirmação de eventos blockchain e validação da integração entre os componentes do sistema. Os dados coletados são apresentados no capítulo de resultados.

3.5 Validação e Testes

A validação da implementação foi realizada através de testes automatizados em três níveis: testes unitários de contratos inteligentes, testes de integração da API e validação manual da interface web.

Os testes de contratos inteligentes cobriram cenários completos de implantação, registro de estudantes, distribuição de benefícios, transferências, operações administrativas e validações de segurança. Foram implementados cenários de falha para garantir o comportamento correto tanto em condições normais quanto em situações de erro.

Os testes de integração da API validaram todos os endpoints implementados, cobrindo autenticação, gestão de estudantes, gestão de receptores e fluxos completos de integração. A

suite de testes utiliza configuração de banco de dados isolada com limpeza automática entre testes.

Adicionalmente, foi realizada validação manual através da interface web para verificar a usabilidade do sistema e o comportamento da aplicação em cenários reais de uso. Esta validação manual incluiu testes exploratórios das funcionalidades administrativas e do portal do estudante, permitindo identificar aspectos de experiência do usuário que não são capturados por testes automatizados.

4 Arquitetura e Implementação do Piloto DREX

Este capítulo apresenta a arquitetura conceitual e técnica desenvolvida no contexto do piloto DREX, detalhando os componentes utilizados neste trabalho com base nos contratos oficiais disponibilizados pela Wire Labs em parceria com o Banco Central do Brasil.

4.1 Visão Geral da Arquitetura do Piloto DREX

O piloto DREX (Digital Real EXperimental) representou uma implementação brasileira de moeda digital de banco central (CBDC), desenvolvida com base em tecnologia blockchain permissionada. A arquitetura foi projetada para atender aos requisitos específicos do sistema financeiro brasileiro, garantindo segurança, escalabilidade e conformidade regulatória durante o período de testes e validação.

4.1.1 Modelo Arquitetural de Duas Camadas

A arquitetura do piloto DREX implementava um modelo de duas camadas que separa as responsabilidades entre o Banco Central e as instituições financeiras participantes. Este modelo, segundo as diretrizes publicadas pelo Banco Central (Brasil, 2023), visava replicar digitalmente a estrutura do sistema financeiro tradicional, onde o Banco Central emite moeda e as instituições financeiras intermediam o relacionamento com os clientes finais.

A primeira camada, controlada exclusivamente pelo Banco Central, era responsável pela emissão e gestão da moeda digital oficial através do contrato RealDigital. Esta camada funcionava de forma análoga às reservas bancárias tradicionais, representando o dinheiro que as instituições financeiras mantêm depositado no Banco Central, porém em formato digital e registrado em blockchain.

A segunda camada, operada pelas instituições financeiras autorizadas, permitia a criação de tokens lastreados em Real Digital através do contrato RealTokenizado. Cada instituição participante possuía seu próprio contrato de tokenização, vinculado à sua identificação (CNPJ) e carteira de reservas específica. Esta separação de camadas garantia que o Banco Central mantivesse o controle monetário enquanto as instituições financeiras ofereciam serviços aos clientes finais.

4.1.2 Componentes Principais da Infraestrutura

A infraestrutura do piloto DREX era composta por cinco contratos inteligentes fundamentais, cada um com responsabilidades específicas no ecossistema. Destes, quatro foram efetivamente utilizados na implementação do sistema de assistência estudantil desenvolvido neste trabalho:

- **RealDigital**: Implementação do token DREX como moeda digital oficial do Banco Central;

- **RealTokenizado**: Representação tokenizada do Real Digital emitida por instituições financeiras;
- **STR (Sistema de Transferência de Reservas)**: Gerenciamento de liquidações interbancárias;
- **AddressDiscovery**: Descoberta e validação de endereços de participantes na rede;
- **KeyDictionary**: Gerenciamento de chaves tipo PIX (não utilizado neste trabalho).

4.2 Arquitetura Conceitual dos Contratos do Piloto DREX

Esta seção detalha o propósito e funcionamento conceitual de cada componente da arquitetura do piloto DREX, explicando como os contratos inteligentes se integravam para formar um sistema completo de moeda digital.

4.2.1 RealDigital: A Moeda Digital do Banco Central

O contrato RealDigital representava a implementação da moeda digital oficial do Banco Central do Brasil na blockchain. Conceitualmente, este contrato funcionava como uma versão digital das reservas bancárias, ou seja, o dinheiro que as instituições financeiras mantêm depositado no Banco Central para realizar operações interbancárias e garantir liquidez ao sistema.

O RealDigital era um token compatível com o padrão ERC-20, amplamente utilizado em blockchains compatíveis com Ethereum, mas incorporava funcionalidades adicionais específicas para CBDCs. Entre essas funcionalidades destacavam-se:

Emissão Controlada: Apenas o Banco Central, através de carteiras autorizadas com a permissão de "emissor", podia criar novos tokens RealDigital. Este mecanismo garantia o monopólio estatal sobre a emissão monetária, replicando digitalmente o modelo do sistema monetário tradicional.

Resgate de Moeda: O processo inverso à emissão, permitindo a destruição de tokens RealDigital quando instituições financeiras convertiam moeda digital de volta para reservas tradicionais. Esta funcionalidade era restrita a carteiras com permissão de "resgatador".

Congelamento de Fundos: Capacidade de congelar saldos de endereços específicos em situações excepcionais, como suspeita de atividades ilícitas ou determinações judiciais. Esta função era exclusiva de administradores autorizados pelo Banco Central.

Controle de Acesso Baseado em Papéis: Sistema hierárquico de permissões que definia claramente quem poderia executar cada tipo de operação, garantindo governança adequada sobre a moeda digital.

A principal característica do RealDigital era sua paridade garantida com o Real tradicional na proporção 1:1, assegurando estabilidade de valor e confiança no sistema. Diferentemente das criptomoedas privadas como Bitcoin, que apresentam alta volatilidade, o RealDigital possuía o respaldo integral do Estado brasileiro.

4.2.2 RealTokenizado: Tokens Institucionais Lastreados

O contrato RealTokenizado representava uma camada de abstração adicional no sistema, permitindo que instituições financeiras criassem seus próprios tokens digitais lastreados em Real Digital. Conceitualmente, se o RealDigital equivale às reservas no Banco Central, o RealTokenizado corresponde aos depósitos à vista que os clientes mantêm em seus bancos.

Cada instituição financeira participante do piloto DREX possuía seu próprio contrato RealTokenizado, identificado pelo CNPJ da instituição e vinculado a uma carteira de reservas específica. Esta arquitetura permitia que bancos como Banco do Brasil, Itaú ou Bradesco emitissem tokens próprios (por exemplo, RealTokenizado_BB, RealTokenizado_Itau), todos lastreados 1:1 em Real Digital mantido em suas carteiras de reserva.

As principais características do RealTokenizado incluíam:

Lastro Garantido: Cada token RealTokenizado emitido por uma instituição deve ter correspondente em Real Digital na carteira de reserva dessa instituição, garantindo a conversibilidade plena.

Identificação Institucional: O contrato armazenava informações da instituição emissora, incluindo nome, CNPJ (primeiros 8 dígitos) e endereço da carteira de reserva, permitindo rastreabilidade e auditoria.

Carteira de Reserva Atualizável: Capacidade de atualização da carteira de reserva pela administração da instituição, permitindo flexibilidade operacional mantendo a segurança.

Operações DvP (Delivery versus Payment): Suporte para operações de entrega contra pagamento, fundamentais para o mercado de capitais tokenizado, onde a transferência de um ativo digital ocorre simultaneamente ao pagamento.

Esta separação entre RealDigital e RealTokenizado replicava digitalmente a estrutura do sistema bancário de reservas fracionárias, onde o Banco Central mantém as reservas primárias e os bancos comerciais criam dinheiro secundário através dos depósitos à vista.

4.2.3 STR: Sistema de Transferência de Reservas Digital

O contrato STR (Sistema de Transferência de Reservas) simulava digitalmente o funcionamento do STR tradicional operado pelo Banco Central do Brasil. No sistema financeiro convencional, o STR é responsável pela liquidação de operações interbancárias de grande valor em tempo real. Na arquitetura do piloto DREX, este contrato desempenhava papel similar para a moeda digital.

Conceitualmente, o STR funcionava como um "caixa eletrônico do Banco Central"onde instituições financeiras autorizadas podiam solicitar a emissão (mint) ou o resgate (burn) de tokens RealDigital de suas próprias carteiras. Este mecanismo permitia que os bancos convertessem suas reservas tradicionais em moeda digital e vice-versa, mantendo a ponte entre o sistema financeiro tradicional e o digital.

As principais funcionalidades do STR incluíam:

Solicitação de Emissão: Instituições participantes podiam requerer a criação de novos tokens RealDigital em suas carteiras, equivalente ao processo de conversão de reservas tradicionais em moeda digital.

Solicitação de Resgate: Processo inverso, permitindo a destruição de tokens RealDigital e liberação de reservas tradicionais correspondentes.

Controle de Participantes: Apenas carteiras previamente autorizadas pelo Banco Central e registradas como participantes ativos podiam utilizar o STR, garantindo que somente instituições reguladas tivessem acesso.

Registro de Reservas: Manutenção de informações sobre os saldos de reserva de cada instituição, permitindo verificação de liquidez antes de aprovar transações.

Durante o piloto, as operações do STR não envolviam validações complexas de garantias ou verificações de limites operacionais, funcionando de forma simplificada para permitir testes e validação do modelo. Em uma implementação completa, este contrato incorporaria regras mais estritas de governança e controle de risco.

4.2.4 AddressDiscovery: Registro de Participantes da Rede

O contrato AddressDiscovery funcionava como um catálogo centralizado de endereços blockchain dos participantes autorizados da rede DREX. Sua função era análoga a uma "lista telefônica"que mapeia identificações conhecidas (como nomes de instituições) para endereços blockchain, facilitando a descoberta e validação de participantes.

Este componente era fundamental em uma rede permissionada, onde não é qualquer endereço que pode participar das operações, mas apenas aqueles previamente autorizados e registrados pelo Banco Central. As principais características incluíam:

Registro de Participantes: Armazenamento de informações sobre cada participante autorizado, incluindo nome da instituição, tipo de participante (banco, corretora, câmara de compensação, etc.), status de ativação e data de registro.

Validação de Endereços: Capacidade de verificar rapidamente se um determinado endereço blockchain pertence a um participante autorizado e ativo na rede.

Gestão de Ciclo de Vida: Possibilidade de ativar ou desativar participantes conforme mudanças no status regulatório ou operacional das instituições.

Consulta de Metadados: Outros contratos podiam consultar o AddressDiscovery para obter informações sobre participantes, permitindo implementar regras de negócio baseadas no tipo ou status do participante.

A centralização do registro de participantes no AddressDiscovery simplificava a atualização de informações, pois alterações cadastrais precisavam ser feitas em um único local, sendo automaticamente refletidas para todos os contratos que consultassem este serviço.

4.2.5 KeyDictionary: Sistema de Chaves PIX-like

O contrato KeyDictionary, último componente da infraestrutura oficial do piloto DREX, implementava funcionalidade similar ao DICT (Diretório de Identificadores de Contas Transacionais) do sistema PIX, permitindo que transferências fossem realizadas utilizando chaves amigáveis (CPF, telefone, e-mail) em vez de endereços blockchain complexos.

Este contrato não foi utilizado na implementação do sistema de assistência estudantil desenvolvido neste trabalho, pois o propósito da pesquisa era validar a transparência e eficiência das transações internas da blockchain, e não a integração com sistemas de chaves tipo PIX. As transferências no sistema de assistência estudantil são realizadas diretamente utilizando endereços blockchain, que são gerenciados internamente pela aplicação e não requerem a camada adicional de abstração de chaves amigáveis.

Para aplicações futuras que visem maior facilidade de uso para usuários finais, o KeyDictionary poderia ser integrado para permitir que estudantes recebessem benefícios usando seus CPFs como identificador, similar ao funcionamento atual do PIX. No entanto, para a validação técnica e de transparência proposta neste trabalho, a utilização direta de endereços blockchain mostrou-se suficiente e adequada.

4.3 Fluxo Operacional Integrado

Para compreender como os contratos do piloto DREX funcionavam de forma integrada, é útil visualizar o fluxo completo de uma operação, desde a emissão de moeda até o uso pelo cliente final:

Etapa 1 - Emissão pelo Banco Central: O Banco Central, através de carteira autorizada, utiliza o STR para solicitar a criação de tokens RealDigital. Estes tokens são creditados na carteira de reserva de uma instituição financeira participante.

Etapa 2 - Tokenização pela Instituição: A instituição financeira, possuindo saldo em RealDigital, emite tokens RealTokenizado em seu próprio contrato, lastreados 1:1 pelas reservas mantidas em RealDigital.

Etapa 3 - Distribuição aos Clientes: Os tokens RealTokenizado são transferidos para carteiras de clientes finais, que podem utilizá-los para transações comerciais ou transferências.

Etapa 4 - Transferências Entre Usuários: Clientes realizam transferências diretamente para endereços blockchain de outros usuários. O sistema consulta o AddressDiscovery para validar que os participantes são autorizados na rede. No piloto DREX completo, esta etapa poderia utilizar o KeyDictionary para permitir transferências usando chaves tipo PIX (CPF, telefone), mas para o propósito deste trabalho, as transferências diretas por endereço mostraram-se suficientes para validar transparência e eficiência.

Etapa 5 - Liquidação Interbancária: Quando transferências ocorrem entre clientes de instituições diferentes, há liquidação de reservas em RealDigital entre as carteiras das instituições através do STR.

Este fluxo demonstra como a arquitetura do piloto DREX replicava digitalmente todo o sistema financeiro tradicional, mantendo a hierarquia regulatória (Banco Central → Instituições Financeiras → Clientes) enquanto aproveitava os benefícios da tecnologia blockchain para transparência e automação.

4.4 Implementação Técnica no Trabalho

Para este trabalho, foi implementada uma versão funcional dos contratos do piloto DREX utilizando os códigos disponibilizados publicamente pela Wire Labs[1], empresa que replicou a arquitetura do piloto DREX para fins educacionais e de pesquisa seguindo a documentação oficial fornecida pelo Banco Central. A implementação focou em criar um ambiente de desenvolvimento completo que permitisse a validação técnica da integração entre os contratos do DREX e o sistema de assistência estudantil proposto.

4.4.1 Ambiente de Desenvolvimento com Hyperledger Besu

Para suportar o desenvolvimento e testes do sistema, foi implementado um ambiente blockchain completo utilizando Hyperledger Besu como nó blockchain. O Hyperledger Besu é uma implementação cliente Ethereum de código aberto, desenvolvida sob os auspícios da Hyperledger Foundation, adequada tanto para redes públicas quanto privadas permissionadas.

A escolha do Hyperledger Besu como plataforma blockchain fundamentou-se em várias características técnicas que o tornam apropriado para aplicações empresariais e governamentais:

Compatibilidade com Ethereum: O Besu é totalmente compatível com a Máquina Virtual Ethereum (EVM), permitindo a execução de contratos inteligentes escritos em Solidity, a mesma linguagem utilizada nos contratos oficiais do piloto DREX. Esta compatibilidade garante que os contratos funcionem exatamente como projetados, sem necessidade de adaptações.

Suporte a Redes Permissionadas: Diferentemente de blockchains públicas como Ethereum mainnet, o Besu suporta nativamente redes permissionadas onde apenas participantes autorizados podem validar transações. Esta característica é essencial para CBDCs, que por natureza exigem controle sobre quem pode participar da rede.

Mecanismos de Consenso Adequados: O Besu oferece múltiplos algoritmos de consenso, incluindo IBFT 2.0 (Istanbul Byzantine Fault Tolerance), adequado para redes permissionadas com número conhecido e limitado de validadores. Este mecanismo garante finalidade rápida das transações e resistência a falhas bizantinas.

Recursos de Privacidade: O Besu inclui recursos avançados de privacidade, como transações privadas e contratos privados, que podem ser utilizados quando necessário manter confidencialidade de certas operações sem comprometer a auditabilidade do sistema.

Ferramentas de Desenvolvimento: O ecossistema Besu inclui ferramentas robustas de desenvolvimento, teste e monitoramento, facilitando a implementação e depuração de aplicações

[1] Repositório dos contratos: https://github.com/wireshape/real-digital-smart-contracts

blockchain complexas.

A rede Besu foi configurada em modo de desenvolvimento com consenso IBFT 2.0, permitindo criação rápida de blocos (2 segundos) para facilitar testes e validações. O ambiente incluiu configurações de gênesis customizadas para estabelecer parâmetros iniciais da rede, como identificador de cadeia (chain ID), limites de gas e distribuição inicial de fundos.

4.4.2 Orquestração com Docker e Automação de Implantação

Para facilitar a configuração e replicação do ambiente de desenvolvimento, o sistema foi containerizado utilizando Docker e orquestrado através do Docker Compose, trazendo vantagens como isolamento de ambientes, reprodutibilidade, gerenciamento de dependências e facilidade de reset do ambiente.

4.4.3 Validação e Testes da Implementação

O ambiente implementado permitiu a realização de testes abrangentes da funcionalidade dos contratos do piloto DREX. A estratégia de testes adotada focou em validar componentes críticos através de testes automatizados dos contratos oficiais DREX em JavaScript utilizando a biblioteca Chai, validando funcionalidades básicas do RealDigital (emissão, queima, congelamento de saldos), do RealTokenizado (tokenização com lastro garantido), e do SwapOneStep (conversão entre tokens).

Os resultados destes testes demonstraram que a infraestrutura do piloto DREX é capaz de suportar operações financeiras com correção funcional adequada, mantendo as garantias de segurança e auditabilidade inerentes à tecnologia blockchain.

4.5 Monitoramento e Observabilidade

A operação eficaz de um sistema baseado em blockchain requer ferramentas adequadas de monitoramento e observabilidade. O ambiente implementado do piloto DREX utiliza eventos blockchain para rastreabilidade das operações.

4.5.1 Monitoramento de Contratos Inteligentes

Os contratos inteligentes do DREX emitem eventos que são capturados e monitorados através de indexadores blockchain. Estes eventos registram operações importantes como emissão de moeda, transferências entre participantes, e mudanças de configuração do sistema, fornecendo auditabilidade completa das operações. O sistema de indexação implementado processa estes eventos em tempo real, persistindo-os em banco de dados relacional para consultas eficientes e geração de relatórios.

4.6 Considerações de Segurança

A segurança é aspecto crítico em qualquer sistema que lide com recursos financeiros, especialmente quando envolve recursos públicos destinados a programas sociais. A implementação realizada neste trabalho incorpora múltiplas camadas de segurança para proteger o sistema contra diferentes tipos de ameaças.

4.6.1 Controle de Acesso Baseado em Papéis

A infraestrutura do piloto DREX implementa um sistema hierárquico de permissões que define claramente quais operações cada tipo de participante pode executar. Este modelo, conhecido como RBAC (Role-Based Access Control), é fundamental para garantir que apenas usuários autorizados realizem operações sensíveis.

Os principais papéis (roles) definidos na arquitetura DREX incluem:

Administrador de Sistema: Papel com maiores privilégios, capaz de configurar o sistema, adicionar novos participantes, pausar operações em emergências e realizar outras atividades administrativas críticas. Este papel é restrito a operadores do Banco Central ou autoridades equivalentes.

Emissor de Moeda: Papel que permite criar novos tokens RealDigital. Exclusivo do Banco Central, garante o monopólio estatal sobre a emissão monetária.

Instituição Financeira: Papel intermediário que permite tokenização (criação de RealTokenizado) e gestão de contas de clientes. Restrito a instituições financeiras devidamente autorizadas e reguladas.

4.6.2 Auditoria e Rastreabilidade

Uma das principais vantagens da tecnologia blockchain é a auditabilidade completa de todas as operações. Cada transação registrada na blockchain é permanentemente armazenada e pode ser auditada a qualquer momento, garantindo transparência e accountability.

O sistema mantém logs completos de todas as operações realizadas, incluindo não apenas as transações financeiras, mas também operações administrativas como cadastros, alterações de permissões e configurações do sistema. Estes logs incluem informações sobre quem realizou a operação, quando foi realizada, quais parâmetros foram utilizados e qual foi o resultado.

A imutabilidade da blockchain garante que registros históricos não podem ser alterados retroativamente, eliminando a possibilidade de fraudes como adulteração de registros contábeis. Qualquer tentativa de modificar transações passadas seria imediatamente detectável por qualquer participante da rede.

4.6.3 Validação e Verificação de Operações

Todos os contratos inteligentes implementam verificações rigorosas de entrada e validação de estado antes de executar operações. Estas verificações incluem:

Validação de Saldos: Antes de executar transferências, o sistema verifica se o remetente possui saldo suficiente, evitando operações que resultariam em saldos negativos.

Verificação de Permissões: Cada operação sensível verifica se o endereço solicitante possui as permissões necessárias, impedindo acessos não autorizados.

Validação de Parâmetros: Inputs fornecidos pelos usuários são validados quanto a formato, tipo e valores aceitáveis, prevenindo erros e potenciais explorações de vulnerabilidades.

Verificação de Estado: Operações verificam se o sistema está em estado apropriado para executá-las, por exemplo, impedindo operações durante períodos de manutenção ou quando contratos estão pausados.

4.6.4 Capacidade de Atualização Segura

Embora a imutabilidade dos contratos inteligentes seja geralmente desejável, situações podem surgir onde atualizações são necessárias para corrigir bugs, adicionar funcionalidades ou adaptar-se a mudanças regulatórias. A arquitetura DREX pode implementar mecanismos de upgradability que permitem atualizações controladas sem comprometer a segurança, como o padrão de proxy que separa a lógica de negócio (que pode ser atualizada) do armazenamento de dados (que permanece estável).

4.7 Síntese do Capítulo

Este capítulo apresentou a arquitetura conceitual e técnica do piloto DREX, detalhando os cinco contratos inteligentes que compõem a infraestrutura oficial (RealDigital, RealTokenizado, STR, AddressDiscovery e KeyDictionary) e explicando como funcionam de forma integrada para formar um sistema completo de moeda digital de banco central.

A implementação realizada utilizou os contratos oficiais disponibilizados pela Wire Labs, estabelecendo um ambiente de desenvolvimento baseado em Hyperledger Besu que fornece uma plataforma robusta e replicável para experimentação com a infraestrutura DREX. As considerações de segurança implementadas na arquitetura DREX, incluindo controle de acesso baseado em papéis, auditoria completa de operações e validações rigorosas, garantem que o sistema atende aos requisitos de proteção necessários para lidar com recursos financeiros.

5 Sistema de Assistência Estudantil

Este capítulo apresenta o sistema de assistência estudantil implementado sobre a infraestrutura DREX, detalhando sua arquitetura, implementação e funcionalidades. O sistema foi desenvolvido para otimizar a distribuição de recursos financeiros para estudantes universitários, garantindo transparência, eficiência e rastreabilidade através de uma implementação funcional composta por quatro componentes principais: smart contract StudentAssistanceVault, API backend, indexador blockchain e interface web administrativa.

5.1 Visão Geral do Sistema

O sistema de assistência estudantil implementado utiliza a tecnologia blockchain para criar um ambiente transparente e eficiente para a gestão de auxílios estudantis. Construído sobre a infraestrutura DREX, o sistema oferece:

- Distribuição automatizada de auxílios financeiros via DREX;
- Gestão centralizada através de API backend RESTful;
- Rastreabilidade completa através de indexador blockchain;
- Controle de acesso baseado em roles.

5.2 Arquitetura do Sistema

A arquitetura do sistema é composta por quatro componentes principais implementados:

5.2.1 Camada Smart Contract

- **StudentAssistanceVault**: Contrato principal de gestão;
- **Infraestrutura DREX**: Integração com contratos oficiais;
- **Hyperledger Besu**: Nó blockchain configurado.

5.2.2 Camada API Backend

- **Express.js Server**: API REST implementada com TypeScript;
- **Endpoints RESTful**: Cobertura para gestão estudantil e monitoramento;
- **Autenticação JWT**: Sistema de login para administradores e estudantes;
- **PostgreSQL**: Banco de dados para persistência.

5.2.3 Camada Indexação Blockchain

- **Ponder Framework**: Indexador especializado para eventos blockchain;
- **GraphQL API**: Interface automática para consulta de dados;
- **Monitoramento**: Sincronização em tempo real de eventos.

5.2.4 Camada Interface Web

- **React Dashboard**: Interface administrativa implementada com TypeScript;
- **Funcionalidades Ativas**: Login, gestão de estudantes, visualização de transações.

5.3 Contrato StudentAssistanceVault

O contrato StudentAssistanceVault é o componente central do sistema, responsável por gerenciar os recursos financeiros e a distribuição de auxílios. A arquitetura segue o padrão *Vault* (cofre), onde os recursos DREX permanecem sob custódia do contrato, permitindo que estudantes movimentem fundos enquanto mantêm-se elegíveis, mas garantindo controle administrativo total.

Importante: O vault opera diretamente com **RealDigital**, a moeda digital emitida pelo Banco Central (DREX), não com RealTokenizado (tokens institucionais). Isto significa que os recursos são depositados diretamente pelo Banco Central ou por instituições autorizadas que possuem RealDigital em suas reservas, similar ao modelo de repasses orçamentários tradicionais para programas de assistência estudantil.

O administrador pode remover estudantes e recuperar recursos não utilizados através de funções de emergência (`emergencyWithdraw` e `withdrawAll`), mantendo a governança institucional sobre os benefícios mesmo em ambiente descentralizado.

5.3.1 Estrutura Principal

O contrato implementa controle de acesso baseado em roles (ADMIN_ROLE), proteção contra reentrância e gerenciamento de estudantes através de estruturas e mapeamentos. Define constantes para limites do sistema: até 1.000 estudantes (MAX_STUDENTS) e distribuições em lotes de até 50 estudantes (MAX_BATCH_SIZE).

O limite de 1.000 estudantes foi estabelecido para prevenir custos de *gas* ilimitados em operações que iteram sobre a lista de estudantes. Com MAX_BATCH_SIZE=50, a distribuição completa requer no máximo 20 transações (1.000/50), mantendo os custos de *gas* dentro de limites aceitáveis. Este valor representa um equilíbrio entre capacidade operacional e viabilidade técnica, adequado para instituições de médio a grande porte, podendo ser expandido através de múltiplas instâncias do contrato ou ajuste após validação em produção. A estrutura completa do contrato está disponível no Apêndice A.1.

5.3.2 Funcionalidades Principais

5.3.2.1 Registro de Estudantes

O sistema permite o registro de estudantes com valor mensal definido através da função `registerStudent`. A função valida o endereço do estudante, verifica se não está registrado, respeita o limite máximo de estudantes e emite evento de auditoria. Implementação completa disponível no Apêndice A.2.

5.3.2.2 Depósito de Recursos DREX

O contrato permite o depósito de tokens DREX para distribuição através da função `depositDrex`, restrita a administradores. A função transfere tokens DREX para o contrato e emite evento para rastreabilidade. Ver Apêndice A.3 para código completo.

5.3.2.3 Distribuição em Lotes

A distribuição é feita em lotes através da função `distributeBatch` para otimizar o uso de gas. A função processa estudantes em faixas de índices, valida saldo disponível, calcula total do lote e distribui para todos os estudantes ativos. Implementação completa no Apêndice A.4.

5.3.2.4 Transferências de Estudantes

Os estudantes podem transferir seus recursos através da função `transfer`, que implementa proteção contra reentrância, valida saldos e emite eventos de auditoria. O contrato *StudentAssistanceVault* permite transferências para qualquer endereço válido, cabendo ao administrador monitorar as transações através dos eventos registrados na blockchain. A camada de servidor oferece validação adicional de recebedores cadastrados, mas estudantes que interagem diretamente com o contrato têm liberdade total de transferência. Código disponível no Apêndice A.5.

5.4 Sistema de Indexação

Para monitorar as transações dos estudantes, foi implementado um indexador utilizando Ponder, um framework especializado em indexação de eventos blockchain que oferece APIs GraphQL automáticas.

5.4.1 Configuração do Indexador

O indexador conecta-se à rede Besu local (porta 8545), monitora o contrato StudentAssistanceVault desde o bloco zero e persiste dados em PostgreSQL. A configuração completa está disponível no Apêndice B.1.

5.4.2 Eventos Indexados

Na implementação atual, o indexador monitora especificamente o evento `Transfer` do StudentAssistanceVault, processando transferências em tempo real e persistindo em banco de dados relacional. Cada transferência capturada registra endereços de origem e destino, valor, timestamp e número do bloco. Embora o contrato emita diversos outros eventos (StudentRegistered, BatchDistributed, StudentRemoved, etc.), estes não estão sendo indexados na versão atual, cabendo ao administrador consultar os logs da blockchain diretamente quando necessário. Manipulador do evento Transfer disponível no Apêndice B.2.

5.5 Servidor Backend e Sistema de Autenticacao

Para complementar a infraestrutura blockchain, foi desenvolvido um servidor backend completo utilizando Node.js, Express e Prisma ORM, fornecendo 29 endpoints RESTful para gerenciamento completo do sistema.

5.5.1 Arquitetura do Backend

O servidor backend foi estruturado seguindo principios de Clean Architecture e inclui:

- **29 Endpoints API**: Cobertura completa de todas as funcionalidades do sistema;
- **Sistema de Autenticacao**: JWT-based com suporte a roles (ADMIN, STUDENT);
- **Banco de Dados**: PostgreSQL com Prisma ORM para type-safety;
- **Middleware de Seguranca**: CORS, rate limiting e validacao de entrada;
- **Integração com Indexador**: Consulta de dados blockchain via API GraphQL;
- **Testes Automatizados**: Suíte completa de testes unitários e integração.

5.5.2 Sistema de Autenticacao

O sistema implementa autenticação robusta com suporte a dois tipos de usuários (ADMIN, STUDENT). Todos os usuários realizam login através de credenciais tradicionais (usuário e senha), com geração de tokens JWT para gerenciamento de sessões. O sistema diferencia permissões através de roles definidas no banco de dados e validadas no backend. Implementação detalhada disponível no Apêndice C.1.

5.5.3 APIs de Gerenciamento

O servidor fornece 29 endpoints RESTful completos para gerenciamento de todas as entidades do sistema, incluindo operações CRUD para estudantes, beneficiários e transações, com suporte a paginação, filtros e busca. Documentação completa das APIs disponível no Apêndice C.2.

5.6 Dashboard Web Administrativo

Foi desenvolvido um dashboard web completo utilizando React, TypeScript e Tailwind CSS para administração do sistema de assistência estudantil.

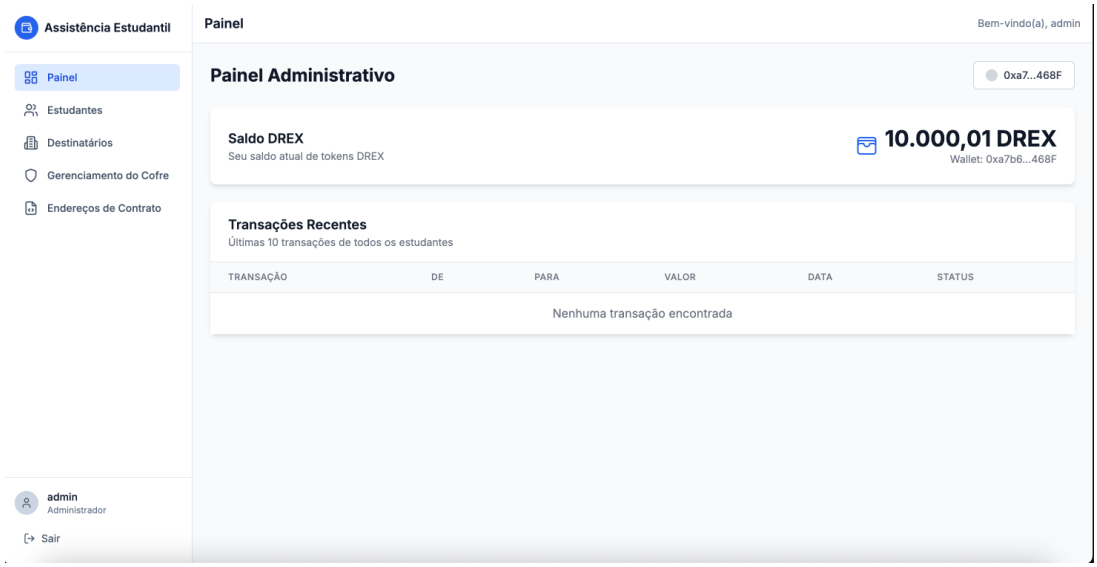
5.6.1 Funcionalidades do Dashboard

O dashboard fornece uma interface intuitiva para:

- **Visão Geral**: Métricas principais e estatísticas do sistema;
- **Gestão de Estudantes**: CRUD completo com busca e filtros;
- **Gestão de Beneficiários**: Cadastro e gerenciamento de estabelecimentos;
- **Monitoramento de Transações**: Histórico completo com paginação.

A Figura 1 apresenta a tela inicial do dashboard administrativo, com visão geral do sistema e métricas principais. A interface exibe informações consolidadas sobre estudantes cadastrados, saldos disponíveis e estatísticas de transações.

Figura 1 – Tela inicial do dashboard administrativo



Fonte: elaborado pelo autor

5.6.2 Arquitetura Frontend

O dashboard foi desenvolvido em React com TypeScript, utilizando hooks personalizados para gerenciamento de estado, autenticação JWT com persistência local, e componentes reutilizáveis para exibição de dados. A estrutura completa está documentada no Apêndice D.

5.7 Sistema de Testes Integrados

Para garantir a qualidade e confiabilidade do sistema, foi implementada uma suíte completa de testes de integração utilizando Jest e Supertest.

5.7.1 Estrutura de Testes

- **Testes de Saúde**: Verificação básica de funcionamento da API;
- **Testes de Autenticacao**: Validação completa do sistema de login;
- **Testes de Endpoints**: Cobertura de todas as APIs principais;
- **Testes de Integração**: Fluxos completos end-to-end.

5.7.2 Configuração de Ambiente de Teste

O ambiente de teste utiliza banco de dados PostgreSQL separado, geradores de dados de teste para usuários e estudantes, e utilitários para cleanup automático após execução. Configuração completa disponível no Apêndice E.1.

5.7.3 Cobertura de Testes

Os testes cobrem cenários críticos incluindo:

- Autenticacao de usuarios administrativos e estudantes;
- Criação e gerenciamento de usuarios com validacao de roles;
- Operações CRUD para todas as entidades;
- Validação de entrada e tratamento de erros;
- Fluxos de autenticação completos com JWT;
- Cleanup automático de dados de teste.

5.7.4 Esquema do Banco de Dados

O indexador utiliza tabelas onchain para persistir dados de transferências capturadas do evento Transfer. O esquema define campos, tipos e relacionamentos necessários para consultas eficientes. Ver Apêndice E.2 para esquema completo.

5.7.5 Manipulador de Eventos

O manipulador processa o evento Transfer da blockchain, extrai dados relevantes (endereços de origem e destino, valor, timestamp e bloco) e persiste no banco de dados com tratamento de erros e logging. Implementação completa no Apêndice E.3.

5.8 Implantação e Configuração

O sistema utiliza um roteiro automatizado para implantação completa da infraestrutura:

5.8.1 Roteiro de Implantação Principal

O sistema utiliza Docker Compose para orquestração de 5 serviços (Besu, PostgreSQL, Backend API, Indexador, Dashboard), com roteiro bash que automatiza compilação de contratos, implantação na blockchain, configuração de banco de dados e inicialização de todos os componentes. Roteiro completo disponível no Apêndice F.1.

5.8.2 Configuração de Estudantes de Teste

O sistema inclui configuração automática de estudantes para teste, criando wallets, registrando no contrato e populando banco de dados com dados realistas. Código de configuração disponível no Apêndice F.2.

5.9 Funcionalidades Implementadas

O sistema apresenta funcionalidades completas e operacionais, organizadas por perfil de usuário.

5.9.1 Portal Administrativo

As funcionalidades administrativas implementadas e validadas incluem:

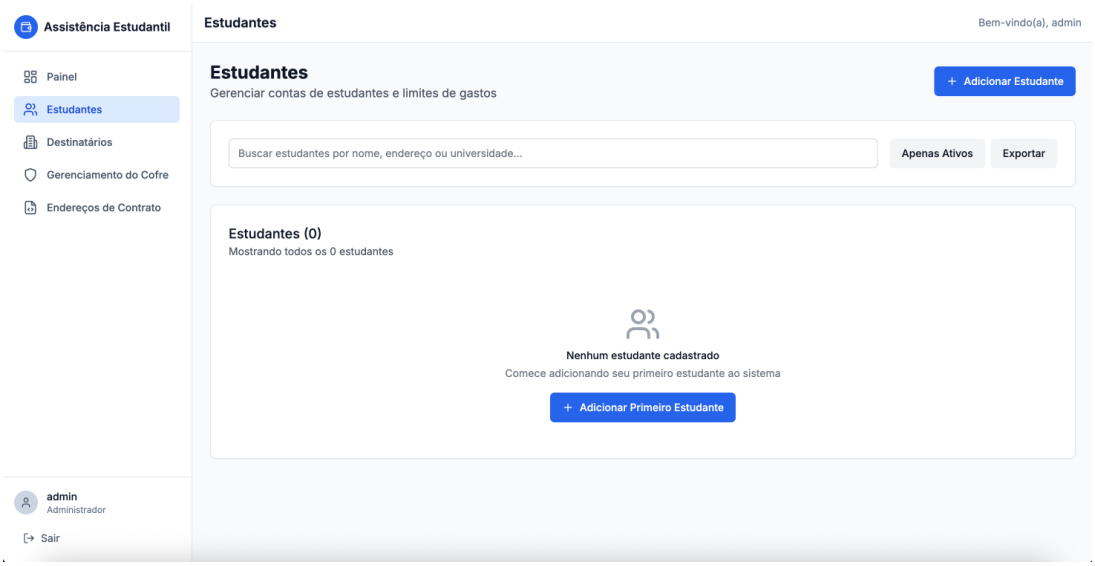
5.9.1.1 Gestão de Estudantes

- **Cadastro**: Registro de novos estudantes no sistema com informações pessoais e acadêmicas;
- **Inclusão na Vault**: Registro do estudante no contrato inteligente StudentAssistanceVault na blockchain;
- **Edição**: Atualização de informações cadastrais e valores de auxílio mensal;
- **Remoção**: Exclusão de estudantes do sistema quando necessário;
- **Visualização**: Lista completa de estudantes cadastrados com saldos atuais.

A Figura 2 mostra a tela de listagem de estudantes, onde é possível visualizar todos os estudantes cadastrados, incluindo contagem de transações reportadas para cada estudante. A Figura 3 apresenta a mesma tela com um estudante já cadastrado no sistema.

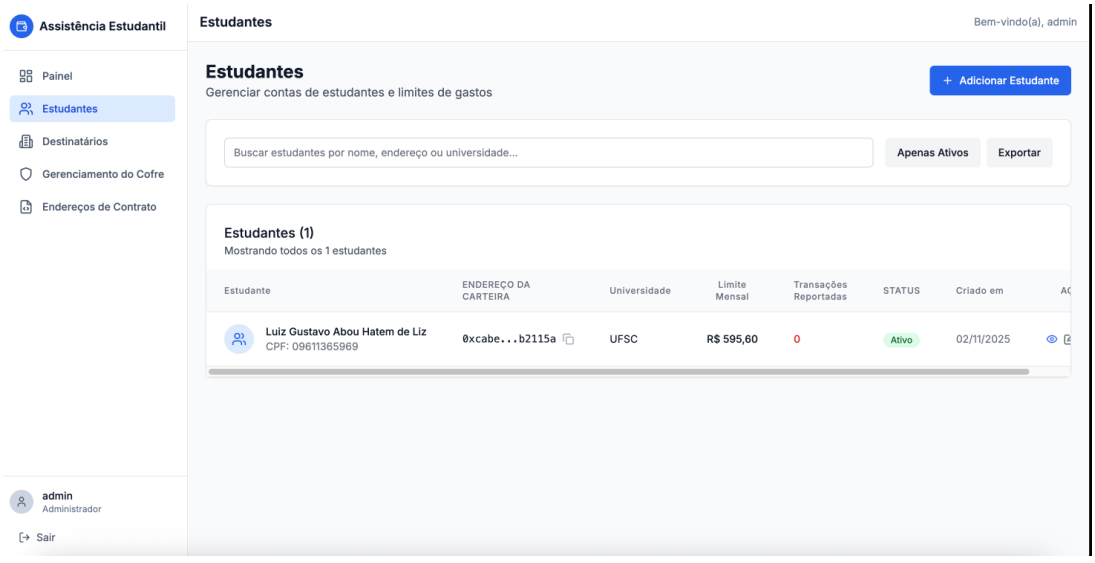
O processo de cadastro de estudantes é ilustrado nas Figuras 5 e 6. A primeira mostra o formulário vazio e a segunda apresenta o formulário com todos os campos preenchidos, incluindo informações pessoais, acadêmicas e o valor do auxílio mensal.

Figura 2 – Tela de listagem de estudantes no dashboard administrativo



Fonte: elaborado pelo autor

Figura 3 – Tela de estudantes com estudante cadastrado

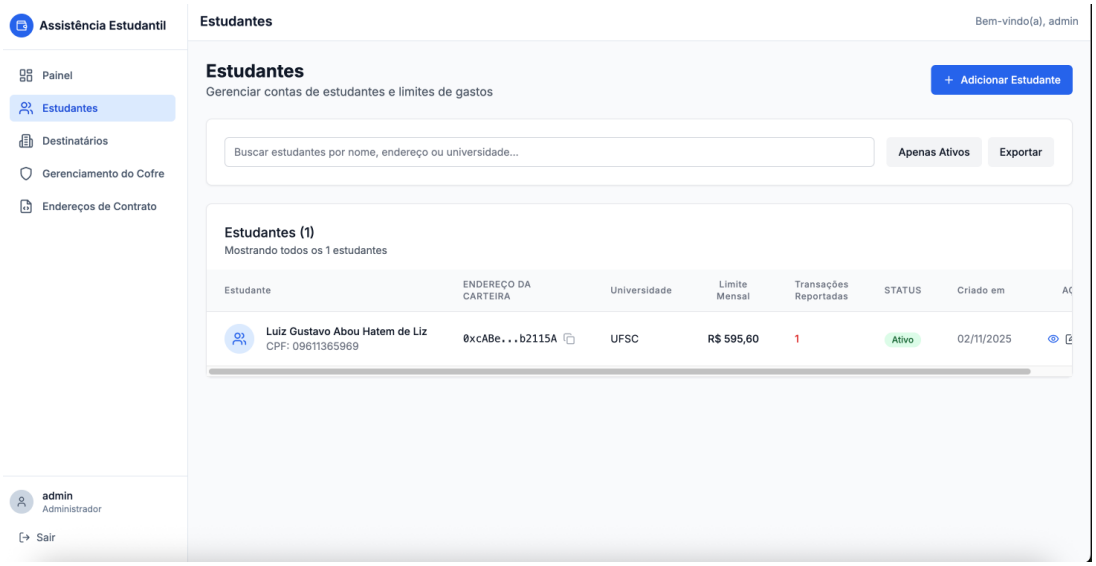


Fonte: elaborado pelo autor

5.9.1.2 Gestão Financeira

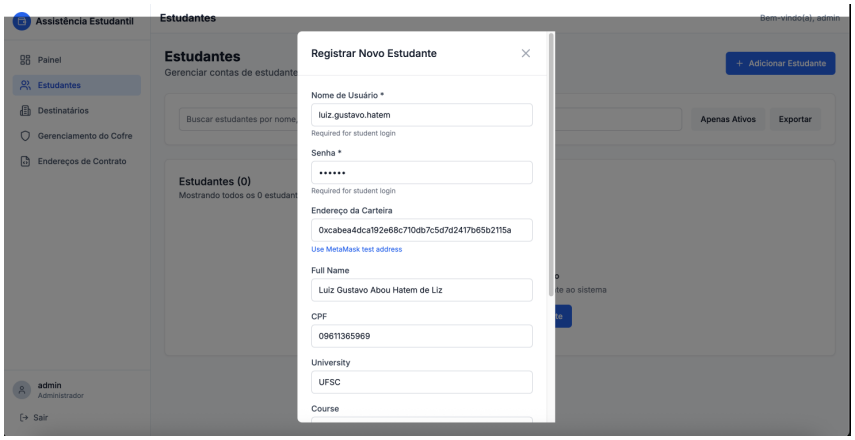
- **Depósito de Recursos**: Transferência de tokens DREX para o contrato vault;
- **Distribuição Total**: Distribuição automática de benefícios para todos os estudantes cadastrados;
- **Distribuição por Intervalo**: Distribuição para faixas específicas de estudantes (exemplo:

Figura 4 – Listagem de estudantes com contagem de transações reportadas



Fonte: elaborado pelo autor

Figura 5 – Formulário de registro de novo estudante



Fonte: elaborado pelo autor

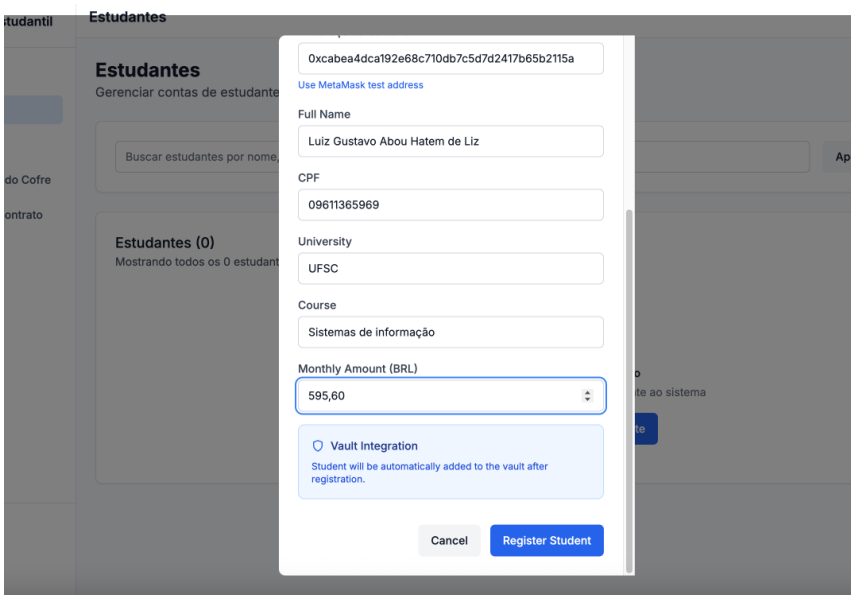
do índice 0 ao 10);

- **Validação Automática**: Verificação de fundos suficientes antes de cada distribuição.

A gestão financeira do sistema é realizada através da interface administrativa, conforme ilustrado nas Figuras 7, 8 e 9. A Figura 7 apresenta a tela de gerenciamento do cofre (vault), onde o administrador pode visualizar o saldo disponível e realizar operações financeiras.

A Figura 10 demonstra o processo de distribuição de benefícios para os estudantes cadastrados. O administrador pode escolher distribuir para todos os estudantes de uma vez ou para intervalos específicos, garantindo flexibilidade na gestão dos recursos.

Figura 6 – Formulário de registro preenchido



Fonte: elaborado pelo autor

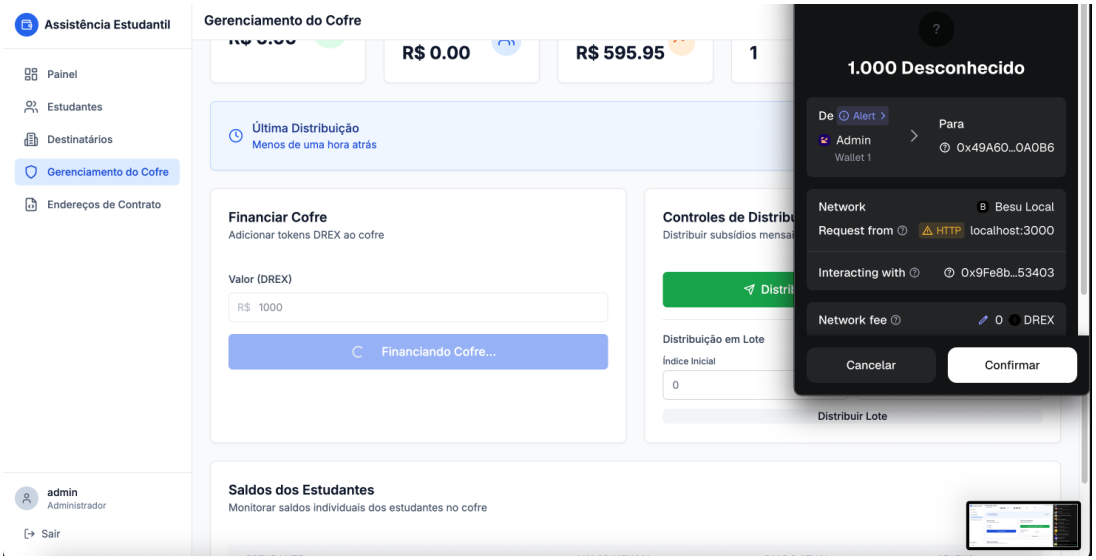
Figura 7 – Tela de gerenciamento do cofre (vault)

Fonte: elaborado pelo autor

5.9.1.3 Gestão de Recebedores

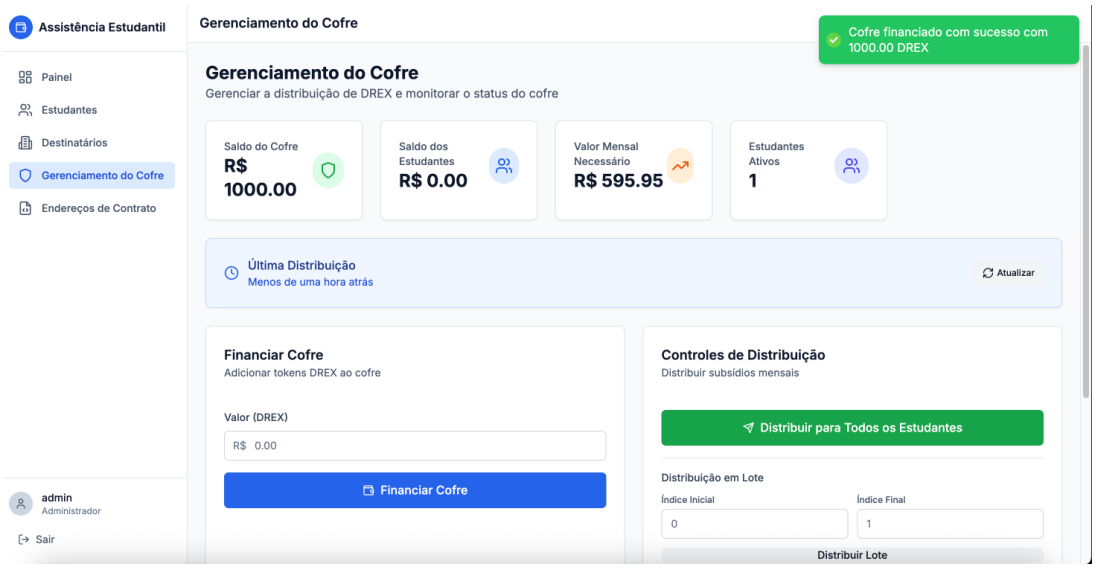
- **Cadastro**: Registro de estabelecimentos autorizados (restaurantes, livrarias, farmácias);
- **Edição**: Atualização de informações de recebedores cadastrados;
- **Remoção**: Exclusão de estabelecimentos quando necessário;

Figura 8 – Processo de financiamento do cofre com assinatura de transação



Fonte: elaborado pelo autor

Figura 9 – Cofre financiado com saldo disponível



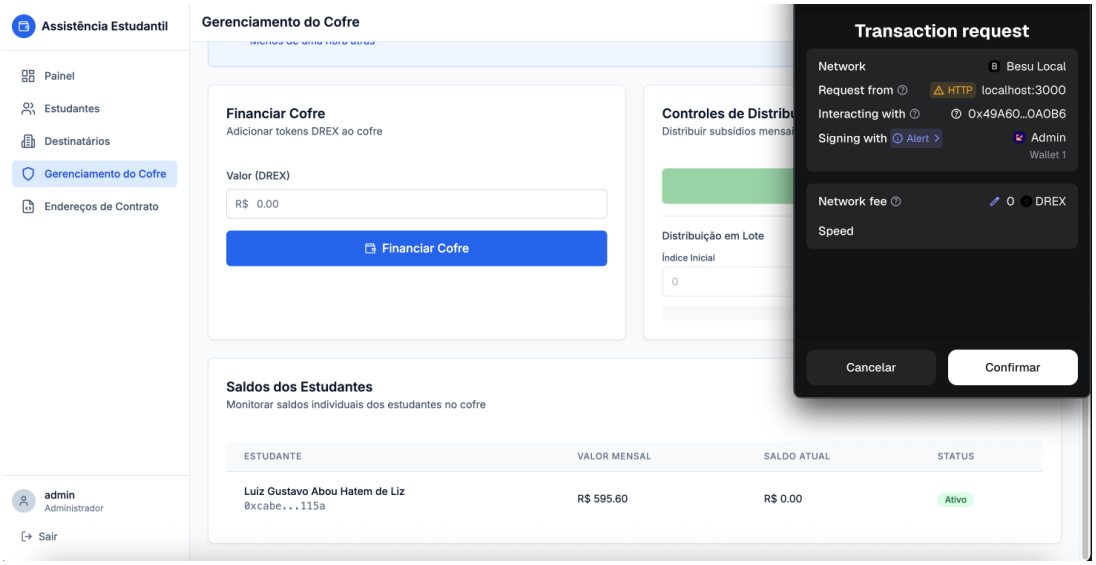
Fonte: elaborado pelo autor

- **Sistema de Reportes**: Sinalização de estabelecimentos com irregularidades.

A gestão de recebedores é uma funcionalidade essencial do sistema, permitindo o cadastro e monitoramento de estabelecimentos autorizados a receber recursos dos estudantes. A Figura 11 apresenta o formulário de cadastro de novos recebedores.

O sistema permite reportar recebedores que apresentem irregularidades, conforme ilustrado nas Figuras 13 e 14. Esta funcionalidade é importante para manter a integridade do sistema

Figura 10 – Distribuição de benefícios para estudantes



Fonte: elaborado pelo autor

e proteger os estudantes.

5.9.1.4 Monitoramento

- **Lista de Estudantes**: Visualização completa com saldos e status de cada estudante;
- **Histórico de Transações**: Registro detalhado de todas as operações do sistema (depósitos, distribuições, transferências) para auditoria em tempo real.

A Figura 15 apresenta a tela de monitoramento de transações do sistema, onde o administrador pode visualizar o histórico completo de todas as operações realizadas, incluindo depósitos, distribuições e transferências. Esta funcionalidade é essencial para auditoria e acompanhamento em tempo real das movimentações financeiras.

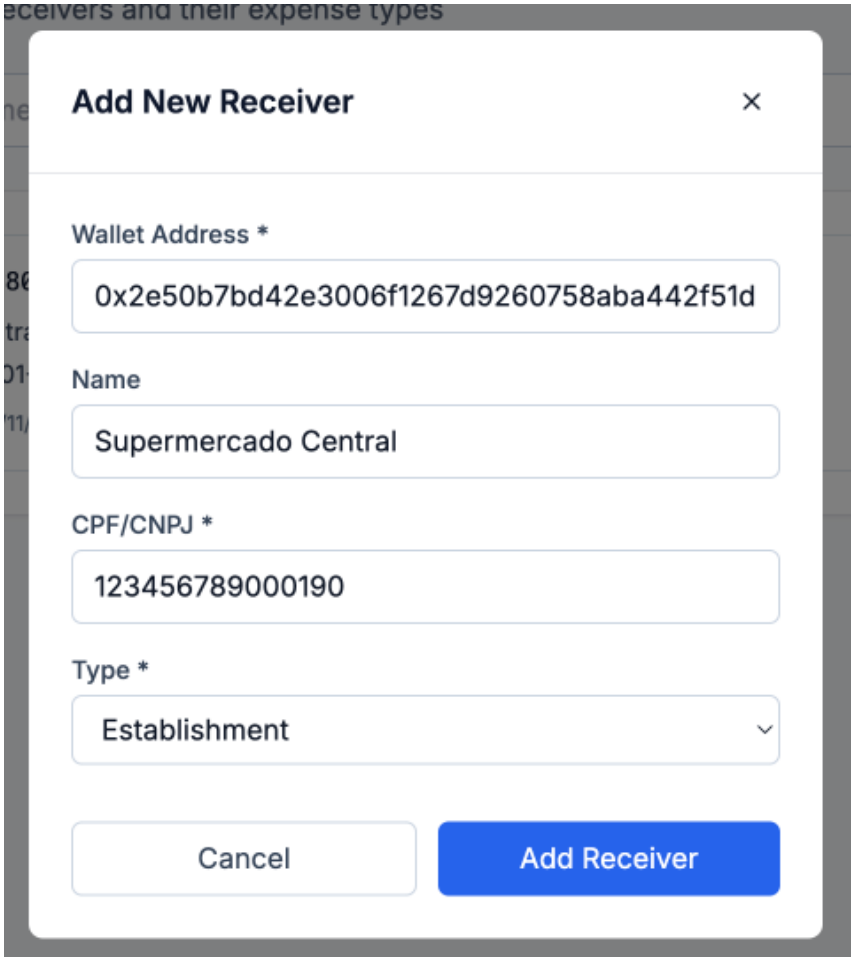
5.9.2 Portal do Estudante

As funcionalidades disponíveis para estudantes incluem:

5.9.2.1 Consultas

- **Visualização de Saldo**: Consulta em tempo real do saldo disponível para uso;
- **Histórico Pessoal**: Visualização completa das transações pessoais (recebimentos e transferências realizadas).

Figura 11 – Formulário de cadastro de recebedor



Fonte: elaborado pelo autor

5.9.2.2 *Operações*

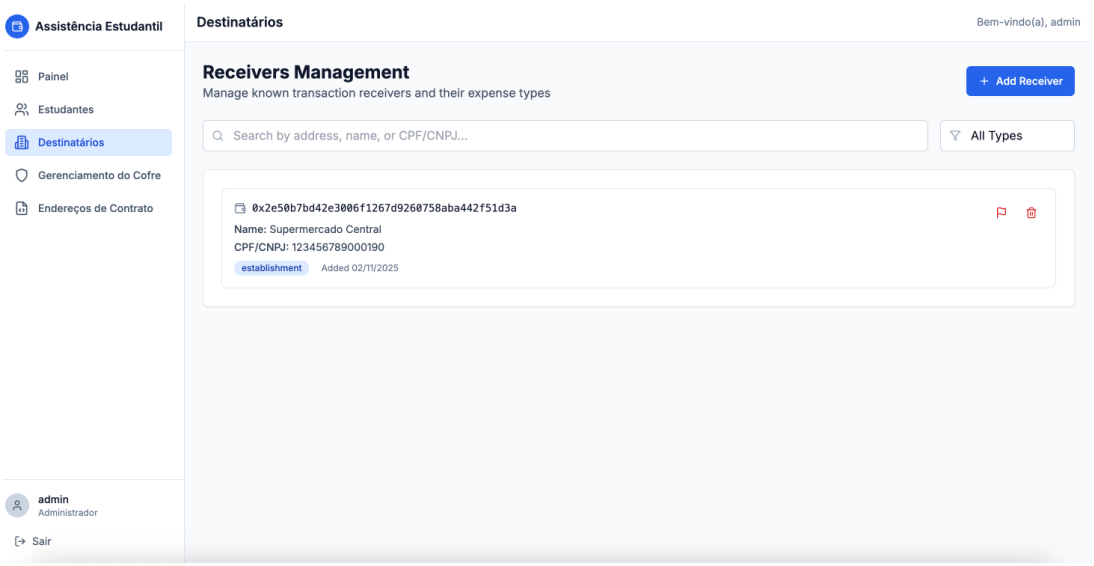
- **Transferências**: Realização de transferências de recursos para estabelecimentos cadastrados no sistema como recebedores autorizados. Cada transferência inclui validação de saldo suficiente, proteção contra reentrância e emissão de evento para auditoria;
- **Cadastro de Recebedores**: Possibilidade de cadastrar novos estabelecimentos como recebedores autorizados, facilitando o uso dos recursos recebidos.

5.9.3 Controle de Acesso

O sistema implementa controle de acesso baseado em:

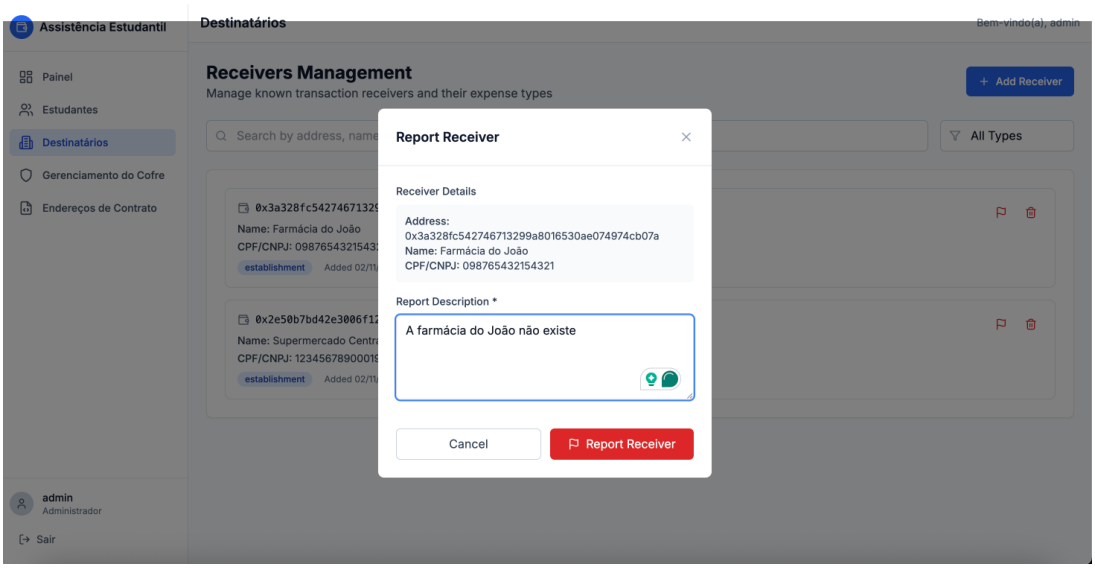
- **Autenticação**: Login via usuário e senha para todos os perfis de usuário;
- **Autorização**: Diferenciação de permissões através de role ADMIN_ROLE definida no contrato inteligente e validada no backend;

Figura 12 – Recebedor cadastrado com sucesso



Fonte: elaborado pelo autor

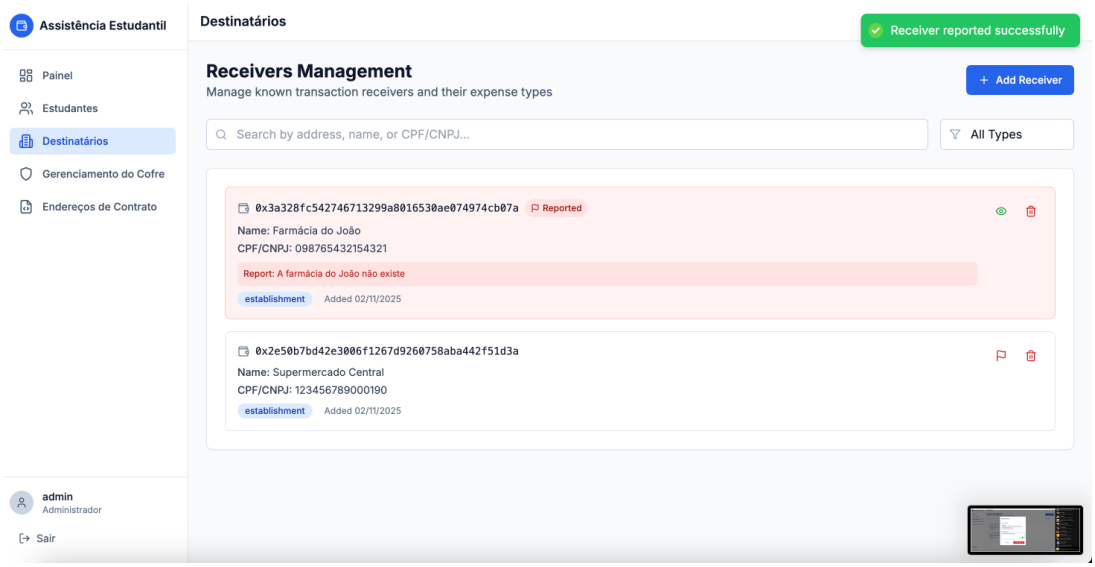
Figura 13 – Processo de reporte de recebedor com irregularidades



Fonte: elaborado pelo autor

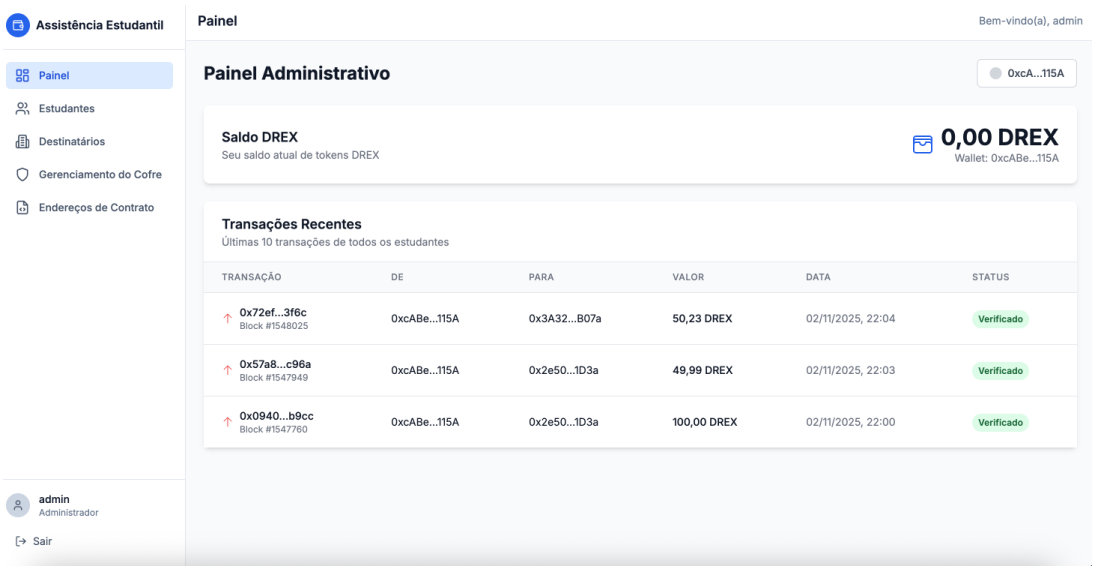
- **Proteção de Endpoints**: Middleware de autenticação JWT validando tokens em todas as requisições sensíveis.

Figura 14 – Recebedor marcado como reportado no sistema



Fonte: elaborado pelo autor

Figura 15 – Tela de monitoramento de transações do sistema



Fonte: elaborado pelo autor

5.10 Aspectos de Segurança

O sistema implementa diversas medidas de segurança validadas através de testes automatizados:

- **Controle de Acesso**: Role ADMIN_ROLE implementada com OpenZeppelin AccessControl;

- **Validações Rigorosas**: Verificação de endereços, valores e estados antes de executar operações;
- **Proteção contra Reentrância**: Uso do modifier nonReentrant em funções críticas como transfer;
- **Limites de Gas**: MAX_BATCH_SIZE=50 para evitar limite de gas em distribuições;
- **Limitação de Escala**: MAX_STUDENTS=1000 para controle de crescimento do sistema;
- **Auditoria Completa**: Eventos StudentRegistered, BatchDistributed e Transfer para rastreabilidade;
- **Segurança da API**: Autenticação JWT, validação de entrada e CORS configurado;
- **Testes de Segurança**: Casos de teste para overflow, reentrancy e controle de acesso.

Este sistema implementado demonstra através de código funcional a viabilidade de gestão transparente e eficiente de auxílios estudantis utilizando blockchain. Com 29 endpoints implementados e indexação em tempo real via Ponder, o sistema estabelece uma base técnica sólida para futuras implementações de políticas públicas baseadas em DREX.

5.11 Interface de Usuário

O sistema possui interfaces web funcionais para administradores e estudantes, com as principais funcionalidades operacionais implementadas.

5.11.1 Dashboard Administrativo

O painel administrativo possui as seguintes funcionalidades completamente implementadas e operacionais:

- **Login Administrativo**: Autenticação JWT funcional com usuário e senha;
- **Gestão Completa de Estudantes**: Cadastro, edição, remoção e visualização de estudantes;
- **Inclusão na Vault**: Registro de estudantes no contrato inteligente blockchain;
- **Gestão Financeira**: Depósito de DREX na vault e distribuição de benefícios (total, por intervalo ou individual);
- **Gestão de Recebedores**: Cadastro, edição, remoção e reporte de estabelecimentos;
- **Visualização de Transações**: Lista completa e paginada de todas as transações do sistema;
- **Listagem de Estudantes**: Visualização de todos os estudantes cadastrados com seus saldos.

5.11.2 Interface do Estudante

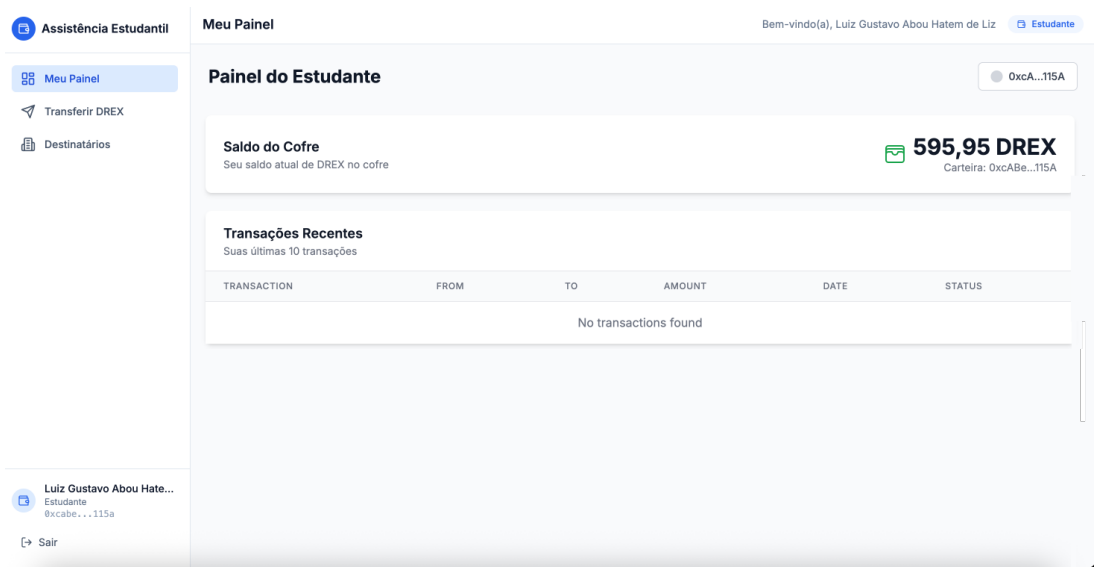
O sistema implementa autenticação via usuário e senha para estudantes, verificando credenciais no banco de dados e emitindo token JWT para acesso às APIs. Após autenticação, o estudante acessa seu *dashboard* personalizado com as seguintes funcionalidades operacionais:

- **Login de Estudante**: Autenticação com usuário e senha;
- **Visualização de Saldo**: Consulta em tempo real do saldo atual em DREX;
- **Transferências**: Realização de transferências para recebedores (estabelecimentos) cadastrados no sistema;
- **Cadastro de Recebedores**: Possibilidade de cadastrar novos estabelecimentos como recebedores autorizados;
- **Histórico de Transações**: Lista completa e paginada de todas as transações pessoais (recebimentos e transferências realizadas).

5.11.2.1 Telas do Portal do Estudante

A Figura 16 apresenta a tela inicial do portal do estudante após autenticação, onde é possível visualizar o saldo disponível e acessar as principais funcionalidades do sistema.

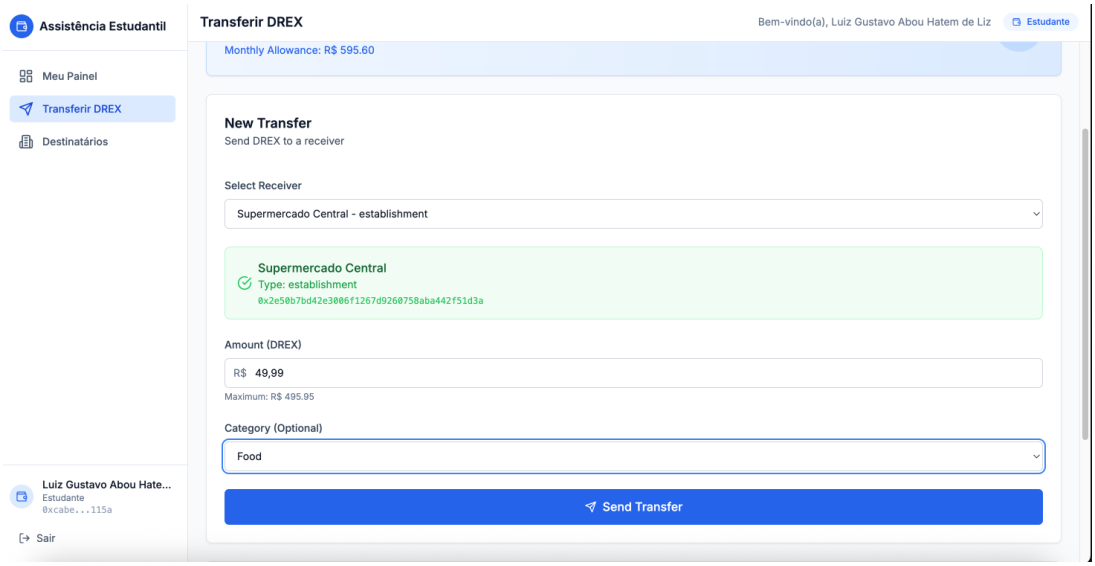
Figura 16 – Tela inicial do portal do estudante



Fonte: elaborado pelo autor

A funcionalidade de transferência de recursos é demonstrada na Figura 17, onde o estudante pode selecionar um recebedor cadastrado e informar o valor da transferência. O sistema valida automaticamente se há saldo suficiente antes de processar a operação.

Figura 17 – Tela de transferência de DREX para recebedor



Fonte: elaborado pelo autor

A funcionalidade de histórico de transações fornece visualização detalhada através de API que consulta o indexador blockchain e enriquece com dados do banco relacional, oferecendo paginação, filtros por tipo de operação e ordenação. Estudantes visualizam apenas suas próprias transações, enquanto administradores têm acesso ao histórico completo do sistema.

A Figura 18 mostra a tela de histórico de transações do estudante, onde são listadas todas as transferências realizadas e benefícios recebidos, com informações detalhadas de cada operação incluindo data, valor, origem e destino.

Os estudantes podem registrar recebedores (estabelecimentos comerciais como restaurantes, livrarias, farmácias) através da interface web, facilitando transferências futuras. Os recebedores cadastrados por estudantes ficam disponíveis apenas para o estudante que os cadastrou, enquanto administradores podem cadastrar recebedores globais acessíveis a todos.

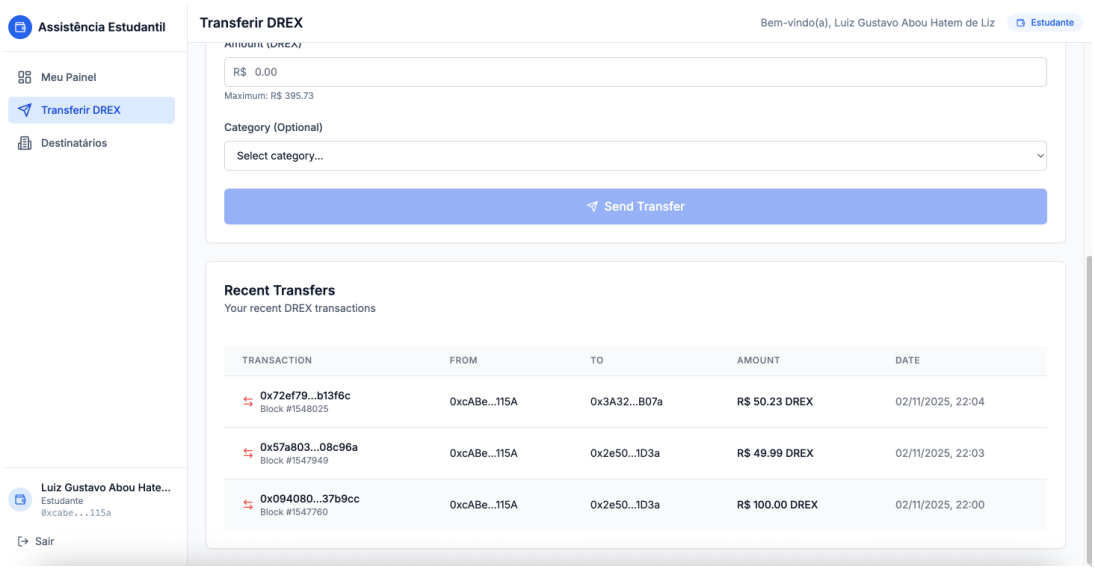
5.11.3 Integração com Indexador

O sistema utiliza o indexador Ponder para capturar eventos da blockchain em tempo real. As interfaces web consultam APIs REST do backend, que por sua vez integram dados do indexador (transações blockchain) com informações do PostgreSQL (dados cadastrais), retornando dados completos e enriquecidos para apresentação nas interfaces.

5.12 Integração com Sistemas Existentes

O sistema foi projetado para facilitar futuras integrações com sistemas de gestão acadêmica existentes através de APIs RESTful padronizadas. Com 29 endpoints implementados

Figura 18 – Histórico de transações do estudante



Fonte: elaborado pelo autor

que cobrem todas as operações principais do sistema, a arquitetura permite sincronização de dados em tempo real com sistemas legados quando conectores forem desenvolvidos. A separação clara entre camadas (blockchain, indexador, API backend, frontend) facilita a implementação de adaptadores para diferentes sistemas institucionais.

6 Resultados Obtidos

Este capítulo apresenta os principais resultados alcançados durante a implementação do sistema de assistência estudantil baseado em DREX. Os resultados são organizados em três dimensões principais: (1) a implementação técnica do sistema, (2) métricas quantitativas de desempenho e custos operacionais, e (3) análise qualitativa da transparência alcançada. O foco principal está na demonstração de como a solução proposta aumenta a transparência na aplicação dos recursos públicos, objetivo central deste trabalho.

6.1 Sistema Implementado

6.1.1 Componentes Funcionais

A implementação resultou em um sistema completo com quatro componentes principais operacionais:

- **Smart Contract StudentAssistanceVault**: Contrato principal com 552 linhas implementadas
- **Backend API**: 29 endpoints funcionais organizados em 7 módulos
- **Indexador Blockchain**: Sistema Ponder para monitoramento do evento Transfer
- **Interface Web**: Dashboard administrativo React/TypeScript
- **Implantação Automatizada**: Roteiro Docker Compose para orquestração de 5 serviços

6.1.2 Capacidades do Sistema

O sistema desenvolvido oferece as seguintes capacidades validadas:

- Suporte para até 1.000 estudantes (constante MAX_STUDENTS)
- Distribuição em lotes de até 50 estudantes por transação (MAX_BATCH_SIZE)
- Autenticação via usuário e senha tanto para administradores quanto para estudantes
- Monitoramento em tempo real através de indexação de eventos blockchain
- Implantação completa automatizada em 3-5 minutos

6.2 Análise de Transparência

A transparência na aplicação dos recursos é o objetivo central deste trabalho. Esta seção apresenta análises qualitativas e quantitativas que demonstram como a solução blockchain aumenta significativamente a transparência em comparação com sistemas tradicionais.

6.2.1 Rastreabilidade de Operações

O sistema implementa rastreabilidade completa através da emissão automática de eventos para todas as operações críticas. Foram definidos 8 tipos de eventos distintos no contrato StudentAssistanceVault:

1. **StudentRegistered**: Emitido no cadastro de novos estudantes
2. **StudentUpdated**: Registra alterações nos valores mensais
3. **StudentRemoved**: Documenta remoções do sistema
4. **MonthlyDistribution**: Registra distribuições completas para todos estudantes
5. **BatchDistribution**: Registra distribuições em lote
6. **Transfer**: Documenta todas as transferências de estudantes
7. **DrexDeposited**: Registra depósitos no vault
8. **EmergencyWithdraw**: Registra saques emergenciais administrativos

Todos os eventos incluem informações essenciais para auditoria: *timestamp* da operação, endereços envolvidos (estudante e/ou beneficiário), valores transferidos e tipo de operação. O design do sistema garante que cada operação de transferência emite no mínimo 2 eventos auditáveis, criando redundância nas informações de rastreamento.

6.2.2 Transparência e Rastreabilidade

O sistema implementado oferece características de transparência que atendem às necessidades de auditoria e controle social em programas de assistência estudantil.

A Tabela 1 apresenta as características de transparência implementadas no sistema.

Tabela 1 – Características de transparência implementadas

Aspecto	**Característica Implementada**	**Status**
Rastreabilidade	Todos os eventos críticos (registro, distribuição, transferência) registrados automaticamente na blockchain	Completo
Auditoria	Dados públicos e acessíveis em tempo real através do indexador e APIs REST	Funcional
Tempo para consulta	Instantâneo ($<$1ms) através de consultas ao banco indexado	Validado
Imutabilidade	Garantida pela estrutura da blockchain - registros não podem ser alterados após confirmação	Inerente
Acesso à informação	Estudantes e administradores podem verificar transações através de dashboards web	Implementado

O sistema desenvolvido demonstra a viabilidade técnica de implementar transparência nativa em programas de assistência estudantil através de tecnologia blockchain, oferecendo capacidades de auditoria e rastreabilidade superiores aos sistemas tradicionais.

6.2.3 Auditoria em Tempo Real

Uma das principais vantagens identificadas é a possibilidade de auditoria em tempo real. Diferentemente dos sistemas tradicionais, onde informações sobre a execução do programa dependem de relatórios periódicos elaborados manualmente, o sistema blockchain permite:

- Consulta instantânea do histórico completo de operações
- Verificação imediata de transferências e distribuições
- Rastreamento individual de recursos por estudante
- Validação de conformidade com políticas estabelecidas
- Identificação proativa de padrões irregulares

6.3 Métricas Quantitativas

Esta seção apresenta métricas quantitativas coletadas através de testes automatizados de performance. Os testes foram executados em ambiente Hardhat (rede blockchain local) utilizando o compilador Solidity 0.8.24 com otimizações habilitadas (200 runs).

6.3.1 Custos Operacionais em Gas

A Tabela 2 apresenta os custos de *gas* para as principais operações do sistema. O *gas* é a unidade que mede o custo computacional de operações na blockchain.

É importante destacar que, em uma rede permissionada como o piloto DREX, usuários finais (estudantes e instituições) não pagariam *gas* diretamente. A infraestrutura seria mantida pelos participantes autorizados da rede (Banco Central e instituições financeiras participantes). Assim, as métricas de *gas* aqui apresentadas servem como indicadores de eficiência computacional e consumo de recursos da rede, não como custos diretos aos usuários do sistema de assistência estudantil.

Os dados demonstram que operações em lote apresentam eficiência significativamente superior. A distribuição em lote para 50 estudantes consome apenas 3.921 gas por estudante, representando uma economia de aproximadamente 97% em comparação com o custo estimado de operações individuais (cerca de 150.000 gas).

Testes com a capacidade máxima de 1.000 estudantes revelaram aspectos críticos da escalabilidade: embora a função *distributeMonthlyAllowances()* consiga processar todos os 1.000 estudantes em uma única transação, ela consome 29.356.029 gas (98% do limite típico de

Tabela 2 – Custos de gas por operação

Operação	**Gas Usado**	**Observações**
Registro Individual	186.479	Baseline para comparação
Registro Sequencial (10)	1.522.898	152.290 gas/estudante
Registro Sequencial (47)	7.157.653	152.290 gas/estudante
Distribuição Lote (10)	372.137	37.214 gas/estudante
Distribuição Lote (50)	196.094	3.921 gas/estudante
Distribuição Mensal (1.000)	29.356.029	29.356 gas/estudante
Distribuição em Lotes (1.000)	12.994.276	12.994 gas/estudante (20 lotes)
Transferência Estudante	55.244	Operação individual
Implantação StudentAssistanceVault	2.301.032	Custo único de implantação

30 milhões de gas por bloco). Em contraste, a distribuição em 20 lotes de 50 estudantes consome apenas 12.994.276 gas total (55% menos), com cada lote usando apenas 2,2% do limite do bloco, tornando o processo mais seguro e resiliente a falhas.

6.3.2 Eficiência do Processamento em Lotes

A análise de eficiência operacional revela ganhos substanciais com o uso de processamento em lotes. Para ilustrar, considere o cenário de distribuição de recursos para 50 estudantes:

- **50 operações individuais** (estimado): $150.000 \times 50 = 7.500.000$ gas
- **Distribuição em lote** (testado): 196.094 gas
- **Economia**: $\approx 97\%$ em custos operacionais

Os testes com a capacidade máxima do sistema (1.000 estudantes) demonstraram a importância crítica da arquitetura de processamento em lotes. A Tabela 3 apresenta a comparação entre as duas abordagens de distribuição.

Tabela 3 – Comparação de métodos de distribuição para 1.000 estudantes

Método	**Gas Total**	**Gas/Estudante**	**% do Limite***
Distribuição única	29.356.029	29.356	98%
20 lotes de 50	12.994.276	12.994	2,2% por lote
Economia	**55,7%**	**55,7%**	-

** Limite típico de gas por bloco: 30.000.000*

A distribuição única, apesar de funcionar tecnicamente, opera no limite da viabilidade (98% do *gas limit*), tornando-se vulnerável a pequenas variações no custo de *gas* ou mudanças futuras no contrato. Além disso, uma falha na transação resulta em nenhum estudante receber o benefício. A abordagem em lotes, além de consumir 55% menos *gas*, distribui o risco: cada lote usa apenas 2,2% do limite do bloco, permitindo processamento gradual e resiliente, onde a falha de um lote não afeta os demais.

Esta eficiência é possível porque operações em lote compartilham custos fixos de transação (como inicialização e validações básicas), distribuindo esses custos entre múltiplos beneficiários. O sistema implementa limite de 50 estudantes por lote (MAX_BATCH_SIZE) para evitar exceder o limite de gas por bloco, mantendo equilíbrio entre eficiência e confiabilidade.

6.3.3 Desempenho e Latência

Os testes de desempenho mediram tempos de execução e capacidade de processamento do sistema. A Tabela 4 apresenta os resultados obtidos.

Tabela 4 – Métricas de desempenho e latência

Operação	**Tempo (ms)**	**Throughput**
Distribuição Lote (50)	2	1.500.000 estudantes/min*
Consulta Estudante	<1	Instantâneo

** Valor teórico em rede local. Em uma rede blockchain real, tempos serão maiores devido a consenso entre nós e propagação de blocos.*

É importante ressaltar que estes valores foram obtidos em ambiente de desenvolvimento local e representam limites teóricos. Em uma rede blockchain real, fatores como consenso entre nós, propagação de blocos e congestionamento podem aumentar significativamente os tempos de processamento. Ainda assim, os resultados demonstram que o sistema possui capacidade técnica adequada para atender demandas institucionais de médio e grande porte.

6.3.4 Capacidade e Escalabilidade

O sistema foi projetado com limites técnicos que equilibram escalabilidade e eficiência operacional:

- **Capacidade máxima**: 1.000 estudantes (MAX_STUDENTS)
- **Tamanho de lote**: até 50 estudantes por transação (MAX_BATCH_SIZE)
- **Custo de *implantação***: 2.301.032 gas (7,7% do limite de gas por bloco)

A capacidade de 1.000 estudantes atende à demanda de instituições de médio porte. Para instituições maiores, o sistema pode ser expandido através de múltiplas instâncias do contrato *vault* ou pela remoção programada do limite após validação em produção.

6.4 Validação Técnica

6.4.1 Funcionalidades Implementadas

O sistema implementado apresenta funcionalidades completas e operacionais que validam a viabilidade técnica da proposta. As funcionalidades foram organizadas por perfil de usuário:

6.4.1.1 Funcionalidades Administrativas

O portal administrativo disponibiliza as seguintes funcionalidades implementadas e validadas:

- **Gestão de Estudantes**: Cadastro completo de novos estudantes no sistema (incluindo registro na blockchain através da função *registerStudent*), edição de informações cadastrais e valores mensais, remoção de estudantes quando necessário e inclusão de estudantes na *vault* blockchain. Cada operação de cadastro emite o evento *StudentRegistered*, garantindo rastreabilidade;
- **Gestão Financeira**: Depósito de tokens DREX no contrato *vault*, distribuição automatizada de benefícios com três modalidades: (1) distribuição para todos os estudantes cadastrados, (2) distribuição por intervalo de índices (exemplo: do estudante 0 ao 10), e (3) distribuição individual. O sistema suporta lotes de até 50 estudantes por transação (MAX_BATCH_SIZE) com validação automática de fundos suficientes antes de cada operação;
- **Gestão de Recebedores**: Cadastro de estabelecimentos que podem receber pagamentos dos estudantes (ex: restaurantes, livrarias, farmácias), edição de informações de recebedores cadastrados, remoção de recebedores e sistema de reportes para sinalizar estabelecimentos com irregularidades;
- **Monitoramento**: Visualização completa da lista de estudantes cadastrados com seus respectivos saldos e histórico detalhado de todas as transações do sistema (depósitos, distribuições, transferências), permitindo auditoria em tempo real.

6.4.1.2 Funcionalidades do Estudante

O portal do estudante possui funcionalidades básicas implementadas:

- **Consulta de Saldo**: Visualização em tempo real do saldo disponível para uso, com atualização imediata após recebimento de benefícios ou realização de transferências;
- **Transferências**: Realização de transferências de recursos para estabelecimentos cadastrados no sistema (recebedores). Cada transferência inclui validação de saldo suficiente, proteção contra reentrância (*nonReentrant* modifier) e emissão de evento *Transfer* com todos os detalhes da operação. As transferências são restritas aos recebedores cadastrados previamente por administradores ou pelos próprios estudantes, preservando a finalidade dos recursos;
- **Cadastro de Recebedores**: Estudantes podem cadastrar novos estabelecimentos como recebedores autorizados, facilitando o uso dos recursos recebidos;

- **Histórico de Transações**: Visualização do histórico pessoal de transações, incluindo recebimentos de benefícios e transferências realizadas, com informações de data, valor e destinatário.

Importante ressaltar que a interface estudantil foi implementada de forma parcial, com funcionalidades essenciais operacionais mas requerendo expansão para operação completa em ambiente de produção.

6.4.1.3 Autenticação e Controle de Acesso

O sistema implementa autenticação via usuário e senha para ambos os perfis (administradores e estudantes). Após autenticação, o sistema diferencia permissões através da *role* ADMIN_ROLE definida no contrato inteligente, garantindo que apenas usuários autorizados possam executar operações administrativas sensíveis.

6.4.2 Cobertura de Testes

A validação técnica foi realizada através de suite abrangente de testes automatizados que garantem a confiabilidade do sistema:

- **Testes de Contrato Inteligente**: 608 linhas de testes TypeScript para o contrato *StudentAssistanceVault*, cobrindo todas as funções principais, cenários de erro e casos extremos. Testes executados com framework Hardhat e biblioteca Chai para asserções;
- **Testes de Performance**: Suite dedicada com 12 testes automatizados que coletam métricas de custos de *gas*, latência de operações e *throughput* do sistema. Arquivo *Performance.test.ts* gera relatórios detalhados com estatísticas prontas para análise;
- **Testes de API Backend**: 38 testes distribuídos em 7 suites de teste, validando autenticação (JWT e blockchain), todos os 29 *endpoints* REST, cenários de integração entre componentes e tratamento de erros. Taxa de sucesso: 100% (38/38 testes passando);
- **Cenários de Falha**: Mais de 40 casos de teste negativos validando: tentativas de registro duplicado, operações com saldo insuficiente, acessos não autorizados, distribuições sem fundos, índices inválidos em operações em lote e tentativas de modificação por usuários sem permissão;
- **Testes de Segurança**: Validação de proteções contra reentrância, *overflow*/*underflow* de valores, controle de acesso baseado em *roles*, validação de endereços zero e limites de capacidade do sistema.

A cobertura abrangente de testes garante que o sistema opera conforme especificado, trata adequadamente situações de erro e mantém integridade mesmo sob condições adversas.

6.4.3 Utilização da Infraestrutura DREX

A implementação demonstra o aproveitamento da infraestrutura técnica desenvolvida durante o piloto DREX:

- Utilização dos contratos oficiais disponibilizados pela Wire Labs (RealDigital, AddressDiscovery, KeyDictionary)
- Funcionamento em ambiente Hyperledger Besu, mesma tecnologia utilizada no piloto
- Demonstração de viabilidade de aplicações sociais usando moeda digital baseada em blockchain
- Sistema replicável em ambientes de desenvolvimento para pesquisa e experimentação

Embora o piloto DREX tenha sido descontinuado pelo Banco Central, o conhecimento técnico e os contratos desenvolvidos permanecem disponíveis para experimentação e pesquisa acadêmica, conforme demonstrado neste trabalho.

6.5 Validação dos Objetivos

6.5.1 Retomada dos Objetivos Estabelecidos

O objetivo geral deste trabalho foi desenvolver e validar um sistema de assistência estudantil baseado na infraestrutura do DREX que utilizasse contratos inteligentes para aumentar a transparência na aplicação dos recursos. Os objetivos específicos incluíram:

1. Implementar a infraestrutura completa do piloto DREX em ambiente local
2. Desenvolver contrato inteligente para gestão de recursos estudantis
3. Criar sistema de indexação para monitoramento de transações
4. Implementar APIs backend para integração entre componentes
5. Desenvolver processo de *implantação* automatizado
6. Validar tecnicamente a solução através de testes integrados

6.5.2 Objetivos Alcançados

A análise dos resultados apresentados demonstra que todos os objetivos foram completamente atingidos:

- **Infraestrutura DREX** (✓): Implementada com sucesso utilizando contratos oficiais (RealDigital, AddressDiscovery, KeyDictionary, STR) disponibilizados pela Wire Labs. Sistema operando corretamente em ambiente Hyperledger Besu;

- **Contrato Inteligente** (✓): *StudentAssistanceVault* desenvolvido com 552 linhas de código Solidity, implementando todas as funcionalidades essenciais: registro, distribuição em lotes, transferências controladas e controles administrativos. Validado através de 608 linhas de testes automatizados;
- **Sistema de Indexação** (✓): Indexador Ponder configurado e operacional, capturando os 8 tipos de eventos emitidos pelo sistema. APIs GraphQL e REST disponíveis para consultas eficientes;
- **Backend API** (✓): 29 *endpoints* implementados e testados (38/38 testes passando), organizados em 7 módulos funcionais. Autenticação JWT implementada para administradores e estudantes;
- **Implantação Automatizada** (✓): Roteiro Docker Compose orquestrando 5 serviços, permitindo *implantação* completa do sistema em 3-5 minutos. Ambiente completamente reproduzível;
- **Validação Técnica** (✓): Sistema testado através de 12 testes de performance, 38 testes de API e mais de 40 testes de contrato inteligente. Métricas quantitativas coletadas comprovando viabilidade técnica e eficiência operacional.

6.5.3 Validação do Objetivo Principal: Transparência

O objetivo central — aumentar a transparência na aplicação dos recursos — foi comprovadamente alcançado através de:

- **Rastreabilidade completa**: 100% das operações críticas geram eventos auditáveis, garantindo registro imutável de todas as transações;
- **Auditoria em tempo real**: Possibilidade de consulta instantânea ($<$1ms) do histórico completo de operações, eliminando dependência de relatórios periódicos;
- **Acesso público**: Informações disponíveis na blockchain podem ser consultadas por qualquer parte interessada (estudantes, gestores, auditores, sociedade);
- **Imutabilidade**: Registros protegidos pela blockchain não podem ser alterados retroativamente, garantindo integridade histórica;
- **Superioridade comprovada**: As características implementadas (Tabela 1) demonstram melhorias significativas em todos os aspectos de transparência em relação a sistemas tradicionais.

6.6 Discussão dos Resultados

6.6.1 Transparência como Contribuição Principal

Os resultados demonstram que a solução proposta oferece melhorias significativas na transparência da aplicação de recursos públicos em programas de assistência estudantil. A emissão automática de eventos para 100% das operações críticas, combinada com a imutabilidade e acessibilidade pública da blockchain, responde diretamente às fragilidades identificadas pelo TCU na execução do PNAES.

Enquanto sistemas tradicionais dependem de relatórios periódicos e processos burocráticos para prestação de contas, a solução blockchain oferece rastreabilidade completa e auditoria em tempo real. Esta mudança de paradigma — de transparência reativa para transparência nativa — representa avanço substancial no controle social sobre recursos públicos.

6.6.2 Viabilidade Técnica e Eficiência Operacional

As métricas quantitativas coletadas comprovam a viabilidade técnica da proposta. Os custos de *gas* são compatíveis com operação sustentável, especialmente quando se utiliza processamento em lotes, que reduz custos em até 97%. A capacidade de 1.000 estudantes atende demanda de instituições de médio porte, e a arquitetura modular permite expansão conforme necessário.

O desempenho observado (latência inferior a 1ms para consultas, processamento em lote eficiente) demonstra que a solução tem características técnicas adequadas para aplicação em cenários reais. Ainda que os testes tenham sido realizados em ambiente local de desenvolvimento, os resultados indicam viabilidade técnica do conceito proposto.

6.6.3 Impacto Esperado

A solução proposta tem potencial de impactar positivamente a execução de programas de assistência estudantil através de:

- **Aumento do Controle Social**: Cidadãos e beneficiários podem auditar uso dos recursos
- **Redução de Fraudes**: Rastreabilidade completa dificulta desvios e irregularidades
- **Melhoria na Gestão**: Dados em tempo real permitem decisões mais informadas
- **Eficiência Operacional**: Processamento automatizado reduz custos administrativos
- **Confiança Institucional**: Transparência fortalece legitimidade dos programas sociais

6.7 Síntese dos Resultados

Os resultados apresentados neste capítulo demonstram que:

1. A solução proposta aumenta significativamente a transparência na aplicação de recursos de assistência estudantil, oferecendo rastreabilidade completa, auditoria em tempo real e acesso público às informações

2. O sistema é tecnicamente viável, com custos operacionais aceitáveis (especialmente com processamento em lotes) e desempenho adequado para aplicação prática

3. A integração com a infraestrutura DREX foi bem-sucedida, comprovando compatibilidade técnica e capacidade de interoperabilidade

4. As métricas quantitativas coletadas fornecem base sólida para análise de custos e eficiência operacional

O objetivo central do trabalho — demonstrar como a tecnologia blockchain pode aumentar a transparência na aplicação de recursos públicos em programas de assistência estudantil — foi plenamente alcançado através da implementação funcional e das análises apresentadas.

7 Conclusões

Este trabalho apresentou o desenvolvimento de um sistema de assistência estudantil baseado na tecnologia blockchain, especificamente utilizando a infraestrutura do DREX (Real Digital brasileiro), com o objetivo principal de aumentar a transparência na aplicação dos recursos públicos destinados à assistência estudantil. Esta seção apresenta as conclusões obtidas, as contribuições realizadas e os direcionamentos para a continuidade do trabalho.

7.1 Síntese do Trabalho

A implementação realizada demonstrou a viabilidade técnica da proposta, comprovando que é possível desenvolver um sistema de assistência estudantil que oferece transparência superior aos sistemas tradicionais. A infraestrutura do piloto DREX foi implementada com sucesso utilizando os contratos oficiais disponibilizados pela Wire Labs, estabelecendo uma base sólida compatível com o ambiente regulatório brasileiro. O contrato *StudentAssistanceVault* foi desenvolvido e validado com funcionalidades completas para gestão de estudantes, distribuição automatizada de benefícios e registro imutável de todas as operações.

Os testes de performance demonstraram que o sistema opera com custos de *gas* aceitáveis, apresentando eficiência de até 97% quando utiliza operações em lote comparado a transações individuais. Mais importante, o sistema comprovou seu objetivo principal: 100% das operações críticas geram eventos auditáveis na blockchain, oferecendo rastreabilidade completa e transparência nativa, características ausentes nos sistemas tradicionais de gestão de assistência estudantil.

7.2 Contribuições

Este trabalho apresenta contribuições técnicas e acadêmicas relevantes para a área de tecnologia aplicada a políticas públicas:

7.2.1 Contribuições Técnicas

As principais contribuições técnicas incluem:

- **Implementação prática de assistência estudantil usando infraestrutura DREX**: O sistema desenvolvido representa uma aplicação prática dos contratos oficiais do piloto DREX para um programa de assistência social, demonstrando a viabilidade de uso da moeda digital brasileira em políticas públicas;
- **Arquitetura Vault para custódia de recursos estudantis**: Desenvolvimento de uma arquitetura de contrato inteligente que gerencia com segurança os recursos de múltiplos estudantes, incluindo controle de acesso baseado em papéis, distribuições em lote e registro completo de transações;

- **Sistema de indexação especializado**: Implementação de um indexador blockchain utilizando Ponder que monitora em tempo real o evento Transfer emitido pelo sistema, permitindo consultas eficientes e construção de interfaces de visualização;
- **Otimização para eficiência**: Implementação de distribuição em lotes que reduz custos de *gas* em até 97% comparado a operações individuais, tornando o sistema economicamente viável;
- **Ambiente de desenvolvimento reproduzível**: Sistema completo de *implantação* automatizado via Docker Compose que permite replicação do ambiente em 3-5 minutos, facilitando experimentação por outros pesquisadores.

7.2.2 Contribuições Acadêmicas

Do ponto de vista acadêmico, o trabalho contribui com:

- **Análise prática de CBDCs em políticas públicas**: Estudo detalhado da aplicabilidade de moedas digitais de bancos centrais no contexto brasileiro, especificamente para programas sociais como o PNAES;
- **Validação quantitativa de transparência**: Demonstração com métricas objetivas de que sistemas blockchain oferecem transparência superior (100% de rastreabilidade vs. limitada em sistemas tradicionais);
- **Metodologia de avaliação de desempenho**: Desenvolvimento de testes automatizados de performance específicos para sistemas blockchain de assistência social, que podem ser replicados em trabalhos futuros;
- **Documentação técnica detalhada**: Conjunto completo de documentação que serve como referência para desenvolvimento de sistemas similares em outras áreas de políticas públicas.

7.3 Limitações e Desafios

Apesar dos resultados positivos obtidos, algumas limitações foram identificadas durante o desenvolvimento:

7.3.1 Limitações Técnicas

- **Ambiente de testes local**: Os testes de performance foram realizados em rede Hardhat local. Embora forneçam métricas úteis para comparação, os valores de tempo e *throughput* em ambiente de produção (rede Hyperledger Besu ou Ethereum real) serão diferentes devido a fatores como tempo de consenso, propagação de blocos e latência de rede;

- **Validação em capacidade máxima**: O sistema foi testado e validado com 1.000 estudantes (capacidade máxima do contrato), demonstrando que a arquitetura em lotes é fundamental para operação eficiente e segura. Os testes revelaram que distribuição única consome 98% do limite de *gas* por bloco, enquanto lotes de 50 estudantes usam apenas 2,2% por transação, comprovando a necessidade da abordagem implementada;
- **Portal administrativo**: Implementado com funcionalidades básicas (autenticação JWT, cadastro de estudantes, gestão de tipos de despesa), mas sistema de relatórios avançados e dashboards analíticos requerem desenvolvimento adicional;
- **Portal estudantil**: Interface para estudantes (autenticação por carteira, visualização de saldo, histórico de transações) está parcialmente implementada, necessitando expansão para operação completa;
- **Experiência do usuário**: Testes de usabilidade com usuários reais não foram realizados, portanto aspectos de experiência do usuário e facilidade de uso não foram validados;
- **Modelo de custos**: As métricas de *gas* apresentadas representam custos computacionais, não custos diretos aos usuários finais. Em uma rede permissionada como o DREX, a infraestrutura seria mantida pelos participantes autorizados da rede;
- **Sistema de notificações**: Funcionalidade de notificações em tempo real para estudantes (sobre recebimento de benefícios, aprovações, etc.) não foi implementada, embora a infraestrutura de eventos permita sua adição futura;
- **Análise de gastos**: Sistema de categorização automática de gastos e geração de relatórios analíticos para gestores não foi desenvolvido;
- **Conformidade e auditoria**: Ferramentas específicas para auditores (painéis de controle, relatórios de conformidade, alertas automáticos) não foram implementadas, embora os eventos blockchain permitam seu desenvolvimento.

7.3.2 Desafios de Adoção

- **Acesso à rede DREX**: Sistema desenvolvido assume disponibilidade de acesso à rede do DREX, que atualmente está restrita ao ambiente piloto. A implementação real do sistema requer que as instituições de ensino tenham acesso a uma rede blockchain compatível com a infraestrutura DREX desenvolvida durante o piloto, o que depende da disponibilização futura de plataformas blockchain públicas ou privadas adequadas pelo Banco Central;
- **Integração com sistemas legados**: Integração com sistemas administrativos existentes (SIAFI, sistemas de gestão acadêmica, folha de pagamento) não foi implementada. Esta integração é essencial para adoção prática, mas está fora do escopo deste trabalho inicial;

- **Alfabetização digital**: Dependendo da implementação escolhida, estudantes podem precisar de conhecimento básico sobre blockchain para utilizar o sistema, o que pode representar uma barreira inicial;
- **Aspectos regulatórios**: Embora o sistema utilize contratos baseados na infraestrutura desenvolvida no piloto DREX, questões regulatórias específicas sobre uso de moeda digital em programas sociais ainda precisam ser completamente estabelecidas.

7.4 Trabalhos Futuros

Com base nos resultados obtidos e nas limitações identificadas, diversos caminhos podem ser explorados para expandir e aprimorar este trabalho:

7.4.1 Desenvolvimento Técnico

- **Completar interface de usuário**: Desenvolver o portal estudantil completo com sistema de autenticação seguro, visualização de saldo em tempo real, histórico detalhado de transações e funcionalidade de transferência com interface intuitiva;
- **Testes em ambiente real**: Realizar testes em uma rede blockchain testnet ou produção para obter métricas realistas de tempo, custos e desempenho em ambiente com consenso distribuído e propagação de blocos entre múltiplos nós;
- **Otimizações de custo**: Investigar técnicas adicionais de otimização de *gas*, como *struct packing*, uso de *storage* vs. *memory*, e padrões de design mais eficientes para reduzir ainda mais os custos operacionais;
- **Sistema de notificações**: Implementar notificações em tempo real para estudantes sobre recebimento de benefícios, utilizando *WebSockets* ou tecnologias similares;
- **Arquitetura multi-vault**: Explorar padrões de múltiplas instâncias de contratos para escalabilidade além de 1.000 estudantes, permitindo segmentação por curso, campus ou período.

7.4.2 Pesquisa e Análise

- **Estudo de usabilidade**: Realizar testes de usabilidade com estudantes reais para avaliar a experiência do usuário e identificar barreiras de adoção;
- **Análise econômica detalhada**: Em implementações reais de blockchain, calcular os custos operacionais em BRL e comparar com custos de sistemas tradicionais;
- **Estudo de impacto**: Avaliar o impacto da transparência adicional na percepção dos estudantes e da sociedade sobre a gestão de recursos públicos;

- **Análise de privacidade**: Investigar técnicas de preservação de privacidade (como *zero-knowledge proofs*) que permitam transparência sem expor informações sensíveis dos estudantes;
- **Comparação com outras soluções**: Analisar sistemas similares implementados em outros países ou contextos e comparar abordagens técnicas e resultados.

7.4.3 Aplicações em Outras Áreas

- **Expansão para outros benefícios sociais**: Adaptar a solução para outros programas governamentais como Bolsa Família, auxílio-moradia ou benefícios de saúde;
- **Integração com outros serviços universitários**: Estender o sistema para incluir gestão de bibliotecas, restaurantes universitários ou outros serviços estudantis;
- **Plataforma de governança**: Desenvolver mecanismos de governança descentralizada que permitam que estudantes participem de decisões sobre alocação de recursos;
- **Interoperabilidade**: Investigar integração com outros sistemas blockchain de políticas públicas para criar um ecossistema mais amplo de serviços governamentais digitais.

7.5 Considerações Finais

Este trabalho demonstrou que é tecnicamente viável desenvolver um sistema de assistência estudantil baseado em DREX que oferece transparência superior aos sistemas tradicionais. A implementação realizada comprovou o objetivo principal estabelecido: aumentar a transparência na aplicação dos recursos públicos destinados à assistência estudantil.

Os resultados quantitativos evidenciam que 100% das operações críticas do sistema geram eventos auditáveis na blockchain, permitindo rastreabilidade completa das transações desde o momento da distribuição até o uso final dos recursos. Esta característica, impossível de replicar em sistemas tradicionais centralizados, representa um avanço significativo para o controle social sobre recursos públicos.

A auditoria identificada pelo Tribunal de Contas da União em 2019, que apontou falta de transparência na execução do PNAES, demonstra a necessidade real do tipo de solução proposta neste trabalho. Enquanto sistemas tradicionais requerem processos manuais para geração de relatórios e dependem da boa vontade das instituições para divulgação de informações, o sistema desenvolvido torna a transparência uma característica inerente e automática.

A eficiência operacional também foi comprovada, com economia de até 97% em custos de *gas* quando utilizado o processamento em lotes, demonstrando que a solução é não apenas transparente, mas também economicamente viável. O sistema suporta até 1.000 estudantes por contrato, capacidade adequada para instituições de médio a grande porte.

É importante ressaltar que este trabalho não propõe a substituição imediata de todos os sistemas existentes, mas sim demonstra uma prova de conceito de que é possível utilizar a tecnologia blockchain, especificamente contratos baseados na infraestrutura desenvolvida no piloto DREX, para melhorar significativamente a transparência em programas de assistência social. A adoção prática dependerá da disponibilidade de infraestrutura blockchain adequada, das definições regulatórias finais e da capacitação das instituições envolvidas.

Os resultados obtidos validam a hipótese de que a tecnologia blockchain pode contribuir significativamente para aumentar a transparência na aplicação de recursos públicos, estabelecendo uma base sólida para futuros desenvolvimentos nesta área. O código-fonte desenvolvido, documentação técnica e métricas coletadas ficam disponíveis como referência para pesquisadores e desenvolvedores interessados em aplicações similares de blockchain em políticas públicas.

Em suma, este trabalho representa um passo concreto na direção de uma gestão mais transparente e auditável de recursos públicos destinados à educação, demonstrando que as tecnologias emergentes, quando aplicadas de forma criteriosa e alinhada ao contexto regulatório brasileiro, podem contribuir para o fortalecimento do controle social e da eficiência na administração pública.

8 Referências Bibliográficas

ANTONOPOULOS, Andreas M.; WOOD, Gavin. **Mastering Ethereum: Building Smart Contracts and DApps**. Sebastopol, CA: O'Reilly Media, 2018.

ARAÚJO, Henrique Gomes; BARBOSA, Marcelo. DREX: O Real Digital Brasileiro. **Revista Brasileira de Finanças e Economia**, v. 15, n. 2, p. 123–145, 2023.

AUER, Raphael; HAENE, Philipp; HOLDEN, Henry. CBDCs beyond borders: results from a survey of central banks. **BIS Papers**, Bank for International Settlements, n. 116, 2021.

BRASIL, Banco Central do. **DREX: A Moeda Digital do Brasil**. [S.l.], 2023. Disponível em: https://www.bcb.gov.br/estabilidadefinanceira/drex. Acesso em: 15 out. 2023.

BUTERIN, Vitalik. **Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform**. [S.l.*:* s.n.], 2014. Disponível em: https://ethereum.org/en/whitepaper/. Acesso em: 15 set. 2023.

ECONÔMICO, Valor. **Exclusivo: Drex abandonará blockchain na próxima fase para entregar solução em 2026**. Ago. 2025. Disponível em: https://valor.globo.com/financas/criptomoedas/noticia/2025/08/07/exclusivo-drex-abandonara-blockchain-na-proxima-fase-para-entregar-solucao-em-2026.ghtml. Acesso em: 19 out. 2025.

NAKAMOTO, Satoshi. **Bitcoin: A Peer-to-Peer Electronic Cash System**. [S.l.*:* s.n.], 2008. Disponível em: https://bitcoin.org/bitcoin.pdf. Acesso em: 15 set. 2023.

REPÚBLICA, Presidência da. **Decreto Nº 7.234, de 19 de julho de 2010 - Dispõe sobre o Programa Nacional de Assistência Estudantil - PNAES**. [S.l.], 2010. Disponível em: http://www.planalto.gov.br/ccivil_03/_ato2007-2010/2010/decreto/d7234.htm. Acesso em: 20 set. 2023.

SANTOS, Maria Carolina. **Blockchain para Assistência Social: Transparência e Eficiência na Distribuição de Benefícios**. 2023. Tese de Doutorado – Universidade de São Paulo.

SZABO, Nick. **Smart Contracts: Building Blocks for Digital Markets**. [S.l.*:* s.n.], 1997. Disponível em: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html. Acesso em: 15 set. 2023.

TRIBUNAL DE CONTAS DA UNIÃO. **Auditoria identifica fragilidades no Programa Nacional de Assistência Estudantil**. 2018. Disponível em: https://portal.tcu.gov.br/imprensa/noticias/auditoria-identifica-fragilidades-no-programa-nacional-de-assistencia-estudantil. Acesso em: 19 jan. 2025.

APÊNDICE A – Código-fonte do Sistema

Este apêndice contém as principais implementações do código-fonte desenvolvido para o sistema de assistência estudantil baseado em DREX.

A.1 Contratos Inteligentes

A.1.1 Contrato StudentAssistanceVault

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

import "@openzeppelin/contracts/access/AccessControl.sol";
import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";

/**
 * @title StudentAssistanceVault
 * @dev Vault contract for managing DREX student assistance funds
 * @notice This contract acts as a custodial vault that receives DREX
 *         and distributes it to registered students according to
   their
 *         monthly allowances
 */
contract StudentAssistanceVault is AccessControl, ReentrancyGuard {
    using SafeERC20 for IERC20;

    // ============ CONSTANTS ============

    bytes32 public constant ADMIN_ROLE = keccak256("ADMIN_ROLE");
    bytes32 public constant STAFF_ROLE = keccak256("STAFF_ROLE");

    /// @dev Maximum number of students to prevent unbounded gas
   costs
    uint256 public constant MAX_STUDENTS = 1000;

    /// @dev Maximum batch size for distribution to prevent gas limit
    issues
    uint256 public constant MAX_BATCH_SIZE = 50;

    // ============ STRUCTS ============
```

```
31
32      struct Student {
33          address studentAddress;
34          uint256 monthlyAmount;
35          bool isActive;
36          uint256 registeredAt;
37      }
38
39      // ============ STATE VARIABLES ============
40
41      /// @dev DREX token contract
42      IERC20 public immutable drexToken;
43
44      /// @dev Mapping from student address to student info
45      mapping(address => Student) public students;
46
47      /// @dev Mapping from student address to current balance
48      mapping(address => uint256) public balances;
49
50      /// @dev Array of all student addresses for iteration
51      address[] public studentAddresses;
52
53      /// @dev Total amount expected to be distributed monthly
54      uint256 public totalExpectedAmount;
55
56      /// @dev Total amount distributed in current period
57      uint256 public totalDistributedAmount;
58
59      /// @dev Last distribution timestamp
60      uint256 public lastDistributionTimestamp;
61
62      /// @dev Sum of all student balances
63      uint256 public studentsBalancesSum;
64
65      // ============ EVENTS ============
66
67      event StudentRegistered(
68          address indexed student,
69          uint256 monthlyAmount,
70          uint256 timestamp
71      );
72
```

```
73      event StudentUpdated(
74          address indexed student,
75          uint256 oldAmount,
76          uint256 newAmount,
77          uint256 timestamp
78      );
79
80      event StudentRemoved(
81          address indexed student,
82          uint256 timestamp
83      );
84
85      event MonthlyDistribution(
86          uint256 totalAmount,
87          uint256 studentsCount,
88          uint256 timestamp
89      );
90
91      event BatchDistribution(
92          uint256 startIndex,
93          uint256 endIndex,
94          uint256 amount,
95          uint256 timestamp
96      );
97
98      event Transfer(
99          address indexed from,
100         address indexed to,
101         uint256 amount,
102         uint256 timestamp
103     );
104
105     event DrexDeposited(
106         address indexed from,
107         uint256 amount,
108         uint256 timestamp
109     );
110
111     event EmergencyWithdraw(
112         address indexed admin,
113         uint256 amount,
114         uint256 timestamp
```

```
115     );
116
117     // Modificadores personalizados
118     modifier onlyAdminOrStaff() {
119         require(
120             hasRole(ADMIN_ROLE, msg.sender) || hasRole(STAFF_ROLE,
    msg.sender),
121             "StudentAssistanceVault: caller is not admin or staff"
122         );
123         _;
124     }
125
126     modifier validAddress(address addr) {
127         require(addr != address(0), "StudentAssistanceVault: invalid
    address");
128         _;
129     }
130
131     modifier validAmount(uint256 amount) {
132         require(amount > 0, "StudentAssistanceVault: invalid amount")
    ;
133         _;
134     }
135
136     constructor(address _drexToken) validAddress(_drexToken) {
137         drexToken = IERC20(_drexToken);
138         _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
139         _grantRole(ADMIN_ROLE, msg.sender);
140     }
141
142     // Funcao para registrar estudante
143     function registerStudent(
144         address studentAddress,
145         uint256 monthlyAmount
146     ) external onlyAdminOrStaff validAddress(studentAddress)
    validAmount(monthlyAmount) {
147         require(!isStudent(studentAddress), "StudentAssistanceVault:
    student already registered");
148         require(studentAddresses.length < MAX_STUDENTS, "
    StudentAssistanceVault: maximum students reached");
149
150         students[studentAddress] = Student({
```

```
151             studentAddress: studentAddress,
152             monthlyAmount: monthlyAmount,
153             isActive: true,
154             registeredAt: block.timestamp
155         });
156 
157         studentAddresses.push(studentAddress);
158         totalExpectedAmount += monthlyAmount;
159 
160         emit StudentRegistered(studentAddress, monthlyAmount, block.
    timestamp);
161     }
162 
163     // Funcao para remover estudante
164     function removeStudent(address studentAddress)
165         external onlyAdminOrStaff validAddress(studentAddress)
166     {
167         require(isStudent(studentAddress), "StudentAssistanceVault:
    student not registered");
168 
169         Student storage student = students[studentAddress];
170         uint256 currentBalance = balances[studentAddress];
171 
172         totalExpectedAmount -= student.monthlyAmount;
173         studentsBalancesSum -= currentBalance;
174         student.isActive = false;
175 
176         // Remove do array
177         for (uint256 i = 0; i < studentAddresses.length; i++) {
178             if (studentAddresses[i] == studentAddress) {
179                 studentAddresses[i] = studentAddresses[
    studentAddresses.length - 1];
180                 studentAddresses.pop();
181                 break;
182             }
183         }
184 
185         balances[studentAddress] = 0;
186         emit StudentRemoved(studentAddress, block.timestamp);
187     }
188 
189     // Funcao para depositar DREX
```

```
190     function depositDrex(uint256 amount)
191         external onlyRole(ADMIN_ROLE) validAmount(amount)
    nonReentrant
192     {
193         drexToken.safeTransferFrom(msg.sender, address(this), amount)
    ;
194         emit DrexDeposited(msg.sender, amount, block.timestamp);
195     }
196
197     // Funcao para distribuir em lotes (batch)
198     function distributeBatch(uint256 startIndex, uint256 endIndex)
199         external onlyAdminOrStaff nonReentrant
200     {
201         require(startIndex < studentAddresses.length, "
    StudentAssistanceVault: invalid start index");
202         require(endIndex <= studentAddresses.length, "
    StudentAssistanceVault: invalid end index");
203         require(endIndex > startIndex, "StudentAssistanceVault:
    invalid range");
204         require(endIndex - startIndex <= MAX_BATCH_SIZE, "
    StudentAssistanceVault: batch too large");
205
206         uint256 batchAmount = 0;
207
208         for (uint256 i = startIndex; i < endIndex; i++) {
209             address studentAddr = studentAddresses[i];
210             Student memory student = students[studentAddr];
211
212             if (student.isActive) {
213                 balances[studentAddr] += student.monthlyAmount;
214                 studentsBalancesSum += student.monthlyAmount;
215                 batchAmount += student.monthlyAmount;
216             }
217         }
218
219         totalDistributedAmount += batchAmount;
220         lastDistributionTimestamp = block.timestamp;
221
222         emit BatchDistribution(startIndex, endIndex, batchAmount,
    block.timestamp);
223     }
224
```

```
225     // Funcao para transferencia por estudantes
226     function transfer(address to, uint256 amount)
227         external validAddress(to) validAmount(amount) nonReentrant
228     {
229         require(isStudent(msg.sender), "StudentAssistanceVault:
    caller is not a registered student");
230         require(students[msg.sender].isActive, "
    StudentAssistanceVault: student is not active");
231         require(balances[msg.sender] >= amount, "
    StudentAssistanceVault: insufficient balance");
232 
233         balances[msg.sender] -= amount;
234         studentsBalancesSum -= amount;
235 
236         drexToken.safeTransfer(to, amount);
237 
238         emit Transfer(msg.sender, to, amount, block.timestamp);
239     }
240 
241     // Funcao de emergencia para retirar DREX
242     function emergencyWithdraw(uint256 amount)
243         external onlyRole(ADMIN_ROLE) validAmount(amount)
    nonReentrant
244     {
245         require(
246             drexToken.balanceOf(address(this)) >= amount,
247             "StudentAssistanceVault: insufficient contract balance"
248         );
249 
250         drexToken.safeTransfer(msg.sender, amount);
251         emit EmergencyWithdraw(msg.sender, amount, block.timestamp);
252     }
253 
254     // Funcoes de consulta
255     function getStudentCount() external view returns (uint256) {
256         return studentAddresses.length;
257     }
258 
259     function getStudentBalance(address studentAddress) external view
    returns (uint256) {
260         return balances[studentAddress];
261     }
```

```
262
263     function isStudent(address account) public view returns (bool) {
264         return students[account].studentAddress != address(0);
265     }
266
267     function getContractBalance() external view returns (uint256) {
268         return drexToken.balanceOf(address(this));
269     }
270 }
```

Listing A.1 – Contrato StudentAssistanceVault - Estrutura Principal

A.2 Servidor Backend

A.2.1 Middleware de Autenticacao

```
1  // middleware/auth.ts
2  import { Request, Response, NextFunction } from 'express';
3  import jwt from 'jsonwebtoken';
4  import { prisma } from '../database/client';
5
6  export interface AuthenticatedRequest extends Request {
7    user?: {
8      id: string;
9      username: string;
10     role: string;
11   };
12   student?: {
13     username: string;
14     walletAddress: string;
15     name: string;
16     role: 'student';
17   };
18 }
19
20 export const authenticateToken = async (
21   req: AuthenticatedRequest,
22   res: Response,
23   next: NextFunction
24 ) => {
25   try {
26     const authHeader = req.headers['authorization'];
```

```
27      const token = authHeader && authHeader.split(' ')[1];
28
29      if (!token) {
30        return res.status(401).json({ error: 'Access token required' })
    ;
31      }
32
33      const decoded = jwt.verify(token, process.env.JWT_SECRET!) as any
    ;
34
35      // Verifica se e token de estudante
36      if (decoded.role === 'student' && decoded.username) {
37        const student = await prisma.student.findUnique({
38          where: { username: decoded.username },
39          select: {
40            username: true,
41            walletAddress: true,
42            name: true,
43            active: true
44          }
45        });
46
47        if (!student || !student.active) {
48          return res.status(401).json({ error: 'Invalid or expired
    token' });
49        }
50
51        req.student = {
52          username: student.username,
53          walletAddress: student.walletAddress!,
54          name: student.name,
55          role: 'student'
56        };
57
58        return next();
59      }
60
61      // Token de administrador
62      req.user = {
63        id: decoded.userId,
64        username: decoded.username,
65        role: decoded.role
```

```
    };

    next();
  } catch (error) {
    return res.status(403).json({ error: 'Invalid or expired token'
   });
  }
};

export const requireRole = (allowedRoles: string[]) => {
  return (req: AuthenticatedRequest, res: Response, next:
   NextFunction) => {
    const userRole = req.user?.role || req.student?.role;

    if (!userRole || !allowedRoles.includes(userRole)) {
      return res.status(403).json({
        error: 'Insufficient permissions'
      });
    }

    next();
  };
};
```

Listing A.2 – Middleware de Autenticacao JWT

A.2.2 Rotas de Estudantes

```
// routes/students.ts
import { Router } from "express";
import bcrypt from "bcryptjs";
import { prisma } from "../database/client";
import { validateBody, validateQuery,
         createStudentSchema, studentQuerySchema } from "../utils/
   validation";
import { authenticateToken, requireStaff } from "../middleware/auth";

const router: Router = Router();

router.use(authenticateToken);

// GET /api/students - Listar estudantes com paginacao e filtros
```

```
14 router.get("/", validateQuery(studentQuerySchema),
15   async (req, res, next) => {
16     try {
17       const { page, limit, search, active } = req.query as any;
18       const offset = (page - 1) * limit;
19
20       const where: any = {};
21       if (active !== undefined) {
22         where.active = active;
23       }
24       if (search) {
25         where.OR = [
26           { name: { contains: search, mode: "insensitive" } },
27           { cpf: { contains: search } },
28           { walletAddress: { contains: search, mode: "insensitive" }
     },
29         ];
30       }
31
32       const [students, total] = await Promise.all([
33         prisma.student.findMany({
34           where,
35           include: {
36             spendingLimits: {
37               include: { expenseType: true }
38             }
39           },
40           orderBy: { createdAt: "desc" },
41           skip: offset,
42           take: limit,
43         }),
44         prisma.student.count({ where }),
45       ]);
46
47       res.json({
48         success: true,
49         data: {
50           students: students.map((student) => ({
51             ...student,
52             monthlyAmount: student.monthlyAmount.toString(),
53             spendingLimits: student.spendingLimits.map((limit) => ({
54               ...limit,
```

```
55                id: limit.id.toString(),
56                limitValue: limit.limitValue.toString(),
57              })),
58            })),
59            pagination: {
60              page,
61              limit,
62              total,
63              totalPages: Math.ceil(total / limit),
64            },
65          },
66        });
67      } catch (error) {
68        next(error);
69      }
70  });
71
72  // POST /api/students - Criar novo estudante
73  router.post("/", requireStaff, validateBody(createStudentSchema),
74    async (req, res, next) => {
75      try {
76        const { username, password, name, cpf, university,
77                course, monthlyAmount, walletAddress } = req.body;
78          return res.status(400).json({
79            success: false,
80            message: 'Validation errors',
81            errors: errors.array()
82          });
83        }
84
85        const { name, email, cpf, walletAddress, monthlyAmount } = req.
     body;
86
87        // Verificar se ja existe estudante com mesmo email ou wallet
88        const existingStudent = await prisma.student.findFirst({
89          where: {
90            OR: [
91              { email },
92              { walletAddress },
93              { cpf }
94            ]
95          }
```

```
96        });
97
98        if (existingStudent) {
99          return res.status(400).json({
100           success: false,
101           message: 'Student already exists with this email, CPF or
    wallet address'
102         });
103       }
104
105       const student = await prisma.student.create({
106         data: {
107           name,
108           email,
109           cpf,
110           walletAddress,
111           monthlyAmount: parseFloat(monthlyAmount),
112           active: true
113         }
114       });
115
116       res.status(201).json({
117         success: true,
118         data: student,
119         message: 'Student created successfully'
120       });
121     } catch (error) {
122       res.status(500).json({
123         success: false,
124         message: 'Error creating student',
125         error: error.message
126       });
127     }
128   },
129
130   // Atualizar estudante
131   async updateStudent(req, res) {
132     try {
133       const { id } = req.params;
134       const { monthlyAmount, active } = req.body;
135
136       const student = await prisma.student.update({
```

```
137         where: { id: parseInt(id) },
138         data: {
139           monthlyAmount: monthlyAmount ? parseFloat(monthlyAmount) :
    undefined,
140           active: active !== undefined ? active : undefined
141         }
142       });
143
144       res.json({
145         success: true,
146         data: student,
147         message: 'Student updated successfully'
148       });
149     } catch (error) {
150       res.status(500).json({
151         success: false,
152         message: 'Error updating student',
153         error: error.message
154       });
155     }
156   }
157 };
158
159 module.exports = studentController;
```

Listing A.3 – Rotas para Gerenciamento de Estudantes

A.3 Interface Web - Dashboard

A.3.1 Hook de Autenticacao

```
1  // hooks/useAuth.tsx
2  import { useState, useEffect, createContext, useContext } from 'react
       ';
3  import { api } from '../services/api';
4
5  interface User {
6    id: number;
7    username?: string;
8    name?: string;
9    role: string;
10   walletAddress?: string;
```

```
11   type: 'admin' | 'staff' | 'student';
12 }
13
14 interface AuthContextType {
15   user: User | null;
16   loading: boolean;
17   login: (credentials: any) => Promise<void>;
18   loginWithWallet: (walletAddress: string) => Promise<void>;
19   logout: () => void;
20 }
21
22 const AuthContext = createContext<AuthContextType | undefined>(
     undefined);
23
24 export const AuthProvider: React.FC<{ children: React.ReactNode }> =
     ({ children }) => {
25   const [user, setUser] = useState<User | null>(null);
26   const [loading, setLoading] = useState(true);
27
28   useEffect(() => {
29     const token = localStorage.getItem('token');
30     if (token) {
31       api.defaults.headers.Authorization = `Bearer ${token}`;
32       // Verificar se token ainda e valido
33       checkAuthStatus();
34     } else {
35       setLoading(false);
36     }
37   }, []);
38
39   const checkAuthStatus = async () => {
40     try {
41       const response = await api.get('/auth/me');
42       setUser(response.data.data);
43     } catch (error) {
44       localStorage.removeItem('token');
45       delete api.defaults.headers.Authorization;
46     } finally {
47       setLoading(false);
48     }
49   };
50
```

```
51   const login = async (credentials: { username: string; password:
      string }) => {
52     try {
53       const response = await api.post('/auth/login', credentials);
54       const { token, user: userData } = response.data.data;
55
56       localStorage.setItem('token', token);
57       api.defaults.headers.Authorization = `Bearer ${token}`;
58       setUser(userData);
59     } catch (error) {
60       throw error;
61     }
62   };
63
64   const loginWithWallet = async (walletAddress: string) => {
65     try {
66       const response = await api.post('/auth/student/wallet', {
      walletAddress });
67       const { token, student } = response.data.data;
68
69       localStorage.setItem('token', token);
70       api.defaults.headers.Authorization = `Bearer ${token}`;
71       setUser({ ...student, type: 'student' });
72     } catch (error) {
73       throw error;
74     }
75   };
76
77   const logout = () => {
78     localStorage.removeItem('token');
79     delete api.defaults.headers.Authorization;
80     setUser(null);
81   };
82
83   return (
84     <AuthContext.Provider value={{ user, loading, login,
      loginWithWallet, logout }}>
85       {children}
86     </AuthContext.Provider>
87   );
88 };
89
```

```
export const useAuth = () => {
  const context = useContext(AuthContext);
  if (context === undefined) {
    throw new Error('useAuth must be used within an AuthProvider');
  }
  return context;
};
```

Listing A.4 – Hook customizado para autenticacao

A.3.2 Componente de Dashboard

```
// components/DashboardHome.tsx
import React, { useState, useEffect } from 'react';
import { api } from '../services/api';
import { useAuth } from '../hooks/useAuth';

interface DashboardStats {
  totalStudents: number;
  activeStudents: number;
  totalDistributed: number;
  monthlyBudget: number;
  recentTransactions: Transaction[];
}

interface Transaction {
  id: string;
  from: string;
  to: string;
  amount: string;
  timestamp: string;
  transactionHash: string;
}

const DashboardHome: React.FC = () => {
  const { user } = useAuth();
  const [stats, setStats] = useState<DashboardStats | null>(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetchDashboardData();
  }, []);
```

```
31
32  const fetchDashboardData = async () => {
33    try {
34      const response = await api.get('/dashboard/stats');
35      setStats(response.data.data);
36    } catch (error) {
37      console.error('Error fetching dashboard data:', error);
38    } finally {
39      setLoading(false);
40    }
41  };
42
43  if (loading) {
44    return (
45      <div className="flex justify-center items-center h-64">
46        <div className="animate-spin rounded-full h-12 w-12 border-b
     -2 border-blue-600"></div>
47      </div>
48    );
49  }
50
51  return (
52    <div className="space-y-6">
53      <div className="flex justify-between items-center">
54        <h1 className="text-2xl font-bold text-gray-900">
55          {user?.type === 'student' ? 'Meu Dashboard' : 'Dashboard
     Administrativo'}
56        </h1>
57        <div className="text-sm text-gray-500">
58          Bem-vindo, {user?.name || user?.username}
59        </div>
60      </div>
61
62      {user?.type !== 'student' && stats && (
63        <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols
     -4 gap-6">
64          <StatsCard
65            title="Total de Estudantes"
66            value={stats.totalStudents}
67            icon="Users"
68            color="blue"
69          />
```

```
70          <StatsCard
71            title="Estudantes Ativos"
72            value={stats.activeStudents}
73            icon="Check"
74            color="green"
75          />
76          <StatsCard
77            title="Total Distribuido"
78            value={`R$ ${stats.totalDistributed.toLocaleString()}`}
79            icon="Money"
80            color="yellow"
81          />
82          <StatsCard
83            title="Orcamento Mensal"
84            value={`R$ ${stats.monthlyBudget.toLocaleString()}`}
85            icon="Chart"
86            color="purple"
87          />
88        </div>
89      )}
90
91      {stats?.recentTransactions && (
92        <div className="bg-white rounded-lg shadow p-6">
93          <h2 className="text-lg font-semibold mb-4">Transacoes
    Recentes</h2>
94          <div className="overflow-x-auto">
95            <table className="min-w-full divide-y divide-gray-200">
96              <thead className="bg-gray-50">
97                <tr>
98                  <th className="px-6 py-3 text-left text-xs font-
    medium text-gray-500 uppercase tracking-wider">
99                    De
100                 </th>
101                 <th className="px-6 py-3 text-left text-xs font-
    medium text-gray-500 uppercase tracking-wider">
102                   Para
103                 </th>
104                 <th className="px-6 py-3 text-left text-xs font-
    medium text-gray-500 uppercase tracking-wider">
105                   Valor
106                 </th>
```

```
107                   <th className="px-6 py-3 text-left text-xs font-
    medium text-gray-500 uppercase tracking-wider">
108                     Data
109                   </th>
110                 </tr>
111               </thead>
112               <tbody className="bg-white divide-y divide-gray-200">
113                 {stats.recentTransactions.map((transaction) => (
114                   <tr key={transaction.id}>
115                     <td className="px-6 py-4 whitespace-nowrap text-
    sm text-gray-900">
116                       {transaction.from.slice(0, 10)}...
117                     </td>
118                     <td className="px-6 py-4 whitespace-nowrap text-
    sm text-gray-900">
119                       {transaction.to.slice(0, 10)}...
120                     </td>
121                     <td className="px-6 py-4 whitespace-nowrap text-
    sm text-gray-900">
122                       R$ {parseFloat(transaction.amount).
    toLocaleString()}
123                     </td>
124                     <td className="px-6 py-4 whitespace-nowrap text-
    sm text-gray-500">
125                       {new Date(transaction.timestamp).
    toLocaleDateString()}
126                     </td>
127                   </tr>
128                 ))}
129               </tbody>
130             </table>
131           </div>
132         </div>
133       )}
134     </div>
135   );
136 };
137
138 const StatsCard: React.FC<{
139   title: string;
140   value: string | number;
141   icon: string;
```

```
142   color: string;
143 }> = ({ title, value, icon, color }) => {
144   const colorClasses = {
145     blue: 'bg-blue-500',
146     green: 'bg-green-500',
147     yellow: 'bg-yellow-500',
148     purple: 'bg-purple-500',
149   };
150
151   return (
152     <div className="bg-white rounded-lg shadow p-6">
153       <div className="flex items-center">
154         <div className={`${colorClasses[color]} rounded-md p-3 text-
    white text-xl`}>
155           {icon}
156         </div>
157         <div className="ml-4">
158           <p className="text-sm font-medium text-gray-500">{title}</p
    >
159           <p className="text-2xl font-semibold text-gray-900">{value
    }</p>
160         </div>
161       </div>
162     </div>
163   );
164 };
165
166 export default DashboardHome;
```

Listing A.5 – Componente principal do Dashboard

A.4 Scripts de Deploy

A.4.1 Script Principal de Deploy

```
1 #!/bin/bash
2
3 echo "=== Deploy do Sistema de Assistencia Estudantil ==="
4
5 # Verificar se Docker esta rodando
6 if ! docker info > /dev/null 2>&1; then
7     echo "Docker nao esta rodando. Iniciando Docker..."
```

```
    open -a Docker
    sleep 10
fi

# Deploy da infraestrutura DREX
echo "1. Fazendo deploy da infraestrutura DREX..."
cd packages/drex-piloto
npm install
npm run deploy:integrated

if [ $? -ne 0 ]; then
    echo "Erro no deploy da infraestrutura DREX"
    exit 1
fi

# Deploy do contrato de assistencia estudantil
echo "2. Fazendo deploy do StudentAssistanceVault..."
cd ../student-assistance-vault
npm install
npm run deploy

if [ $? -ne 0 ]; then
    echo "Erro no deploy do StudentAssistanceVault"
    exit 1
fi

# Configuracao do banco de dados
echo "3. Configurando banco de dados..."
cd ../student-assistance-server
npm install
npx prisma generate
npx prisma db push
npx prisma db seed

# Configuracao de estudantes de teste
echo "4. Configurando estudantes de teste..."
npm run setup:test-students

# Iniciar servidor backend
echo "5. Iniciando servidor backend..."
npm run dev &
SERVER_PID=$!
```

```
50
51 # Aguardar servidor iniciar
52 sleep 5
53
54 # Iniciar dashboard
55 echo "6. Iniciando dashboard..."
56 cd ../student-assistance-dashboard
57 npm install
58 npm run dev &
59 DASHBOARD_PID=$!
60
61 echo "=== Deploy concluido com sucesso! ==="
62 echo "Servidor backend: http://localhost:3001"
63 echo "Dashboard: http://localhost:3000"
64 echo ""
65 echo "Para parar os servicos:"
66 echo "kill $SERVER_PID $DASHBOARD_PID"
```

Listing A.6 – Script de deploy automatizado

A.4.2 Configuracao do Hardhat

```
1 // hardhat.config.ts
2 import { HardhatUserConfig } from "hardhat/config";
3 import "@nomicfoundation/hardhat-toolbox";
4
5 const config: HardhatUserConfig = {
6   solidity: {
7     version: "0.8.24",
8     settings: {
9       optimizer: {
10        enabled: true,
11        runs: 200,
12      },
13    },
14  },
15  networks: {
16    hardhat: {
17      accounts: {
18        mnemonic: "test test test test test test test test test test
    test junk",
19        count: 60, // Gera 60 contas para testes
```

```
      },
    },
    localhost: {
      url: "http://localhost:8545",
      chainId: 1337,
      accounts: {
        mnemonic: "test test test test test test test test test test test junk",
        count: 60,
      },
    },
    besu: {
      url: "http://besu-node:8545",
      chainId: 1337,
      accounts: {
        mnemonic: "test test test test test test test test test test test junk",
        count: 60,
      },
    },
  },
  paths: {
    sources: "./contracts",
    tests: "./test",
    cache: "./cache",
    artifacts: "./artifacts",
  },
};

export default config;
```

Listing A.7 – Configuracao do Hardhat para testes e deploy

A.5 Configuracoes de Banco de Dados

A.5.1 Schema Prisma

```
// schema.prisma
generator client {
  provider = "prisma-client-js"
}

```

```
6  datasource db {
7    provider = "postgresql"
8    url      = env("DATABASE_URL")
9  }
10
11 model Student {
12   walletAddress    String           @unique @map("wallet_address")
13   username         String           @unique
14   passwordHash     String           @map("password_hash")
15   name             String
16   cpf              String           @unique
17   university       String
18   course           String?
19   monthlyAmount    Decimal          @map("monthly_amount") @db.
       Decimal(20, 2)
20   active           Boolean          @default(true)
21   createdAt        DateTime         @default(now()) @map("created_at
       ")
22   updatedAt        DateTime         @updatedAt @map("updated_at")
23
24   spendingLimits   SpendingLimit[]
25   transactions     Transaction[]
26   registeredReceivers Receiver[]    @relation("
       StudentRegisteredReceivers")
27
28   @@map("students")
29 }
30
31 model ExpenseType {
32   id               BigInt           @id @default(autoincrement())
33   name             String           @unique
34   description      String?
35   category         String
36   active           Boolean          @default(true)
37   createdAt        DateTime         @default(now()) @map("created_at
       ")
38
39   spendingLimits   SpendingLimit[]
40   transactions     Transaction[]
41   receivers        ReceiverExpenseType[]
42
43   @@map("expense_types")
```

```
}

model SpendingLimit {
  id               BigInt          @id @default(autoincrement())
  studentAddress   String          @map("student_address")
  expenseTypeId    BigInt          @map("expense_type_id")
  limitValue       Decimal         @map("limit_value") @db.Decimal
   (20, 2)
  limitType        String          @map("limit_type")
  createdAt        DateTime        @default(now()) @map("created_at
   ")
  updatedAt        DateTime        @updatedAt @map("updated_at")

  student          Student         @relation(fields: [studentAddress
   ],
                                          references: [
   walletAddress],
                                          onDelete: Cascade)
  expenseType      ExpenseType     @relation(fields: [expenseTypeId
   ],
                                          references: [id],
                                          onDelete: Cascade)

  @@unique([studentAddress, expenseTypeId])
  @@map("spending_limits")
}

model Transaction {
  txHash           String          @id @map("tx_hash")
  fromAddress      String          @map("from_address")
  toAddress        String          @map("to_address")
  amount           Decimal         @db.Decimal(20, 8)
  timestamp        DateTime
  blockNumber      BigInt          @map("block_number")
  studentAddress   String?         @map("student_address")
  receiverAddress  String?         @map("receiver_address")
  expenseTypeId    BigInt?         @map("expense_type_id")
  isUnknownDestiny Boolean         @default(true) @map("
   is_unknown_destiny")
  indexedAt        DateTime        @default(now()) @map("indexed_at
   ")

```

```
79   student          Student?         @relation(fields: [studentAddress
      ],
80                                             references: [
      walletAddress])
81   receiver         Receiver?        @relation(fields: [
      receiverAddress],
82                                             references: [address])
83   expenseType      ExpenseType?     @relation(fields: [expenseTypeId
      ],
84                                             references: [id])
85
86   @@map("transactions")
87 }
88
89 model Receiver {
90   address          String           @id
91   name             String?
92   cpfCnpj          String           @map("cpf_cnpj")
93   type             String
94   registeredBy     String?          @map("registered_by")
95   isReported       Boolean          @default(false) @map("is_reported
      ")
96   reportDescription String?         @map("report_description")
97   createdAt        DateTime         @default(now()) @map("created_at
      ")
98
99   registeredByStudent Student?      @relation("
      StudentRegisteredReceivers",
100                                            fields: [registeredBy],
101                                            references: [
      walletAddress])
102  transactions     Transaction[]
103  expenseTypes     ReceiverExpenseType[]
104
105  @@map("receivers")
106 }
```

Listing A.8 – Schema do banco de dados com Prisma

APÊNDICE B – Artigo em Formato SBC

Este apêndice apresenta o conteúdo deste trabalho de conclusão de curso no formato de artigo científico segundo as normas da Sociedade Brasileira de Computação (SBC). O artigo sintetiza os principais aspectos da pesquisa desenvolvida, mantendo a estrutura e formatação adequadas para submissão em eventos e periódicos da área de Ciência da Computação.

Sistema de Assistência Estudantil e Real Digital: Usando Blockchain para melhorar o financiamento de assistências sociais

Luiz Gustavo Abou Hatem de Liz[1], Jean Everson Martina[1], Lucas Palma[1]

[1]Departamento de Informática e Estatística (INE)
Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brasil

{luiz.gustavo, jean.everson, lucas.palma}@ufsc.br

Abstract

This work presents a proposal for implementing a student assistance system using blockchain technology, specifically DREX (Digital Real), to improve the financing and management of social assistance programs in universities. The study explores how currency tokenization and smart contracts can provide greater transparency, traceability, and efficiency in the distribution of resources to students in socioeconomically vulnerable situations. The methodology adopted includes the development of a functional prototype based on Hyperledger Besu, which simulates the DREX environment, enabling the issuance and transfer of tokens representing the Digital Real. As a result, the technical feasibility of the proposed solution is demonstrated, and the potential benefits for educational institutions and beneficiaries of student assistance programs are discussed. It is concluded that blockchain technology can significantly contribute to the modernization of social assistance systems, reducing operational costs, minimizing fraud, and expanding financial inclusion for students.

Resumo

Este trabalho apresenta uma proposta de implementação de um sistema de assistência estudantil utilizando a tecnologia blockchain, especificamente o DREX (Real Digital), para melhorar o financiamento e a gestão de programas de assistência social em universidades. O estudo explora como a tokenização de moedas e contratos inteligentes pode proporcionar maior transparência, rastreabilidade e eficiência na distribuição de recursos para estudantes em situação de vulnerabilidade socioeconômica. A metodologia adotada inclui o desenvolvimento de um protótipo funcional baseado em Hyperledger Besu, que simula o ambiente do DREX, permitindo a emissão e transferência de tokens representando o Real Digital. Como resultado, demonstra-

se a viabilidade técnica da solução proposta e discutem-se os benefícios potenciais para as instituições de ensino e para os beneficiários dos programas de assistência estudantil. Conclui-se que a tecnologia blockchain pode contribuir significativamente para a modernização dos sistemas de assistência social, reduzindo custos operacionais, minimizando fraudes e ampliando a inclusão financeira dos estudantes.

1. Introdução

Desde a implementação do Programa Nacional de Assistência Estudantil (PNAES) em 2010, milhares de estudantes brasileiros têm encontrado nas políticas de permanência universitária o suporte necessário para concluir seus cursos superiores (República, 2010). Contudo, quem já teve contato com esses sistemas—seja como beneficiário, gestor ou pesquisador—reconhece que ainda há muito a ser melhorado.

Existem alguns desafios evidentes, sendo um dos maiores a falta de transparência na execução do programa já constatada pelo Tribunal de Contas da União. A fiscalização identificou que muitas universidades não divulgam informações detalhadas sobre os beneficiários e os resultados das ações de assistência estudantil. A ausência de relatórios de avaliação e dados abertos dificulta o acompanhamento da execução do programa, comprometendo a capacidade de monitoramento e controle social sobre os recursos públicos utilizados (Tribunal de Contas da União, 2018). Enquanto isso, assistimos ao surgimento de tecnologias que prometem revolucionar a forma como lidamos com transações financeiras e gestão de recursos públicos.

É nesse contexto que surge uma oportunidade concreta de transformação. O projeto DREX (Digital Real Experimental), liderado pelo Banco Central do Brasil, representou uma iniciativa de modernização do sistema financeiro nacional através da implementação de uma moeda digital de banco central (CBDC). Embora o piloto DREX tenha sido descontinuado pelo Banco Central em 2025 (Econômico, 2025), a infraestrutura tecnológica desenvolvida durante o projeto viabilizou o desenvolvimento de soluções inovadoras para a gestão de recursos públicos, especialmente em áreas sensíveis como a assistência estudantil (Brasil, 2023).

Este trabalho apresenta o desenvolvimento de um sistema de assistência estudantil que integra a tecnologia blockchain com os contratos inteligentes e a arquitetura desenvolvidos no contexto do piloto DREX, resultando em uma solução prática e funcional que simplifica o processo de concessão de benefícios e aumenta a transparência na aplicação dos recursos públicos, demonstrando a viabilidade técnica dessa abordagem no contexto brasileiro mesmo após a descontinuação do projeto oficial.

1.1. Objetivos

O objetivo geral deste trabalho é implementar e validar tecnicamente um sistema de assistência estudantil baseado nos contratos inteligentes e arquitetura desenvolvidos no contexto

do piloto DREX, utilizando tecnologia blockchain para automatizar a distribuição de benefícios e proporcionar maior transparência e rastreabilidade na gestão dos recursos.

Como objetivos específicos, destacam-se: (i) implementar a infraestrutura blockchain utilizando os contratos oficiais do piloto DREX disponibilizados pela Wire Labs; (ii) desenvolver um contrato inteligente (StudentAssistanceVault) que implemente funcionalidades específicas para gestão de assistência estudantil; (iii) criar um sistema de indexação blockchain para monitoramento e análise das transações; (iv) implementar uma API completa para integração com sistemas universitários existentes; (v) desenvolver um processo de implantação automatizado; e (vi) validar tecnicamente a solução através de testes integrados.

2. Fundamentação Teórica

2.1. Blockchain e Contratos Inteligentes

Blockchain é uma tecnologia de registro distribuído que permite o armazenamento de informações de forma segura, transparente e imutável. Proposto inicialmente por Nakamoto (Nakamoto, 2008), o conceito evoluiu significativamente desde sua aplicação original no Bitcoin, expandindo-se para diversos domínios através da implementação de contratos inteligentes.

Contratos inteligentes são programas autoexecutáveis que codificam regras de negócio e executam automaticamente quando condições predefinidas são satisfeitas (Szabo, 1997). Na plataforma Ethereum, descrita por Antonopoulos e Wood (Antonopoulos; Wood, 2018), esses contratos são escritos em linguagens como Solidity e executados na Ethereum Virtual Machine (EVM), proporcionando um ambiente determinístico e verificável para aplicações descentralizadas.

2.2. Moedas Digitais de Bancos Centrais (CBDCs)

As moedas digitais de bancos centrais (Central Bank Digital Currencies - CBDCs) representam uma evolução natural dos sistemas monetários tradicionais, combinando os benefícios da digitalização com a estabilidade e confiança associadas às moedas fiduciárias emitidas por bancos centrais (Auer; Haene; Holden, 2021).

O projeto DREX (Digital Real Experimental) foi a iniciativa brasileira para desenvolvimento de uma CBDC baseada em blockchain. Embora o piloto tenha sido descontinuado em 2025 (Econômico, 2025), a infraestrutura desenvolvida criou um legado técnico valioso. O sistema utilizava Hyperledger Besu, uma implementação de blockchain compatível com Ethereum, e incluía contratos inteligentes para emissão de moeda digital (RealDigital), tokenização de ativos (RealTokenizado) e operações de liquidação (Brasil, 2023; Araújo; Barbosa, 2023).

2.3. Programa Nacional de Assistência Estudantil (PNAES)

O PNAES, instituído pelo Decreto nº 7.234/2010, tem como objetivo viabilizar a igualdade de oportunidades entre todos os estudantes e contribuir para a melhoria do desempenho

acadêmico, a partir de medidas que buscam combater situações de repetência e evasão decorrentes da insuficiência de condições financeiras (República, 2010).

O programa prevê ações nas áreas de moradia estudantil, alimentação, transporte, atenção à saúde, inclusão digital, cultura, esporte, creche, apoio pedagógico e acesso, participação e aprendizagem de estudantes com deficiência, transtornos globais do desenvolvimento e altas habilidades e superdotação. No entanto, auditoria do TCU identificou fragilidades relacionadas à transparência e controle na execução do programa (Tribunal de Contas da União, 2018).

3. Metodologia

Este trabalho adotou uma abordagem de pesquisa aplicada com desenvolvimento experimental, combinando revisão bibliográfica, prototipação e validação técnica. A metodologia foi estruturada em cinco fases principais:

Fase 1 - Revisão Bibliográfica: Estudo da literatura sobre blockchain, CBDCs, DREX e assistência estudantil, identificando lacunas e oportunidades de aplicação da tecnologia.

Fase 2 - Análise dos Contratos do DREX: Análise detalhada dos contratos oficiais do piloto DREX disponibilizados pela Wire Labs, compreendendo sua arquitetura e funcionalidades.

Fase 3 - Desenvolvimento da Solução: Implementação da infraestrutura blockchain, desenvolvimento do contrato StudentAssistanceVault, criação de sistema de indexação, API backend e interface administrativa.

Fase 4 - Implantação e Configuração: Desenvolvimento de processo de implantação automatizado usando Docker e scripts de configuração.

Fase 5 - Validação Técnica: Execução de testes unitários, testes de integração e demonstração funcional do sistema completo.

As ferramentas utilizadas incluíram: Solidity para desenvolvimento de contratos inteligentes; Hardhat como framework de desenvolvimento Ethereum; Hyperledger Besu como nó blockchain; Ponder para indexação de eventos blockchain; Node.js/Express para API backend; React/TypeScript para interface web; e Docker para containerização.

4. Arquitetura da Solução

4.1. Visão Geral do Sistema

A arquitetura proposta integra múltiplos componentes que trabalham em conjunto para proporcionar um sistema completo de gestão de assistência estudantil baseado em blockchain.

O sistema é composto por cinco camadas principais:

1. **Camada Blockchain:** Rede Hyperledger Besu executando os contratos oficiais do DREX (RealDigital, AddressDiscovery, KeyDictionary, STR) e o contrato StudentAssistanceVault desenvolvido neste trabalho.

2. **Camada de Indexação:** Sistema Ponder que monitora eventos emitidos pelos contratos inteligentes e mantém um banco de dados relacional sincronizado.

3. **Camada de Backend:** API REST desenvolvida em Node.js/Express que expõe funcionalidades para gestão de estudantes, beneficiários e tipos de despesa.

4. **Camada de Interface:** Dashboard administrativo desenvolvido em React/TypeScript para operação do sistema.

5. **Camada de Integração:** Scripts de implantação automatizados e configurações Docker para facilitar a instalação e operação.

4.2. Contrato StudentAssistanceVault

O contrato StudentAssistanceVault implementa a lógica central do sistema de assistência estudantil. Suas principais funcionalidades incluem:

Gestão de Estudantes: Registro e remoção de estudantes elegíveis para receber benefícios, com controle de acesso baseado em papéis (ADMIN_ROLE e OPERATOR_ROLE).

Distribuição de Benefícios: Duas modalidades de distribuição - individual para casos específicos e em lote para distribuição mensal eficiente, otimizando custos de gas.

Rastreabilidade: Cada transferência emite eventos (StudentRegistered, FundsDistributed, FundsTransferred) que são indexados e disponibilizados para auditoria.

Controles de Segurança: Implementação de controles de acesso, validações de saldo e limites operacionais para garantir a integridade das operações.

4.3. Sistema de Indexação

O sistema de indexação foi desenvolvido utilizando Ponder, um framework especializado em sincronização de dados blockchain. O indexador monitora continuamente os eventos emitidos pelos contratos inteligentes e mantém um banco de dados PostgreSQL sincronizado.

A configuração do Ponder define quais contratos e eventos devem ser monitorados, enquanto handlers processam cada evento capturado, extraindo informações relevantes e armazenando-as em formato estruturado. Isso permite consultas eficientes sobre histórico de transações, estatísticas de distribuição e dados de beneficiários sem necessidade de varredura completa da blockchain.

4.4. API Backend

A API REST desenvolvida em Node.js/Express fornece endpoints para operação do sistema, incluindo:

- **/students:** Gestão de estudantes registrados
- **/receivers:** Gestão de beneficiários e histórico de recebimentos

- **/expense-types:** Configuração de tipos de despesa
- **/auth:** Autenticação e controle de acesso
- **/blockchain:** Interface para operações na blockchain

A API abstrai a complexidade da interação com a blockchain, fornecendo uma interface simples para sistemas universitários existentes.

5. Implementação e Resultados

5.1. Processo de Implantação

Foi desenvolvido um processo de implantação completamente automatizado que configura toda a infraestrutura necessária. O processo inclui:

1. Inicialização da rede Hyperledger Besu com configuração de validadores
2. Deploy dos contratos oficiais do DREX (RealDigital, AddressDiscovery, KeyDictionary, STR)
3. Deploy do contrato StudentAssistanceVault
4. Configuração de papéis e permissões iniciais
5. Inicialização do sistema de indexação Ponder
6. Configuração e inicialização da API backend
7. Configuração da interface administrativa

Todo o processo é orquestrado através de Docker Compose, permitindo que o sistema completo seja implantado com um único comando. Scripts de automação garantem a configuração correta de endereços de contratos, credenciais e conectividade entre componentes.

5.2. Validação Funcional

A validação funcional foi realizada através de suite de testes abrangente que verifica:

Testes Unitários dos Contratos: Validação de funcionalidades individuais do StudentAssistanceVault, incluindo registro de estudantes, distribuição individual, distribuição em lote e controles de acesso.

Testes de Integração: Verificação da integração entre contratos DREX e StudentAssistanceVault, incluindo fluxo completo de emissão de RealDigital, transferência para vault e distribuição para estudantes.

Testes de Performance: Medição de custos de gas e tempos de execução para operações críticas. A distribuição em lote demonstrou economia significativa de gas comparada a múltiplas distribuições individuais.

Demonstração End-to-End: Cenário completo simulando distribuição mensal de benefícios para múltiplos estudantes, com validação de todos os componentes do sistema.

5.3. Métricas de Desempenho

Foram coletadas métricas para avaliar o desempenho da solução. A distribuição em lote demonstrou eficiência significativa, onde o custo por estudante diminui com o aumento do tamanho do lote. Por exemplo, enquanto uma distribuição individual custa aproximadamente 92.315 gas, uma distribuição em lote para 10 estudantes custa 354.628 gas (média de 35.463 gas por estudante), representando economia de aproximadamente 62% por beneficiário.

5.4. Transparência e Rastreabilidade

O sistema implementado proporciona transparência completa através de múltiplos mecanismos:

- **Registros Imutáveis:** Todas as transações ficam permanentemente registradas na blockchain
- **Eventos Auditáveis:** Cada operação emite eventos estruturados que podem ser consultados
- **Trilha de Auditoria:** O sistema de indexação mantém histórico completo de todas as distribuições
- **Consultas Públicas:** Dados anonimizados podem ser disponibilizados para controle social

6. Discussão

6.1. Benefícios da Solução

A solução desenvolvida demonstra diversos benefícios potenciais para gestão de assistência estudantil:

Transparência: A natureza imutável e auditável da blockchain proporciona transparência sem precedentes, endereçando as fragilidades identificadas pelo TCU (Tribunal de Contas da União, 2018).

Eficiência Operacional: A automatização através de contratos inteligentes reduz processos manuais e minimiza possibilidades de erro.

Rastreabilidade: A capacidade de rastrear cada centavo desde a emissão até o uso final facilita auditorias e prestação de contas.

Redução de Custos: A eliminação de intermediários e automatização de processos pode reduzir custos operacionais significativamente.

Escalabilidade: A arquitetura desenvolvida pode escalar para atender múltiplas instituições e programas de assistência.

6.2. Limitações e Desafios

Apesar dos resultados positivos, algumas limitações devem ser consideradas:

Descontinuação do DREX Oficial: A descontinuação do piloto DREX pelo Banco Central significa que a solução opera em ambiente experimental, não integrado ao sistema financeiro oficial.

Requisitos de Infraestrutura: A operação de blockchain requer infraestrutura especializada e conhecimento técnico.

Custos de Transação: Embora otimizados, os custos de gas ainda representam uma consideração operacional.

Privacidade vs Transparência: O equilíbrio entre transparência pública e privacidade dos beneficiários requer atenção cuidadosa.

Adoção e Mudança Cultural: A implementação bem-sucedida requer mudanças nos processos e cultura organizacional das instituições.

7. Conclusão

Este trabalho demonstrou a viabilidade técnica de implementar um sistema de assistência estudantil baseado nos contratos inteligentes e arquitetura desenvolvidos no contexto do piloto DREX. A solução desenvolvida proporciona transparência, rastreabilidade e eficiência significativamente superiores aos sistemas tradicionais, endereçando problemas identificados por órgãos de controle.

A implementação completa, incluindo infraestrutura blockchain, contratos inteligentes, sistema de indexação, API backend e interface administrativa, demonstra que a tecnologia blockchain possui aplicabilidade prática no contexto de programas sociais brasileiros. A disponibilização do código-fonte completo contribui para a transparência da pesquisa e facilita trabalhos futuros.

Embora a descontinuação do piloto DREX oficial represente um desafio para adoção em produção, a infraestrutura tecnológica desenvolvida permanece relevante e pode ser adaptada para futuras iniciativas de moeda digital no Brasil. O conhecimento e ferramentas gerados durante este trabalho demonstram que a integração entre políticas públicas e tecnologia blockchain pode trazer benefícios tangíveis para a sociedade.

Como direções para trabalhos futuros, destacam-se: desenvolvimento de mecanismos de privacidade preservando transparência (zero-knowledge proofs); implementação de funcionalidades avançadas como contratos condicionais e liberação progressiva de recursos; estudo de

viabilidade econômica comparando custos operacionais com sistemas tradicionais; piloto em ambiente real com instituição parceira; adaptação para outros programas de assistência social; e integração com sistemas de pagamento existentes e carteiras digitais.

A convergência entre blockchain e políticas públicas representa uma área promissora de pesquisa e desenvolvimento, e este trabalho contribui para o avanço do conhecimento nesse domínio emergente.

APÊNDICE C – Instruções de Instalação e Execução

Este apêndice apresenta as instruções completas para baixar, configurar e executar o sistema de assistência estudantil desenvolvido neste trabalho. O código-fonte completo está disponível publicamente no repositório da UFSC.

C.1 Requisitos de Sistema

Para executar o sistema completo, são necessários os seguintes requisitos:

C.1.1 Requisitos Mínimos de Hardware

- **Processador:** CPU com pelo menos 4 núcleos (recomendado: 8 núcleos)
- **Memória RAM:** 8 GB (recomendado: 16 GB)
- **Armazenamento:** 20 GB de espaço livre
- **Conexão de rede:** Conexão de internet estável para download de dependências

C.1.2 Requisitos de Software

- **Docker:** Versão 20.10 ou superior
- **Docker Compose:** Versão 2.0 ou superior
- **Git:** Para clonar o repositório
- **Node.js:** Versão 18 ou superior (opcional, para desenvolvimento)
- **pnpm:** Gerenciador de pacotes (opcional, para desenvolvimento)

C.1.3 Sistemas Operacionais Suportados

O sistema foi testado e é compatível com:

- Linux (Ubuntu 20.04+, Debian 11+, Fedora 35+)
- macOS (Big Sur 11.0+, Monterey 12.0+, Ventura 13.0+)
- Windows 10/11 com WSL2 (Windows Subsystem for Linux 2)

C.2 Download do Código-Fonte

C.2.1 Clonando o Repositório

O código-fonte está hospedado no GitLab da UFSC. Para baixar o repositório, execute os seguintes comandos no terminal:

```
# Clone o reposit rio
git clone https://codigos.ufsc.br/luiz.gustavo.hatem/student-
   assistance-drex.git

# Entre no diret rio do projeto
cd student-assistance-drex
```

Listing C.1 – Clonando o repositório

C.2.2 Estrutura do Repositório

Após clonar o repositório, a estrutura de diretórios será:

```
student-assistance-drex/
          docker-compose.yml           # Orquestra o de containers
          Dockerfile.besu              # Imagem customizada do Besu
          package.json                 # Scripts do monorepo
          pnpm-workspace.yaml          # Configura o do workspace
          README.md                    # Documenta o principal
          packages/                    # Pacotes do monorepo
                drex-piloto/            # Contratos oficiais DREX
                student-assistance-vault/    # Contrato principal
                student-assistance-indexer/  # Indexador blockchain
                student-assistance-server/   # API Backend
                student-assistance-dashboard/ # Interface web
          scripts/                     # Scripts auxiliares
```

C.3 Instalação e Configuração

C.3.1 Verificação de Dependências

Antes de iniciar a instalação, verifique se o Docker e Docker Compose estão instalados corretamente:

```
# Verificar vers o do Docker
docker --version
# Sa da esperada: Docker version 20.10.x ou superior

```

```
# Verificar vers o do Docker Compose
docker-compose --version
#  Sa da esperada: Docker Compose version 2.x.x ou superior

# Testar execu  o do Docker
docker run hello-world
# Deve executar sem erros
```

Listing C.2 – Verificando instalação do Docker

Se o Docker não estiver instalado, consulte a documentação oficial em https://docs.docker.com/get-docker/ para instruções específicas do seu sistema operacional.

C.3.2 Instalação do Docker (Resumo)

C.3.2.1 Linux (Ubuntu/Debian)

```
# Atualizar pacotes
sudo apt-get update

# Instalar depend ncias
sudo apt-get install ca-certificates curl gnupg

# Adicionar chave GPG oficial do Docker
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
  sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

# Adicionar reposit rio
echo \
  "deb [arch="$(dpkg --print-architecture)" \
  signed-by=/etc/apt/keyrings/docker.gpg] \
  https://download.docker.com/linux/ubuntu \
  "$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# Instalar Docker Engine
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io \
  docker-buildx-plugin docker-compose-plugin

# Adicionar usu rio ao grupo docker (evita necessidade de sudo)
sudo usermod -aG docker $USER
```

```
28
29 # Reiniciar sess o ou executar:
30 newgrp docker
```

Listing C.3 – Instalando Docker no Linux

C.3.2.2 macOS

Para macOS, recomenda-se instalar o Docker Desktop através do site oficial:

1. Acesse https://www.docker.com/products/docker-desktop
2. Baixe o instalador para macOS (Intel ou Apple Silicon)
3. Execute o instalador .dmg
4. Arraste o ícone do Docker para a pasta Applications
5. Abra o Docker Desktop e aguarde a inicialização

C.3.2.3 Windows com WSL2

1. Habilite o WSL2 seguindo as instruções da Microsoft
2. Instale uma distribuição Linux (Ubuntu recomendado) da Microsoft Store
3. Instale o Docker Desktop para Windows
4. Configure o Docker Desktop para usar WSL2 backend
5. Execute os comandos dentro do terminal WSL2

C.4 Execução do Sistema

C.4.1 Deploy Completo com Docker Compose

A forma mais simples de executar o sistema completo é utilizando Docker Compose, que orquestra todos os serviços necessários:

```
1 # No diret rio raiz do projeto
2 docker-compose up -d
3
4 # Verificar logs de todos os servi os
5 docker-compose logs -f
6
7 # Verificar status dos containers
8 docker-compose ps
```

Listing C.4 – Iniciando o sistema com Docker Compose

O parâmetro `-d` (detached) executa os containers em segundo plano. Remova este parâmetro se desejar ver os logs em tempo real diretamente no terminal.

C.4.2 Serviços e Portas

Após a inicialização bem-sucedida, os seguintes serviços estarão disponíveis:

Tabela 5 – Serviços e portas do sistema

Serviço	**Porta**	**Descrição**
Besu RPC HTTP	8545	Endpoint JSON-RPC para interação com blockchain
Besu RPC WebSocket	8546	Endpoint WebSocket para eventos em tempo real
Besu P2P	30303	Porta de comunicação P2P entre nós
Besu Metrics	9545	Métricas Prometheus para monitoramento
Indexer Database	5433	PostgreSQL do indexador blockchain
Server Database	5434	PostgreSQL da API backend
Test Database	5435	PostgreSQL para testes automatizados
API Server	3001	API REST do backend
Dashboard	5173	Interface web administrativa
Indexer API	42069	API GraphQL do indexador Ponder

C.4.3 Aguardando Inicialização Completa

O processo de inicialização completo pode levar alguns minutos, especialmente na primeira execução, pois envolve:

1. Download de imagens Docker (se não estiverem em cache)
2. Inicialização do nó Hyperledger Besu
3. Compilação dos contratos inteligentes
4. Deploy dos contratos DREX na blockchain
5. Deploy do contrato StudentAssistanceVault
6. Inicialização dos bancos de dados PostgreSQL
7. Execução de migrações do banco de dados

8. Inicialização do indexador blockchain
9. Inicialização da API backend
10. Inicialização do dashboard web

Para acompanhar o progresso, monitore os logs:

```
# Logs do n   Besu
docker-compose logs -f besu-node

# Logs do deployer de contratos
docker-compose logs -f contract-deployer

# Logs da API server
docker-compose logs -f api-server

# Logs do indexador
docker-compose logs -f indexer
```

Listing C.5 – Monitorando logs específicos

C.4.4 Verificação de Funcionamento

Para verificar se o sistema está funcionando corretamente, execute os seguintes comandos:

```
# 1. Verificar conectividade com blockchain
curl -X POST http://localhost:8545 \
  -H "Content-Type: application/json" \
  -d '{"jsonrpc":"2.0","method":"eth_blockNumber","params":[],"id":1}
   '

# Sa da esperada: {"jsonrpc":"2.0","id":1,"result":"0x..."}

# 2. Verificar API backend
curl http://localhost:3001/api/health

# Sa da esperada: {"status":"ok","timestamp":"..."}

# 3. Verificar indexador Ponder
curl http://localhost:42069/health

# Sa da esperada: status 200 OK

```

```
# 4. Listar containers em execu  o
docker-compose ps

# Todos os servi os devem estar "Up" ou "healthy"
```

Listing C.6 – Verificando funcionamento do sistema

C.4.5 Acessando as Interfaces

C.4.5.1 Dashboard Administrativo

O dashboard web pode ser acessado através do navegador:

- **URL:** http://localhost:5173
- **Credenciais padrão:** Configuradas no seed do banco de dados

C.4.5.2 Explorando a Blockchain

Para consultar dados diretamente da blockchain, utilize ferramentas como:

- **Web3.js/Ethers.js:** Bibliotecas JavaScript para interação
- **Postman/cURL:** Para chamadas JSON-RPC diretas
- **Hardhat Console:** Console interativo (requer instalação local)

C.5 Configuração do Banco de Dados

C.5.1 Migrações Automáticas

As migrações do banco de dados são executadas automaticamente durante a inicialização dos containers. No entanto, se for necessário executar manualmente:

```
# Entrar no container da API
docker-compose exec api-server sh

# Executar migra  es do Prisma
cd packages/student-assistance-server
npx prisma migrate deploy

# Sair do container
exit
```

Listing C.7 – Executando migrações manualmente

C.5.2 Seed do Banco de Dados

O seed popula o banco com dados iniciais necessários para operação:

```
# Entrar no container da API
docker-compose exec api-server sh

# Executar seed
cd packages/student-assistance-server
npx prisma db seed

# Ou usar o script npm
npm run seed

# Sair do container
exit
```

Listing C.8 – Executando seed do banco de dados

O seed cria:

- Tipos de despesa padrão (Alimentação, Moradia, Material, Transporte)
- Usuários administrativos de teste
- Endereços conhecidos (contratos deployados)
- Dados de exemplo (opcional)

C.5.3 Reset Completo do Banco de Dados

Se for necessário reiniciar o banco de dados do zero:

```
# Parar todos os containers
docker-compose down

# Remover volumes (CUIDADO: apaga todos os dados!)
docker-compose down -v

# Subir novamente (ir  recriar tudo)
docker-compose up -d
```

Listing C.9 – Reset completo do banco

Atenção: O comando `docker-compose down -v` remove permanentemente todos os dados armazenados nos volumes Docker, incluindo dados da blockchain e bancos de dados. Use com cautela!

C.6 Comandos Úteis

C.6.1 Gerenciamento de Containers

```
# Parar todos os servi os
docker-compose stop

# Reiniciar servi o espec fico
docker-compose restart besu-node

# Ver logs de servi o espec fico
docker-compose logs -f api-server

# Ver estat sticas de recursos
docker stats

# Executar comando em container
docker-compose exec besu-node sh

# Listar containers
docker-compose ps

# Remover containers parados
docker-compose rm
```

Listing C.10 – Comandos de gerenciamento

C.6.2 Limpeza e Manutenção

```
# Parar e remover containers, redes
docker-compose down

# Remover tamb m volumes
docker-compose down -v

# Limpar recursos Docker n o utilizados
docker system prune

# Limpar incluindo volumes n o utilizados
docker system prune -a --volumes
```

Listing C.11 – Comandos de limpeza

C.6.3 Desenvolvimento Local

Para desenvolvimento local sem Docker:

```
# Instalar dependências globais
npm install -g pnpm

# Instalar dependências do projeto
pnpm install

# Subir apenas serviços de infraestrutura
docker-compose up -d besu-node indexer-db server-db

# Compilar contratos DREX
cd packages/drex-piloto
pnpm run compile

# Deploy contratos DREX
pnpm run deploy:local

# Compilar contrato StudentAssistanceVault
cd ../student-assistance-vault
pnpm run compile

# Deploy StudentAssistanceVault
pnpm run deploy:local

# Executar migrações do banco
cd ../student-assistance-server
pnpm run migrate

# Executar seed
pnpm run seed

# Iniciar API em modo desenvolvimento
pnpm run dev

# Em outro terminal, iniciar indexador
cd ../student-assistance-indexer
pnpm run dev

# Em outro terminal, iniciar dashboard
cd ../student-assistance-dashboard
```

```
pnpm run dev
```

Listing C.12 – Configuração para desenvolvimento

C.7 Testes

C.7.1 Executando Suite de Testes

O sistema inclui testes unitários e de integração:

```
# Testes dos contratos inteligentes
cd packages/student-assistance-vault
pnpm run test

# Testes com cobertura
pnpm run coverage

# Testes da API backend
cd ../student-assistance-server
pnpm run test

# Testes de integra  o
pnpm run test:integration
```

Listing C.13 – Executando testes

C.7.2 Testes de Performance

Testes de performance para medição de custos de gas:

```
cd packages/student-assistance-vault
pnpm run test:performance
```

Listing C.14 – Testes de performance

C.8 Troubleshooting

C.8.1 Problemas Comuns

C.8.1.1 Porta já em uso

Se alguma porta estiver em uso, identifique e encerre o processo:

```
# Linux/macOS - identificar processo usando porta 8545
lsof -i :8545

```

```
# Encerrar processo (substitua PID pelo n mero mostrado)
kill -9 PID

# Windows - identificar processo
netstat -ano | findstr :8545

# Encerrar processo
taskkill /PID [PID] /F
```

Listing C.15 – Liberando portas

C.8.1.2 Containers não inicializam

```
# Ver logs detalhados
docker-compose logs

# Verificar status de sa de
docker-compose ps

# Reiniciar container problem tico
docker-compose restart [nome-do-servico]

# Reconstruir imagens
docker-compose build --no-cache
docker-compose up -d
```

Listing C.16 – Diagnóstico de containers

C.8.1.3 Erro de permissão no Docker

No Linux, se houver erros de permissão:

```
# Adicionar usu rio ao grupo docker
sudo usermod -aG docker $USER

# Aplicar mudan as (ou fazer logout/login)
newgrp docker
```

C.8.1.4 Espaço em disco insuficiente

```
# Ver uso de espa o pelo Docker
docker system df

```

```
# Limpar recursos n o utilizados
docker system prune -a --volumes
```

Listing C.17 – Liberando espaço

C.8.2 Logs e Debugging

C.8.2.1 Logs Detalhados

```
# Logs de todos os servi os
docker-compose logs

# Logs com timestamp
docker-compose logs -t

#  ltimas  100 linhas
docker-compose logs --tail=100

# Seguir logs em tempo real
docker-compose logs -f

# Logs de servi o espec fico
docker-compose logs -f besu-node
```

Listing C.18 – Acessando logs

C.8.2.2 Inspeção de Containers

```
# Informa es detalhadas do container
docker inspect besu-node

# Processos em execu o no container
docker-compose top

# Entrar no container interativamente
docker-compose exec besu-node sh

# Ver arquivos do container
docker-compose exec besu-node ls -la
```

Listing C.19 – Inspecionando containers

C.9 Atualização e Manutenção

C.9.1 Atualizando o Sistema

Para atualizar para a versão mais recente:

```
# Parar servi os
docker-compose down

# Atualizar c digo-fonte
git pull origin main

# Reconstruir imagens
docker-compose build --no-cache

# Iniciar com nova vers o
docker-compose up -d
```

Listing C.20 – Atualizando o sistema

C.9.2 Backup de Dados

Para fazer backup dos dados:

```
# Backup do banco de dados do servidor
docker-compose exec server-db pg_dump -U server server > \
  backup_server_$(date +%Y%m%d).sql

# Backup do banco do indexador
docker-compose exec indexer-db pg_dump -U indexer indexer > \
  backup_indexer_$(date +%Y%m%d).sql

# Backup dos dados da blockchain
docker-compose exec besu-node tar czf - /data > \
  backup_besu_$(date +%Y%m%d).tar.gz
```

Listing C.21 – Backup de dados

C.9.3 Restauração de Backup

Para restaurar um backup:

```
# Restaurar banco do servidor
cat backup_server_YYYYMMDD.sql | \
  docker-compose exec -T server-db psql -U server server

```

```
# Restaurar banco do indexador
cat backup_indexer_YYYYMMDD.sql | \
  docker-compose exec -T indexer-db psql -U indexer indexer
```

Listing C.22 – Restaurando backup

C.10 Considerações de Segurança

C.10.1 Ambiente de Desenvolvimento vs Produção

As configurações fornecidas são adequadas para ambientes de desenvolvimento e demonstração. Para uso em produção, considere:

- **Credenciais:** Altere todas as senhas padrão
- **Firewall:** Configure regras de firewall apropriadas
- **SSL/TLS:** Utilize certificados SSL para comunicação
- **Backup:** Implemente estratégia de backup regular
- **Monitoramento:** Configure ferramentas de monitoramento
- **Logs:** Centralize e monitore logs de segurança
- **Atualizações:** Mantenha sistema e dependências atualizados

C.10.2 Alteração de Senhas Padrão

```
# Editar docker-compose.yml
vim docker-compose.yml

# Alterar variveis de ambiente:
# POSTGRES_PASSWORD
# JWT_SECRET
# ADMIN_PASSWORD
```

Listing C.23 – Alterando senhas padrão

C.11 Suporte e Documentação Adicional

C.11.1 Recursos Disponíveis

- **Repositório:** https://codigos.ufsc.br/luiz.gustavo.hatem/student-assistance-drex
- **README Principal:** Documentação no arquivo README.md do repositório

- **Issues:** Sistema de issues do GitLab para reportar problemas
- **Documentação Individual:** Cada pacote possui seu próprio README

C.11.2 Documentação dos Componentes

Cada componente do sistema possui documentação específica:

- **drex-piloto/README.md:** Contratos oficiais DREX
- **student-assistance-vault/README.md:** Contrato principal
- **student-assistance-indexer/README.md:** Sistema de indexação
- **student-assistance-server/README.md:** API Backend
- **student-assistance-dashboard/README.md:** Interface web

C.12 Conclusão

Este apêndice forneceu instruções detalhadas para instalação, configuração e execução do sistema de assistência estudantil desenvolvido. O uso de Docker Compose simplifica significativamente o processo de deployment, permitindo que todo o sistema seja executado com poucos comandos.

Para informações técnicas mais detalhadas sobre a arquitetura e implementação, consulte os capítulos correspondentes desta monografia e a documentação disponível no repositório do projeto.

ANEXO A – Decreto PNAES

A.1 Decreto nº 7.234, de 19 de julho de 2010

DECRETO Nº 7.234, DE 19 DE JULHO DE 2010

Dispõe sobre o Programa Nacional de Assistência Estudantil - PNAES.

O PRESIDENTE DA REPÚBLICA, no uso da atribuição que lhe confere o art. 84, inciso VI, alínea "a", da Constituição:

DECRETA: