UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS REITOR JOÃO DAVID FERREIRA LIMA
PROGRAMA DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Arthur Philippi Bianco

TOKEN CRIPTOGRÁFICO PARA PAGAMENTO DE PROGRAMAS DE PERMANÊNCIA ESTUDANTIL

Florianópolis, Santa Catarina – Brasil
2022

Arthur Philippi Bianco

TOKEN CRIPTOGRÁFICO PARA PAGAMENTO DE PROGRAMAS DE PERMANÊNCIA ESTUDANTIL

Trabalho de Conclusão de Curso submetido ao Programa de Graduação em Ciências da Computação da Universidade Federal de Santa Catarina para a obtenção do Grau de Bacharel em Ciências da Computação.
Orientador(a): Prof. Jean Everson Martina, Dr.

Florianópolis, Santa Catarina – Brasil
2022

Catalogação na fonte pela Biblioteca Universitária da Universidade Federal de Santa Catarina.
Arquivo compilado às 22:30h do dia 24 de março de 2022.

Arthur Philippi Bianco
Token criptográfico para pagamento de programas de permanência estudantil / Arthur Philippi Bianco; Orientador(a), Prof. Jean Everson Martina, Dr. - Florianópolis, Santa Catarina - Brasil, 30 de Fevereiro de 2022.
63 p.

Trabalho de Conclusão de Curso - Universidade Federal de Santa Catarina, INE - Departamento de Informática e Estatística, CTC - Centro Tecnológico, Programa de Graduação em Ciências da Computação.

Inclui referências

1. Token, 2. Blockchain, 3. Cripto Moeda, I. Prof. Jean Everson Martina, Dr. II. Programa de Graduação em Ciências da Computação III. Token criptográfico para pagamento de programas de permanência estudantil

CDU 02:141:005.7

Arthur Philippi Bianco

TOKEN CRIPTOGRÁFICO PARA PAGAMENTO DE PROGRAMAS DE PERMANÊNCIA ESTUDANTIL

Este(a) Trabalho de Conclusão de Curso foi julgado adequado(a) para obtenção do Título de Bacharel em Ciências da Computação, e foi aprovado em sua forma final pelo Programa de Graduação em Ciências da Computação do INE – Departamento de Informática e Estatística, CTC – Centro Tecnológico da Universidade Federal de Santa Catarina.

Florianópolis, Santa Catarina – Brasil, 30 de Fevereiro de 2022.

Prof. Jean Everson Martina, Dr.
Coordenador(a) do Programa de Graduação em Ciências da Computação

Banca Examinadora:

Prof. Jean Everson Martina, Dr.
Orientador(a)
Universidade Federal de Santa Catarina – UFSC

Giovani Pieri, Dr.
Universidade Federal de Santa Catarina – UFSC

Thaís Bardini Idalino, Dr.
Universidade Federal de Santa Catarina – UFSC

Dedico este trabalho primeiramente à Deus, à minha família e aos meus colegas, sem os quais não seria capaz de realizá-lo.

"He who would do great things should not attempt them all alone."
Seneca Proverb

"I would rather wear out than rust out."
George Whitefield, British Methodist preacher

RESUMO

Métodos utilizados por instituições federais públicas de ensino superior em ações de assistência estudantil estão desatualizados e faltam com transparência. Uma solução ao problema é a utilização de tecnologia de blockchain, que possui o benefício da rastreabilidade e transparência nas transações. Esta tecnologia a torna ideal para uso por instituições públicas de assistência à permanência estudantil em programas como o Programa Nacional de Assistência Estudantil, que tem por objetivo garantir a permanência e conclusão dos estudantes no ensino superior. Este projeto desenvolve um protótipo de *token* criptográfico para ser utilizando como moeda por estudantes de instituições de ensino superior, de modo que alunos e beneficiários de programas de assistência estudantil possam utilizá-lo para comprar o passe do restaurante universitário, pagar aluguel e comprar produtos em empresas associadas.

Palavras-chaves: Token. Blockchain. Cripto Moeda.

ABSTRACT

Methods used by public federal institutions of higher education in student assistance actions are outdated and lack transparency. A solution to this problem is to use blockchain technology, which has the benefit of traceability and transparency in transactions. This technology makes it ideal for use by public institutions of student assistance in programs such as the National Student Assistance Program, which aims to guarantee the permanence and completion of students in higher education. This project develops a cryptographic token prototype to be used as currency by students of higher education institutions, so that students and beneficiaries of student assistance programs can use it to buy the university restaurant pass, pay rent and buy products from associated companies.

Keywords: Token. Blockchain. Cryptocurrency.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

PNAES	Programa Nacional de Assistência Estudantil
RU	Restaurante Universitário
p2p	Peer to Peer
UTXO	Unspent Transaction Output
MEC	Ministério da Educação
CLI	Command-line Interface
CIA	Confidentiality, Integrity, and Availability
JWT	JSON Web Token
PoW	Proof of Work

SUMÁRIO

1 INTRODUÇÃO

Com a promulgação do decreto 7.234, que institui a criação do Programa Nacional de Assistência Estudantil (PNAES), estudantes de graduação em universidades públicas brasileiras passaram a receber incentivos à sua permanência e conclusão do ensino superior (BRASIL, 2010). Beneficiários deste programa podem receber assistência na moradia estudantil, alimentação, transporte, entre outros. Usualmente, cada universidade recebe uma fração da verba do programa, e a universidade fica responsável por distribuir estes recursos para os contemplados. Os beneficiários destas políticas recebem os benefícios, muitas vezes, diretamente nas próprias contas pessoais. Os alunos por sua vez, podem utilizar dos recursos da forma como quiserem.

Estes recursos podem, eventualmente, ser utilizados indevidamente e é difícil verificar sua correta utilização. Além disso, pode ocorrer também desvio destes recursos por parte de autoridades públicas, antes mesmo dos valores chegarem nas mãos dos alunos. Por exemplo, na Universidade Federal de Viçosa, onde um grupo de funcionários foram acusados de desviar alimentos dos estoques do refeitório custeados pelo PNAES (MACEDO, 2016).

O método comumente utilizado para transferir o benefício no nome do aluno é eletrônico. Este método é seguro e eficiente. Entretanto, quando efetivamente ocorre um desvio, o processo de investigação é lento e depende de denúncia ou averiguação por órgãos competentes. Também é preciso levar em conta que processos investigativos são demorados por estarem sujeitos aos labirintos inerentes aos trâmites legais. Com efeito, em março de 2022, ocorre a operação com o sugestivo nome de "Quadro Negro". A atividade policial tem como objetivo desarticular organização criminosa envolvida em desvio de verbas destinadas à educação pública na Universidade Federal Fluminense (UFF) (MIRELLE PINHEIRO, 2022). A investigação identificou suposto desvio de R$ 3,7 milhões em contratações emergenciais e ordinárias entre a UFF e uma grande empresa de terceirização de mão de obra, no período de 2011 a 2015.

Diante disto, propõe-se uma solução utilizando tecnologia *blockchain* para lidar com os problemas mencionados, dificultando ações ilícitas. Esta tecnologia inventada pelo lendário Satoshi Nakamoto (NAKAMOTO, 2009), possui propriedades intrínsecas de transparência e imutabilidade dos dados. Com ela, pode-se facilitar a constatação de utilização inadequada de recursos, uma vez que qualquer transação terá visibilidade pública e sua autenticidade será inegável. Neste trabalho, apresenta-se um esquema que utiliza contratos inteligentes e a implementação *Open-Source* de *blockchain Hyperledger Fabric* para implementar uma solução ao problema descrito, resultando na criação de um *token* criptográfico.

1.1 MOTIVAÇÃO

O índice de corrupção percebida é um dos indicadores de corrupção mais importantes do mundo (INTERNATIONAL, 2020). Segundo ele, para o ano de 2020, o Brasil está na posição 94, junto com Sérvia, Sri Lanka, Suriname, Tanzânia, Etiópia, Peru e Cazaquistão (INTERNATIONAL, 2020). Estima-se que o governo tenha investido cerca de R$ 849 milhões no

PNAES apenas no ano de 2021, e contemplado 336.770 estudantes em situação de vulnerabilidade socioeconômica entre os exercícios de 2016 e 2020 (BRASIL, 2021). Diante do risco que correm os investimentos estatais em educação de nível superior, são necessárias soluções novas, que por meio da tecnologia, ajudem a combater um dos maiores problemas que o Brasil enfrenta.

1.2 JUSTIFICATIVA

Atualmente vivemos em um mundo onde a utilização de criptomoedas já é bastante disseminada, com estimativas de usuários de criptomoedas ultrapassando 300 milhões (TRIPLEA, 2021). E mesmo com milhões de usuários, a fração da população mundial não ultrapassa 4%. Instituições, indivíduos e a sociedade como um todo podem beneficiar-se muito com a adoção de tecnologia de *blockchain*. Este tema foi escolhido com o intuito de contribuir em uma área que influencia a vida de muitas pessoas, e acelerar a adoção de uma inovação tecnológica que será utilizada para facilitar o acesso à informação por todos.

1.3 OBJETIVOS

1.3.1 OBJETIVO GERAL

Este trabalho procura criar um protótipo de *token* criptográfico para uma rede *blockchain* para servir de intermédio de pagamento entre instituições e seus beneficiários de modo a dificultar transações ilícitas e desvios de recursos, mas ainda servir ao público eficientemente e naturalmente.

1.3.2 OBJETIVOS ESPECIFICOS

i Desenvolver um *token* criptográfico que permita a sua troca entre usuários cadastrados na rede para pagamento de serviços;

ii Implementar um sistema que permita a emissão e distribuição dos *tokens* por parte da autoridade responsável;

iii Implementar uma interface pela qual os usuários possam interagir com a rede e administrar seus *tokens*;

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção apresenta a fundamentação teórica da área de estudo deste trabalho, necessária para compreender a proposta e como ela resolve o problema. Serão contemplados tópicos relevantes como segurança da informação, criptografia, *blockchain*, tecnologias relacionadas, e criptomoedas.

2.1 SEGURANÇA DA INFORMAÇÃO

Segurança da informação é definido como a proteção fornecida a um sistema com vistas de preservar a integridade, disponibilidade e confidencialidade das informações pertinentes ao sistema (STALLINGS, 2013). Esta definição toca em 3 grandes tópicos dentro da área de segurança da informação, conhecidos como a grande tríade CIA, que em português, são a confidencialidade, integridade e disponibilidade. É interessante detalhar esta tríade, mas também, vale a pena introduzir outros 2 que são bem estabelecidos na área: Autenticidade e Não Repúdio (STALLINGS, 2013). A seguir os 5 grandes conceitos por trás da segurança da informação:

1. **Confidencialidade**: Diz respeito a limitar o acesso à informação a apenas as partes envolvidas que tenham autorização e ninguém mais.
2. **Integridade**: Garantir que mudanças em dados e programas ocorram de maneira consentida e autorizada, e garantir que um sistema funcione da maneira para o qual foi programado e sem interferências de terceiros desautorizados.
3. **Disponibilidade**: Garante que o sistema esteja funcional e que usuários legítimos não sejam negados de utilizarem o serviço.
4. **Autenticidade**: Trata sobre a verificação da legitimidade de uma mensagem. Isto é, afirmar com certeza de que os agentes no sistema são quem eles realmente dizem ser.
5. **Não Repúdio**: Diz respeito à irrefutabilidade da autoria de uma mensagem no contexto do sistema.

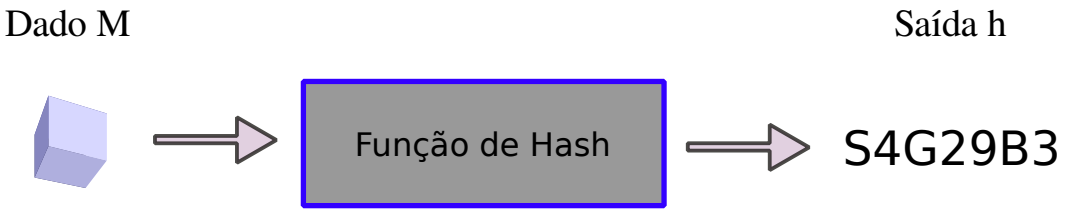
2.2 FUNÇÃO DE HASH

Função de hash H é uma função que aceita uma entrada M de qualquer tamanho e produz uma saída h de tamanho fixo. Em geral, uma função deste tipo é dita boa caso conjuntos grandes de entradas produzam saídas aparentemente aleatórias (STALLINGS, 2013). A função é chamada de função de hash criptográfica quando é computacionalmente inviável encontrar:

1. a entrada M a partir de uma saída conhecida h Ou
2. duas entradas M, M' que possuam a mesma saída $h = h'$

Em razão destas características, funções deste tipo são bastante utilizadas para determinar se algum dado mudou ou não. A imagem 1 representa o funcionamento de uma função de *Hash*.

Figura 1 – Representação de uma função de hash.



Fonte: Desenvolvimento próprio

2.3 REDES PEER TO PEER (P2P)

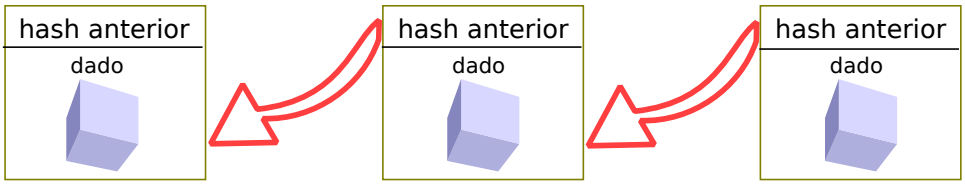
Antes de chegar ao conceito de blockchain propriamente, é necessário entender o que é Peer-to-Peer, que significa ponto a ponto. Trata-se de uma conexão estabelecida diretamente entre participantes, sem interlocução nem desvios. Em geral, redes P2P permitem a uma comunidade de usuários reunirem recursos, sejam eles arquivos, banda de rede, poder de processamento, armazenamento, e disponibilizarem acesso a uma coleção de dados, conferências de video maiores, buscas ou computações mais complexas, entre si do que um usuário individualmente seria capaz (PETERSON; DAVIE, 2011).

Redes deste tipo são especialmente relevantes no contexto de *blockchain*, posto que o principal benefício que *blockchain* proporciona é a exclusão da necessidade de confiança entre os participantes. Ou seja, é interessante que não tenha uma autoridade centralizadora para servir pedidos, mas que vários participantes se ajudem, uma vez que a sobrevivência deste sistema hipotético dependeria de confiar neste servidor.

2.4 BLOCKCHAIN

Blockchain, como o nome sugere, significa uma lista encadeada de blocos. É útil entender *blockchain* como sendo um banco de dados distribuído, ou seja, uma coleção de informações armazenada digitalmente. Ao invés, porém, de uma estrutura centralizada que contém todos os dados e serve pedidos, trata-se de uma lista encadeada e, geralmente, descentralizada com os pedaços de informação, geralmente transações, armazenados e lacrados em blocos. Estes blocos são encadeados utilizando ponteiros *hash* para o bloco anterior da lista. Este tipo ponteiro nada mais é que um par composto por um ponteiro ao bloco, e o *hash* deste bloco. Desta maneira, além do bloco conter um ponteiro para o elemento anterior da lista, este ponteiro também contém o *hash* daquele bloco (NARAYANAN *et al.*, 2016), conforme a imagem 2. Definindo desta maneira, é fácil de perceber que uma *blockchain* é resistente a alterações em qualquer parte, uma vez que ao alterar os dados de qualquer bloco, todos os *hashes* daquele ponto em diante ficarão inconsistentes, começando com o *hash* do próprio bloco alterado.

Figura 2 – Representação de uma *hash list*.



Fonte: Desenvolvimento próprio

Em essência, *Blockchain* é composta por de 6 grandes propriedades (LIN; LIAO, 2017), são eles:

1. **Descentralização**: Talvez a funcionalidade central de qualquer *blockchain* é não ser controlado por uma autoridade central. Uma rede, tipicamente, serve a todos os participantes e realiza as modificações na *blockchain* baseadas em algum algoritmo de consenso.

2. **Transparência**: É comum que as transações que ocorrem dentro de uma rede *blockchain* sejam públicas e, por este motivo, que existe confiança entre as partes. Ou seja, os usuários podendo verificar as transações dentro da rede, percebem que não houve geração espontânea da moeda na rede, ou que alguém está ferindo o protocolo, gerando confiança.

3. **Código aberto**: Possuir um repositório público com o código. Outro elemento que toca no quesito confiança, permite diferenciar entre *forks* dentro da mesma moeda.

4. **Autonomia**: Autonomia diz respeito à capacidade de qualquer nodo agir para atualizar a blockchain sem interferência de outros nodos.

5. **Imutabilidade**: Dados escritos na *blockchain* são permanentes. Isso só pode ser mudado por meio de ataques específicos. Por exemplo, na rede do Bitcoin onde, caso de 51% dos nodos colaborarem, é possível alterar o histórico da *chain*.

6. **Anonimidade**: Não existe uma associação direta entre o usuário com seus recursos na *blockchain* e o nome dele, apenas o endereço dele.

2.4.1 Bloco

Os blocos desta lista encadeada carregam as informações que são importantes para a rede. No caso do *Bitcoin*, cada bloco possui uma lista de transações que, em conjunto com todos os outros blocos, definem o estado atual dos fundos de cada participante. A título de exemplo, apresenta-se aqui a estrutura de um bloco da rede *Bitcoin*: Cada bloco possui um *header* de 80 bytes, com o corpo de transações contendo 500 transações ao máximo, cada uma com 250 bytes (ANTONOPOULOS, 2015). O *bloco* é composto de:

1. **Block Size**: Contém o tamanho do bloco, em bytes.

2. **Block Header**: O *header* do bloco.
3. **Transaction Counter**: A quantidade de tansações neste bloco.
4. **Transactions**: Contém as transações.

O *header* do bloco contém os seguintes dados:

1. **Version**: Contém a versão para fins de acompanhamento de atualizações.
2. **Previous Block Hash**: Contém o *hash* do bloco anterior.
3. **Merkle Root**: Contém a hash da raiz da *Merkle tree* das transações deste bloco.
4. **Timestamp**: O instante aproximado de criação deste bloco.
5. **Difficulty Target**: A dificuldade alvo do algoritmo de consenso para este bloco.
6. **Nonce**: Um contado utilizado no algoritmo de consenso.

Alterações à rede ocorrem por meio destes blocos: participantes da rede criam um bloco contendo informação relevante para a rede e adicionam este bloco à rede. Caso este bloco respeite as regras da rede, ele é adicionado e este novo *bloco* é transmitido para os outros participantes. Visto que estas alterações feitas na blockchain são replicadas para cada um dos nodos participantes dela, é necessário, de algum modo, decidir quais alterações serão aceitas, em outras palavras, qual dos blocos aceitar como próximo. Para lidar com este problema, utilizam-se algoritmos de consenso.

2.4.1.1 Consenso

Os algoritmos de consenso são utilizados para obter consenso sobre uma transação, e neste caso para decidir sobre o próximo bloco. Existe uma miríade de algoritmos desenvolvidos com este intuito. O mais conhecido é o algoritmo de prova de trabalho (Proof of Work (PoW)) do *Bitcoin* (NAKAMOTO, 2009). Em essência, a rede do *Bitcoin* faz a escolha do próximo bloco ao verificar que o trabalho computacional por ele foi feito. O processo envolve buscar por um valor cujo *hash* comece com um certo número de zeros determinado pela rede naquele momento (NAKAMOTO, 2009).

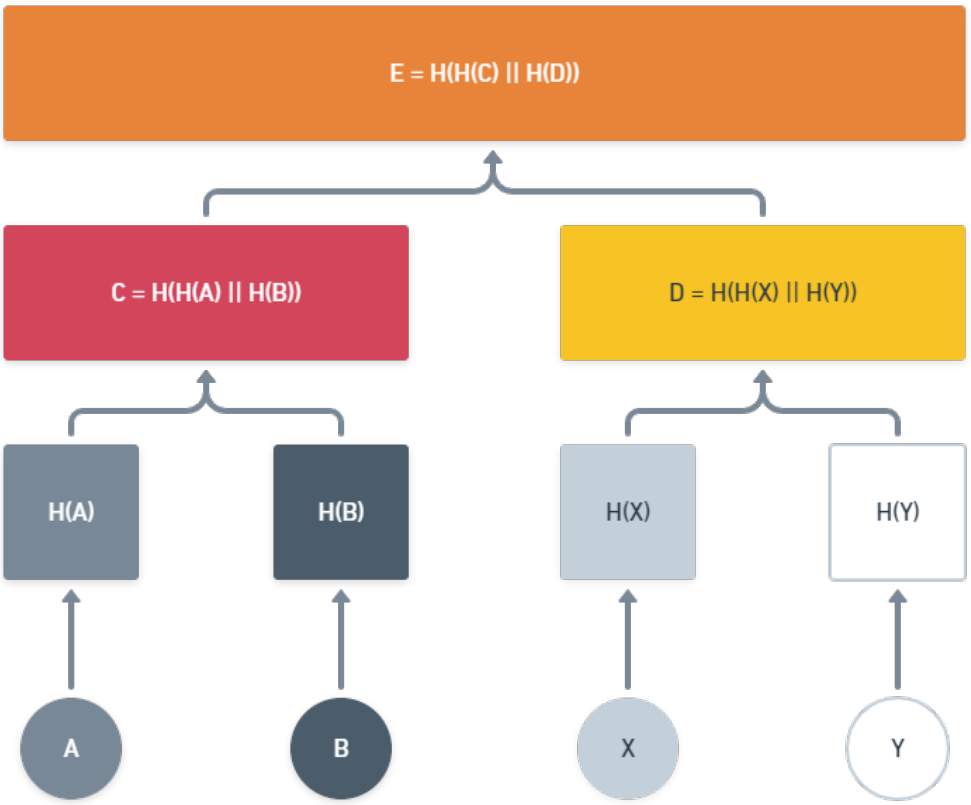
2.4.2 Merkle Tree

Um conceito que acompanha o estudo de *blockchain* é o de *Merkle Tree*. Fundamentalmente, *Merkle Tree* é uma árvore binária. Porém, similarmente à *hash list* ela é construída utilizando ponteiros *hash*. Uma das principais utilidades desta estrutura é incrementar na eficiência computacional. Uma operação que é auxiliada, por exemplo, é o cômputo da prova de pertencimento. Para verificar que um bloco faz parte de uma *hash list*, é necessário verificar todos os hashes de todos os blocos anteriores ao bloco em questão. Utilizando a *Merkle Tree*, por outro lado, é

necessário verificar apenas $log(n)$ blocos, onde n é o número de blocos (NARAYANAN *et al.*, 2016). A imagem 3 mostra um exemplo de uma *Merkle Tree*.

Além disso esta estrutura permite aos participantes de uma rede economizar espaço em disco. Por exemplo, a proposta de Nakamoto na rede *Bitcoin*. Ao invés de organizar as transações linearmente, elas devem ser organizadas de acordo com uma *Merkle Tree*. Na imagem 3, para verificar que a transação A está presente no bloco, é necessário apenas as informações de A, $H(B)$ e $H(D)$ para recalcular a raiz. Para demonstrar que um nodo folha faz parte da *Merkle Tree* é necessário processar apenas uma quantidade proporcional ao logaritmo do número de nodos da árvore (NAKAMOTO, 2009). Portanto, economiza-se espaço desta forma, pois as transações que não são necessárias podem ser descartadas, sem implicar prejuízos à árvore.

Figura 3 – Representação de uma *Merkle Tree* onde H é a função de *hash*, e A, B, X, Y são transações ou blocos



Fonte: Desenvolvimento próprio

2.4.3 Permissioned vs Permissionless blockchain

Blockchains como a do *Bitcoin* são *permissionless* (não permissivo). Isto significa que não existe uma entidade que concede permissões para novos usuários entrarem na rede e fazerem modificações. A rede é aberta ao público e qualquer um pode interagir com ela. Isto é favorável ao *Bitcoin* pois cria uma descentralização maior e permite com que mais usuários utilizem a

rede. Ademais, participantes da rede são livres de censura, por não depender de uma autoridade centralizada decidir sobre a validade ou não de transações.

Por outro lado, redes *Permissioned* (permissivas), ou *blockchains* privadas, são fechadas a um grupo de agentes. Geralmente, estes agentes são previamente determinados, e por vezes, fazem parte de algumas organizações que desejam realizar negócios entre si, mas necessitam da privacidade e segurança provida por uma *blockchain*. Participantes de redes deste tipo abrem mão de anonimidade, necessitando autenticar-se. No caso da proposta atual, este seria o modelo ideal, pois nem todos devem ter acesso ao *token*, visto que nem todas as pessoas devem ter acesso a um restaurante universitário, por exemplo.

Redes privadas também contam com um algoritmo de consenso. Entretanto, estes algoritmos não utilizam algoritmos de mineração que consomem recursos computacionais, justamente porque há um nível maior de confiança entre os participantes. Usuários maliciosos podem ser facilmente identificados e excluídos da rede, visto que, em caso de uso malicioso dentro da rede, sabe-se exatamente qual usuário fez a ação, e inclusive os dados pessoais dele, em função da transparência e necessidade de autenticação provida pela blockchain privada. Em geral estes algoritmos de consenso são mais rápidos e mais computacionalmente eficientes (YAGA *et al.*, 2019). Algumas implementações de *blockchain*, como o *hyperledger fabric*, utilizam um serviço de ordenação que garante que todas as mensagens são vistas na mesma ordem por todos os participantes, definindo para a rede, qual a próxima ação a ser minerada em um bloco.

2.5 CRIPTOMOEDAS

Criptomoedas e moedas emitidas pelo governo (conhecidas como moedas fiduciárias), são dinheiro, com seu valor determinado pela lei da oferta e demanda. Sendo assim, elas possuem características essenciais inerentes a qualquer moeda. As principais funcionalidades de uma moeda são (TEAM, 2020):

1. Fungibilidade: Diz respeito à intercambiabilidade entre instâncias iguais de uma mesma moeda. Unidades de uma mesma moeda devem ser da mesma qualidade para que sejam intercambiáveis entre si.

2. Durabilidade: As características físicas da moeda devem ser duráveis o suficiente para que ela possa ser reutilizada múltiplas vezes. Como moedas digitais não contam com um componente físico, são naturalmente duráveis.

3. Portabilidade: Deve ser capaz de subdividir a moeda em valores menores, mas de modo que estas pequenas frações sejam convenientes de transportar e ainda valham o suficiente para valer a pena transportar.

4. Reconhecibilidade: Deve ser prontamente reconhecível que uma certa instância da moeda é autêntica. Ausência desta característica aumenta custos de transação entre usuários.

5. Estabilidade: Diz respeito ao valor da moeda com relação a outros produtos ser estável. É inconveniente que uma moeda tenha valor altamente volátil para os usuários, uma vez que eles incorrerão em custos transacionais por ter de verificar o valor da moeda a cada momento.

Criptomoedas diferem das moedas fiduciárias pelo fato de que aqueles, comumente, são o principal recurso de uma blockchain. Com o crescimento da popularidade dessas moedas, muito em função da perspectiva de ganho econômico, torna-se necessário analisar as reais utilidades delas, fora a especulação econômica.

2.5.1 Bitcoin

Bitcoin foi inventado por Satoshi Nakamoto que criou uma nova maneira das pessoas realizarem transações monetárias entre si. Em verdade, é importante entender que Bitcoin não é apenas o nome da criptomoeda. É também do protocolo e da blockchain que hospeda a moeda (SWAN, 2015).

Bitcoin é revolucionário, não apenas por introduzir a tecnologia blockchain, mas pela capacidade de minar a autoridade estatal ao permitir que os usuários contornem os controles estatais sobre o dinheiro. Isto ocorre pela proposta de eliminação de confiança entre as partes em um sistema econômico. Bitcoin influenciou quase todas as criptomoedas e *tokens* que foram criadas depois, muitas das quais não possuem os mesmas propostas do *Bitcoin*.

2.5.1.1 Wallets

Carteiras de criptomoedas são baseadas no esquema de par de chaves (pública e privada), onde cada participante detém a sua chave privada, mas todos conhecem a chave pública. As carteiras, no caso do *Bitcoin*, são utilizadas para assinar transações, ou seja, enviar *Bitcoin* entre usuários. Na rede *Ethereum*, por outro lado, as carteiras são usadas para assinar uma gama de transações, podendo implantar contratos, executar uma função de contrato e enviar não somente *Ethereum*, mas também uma séries de outros *tokens* presentes na mesma rede. *Bitcoin* e *Ethereum* podem ser considerados *tokens*. Vale a pena entender por que, tendo em vista a criação de um *token* no presente trabalho.

2.5.2 Token

Tokens podem ser definidos como entidades que representam valor econômico. Originalmente, conchas e miçangas foram as primeiras formas de *token* usadas. Outros exemplos de *tokens* surgiram ao longo da história da humanidade, como por exemplo, fichas de cassino, vouchers, cartas de baralho, títulos de dívida, cartões de identificação, cartões de fidelidade e bilhetes de loteria. Em maioria, os *tokens* possuem alguma característica que dificulta ou impede a falsificação. Cédulas e moedas também podem ser considerados como exemplos de *tokens* (VOSHMGIR, 2020). Dentro da computação, *tokens* podem ser utilizados para representar o

direito de realizar um conjunto de ações, como é o caso do *Json Web Token* (JWT) que permite um mecanismo de autenticação *stateless* (sem estado) (JONES; BRADLEY; SAKIMURA, 2015).

2.5.2.1 Token criptográfico

Tokens criptográficos são aqueles que existem em uma blockchain. Estes *tokens* representam um conjunto de regras, definidas no contrato que os emite, o contrato de *tokens*. *Tokens* criptográficos são acessíveis apenas por meio da carteira que as possui (VOSHMGIR, 2020).

Os primeiros *tokens* deste tipo foram os nativos às redes blockchain que fazem parte do esquema de incentivo da rede, também chamados de *tokens* de protocolo. Na rede do Bitcoin, por exemplo, o *token* nativo chama-se Bitcoin. Nesta rede, o Bitcoin é criado e transacionado com base em um conjunto de regras cripto-econômicas que determinam as circunstâncias nas quais as transações Bitcoin são válidas e quando novos blocos são criados (VOSHMGIR, 2020).

Com o advento do *Ethereum*, por outro lado, *Tokens* criptográficos passaram a ser criados por meio de contratos que podem ser escritos pelos próprios usuários. Isto levou à criação de uma série de padrões na rede *Ethereum* dispondo sobre as interfaces de diversos tipos de *tokens* capazes de existir na mesma rede do *Ethereum*. Estas interfaces foram definidas por meio de *Ethereum request for comment* (ERC), similarmente aos *Request For Comments* (RFC) criados para padronizar a internet.

Por exemplo, o surgimento do padrão ERC-721 (ENTRIKEN *et al.*, 2018) dentro do *Ethereum* que dispõe sobre a criação de *Tokens* Não Fungíveis (NFT), permite que usuários da rede criem com facilidade *tokens* que representem quaisquer tipos de colecionáveis, obras de arte, propriedade digital, entre outros. Para isso, o ERC-721, possui propriedades especiais que permitem com que um *token* da rede seja único. Ou seja pode-se interpretar um *token* criptográfico como sendo uma interface para um objeto num contrato inteligente.

2.6 CONTRATOS INTELIGENTES

A noção de um contrato não é nova no histórico da humanidade. Desde contratos firmados entre duas potências europeias sobre a delimitação de território sul americano, até um simples contrato de aluguel de imóvel, eles são indispensáveis e hoje são pilar das relações econômicas. Nick Szabo, porém, em 1997 idealizou uma evolução natural do contrato: embutir as clausulas do contrato em software e hardware, de modo a tornar caro ou proibitivamente caro o rompimento delas (SZABO, 1997).

O protocolo do *Bitcoin*, segundo o *whitepaper*, permite utilização de *scripts* limitados nas transações. Isto permite a usuários certo nível de expressividade, permitindo algo como disponibilização dos fundos de uma transação, somente após uma certa data. A linguagem de *script* do Bitcoin é propositalmente limitada, pois um ator malicioso poderia utilizar desta característica para ocupar a rede com processamento intenso. Portanto, não existem laços ou estruturas de controle complexas, exceto o controle de fluxo condicional na linguagem. Isto garante a Turing-incompletude dela, e dificulta a utilização de bombas lógicas (ANTONOPOULOS,

2015). Em 2014, porém, O *Ethereum* foi lançado com suporte a contratos inteligentes Turing completos.

2.6.1 Ethereum

Ethereum não é simplesmente uma criptomoeda, é uma plataforma distribuída de desenvolvimento, concebida por Vitalik Buterin, cujo intuito é, entre outras coisas, permitir a desenvolvedores criarem aplicações e definirem contratos inteligentes de acordo com suas necessidades de negócio (BUTERIN *et al.*, 2014). Utilizando contratos inteligentes do *Ethereum*, é possível, por exemplo, realizar uma venda de produto a prazo, e caso ocorra perda do prazo de pagamento, fazer com que o contrato cobre os juros automaticamente.

Atualmente, o algoritmo de consenso utilizado pela rede é o *Proof of Work* (PoW). Está prevista uma alteração no algoritmo de consenso da rede *Ethereum* no futuro, com vistas de contornar os gastos energéticos envolvidos na mineração. Quando ocorrer a mudança, a rede passará a utilizar o algoritmo Proof of Stake (PoS).

3 DESENVOLVIMENTO

Neste capítulo, o objetivo é descrever a implementação e como ela resolve o problema.

3.1 BLOCKCHAIN

Para criar um *token* criptográfico em acordo com a proposta, é necessário escolher uma *blockchain* para hospedar o *token*. Como discutido anteriormente, a rede *Bitcoin* não é adequada, visto que sua linguagem de *scripting* não possui a complexidade necessária para poder expressar as regras de negócio inerentes à proposta. *Ethereum*, por outro lado, possui suporte a contratos inteligentes e é utilizado em larga escala. Porém, *Ethereum* é uma rede *Permissionless*. Como mecionado, estaria fora da proposta permitir com que qualquer usuário tivesse acesso aos *tokens*. Além disso, a rede do *Ethereum* atualmente apresenta taxa de transação muito alta, o que tornaria inviável realizar uma simples transferência entre dois usuários. Por exemplo, na imagem 4, vê-se que o preço para realizar uma transação instantânea é de $3,97, que equivale, com dólar valendo aproximadamente 5,13 reais, a R$20,40.

Diante disto, optou-se por criar uma *Blockchain*, utilizando o *framework Hyperledger **Fabric***. Hyperledger é um framework de código aberto, e foi desenvolvido pela Fundação Linux em 2015. O *Hyperledger **Fabric*** é uma plataforma de *Distributed Ledger Technology* que foi desenvolvida pela IBM para uso corporativo. Usar *Hyperledger **Fabric*** é bastante conveniente quando se quer criar uma rede com alta modularidade. Isto significa que, caso se queira restringir identidades a certas atividades, ou criar canais de comunicação ocultos entre certos participantes da rede, é conveniente de se fazer utilizando ***Fabric*** (FABRIC, 2020).

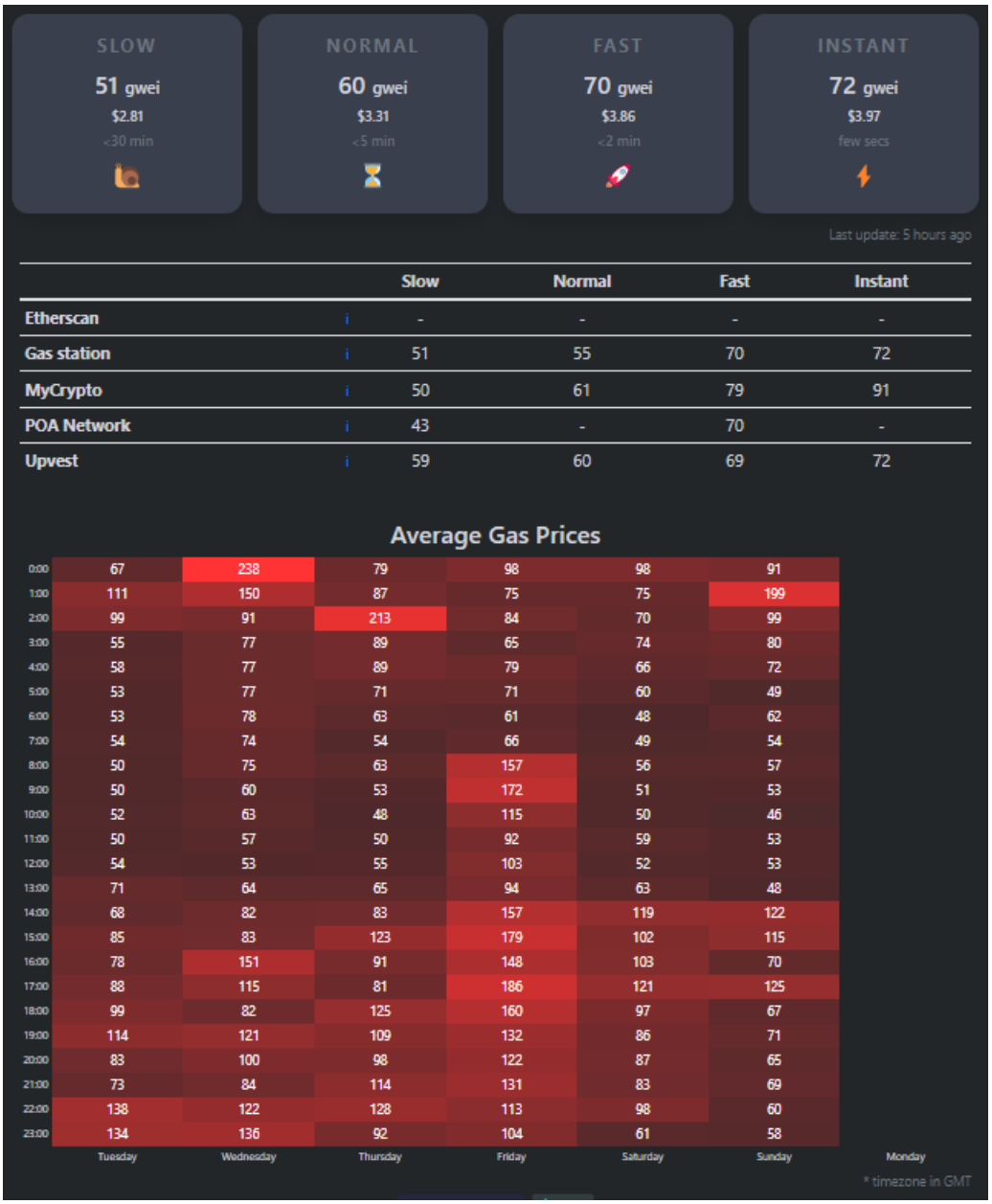
3.1.1 Minifabric

Ainda no que tange a criação da rede *blockchain*, utilizou-se uma ferramenta oficial lançada pela *Hyperledger Foundation*, chamado *Hyperledger Minifabric* que permite a especificação rápida de uma rede e seus participantes por meio de um arquivo de configuração *yaml*. *Minifabric* é apenas um utilitário que serve de interface para o próprio *Hyperledger **Fabric***. Assim decidiu-se pela simplicidade de utilização da ferramenta para criar uma rede, adicionar os participantes nela, criar canais, e outras operações relevantes no contexto da arquitetura.

3.1.2 Arquitetura

O *Hyperledger **Fabric*** permite a criação de organizações para compor uma solução e melhor representar o modelo de negócio. Uma organização neste contexto representa uma parte interessada, capaz de acessar canais e emitir identidades para participantes desta organização (HYPERLDGER, 2020). Decidiu-se por uma arquitetura de *blockchain* onde estão presentes 3 organizações: o MEC, a UFSC e os estudantes. A ideia por trás disto é permitir com que usuários da organização MEC sejam capazes de emitir *tokens*, enquanto a organização dos

Figura 4 – Captura mostrando as taxas da rede *Ethereum*

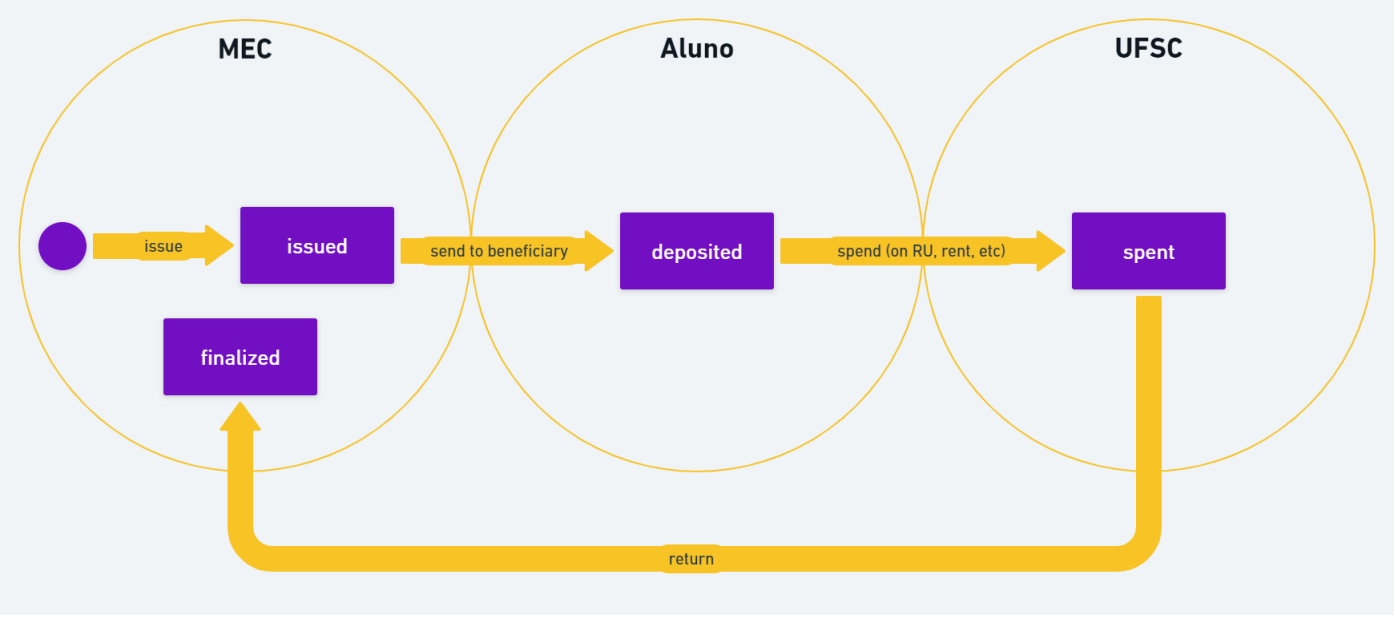


Fonte: Ethgas.watch (2022)

estudantes não, tampouco a UFSC. Além disso, deve estar presente um nodo ordenador, que é responsável por fazer o consenso na blockchain, mas que em nada interfere na implementação. Fora a emissão dos *tokens*, todas as organizações possuem acesso a todas as funcionalidades. A organização que representa a UFSC, não possui utilidade aparente, porém ela pode ser utilizada como intermediária para envio de *tokens*, uma vez que a escolha dos beneficiários é por parte dela. Da forma como foi feito, porém, a UFSC envia ao MEC a lista de beneficiários e o MEC realiza a transferência para eles. Ambas alternativas estão de acordo com a lei e resolvem o problema.

O esquema proposto para o ciclo de vida dos *tokens* pode ser visualizado na figura 5. De acordo com ele, o MEC emite um número de *tokens*, e os transfere para os beneficiários cadastrados. Estes, por sua vez, podem utilizar o *token* em serviços aprovados pela UFSC, tais como RU, pagamento de aluguel, e outros. Ao final de cada mês, a UFSC concentrará um número de *tokens* os quais serão enviados de volta para o MEC, fechando o ciclo. Uma vez finalizado o *token*, o MEC envia o dinheiro na mesma quantia que o total em *tokens* recebido pela UFSC naquele mês.

Figura 5 – Ciclo de vide de um *token*, como definido no contrato.



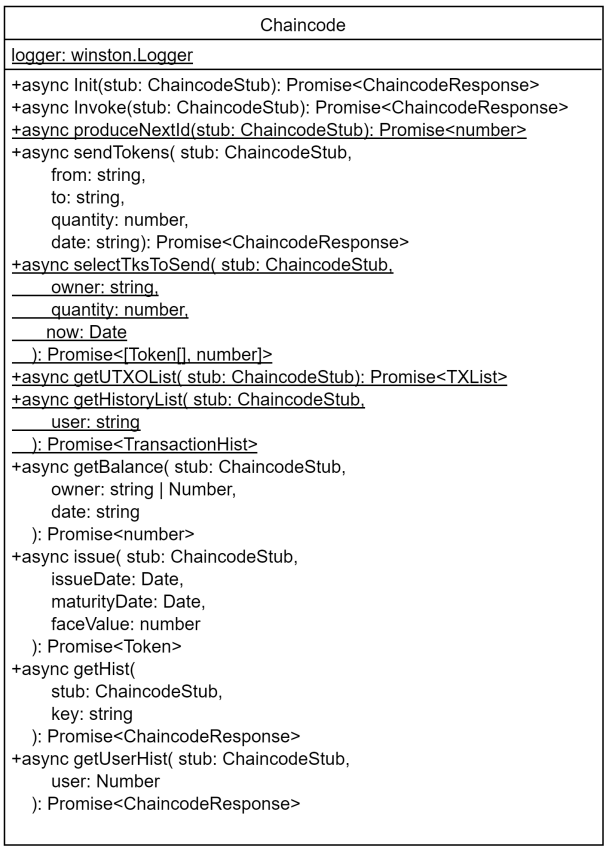
Fonte: Desenvolvimento próprio

3.2 O CONTRATO INTELIGENTE

Os contratos foram escritos em *Typescript*, e depois transpilados [1] para *Javascript*, linguagem para a qual existe uma interface com o *Hyperledger* ***Fabric***. A solução foi escrita em

[1] Transpilar significa transformar código fonte de uma linguagem em um código fonte equivalente em outra linguagem de programação.

Figura 6 – Diagrama de classes do contrato mycc



Fonte: Desenvolvimento próprio

4 classes que compõem o mesmo contrato batizado de *mycc*, cujo diagrama da classe está apresentado na figura 6. A estrutura de dados central para o funcionamento chama-se *Token*. Esta estrutura, que aparece no diagrama da imagem 7, representa alguma quantia de *tokens*. Os atributos dela são:

- **currentState**: Diz respeito ao estado atual do *Token*, valores possíveis são *ACTIVE* (ativo) e FINALIZED (finalizado). Este atributo pode ser utilizado para tirar um *token* de circulação, bastando definir como finalizado. Isto é possível, pois nas funções que calculam o balanço de um usuário, é verificado que este atributo é ativo, como na linha 382 do Apêndice A.1.1.

- **faceValue**: Representa a quantidade de *tokens* deste objeto. Para fins de simplificação, este atributo pode possuir qualquer número acima de 0,1, ou seja escolheu-se permitir representar apenas valores maiores que um décimo. Diferentemente do *Bitcoin*, onde a menor fração do *token* é 1 *satoshi*, ou 10e-8, não se espera que o *token* chegue a um valor maior que o real. Em verdade, assume-se que o *token* tenha valor de exatamente 1 real.

- **issueDate**: Este atributo serve para anotar, no próprio objeto, qual a data de sua criação ou emissão.

- **maturityDate**: A data de expiração do *token*. Este atributo é necessário pois permite que beneficiários tenham acesso ao recurso por um tempo definido, como por exemplo, pela duração de sua bolsa, ou até o fim de seu curso. A checagem da expiração do *token* é feita juntamente com o atributo *currentState*, como na linha 382 do Apêndice A.1.1.
- **owner**: Especifica a identidade que é dona e tem permissão de manipular o *token*.
- **tokenId**: Este atributo serve para identificar um *token* em particular, pois quaisquer dois *tokens* podem possuir todos os atributos idênticos, menos o tokenId.

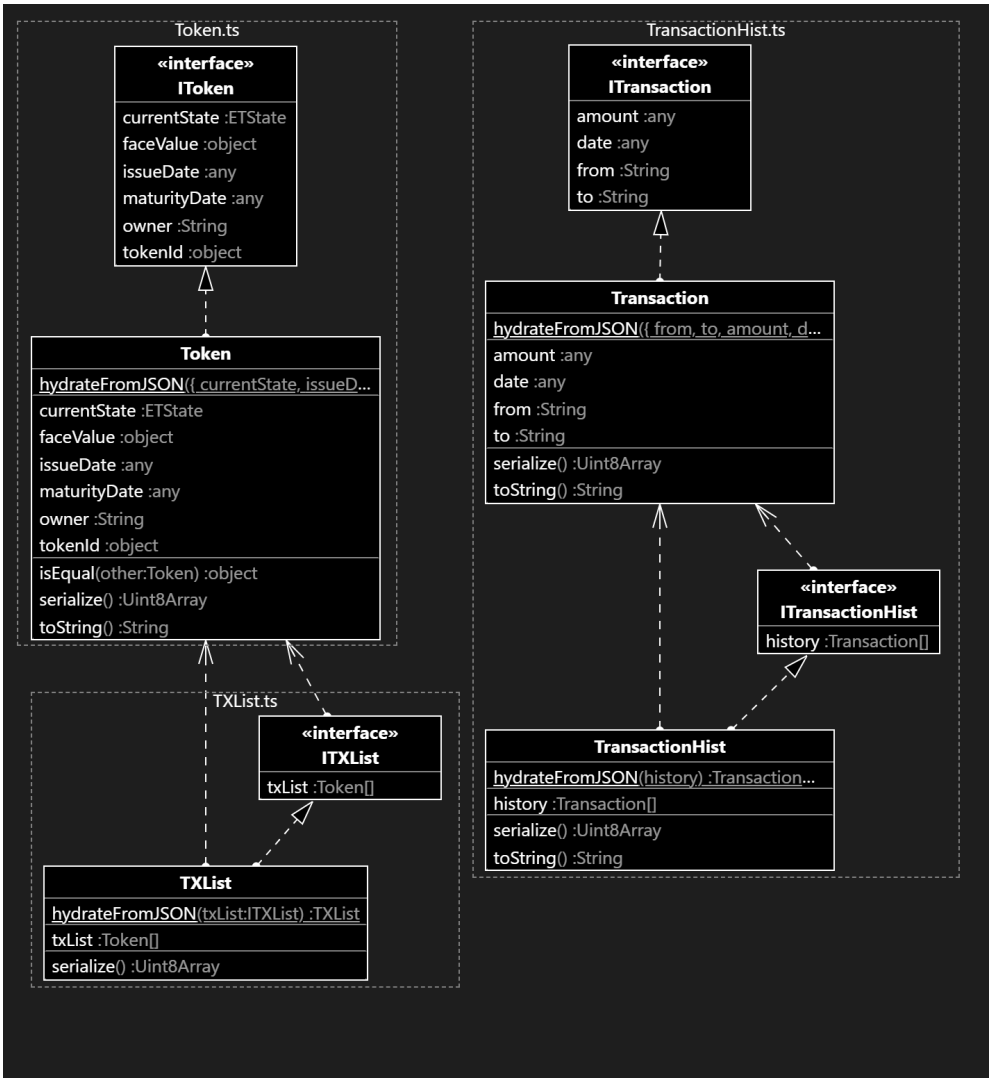
Em essência o *token* se comporta como uma criptomoeda, pois representa valor monetário no mundo físico. Por isso, este projeto baseou-se na implementação do *Bitcoin*. Assim como no Bitcoin, os usuários da rede não possuem propriamente uma estrutura de carteira onde estão armazenados os ativos (ANTONOPOULOS, 2015). Esta é apenas uma abstração que facilita o entendimento. Na realidade, cada fração de cripto-ativo possui um dono, e portanto, a maneira de se saber a quantidade de ativos que certa identidade possui, é contando a quantidade de ativos não gastos no nome dela na lista de *Unspent Transaction Outputs* (UTXO). No contrato implementado, a lista de UTXO, que é da classe *TXList*, conforme a figura 7, contém todas as frações de *token* que podem ser gastas.

Quando um *token* é gasto, ou seja, quando um usuário envia uma certa quantidade para outro, o contrato não muda o dono dos *tokens* que foram utilizados nesta transação, até porque nem sempre existe um só *token* com exatamente o valor a ser enviado. Em realidade, os *tokens* que foram escolhidos são destruídos, e em seu lugar é criado um novo *token* com o valor que se quer enviar em nome do destinatário, e, caso necessário, cria-se também um *token* com o valor do troco em nome do remetente. Assim como no Bitcoin, a lista de UTXO é pública e pode ser consultada por qualquer participante de qualquer organização na rede. O que impede usuários de consultarem os fundos de outros usuários é o *back-end*. Toda esta lógica esta implementada na função *sendTokens* do contrato nas linhas 162 a 275 do Apêndice A.1.1.

Para uma identidade dentro da rede enviar *tokens* para outra, é necessário fornecer a identidade alvo da transação, a identidade origem, a data atual, e a quantia. De forma a mais rapidamente escolher quais *tokens* gastar na transação, é utilizado um algoritmo guloso de busca equivalente ao algoritmo na figura 8. Este algoritmo também foi implementado no contrato na função *selectTksToSend*, nas linhas 285 a 327 do Apêndice A.1.1, porém optou-se por incluir também uma versão em *Python* por ser mais fácil de entender do que em *Typescript*. Este algoritmo busca primeiro o *token* de menor valor dentre os *tokens* com valor maior que a quantia. Caso encontre, retorna este *token*, juntamente com o valor do troco. Se não houver valor maior do que a quantia a ser enviada, o algoritmo itera sobre os *tokens* menores do que a quantia, acumulando o total e retorna os *tokens* acumulados caso encontre ou passe da quantia. Caso não se verifique saldo suficiente entre os menores, o algoritmo falha e a transação é revertida.

Na imagem 7, também pode-se ver a presença da classe *Transaction*. Esta classe representa uma transação, e é utilizada para facilitar a visualização do histórico de um usuário na interface.

Figura 7 – Diagrama de classes das estruturas utilizadas pelo contrato *mycc*



Fonte: Desenvolvimento próprio

Cada usuário possui um *TransactionHist*, que nada mais é que uma lista de transações. Cada emissão ou transação feita entre usuários na rede é adicionada a esta lista. Isto permite com que a busca por todas as transações de um usuário ocorra em tempo constante ($O(1)$) com relação ao número de usuários na plataforma, por meio da função *getUserHist* definida nas linhas 532 a 540 do Apêndice A.1.1. Isto porque, para obter as transações de um usuário é necessário um simples acesso à blockchain, com a chave recebida.

O contrato desenvolvido também permite ao Ministério da Educação (MEC), e somente o MEC, emitir os *tokens* à vontade. Isto é garantido, pois a função de emissão do contrato checa se a identidade que iniciou a transação pertence à organização MEC, como mostra o código da função *Issue* localizado nas linhas 411 a 414 do Apêndice A.1.1. *Tokens* criados desta forma entram na carteira do MEC, a partir do qual podem ser enviados para os beneficiários utilizando a função de transação.

Figura 8 – Código em *python* do algoritmo de seleção de Tokens.

```python
def select_outputs_greedy(unspent, min_value):
    # Fail if empty.
    if not unspent:
        return None
    # Partition into 2 lists.
    lessers = [utxo for utxo in unspent if utxo.value < min_value]
    greaters = [utxo for utxo in unspent if utxo.value >= min_value]
    key_func = lambda utxo: utxo.value
    if greaters:
        # Not-empty. Find the smallest greater.
        min_greater = min(greaters)
        change = min_greater.value - min_value
        return [min_greater], change
    # Not found in greaters. Try several lessers instead.
    # Rearrange them from biggest to smallest. We want to use the least
    # amount of inputs as possible.
    lessers.sort(key=key_func, reverse=True)
    result = []
    accum = 0
    for utxo in lessers:
        result.append(utxo)
        accum += utxo.value
        if accum >= min_value:
            change = accum - min_value
            return result, "Change: %d Satoshis" % change
    # No results found.
    return None, 0
```

Fonte: Excerto tirado de Antonopoulos (2015)

3.3 APLICAÇÃO WEB

Durante o desenvolvimento, a interação com o contrato inteligente ocorreu por meio de *Command-line Interface* (CLI). Tendo em vista a principal função da moeda ser o pagamento de serviços, optou-se por uma interface mais parecida com um banco. Para implementar esta parte do trabalho, utilizou-se a biblioteca *express* para o *Back-end* e *NEXTJS* para o *Front-end*, ambos em *typescript*. Um *back-end* ou servidor, é necessário pois cada usuário dentro da rede precisa criar uma carteira em disco, contendo a chave privada e a organização à qual pertence. Não é possível realizar alterações no sistema de arquivos apenas com a interface.

As visões de usuário contam com design responsivo, ou seja, são compatíveis com diferentes tipos de dispositivos, portanto um usuário poderia utilizar a interface web tanto no seu celular, quanto em seu computador pessoal. Foram implementadas 6 visões de usuário, com uma delas, a de emissão de *tokens*, visível apenas pelo MEC. Para se referir ao *token* implementado,

escolheu-se o nome Cripto RU (CRU), ou seja, usuários receberão *tokens* CRU como pagamento de auxílios.

3.3.1 Visões

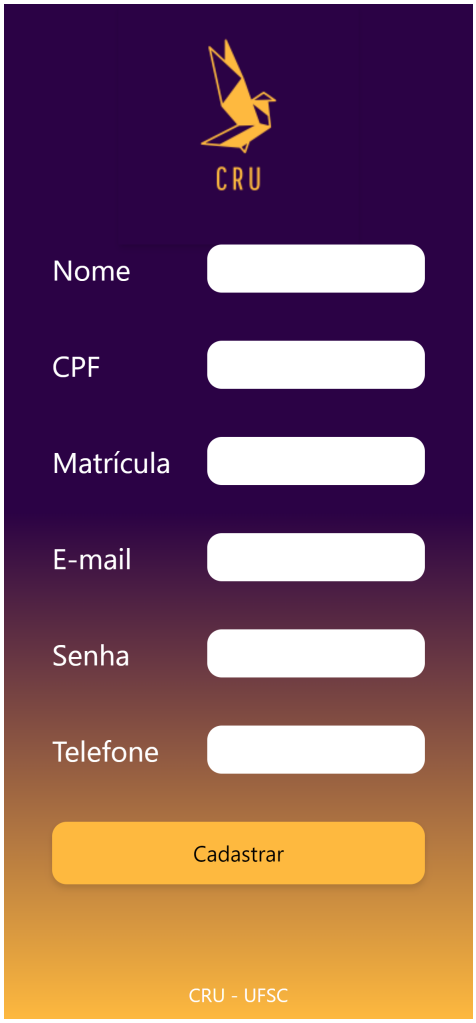
A primeira tela é a de *login* que é bastante simples, como se pode ver na imagem 9. Ao clicar em *login*, o usuário será redirecionado à sua página inicial caso possua conta. Para criar uma conta nova, o usuário deve clicar em Cadastrar.

Figura 9 – Tela de autenticação do aplicativo.

Fonte: Próprio

Ao clicar em Cadastrar, o usuário será redirecionado para a tela de cadastro, como na imagem 10. A tela de cadastro pede dados básicos sobre o aluno. Uma vez preenchidos, ao clicar em cadastrar, o usuário é automaticamente redirecionado à página inicial, já logado.

Figura 10 – Tela de cadastro do aplicativo.

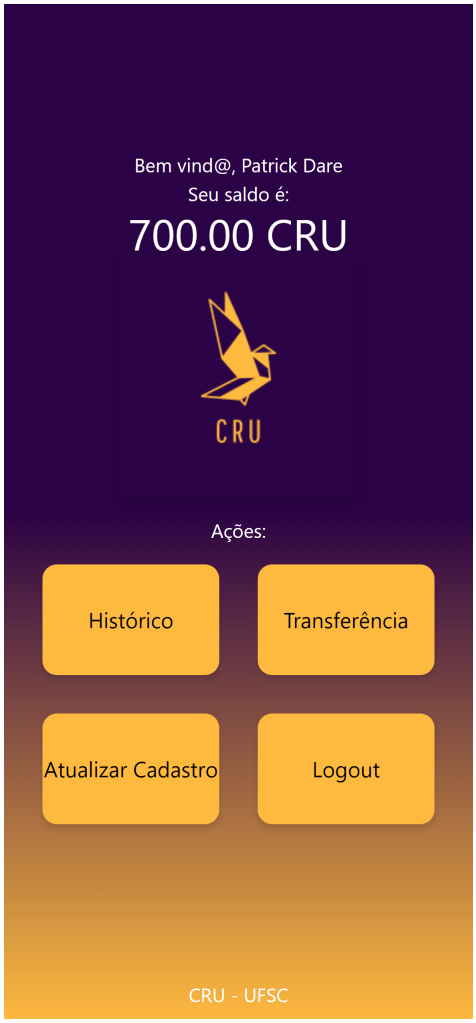


Fonte: Próprio

Ao se autenticar com sucesso, a página principal apresenta para o usuário as quatro opções: Logout, Atualizar Cadastro, Transferir e Ver Histórico conforme mostra a imagem 11. Caso o usuário administrador (MEC) faça login, a interface também mostrará a opção de emitir *tokens*, como na imagem 12. Por padrão, o contrato já inicializa a identidade do MEC possuindo 1000000 de *tokens*.

Para fazer um envio de *tokens*, o usuário deve clicar na opção de transferência, onde ele visualizará uma tela similar à imagem 13. O usuário deve inserir a matrícula do destinatário no primeiro campo, e a quantidade de *tokens* no segundo. Aqui o usuário tem a opção de procurar por outro usuário dentro da rede antes de fazer a transferência, para assegurar que os fundos vão chegar no destinatário apropriado. Para isto, após preenchido o destinatário, deve-se clicar no ícone da lupa, que vai informar ao usuário dentro da própria página se o destinatário é válido ou não. Assim que o usuário estiver satisfeito com o preenchimento dos campos, ele deve clicar em transferir. Caso haja algum erro, de qualquer natureza, a interface avisará na mesma janela o motivo do erro.

Figura 11 – Página inicial do aplicativo – tela de usuário.



Fonte: Próprio

A tela de Atualizar Cadastro, como o nome sugere, permite ao usuário corrigir eventuais erros no processo de cadastro. Ao clicar nesta opção, o servidor envia os dados atuais, para que o usuário saiba como eles estão no banco. O usuário pode diretamente alterar seus dados nos campos que desejar, e ao fim, clicar em atualizar que as modificações serão persistidas no banco.

Usuários contam com a visualização de seu histórico de transações por meio da opção "Histórico"na página inicial. Conforme a figura 15, o histórico de um usuário possui todas as transações que ele já fez ou já recebeu, com informações sobre a data, o remetente, o destinatário e o valor.

Por fim, a janela de emissão dos *tokens* acessível apenas pelo perfil administrador (MEC). Na janela ilustrada pela figura 16, o representante do MEC simplesmente insere a quantidade de *tokens* a emitir e clica em emitir. A quantidade desejada será criada na carteira do MEC e já estará disponível assim que a página redirecionar automaticamente à página inicial.

Figura 12 – Página inicial do aplicativo – tela de administrador.

Fonte: Próprio

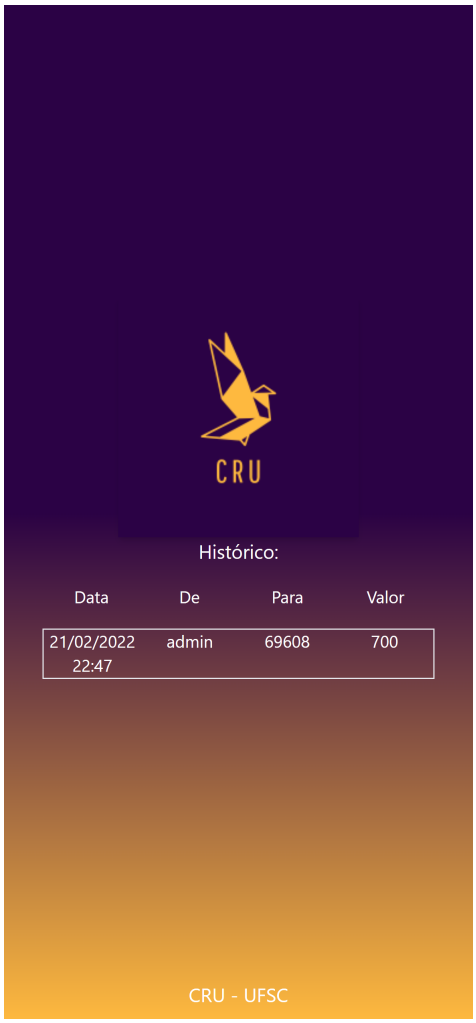
Figura 13 – Tela de transferência de *tokens* do aplicativo.

Fonte: Próprio

Figura 14 – Tela de atualização de cadastro do aplicativo.

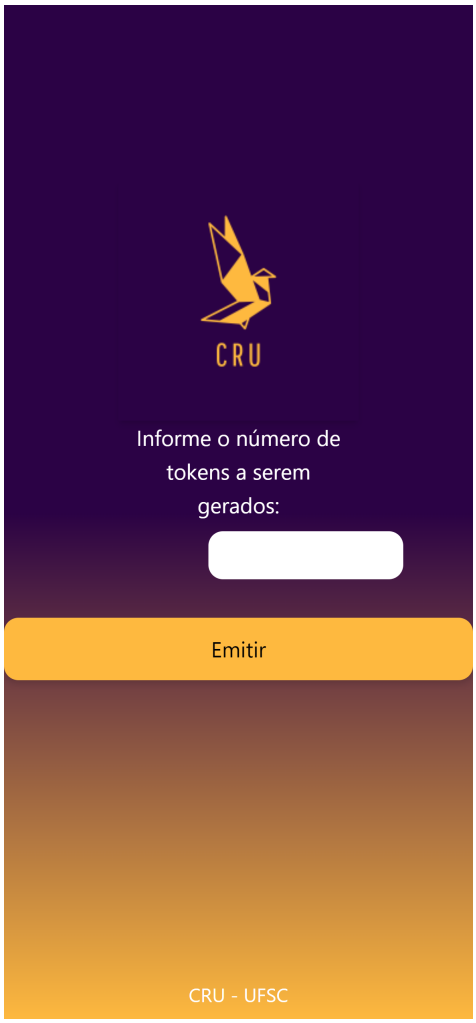
Fonte: Próprio

Figura 15 – Tela de histórico de transações.



Fonte: Próprio

Figura 16 – Tela de emissão de *tokens* do aplicativo.



Fonte: Próprio

4 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

4.1 CONSIDERAÇÕES FINAIS

A proposta inicial de conceber um *token* criptográfico e uma interface que permita sua manipulação foi desenvolvida com sucesso. Vale a pena discutir as consequências da maneira como foi desenvolvida a solução e compará-la com as soluções utilizadas por instituições públicas atualmente.

Existem alguns benefícios de se ter um esquema assim, ou seja um blockchain e não simplesmente um banco de dados centralizado na mão do governo. A vantagem principal é permitir a implantação de contratos. Isto é, é possível descrever situações altamente complexas que reflitam a realidade e fazer com que a rede realize toda a operação por meio do contrato. Isto é bastante interessante para o sistema, pois assegura maior transparência e segurança aos participantes. Por exemplo, um dos auxílios atualmente pagos pela UFSC é o auxilio moradia. O aluno recebe o dinheiro do auxílio diretamente na conta dele, mediante comprovação por meio do contrato. Utilizando a solução desenvolvida, é possível com que as regras e cláusulas deste contrato físico sejam modeladas no contrato inteligente. Isto faria com que o dinheiro fosse automaticamente cobrado da carteira dele, bem como quaisquer taxas ou compensações que sejam aplicáveis dependendo de acontecimentos futuros. Todo o processo de aprovação do benefício e pagamento ocorreria com bem menos atrito, além de reduzir o risco de haver erros.

A próxima vantagem é a imutabilidade e rastreabilidade simplificada dos recursos. Não é simples de esconder das autoridades uma transação que crie os recursos no nome de um intruso, até porque a rede não vai permitir. Para poder fazer algo neste estilo, seria necessário atualizar o contrato com código malicioso. Além disso, os usuários, ao utilizarem a rede para transacionar no dia a dia, naturalmente vão indicar mau-uso de recursos, uma vez que atores malignos na rede concentrarão grande parte dos recursos, permitindo fácil identificação. Atualmente, para congelar fundos sob mau uso por parte de estudantes, é necessário passar por todo um processo investigativo jurídico que é lento. Por outro lado, pelo menos tecnicamente, é simples para as autoridades comandar a invalidez dos *tokens* assim que detectar mau-uso, situação que não se verifica com passes físicos ou dinheiro vivo, ou até mesmo dinheiro em conta bancária.

Outro aspecto relevante diz respeito à autonomia universitária, como enquadrar programas não relacionados ao PNAES que cada universidade pode possuir? Em se tratando de outros programas próprios de assistência estudantil de cada universidade, esta proposta pode ser adaptada para servir as regras de cada programa de acordo com as necessidades. Isto é factível pois para cada programa pode-se criar seu próprio contrato que dispõe sobre suas regras específicas, criando um novo *token* ou utilizando o mesmo apresentado neste trabalho, tudo isto sem interferir no funcionamento do *token* proposto.

4.2 TRABALHOS FUTUROS

Este trabalho pode servir de base para realizar alguns outros projetos no futuro, como por exemplo:

1. Unificar com o sistema de transporte coletivo, permitindo com que usuários utilizem seus *tokens* para comprar passagens de ônibus.

2. Integração com o sistema de controle de acesso do RU, permitindo com que usuários utilizem as suas carteirinhas para pagar com os seus *tokens* na entrada do restaurante.

3. Ampliação/Adaptação das funcionalidades para incluir outras modalidades de bolsas e/ou outras instituições de ensino superior. Este trabalho teve como principal objetivo atender ao PNAES, porém, outros tipos de assistência, inclusive providenciados pela UFSC, poderiam também ser contemplados pelo esquema proposto.

REFERÊNCIAS

ANTONOPOULOS, Andreas M. **Mastering Bitcoin**. [*S.l.*: *s.n.*], 2015. P. xxi + 272. ISBN 1-4493-7404-2, 1-4919-0260-4. Citado nas pp. 16, 21, 27, 29.

BRASIL. Decreto nº 7.234, de 19 de julho de 2010. Dispõe sobre o Programa Nacional de Assistência Estudantil - PNAES. **Diário Oficial [da] República Federativa do Brasil**, Brasília, DF, 19 jul. 2010. Disponível em: <http://www.planalto.gov.br/ccivil_03/_ato2007-2010/2010/decreto/d7234.htm>. Acesso em: 5 out. 2021. Citado na p. 12.

BRASIL. **portal gov**. [*S.l.*: *s.n.*], 2021. Disponível em: <https://www.gov.br/mec/pt-br/acesso-a-informacao/institucional/secretarias/secretaria-de-educacao-superior/pnaes>. Citado na p. 13.

BUTERIN, Vitalik *et al.* A next-generation smart contract and decentralized application platform, 2014. Citado na p. 22.

ENTRIKEN, William *et al.* **EIP-721: Non-Fungible Token Standard**. [*S.l.*], jan. 2018. Disponível em: <https://eips.ethereum.org/EIPS/eip-721>. Citado na p. 21.

ETHGAS.WATCH. **Eth Gas Watch**. 2022. Disponível em: <ethgas.watch>. Citado na p. 24.

FABRIC, Hyperledger. **Hyperledger Fabric White paper**. 2020. Disponível em: <https://www.hyperledger.org/wp-content/uploads/2020/03/hyperledger_fabric_whitepaper.pdf>. Acesso em: 13 out. 2021. Citado na p. 23.

HYPERLDGER, Foundation. **Hyperledger Fabric Docs**. 2020. Disponível em: <https://hyperledger-fabric.readthedocs.io/en/latest/network/network.html>. Citado na p. 23.

INTERNATIONAL, Transparency. **Indice de corrupção percebida**. 2020. Disponível em: <https://www.transparency.org/en/cpi/2020/index/nzl>. Acesso em: 13 out. 2021. Citado na p. 12.

JONES, M.; BRADLEY, J.; SAKIMURA, N. **JSON Web Token (JWT)**. [*S.l.*], mai. 2015. http://www.rfc-editor.org/rfc/rfc7519.txt. Disponível em: <http://www.rfc-editor.org/rfc/rfc7519.txt>. Citado na p. 21.

LIN, Iuon-Chang; LIAO, Tzu-Chun. A survey of blockchain security issues and challenges. **Int. J. Netw. Secur.**, v. 19, n. 5, p. 653–659, 2017. Citado na p. 16.

MACEDO, Fausto. PF investiga fraudes até no bandejão da Universidade de Viçosa. **ADPF**, 2016. Disponível em: <http://www.adpf.org.br/adpf/admin/painelcontrole/materia/materia_portal.wsp?tmp.edt.materia_codigo=8229&wi.redirect=327R0K9DO174MA10MI1S#.Yhb7ncnMJrp>. Citado na p. 12.

MIRELLE PINHEIRO, Carlos Carone. PF investiga professor pelo desvio de R$ 3,7 milhões de universidade. **Metrópoles**, 2022. Disponível em: <https://www.metropoles.com/distrito-federal/na-mira/pf-investiga-professor-pelo-desvio-de-r-37-milhoes-de-universidade>. Citado na p. 12.

NAKAMOTO, Satoshi. **Bitcoin: A peer-to-peer electronic cash system**. [*S.l.*: *s.n.*], 2009. Disponível em: <http://www.bitcoin.org/bitcoin.pdf>. Citado nas pp. 12, 17, 18.

NARAYANAN, Arvind *et al.* **Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction**. USA: Princeton University Press, 2016. ISBN 0691171696. Citado nas pp. 15, 18.

PETERSON, L.L.; DAVIE, B.S. **Computer Networks: A Systems Approach**. [*S.l.*]: Elsevier Science, 2011. P. 769. (ISSN). ISBN 9780123850607. Disponível em: <https://books.google.com.br/books?id=BvaFreun1W8C>. Citado na p. 15.

STALLINGS, William. **Cryptography and Network Security: Principles and Practice**. 6th. USA: Prentice Hall Press, 2013. ISBN 0133354695. Citado na p. 14.

SWAN, Melanie. **Blockchain: blueprint for a new economy**. pub-ORA-MEDIA:adr: pub-ORA-MEDIA, 2015. P. xviii + 130. ISBN 1-4919-2049-1, 1-4919-2045-9 (e-book), 1-4919-2047-5 (e-book). Disponível em: <http://proquest.safaribooksonline.com/?fpi=9781491920480>. Citado na p. 20.

SZABO, Nick. Formalizing and securing relationships on public networks. **First monday**, 1997. Citado na p. 21.

TEAM, THE INVESTOPEDIA. **Investopedia: Money**. 2020. Disponível em: <https://www.investopedia.com/terms/m/money.asp>. Acesso em: 13 out. 2021. Citado na p. 19.

TRIPLEA. **Estatísticas realcionadas a adoção de criptomoedas**. 2021. Disponível em: <https://triple-a.io/crypto-ownership/>. Acesso em: 13 out. 2021. Citado na p. 13.

VOSHMGIR, S. **Token Economy: How the Web3 reinvents the Internet**. [*S.l.*]: Shermin Voshmgir, 2020. P. 148–152. ISBN 9783982103839. Disponível em: <https://books.google.com.br/books?id=vWo-EAAAQBAJ>. Citado nas pp. 20, 21.

YAGA, Dylan *et al.* Blockchain technology overview. **arXiv preprint arXiv:1906.11078**, 2019. Citado na p. 19.

Apêndices

APÊNDICE A – ARQUIVOS CONTENDO OS CÓDIGOS FONTE

A.1 CÓDIGO DO CONTRATO

A.1.1 index.ts

```
import { ChaincodeResponse, ChaincodeStub, Shim } from "fabric-shim";
import { Token, ETState } from "./Token";
import { ITXList, TXList } from "./TXList";
import {
    TransactionHist,
    Transaction,
    ITransactionHist,
} from "./TransactionHist";
const MEC = "mec-example-com";

/**
 * Business unrelated function that takes a vector.
 * and returns the appropriate type of each element of the array
 * @param {Array} arr the parameter array
 * @return {(string | number | Boolean | Date)[]} an array containing the elements
 ↪  with the appropriate type
 */
function parseStringArray(
    arr: Array<string>
): (string | number | Boolean | Date)[] {
    return arr.map((el) => {
        if (!Number.isNaN(Number(el))) {
            // Try Number
            return Number(el);
        } else if (!Number.isNaN(new Date(el).getDate())) {
            // Try Date
            return new Date(el);
        } else if (
            el.toLowerCase() === "true" ||
            el.toLowerCase() === "false"
        ) {
            // Try Boolean
            return new Boolean(el);
        } else {
            // Return as String
            return el;
        }
```

```
37      });
38  }
39
40  /**
41   * Business unrelated function that takes an object
42   * and returns the appropriate type in string form
43   * @param {number | Token | ChaincodeResponse} obj The object to get the string
    ↪  type of
44   * @return {string} the string type
45   */
46  function getType(obj: number | Token | ChaincodeResponse): string {
47      if (!Number.isNaN(Number(obj))) {
48          return "Number";
49      } else if (Object.keys(obj).includes("faceValue")) {
50          return "Token";
51      } else {
52          return "ChaincodeResponse";
53      }
54  }
55
56  class Chaincode {
57      static logger = Shim.newLogger("LOGGING_OUT");
58
59      // doesn't require any parameters, for now
60      async Init(stub: ChaincodeStub): Promise<ChaincodeResponse> {
61          console.info("========= example02 Init =========");
62          Chaincode.logger.level = "debug";
63
64          try {
65              // Create UTXOLIST
66              let initialIssue = new Token(
67                  await Chaincode.produceNextId(stub),
68                  "admin",
69                  new Date("2022-01-01"),
70                  new Date("2023-01-01"),
71                  1e6
72              );
73              initialIssue.currentState = ETState.ACTIVE;
74              await stub.putState(
75                  "UTXOLIST",
76                  new TXList([initialIssue]).serialize()
77              );
```

```
78              // create very first emission
79              return Shim.success(initialIssue.serialize());
80              // return Shim.success(Buffer.from("initializedMen"));
81          } catch (err: any) {
82              return Shim.error(err);
83          }
84      }
85
86      async Invoke(stub: ChaincodeStub): Promise<ChaincodeResponse> {
87          const ret = stub.getFunctionAndParameters();
88          console.info(ret);
89          const self = this;
90          type AllowedMethod =
91              | "getBalance"
92              | "issue"
93              | "query"
94              | "Init"
95              | "sendTokens"
96              | "getHist"
97              | "getUserHist";
98          const method = self[ret.fcn as AllowedMethod];
99          if (!method) {
100             console.log("no method of name:" + ret.fcn + " found");
101             return Shim.error(
102                 Buffer.from("no method of name:" + ret.fcn + " found")
103             );
104         }
105         try {
106             const params = [
107                 stub,
108                 ...parseStringArray(ret.params),
109             ] as Parameters<typeof method>;
110             // @ts-ignore
111             const payload = await method(...params);
112             switch (getType(payload)) {
113                 case "Number":
114                     return Shim.success(Buffer.from(payload.toString()));
115                 case "Token":
116                     return Shim.success((payload as Token).serialize());
117                 default:
118                     return payload as ChaincodeResponse;
119             }
```

```
120         } catch (err: any) {
121             console.log(err);
122             return Shim.error(err);
123         }
124     }
125 
126     /**
127      * Increments current tokenIDCounter and returns it. Creates if not exists.
128      *
129      * @param {ChaincodeStub} stub the transaction context
130      * @Return the updated counter
131      */
132     static async produceNextId(stub: ChaincodeStub): Promise<number> {
133         let id = 0;
134         try {
135             if (
136                 new TextDecoder().decode(
137                     await stub.getState("tokenIDCounter")
138                 ) !== null
139             ) {
140                 id =
141                     1 +
142                     Number(
143                         new TextDecoder().decode(
144                             await stub.getState("tokenIDCounter")
145                         )
146                     );
147             }
148             stub.putState("tokenIDCounter", Buffer.from(id.toString()));
149         } catch (err) {
150             console.log(err);
151         }
152         return id;
153     }
154 
155     /**
156      * Checks if the owner has the tokens required and sends the tokens to the target
157      *
158      * @param {ChaincodeStub} stub the transaction context
159      * @param {string} from
160      * @param {string} to
161      */
```

```
162     async sendTokens(
163         stub: ChaincodeStub,
164         from: string,
165         to: string,
166         quantity: number,
167         date: string
168     ): Promise<ChaincodeResponse> {
169         // from = from.toString()
170         // to = to.toString()
171         // check whoever is sending can send tokens
172         if (!from || !to || !quantity) {
173             let error = Shim.error(
174                 Buffer.from("Incorrect number of parameters")
175             );
176             error.payload = Buffer.from(JSON.stringify({ code: "params" }));
177             return error;
178         }
179         if (from === to) {
180             let error = Shim.error(
181                 Buffer.from("Not allowed to send tokens to same place")
182             );
183             error.payload = Buffer.from(JSON.stringify({ code: "same" }));
184             return error;
185         }
186         if (quantity < 0.1) {
187             let error = Shim.error(
188                 Buffer.from("Not allowed to send less than 0.1 tokens")
189             );
190             error.payload = Buffer.from(JSON.stringify({ code: "min" }));
191             return error;
192         }
193         quantity = Number.parseFloat(quantity.toFixed(1));
194         const now = new Date(date);
195         // check balance
196         let [tks, change] = await Chaincode.selectTksToSend(
197             stub,
198             from,
199             quantity,
200             now
201         );
202         Chaincode.logger.debug(JSON.stringify(tks));
203         Chaincode.logger.debug(change);
```

```
204         // if no tokens found, return error
205         if (tks.length === 0) {
206             let error = Shim.error(Buffer.from(from + " lacks funds"));
207             error.payload = Buffer.from(JSON.stringify({ code: "funds" }));
208             return error;
209         }
210 
211         let tokenList = await Chaincode.getUTXOList(stub);
212 
213         let expirationDate = new Date(tks[0].maturityDate);
214         let issueDate = new Date(tks[0].issueDate);
215         let tokenState = tks[0].currentState;
216 
217         if (change !== 0) {
218             // create change
219             let newToken = new Token(
220                 await Chaincode.produceNextId(stub), // produceNextId(stub)
221                 from,
222                 issueDate,
223                 expirationDate,
224                 change
225             );
226             newToken.currentState = tokenState;
227             tokenList.txList.push(newToken);
228         }
229         // create the token with the amount sent in the name of the recipient
230         let newToken = new Token(
231             await Chaincode.produceNextId(stub),
232             to,
233             issueDate,
234             expirationDate,
235             quantity
236         );
237         newToken.currentState = tokenState;
238         tokenList.txList.push(newToken);
239         Chaincode.logger.warn(JSON.stringify(tokenList));
240 
241         // delete these tokens
242         for (const tk of tks) {
243             const deleted = tokenList.txList.splice(
244                 tokenList.txList.findIndex((token) => token.isEqual(tk)),
245                 1
```

```
246             );
247             Chaincode.logger.debug("Deleted: " + deleted);
248         }
249         // write this back on chaincode
250         await stub.putState("UTXOLIST", tokenList.serialize());
251         // save on the to and froms user history
252         let fromHist = await Chaincode.getHistoryList(stub, from);
253         let toHist = await Chaincode.getHistoryList(stub, to);
254         const newTransaction = new Transaction(from, to, quantity, now);
255         fromHist.history.push(newTransaction);
256         toHist.history.push(newTransaction);
257 
258         await stub.putState(from, fromHist.serialize());
259         await stub.putState(to, toHist.serialize());
260 
261         let ccresponse = Shim.success(
262             Buffer.from(
263                 "Transferred " +
264                     quantity +
265                     " from " +
266                     from +
267                     " to " +
268                     to +
269                     " tokenId: " +
270                     tks[0].tokenId
271             )
272         );
273         ccresponse.payload = Buffer.from(JSON.stringify({ code: "success" }));
274         return ccresponse;
275     }
276 
277     /**
278      * Choses the tokens required to create a transaction from the owner with the
    ↪  quanity required
279      *
280      * @param {ChaincodeStub} stub the transaction context
281      * @param {string} owner the identity which will give the tokens
282      * @param {string} quanity the total tokens to send
283      * @param {string} now the current time - to maintain consistency between peers
    ↪  when checking expiry
284      */
285     static async selectTksToSend(
```

```
286         stub: ChaincodeStub,
287         owner: string,
288         quantity: number,
289         now: Date
290     ): Promise<[Token[], number]> {
291         // get only the tokens that belong to the owner and are still valid
292         let tokensOfOwner = (await Chaincode.getUTXOList(stub)).txList.filter(
293             (token) => ((token.owner === owner) && (now <= token.maturityDate) &&
                ↪  (token.currentState !== ETState.FINALIZED))
294         );
295         // seprate the ones bigger and smaller from the list
296         const greaters = tokensOfOwner.filter(
297             (token) => token.faceValue >= quantity
298         );
299         let change = 0;
300         if (typeof greaters !== "undefined" && greaters.length > 0) {
301             // get min value
302             let min = greaters[0];
303             for (const e of greaters) {
304                 if (e.faceValue < min.faceValue) min = e;
305             }
306             change = min.faceValue - quantity;
307             return [[min], change];
308         }
309         // no values above the required, look for the sum in the lessers
310         let lessers = tokensOfOwner.filter(
311             (token) => token.faceValue < quantity
312         );
313         // sort values
314         lessers.sort((left, right) => left.faceValue - right.faceValue);
315         let output = [];
316         let sum = 0;
317         for (const token of lessers) {
318             output.push(token);
319             sum += token.faceValue;
320             if (sum >= quantity) {
321                 change = sum - quantity;
322                 return [output, change];
323             }
324         }
325         // not enough funds
326         return [[], -1];
```

```
327     }
328
329     /**
330      * Get the list of unspent transactions
331      *
332      * @param {ChaincodeStub} stub the transaction context
333      * @Return the transaction list
334      */
335     static async getUTXOList(stub: ChaincodeStub): Promise<TXList> {
336         return TXList.hydrateFromJSON(
337             JSON.parse(
338                 new TextDecoder().decode(await stub.getState("UTXOLIST"))
339             ) as ITXList
340         );
341     }
342
343     /**
344      * Returns a fresh History list in the name of the user, or the list if it exists
345      *
346      * @param {ChaincodeStub} stub the transaction context
347      * @param {string} user the user to get the hist from
348      * @Return the transaction list
349      */
350     static async getHistoryList(
351         stub: ChaincodeStub,
352         user: string
353     ): Promise<TransactionHist> {
354         if ((await stub.getState(user)).length === 0) {
355             this.logger.debug("creating fresh hist");
356             return new TransactionHist([]);
357         }
358         return TransactionHist.hydrateFromJSON(
359             JSON.parse(
360                 new TextDecoder().decode(await stub.getState(user))
361             ) as ITransactionHist
362         );
363     }
364
365     /**
366      *  Gets the number of tokens of a certain identity.
367      *
368      * @param {ChaincodeStub} stub the transaction context
```

```
369         * @param {string} owner the identity
370         * @Return the current balance
371         */
372        async getBalance(
373            stub: ChaincodeStub,
374            owner: string | Number,
375            date: string
376        ): Promise<number> {
377            // query the elements in the UTXOLIST
378            if (typeof owner === "number") owner = owner.toString();
379            let accum = 0;
380            const now = new Date(date);
381            (await Chaincode.getUTXOList(stub)).txList.forEach((token) => {
382                if ((token.owner === owner) && (now <= token.maturityDate) &&
                ↪  (token.currentState !== ETState.FINALIZED))
383                    accum += token.faceValue;
384            });
385
386            return accum;
387        }
388
389        /**
390         * Issue token, Mec is the only valid issuer
391         *
392         * @param {ChaincodeStub} stub the transaction context
393         * @param {string} issueDate token issue date
394         * @param {string} maturityDate token maturity date
395         * @param {number} faceValue face value of token
396         */
397        async issue(
398            stub: ChaincodeStub,
399            issueDate: Date,
400            maturityDate: Date,
401            faceValue: number
402        ): Promise<Token> {
403            // incorrect params
404            if (!issueDate || !maturityDate || !faceValue) {
405                console.log(issueDate);
406                console.log(maturityDate);
407                console.log(faceValue);
408                throw new Error("Incorrent number of parameters");
409            }
```

```
410         // stop from transacting if identity not Mec
411         if (stub.getCreator().mspid !== MEC)
412             throw new Error(
413                 "Command issuer not Mec, and therefore not allowed to issue tokens"
414             );
415         const adminId = "admin";
416         // create token
417         const token = new Token(
418             await Chaincode.produceNextId(stub),
419             adminId, // owner is hard-coded
420             issueDate,
421             maturityDate,
422             faceValue
423         );
424         // set Active
425         token.currentState = ETState.ACTIVE;
426         // add to world state
427         let UTXOLIST = await Chaincode.getUTXOList(stub);
428         UTXOLIST.txList.push(token);
429         await stub.putState("UTXOLIST", UTXOLIST.serialize());
430 
431         // save on the to and froms user history
432         let adminHist = await Chaincode.getHistoryList(stub, adminId);
433         const now = issueDate;
434         const newTransaction = new Transaction(
435             "Nova Emissão",
436             adminId,
437             faceValue,
438             now
439         );
440         adminHist.history.push(newTransaction);
441 
442         await stub.putState(adminId, adminHist.serialize());
443         // return it
444         return token;
445     }
446 
447     // Deletes an entity from state
448     async delete(stub: ChaincodeStub, args: string[]) {
449         if (args.length != 1) {
450             throw new Error("Incorrect number of arguments. Expecting 1");
451         }
```

```
452
453         const A = args[0];
454
455         // Delete the key from the state in ledger
456         await stub.deleteState(A);
457     }
458
459     /**
460      * Returns the history of the UTXO, i.e. every possible way it has look like
    ↪  since creation, including
461      *
462      * @param {ChaincodeStub} stub the transaction context
463      * @param {string} key UTXO
464      */
465     async getHist(
466         stub: ChaincodeStub,
467         key: string
468     ): Promise<ChaincodeResponse> {
469         if (!key) throw new Error("Incorrent number of parameters");
470         const list = [];
471         const historyQueryIterator = await stub.getHistoryForKey(key);
472         while (true) {
473             let jsonResp: {
474                 txId?: string;
475                 timestamp?: Date;
476                 isDelete?: boolean;
477                 value?: TXList;
478             } = {};
479             const res = await historyQueryIterator.next();
480             if (res.done) {
481                 await historyQueryIterator.close();
482                 return Shim.success(Buffer.from(JSON.stringify(list)));
483             }
484             jsonResp.txId = res.value.txId;
485             jsonResp.timestamp = new Date(res.value.timestamp.nanos * 1e9);
486             if (res.value.isDelete) {
487                 jsonResp.isDelete = res.value.isDelete;
488             } else {
489                 jsonResp.value = TXList.hydrateFromJSON(
490                     JSON.parse(new TextDecoder().decode(res.value.value))
491                 );
492             }
```

```
493             list.push(jsonResp);
494         }
495     }
496     // query callback representing the query of a chaincode
497     async query(
498         stub: ChaincodeStub,
499         ...args: string[]
500     ): Promise<ChaincodeResponse> {
501         console.log(args);
502         if (args.length != 1) {
503             throw new Error(
504               "Incorrect number of arguments. Expecting name of the person to query"
505             );
506         }
507 
508         const jsonResp = { name: "", amount: "" };
509         const A = args[0];
510 
511         // Get the state from the ledger
512         const Avalbytes = await stub.getState(A);
513         if (!Avalbytes) {
514             throw new Error("Failed to get state for " + A);
515         }
516 
517         jsonResp.name = A;
518         jsonResp.amount = Avalbytes.toString();
519         console.info("Query Response:");
520         console.info(jsonResp);
521         if (jsonResp.amount === "")
522             return Shim.success(Buffer.from("no data here, amount empty"));
523         return Shim.success(Avalbytes);
524     }
525 
526     /**
527      * Returns the users history
528      *
529      * @param {ChaincodeStub} stub the transaction context
530      * @param {Number} user the user whose hist is required
531     */
532     async getUserHist(
533         stub: ChaincodeStub,
534         user: Number
```

```
535     ): Promise<ChaincodeResponse> {
536         return Shim.success(
537             (await Chaincode.getHistoryList(stub, user.toString())).serialize()
538         );
539     }
540 }
541
542 Shim.start(new Chaincode());
```

A.1.2 Token.ts

```
1  export enum ETState {
2      ACTIVE = 1,
3      FINALIZED,
4  }
5
6  export interface IToken {
7      // Token current state
8      currentState: ETState;
9      // Token issue number id
10     tokenId: number;
11     // Token owner
12     owner: string;
13     // Token issue date
14     issueDate: Date;
15     // Maturity date
16     maturityDate: Date;
17     // Face value
18     faceValue: number;
19 }
20 export class Token implements IToken {
21     constructor(
22         tokenID: number,
23         owner: string,
24         issueDate: Date,
25         maturityDate: Date,
26         faceValue: number
27     ) {
28         this.tokenId = tokenID;
29         this.owner = owner;
30         this.issueDate = issueDate;
31         this.maturityDate = maturityDate;
32         this.faceValue = faceValue;
```

```
33      }
34      // Token current state
35      currentState!: ETState;
36      // Token issue namehydrateFromJSON
37      tokenId!: number;
38      // Token owner
39      owner!: string;
40      // Token issue date xx-xx-xxxx
41      issueDate!: Date;
42      // Maturity date
43      maturityDate!: Date;
44      // Face value
45      faceValue!: number;
46
47      /**
48       * Convert this object to buffer containing JSON data serialization
49       * Typically used before putState()ledger API
50       * @return {Buffer} buffer with the data to store
51       */
52      serialize(): Uint8Array {
53          return Buffer.from(JSON.stringify(this));
54      }
55
56      /**
57       * Compare utility
58       * @return {boolean} true if 2 tokens are the equal in value
59       */
60      isEqual(other: Token): boolean {
61          return this.currentState === other.currentState &&
62          this.tokenId === other.tokenId &&
63          this.owner === other.owner &&
64          this.issueDate.getTime() === other.issueDate.getTime() &&
65          this.maturityDate.getTime() === other.maturityDate.getTime() &&
66          this.faceValue === other.faceValue;
67      }
68
69      /**
70       * Creates a Token object from a raw JSON object
71       * @param {IToken} token The object to rehydrate (typically the return of
    ↪  JSON.parse())
72       * @return {Token} The hydrated Token
73       */
```

```
74      static hydrateFromJSON({
75          currentState,
76          issueDate,
77          faceValue,
78          maturityDate,
79          owner,
80          tokenId,
81      }: IToken): Token {
82          const token = new Token(tokenId, owner, new Date(issueDate), new
            ↪  Date(maturityDate), faceValue);
83          token.currentState = currentState;
84          return token;
85      }
86
87      /**
88       * Convert this object to string representation
89       * @return {string}
90       */
91      toString(): string {
92          return JSON.stringify(this);
93      }
94  }
```

A.1.3 TransactionHist.ts

```
1   export interface ITransaction {
2       from: string;
3       to: string;
4       amount: Number;
5       date: Date;
6   }
7
8   export class Transaction implements ITransaction {
9       from!: string;
10      to!: string;
11      amount!: Number;
12      date!: Date;
13
14      constructor(from: string, to: string, amount: Number, date: Date) {
15          this.from = from;
16          this.to = to;
17          this.amount = amount;
18          this.date = date;
```

```
19      }
20
21      static hydrateFromJSON({ from, to, amount, date }: ITransaction): Transaction {
22          return new Transaction(from, to, amount, date);
23      }
24      /**
25       * Convert this object to buffer containing JSON data serialization
26       * Typically used before putState() ledger API
27       * @return {Buffer} buffer with the data to store
28       */
29       serialize(): Uint8Array {
30          return Buffer.from(JSON.stringify(this));
31      }
32
33      /**
34       * Convert this object to string representation
35       * @return {string}
36       */
37       toString(): string {
38          return JSON.stringify(this);
39      }
40  }
41
42  export interface ITransactionHist {
43      history: Transaction[];
44  }
45
46  export class TransactionHist implements ITransactionHist {
47      history: Transaction[];
48      constructor(history: Transaction[]) {
49          this.history = history;
50      }
51
52      /**
53       * Convert this object to buffer containing JSON data serialization
54       * Typically used before putState()ledger API
55       * @return {Buffer} Buffer with the data to store
56       */
57      serialize(): Uint8Array {
58          return Buffer.from(JSON.stringify(this));
59      }
60      /**
```

```
61      * Creates a TransactionHist object from a raw JSON object
62      * @param {ITransactionHist} history The object to rehydrate (typically the
    ↪  return of JSON.parse())
63      * @return {TransactionHist} The hydrated list
64      */
65     static hydrateFromJSON(history: ITransactionHist): TransactionHist {
66         return new TransactionHist(
67             history.history.map((tx) => Transaction.hydrateFromJSON(tx))
68         );
69     }
70     /**
71      * Convert this object to string representation
72      * @return {string}
73      */
74     toString(): string {
75         return JSON.stringify(this);
76     }
77 }
```

A.1.4 TXList.ts

```
1  import { Token } from "./Token";
2
3  export interface ITXList {
4      txList: Token[];
5  }
6
7  export class TXList implements ITXList {
8      constructor(txList: Token[]) {
9          this.txList = txList;
10     }
11     // list of tokens
12     txList!: Token[];
13
14     /**
15      * Convert this object to buffer containing JSON data serialization
16      * Typically used before putState()ledger API
17      * @return {Buffer} Buffer with the data to store
18      */
19     serialize(): Uint8Array {
20         return Buffer.from(JSON.stringify(this));
21     }
22     /**
```

```
23      * Creates a TXList object from a raw JSON object
24      * @param {ITXList} txList The object to rehydrate (typically the return of
    ↪  JSON.parse())
25      * @return {TXList} The hydrated list
26      */
27     static hydrateFromJSON(txList: ITXList): TXList {
28         return new TXList(
29             txList.txList.map((token) => Token.hydrateFromJSON(token))
30         );
31     }
32 }
```

APÊNDICE B – ARTIGO DO TCC

Token Criptográfico Para Pagamento De Programas De Permanência Estudantil

Arthur Philippi Bianco[1]

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

Abstract. *Methods used by public federal institutions of higher education in student assistance actions are outdated and lack transparency. A solution to this problem is to use blockchain technology, which has the benefit of traceability and transparency in transactions. This technology makes it ideal for use by public institutions of student assistance in programs such as the National Student Assistance Program, which aims to guarantee the permanence and completion of students in higher education. This project develops a cryptographic token prototype to be used as currency by students of higher education institutions, so that students and beneficiaries of student assistance programs can use it to buy the university restaurant pass, pay rent and buy products from associated companies.*

Resumo. *Métodos utilizados por instituições federais públicas de ensino superior em ações de assistência estudantil estão desatualizados e faltam com transparência. Uma solução ao problema é a utilização de tecnologia de blockchain, que possui o benefício da rastreabilidade e transparência nas transações. Esta tecnologia a torna ideal para uso por instituições públicas de assistência à permanência estudantil em programas como o Programa Nacional de Assistência Estudantil, que tem por objetivo garantir a permanência e conclusão dos estudantes no ensino superior. Este projeto desenvolve um protótipo de token criptográfico para ser utilizando como moeda por estudantes de instituições de ensino superior, de modo que alunos e beneficiários de programas de assistência estudantil possam utilizá-lo para comprar o passe do restaurante universitário, pagar aluguel e comprar produtos em empresas associadas.*

1. Introdução

Com a promulgação do decreto 7.234, que institui a criação do Programa Nacional de Assistência Estudantil (PNAES), estudantes de graduação em universidades públicas brasileiras passaram a receber incentivos à sua permanência e conclusão do ensino superior [Brasil 2010]. Beneficiários deste programa podem receber assistência na moradia estudantil, alimentação, transporte, entre outros. Usualmente, cada universidade recebe uma fração da verba do programa, e a universidade fica responsável por distribuir estes recursos para os contemplados. Os beneficiários destas políticas recebem os benefícios, muitas vezes, diretamente nas próprias contas pessoais. Os alunos por sua vez, podem utilizar dos recursos da forma como quiserem.

Estes recursos podem, eventualmente, ser utilizados indevidamente e é difícil verificar sua correta utilização. Além disso, pode ocorrer também desvio destes recursos

por parte de autoridades públicas, antes mesmo dos valores chegarem nas mãos dos alunos. Por exemplo, na Universidade Federal de Viçosa, onde um grupo de funcionários foram acusados de desviar alimentos dos estoques do refeitório custeados pelo PNAES [Macedo 2016].

O método comumente utilizado para transferir o benefício no nome do aluno é eletrônico. Este método é seguro e eficiente. Entretanto, quando efetivamente ocorre um desvio, o processo de investigação é lento e depende de denúncia ou averiguação por órgãos competentes. Também é preciso levar em conta que processos investigativos são demorados por estarem sujeitos aos labirintos inerentes aos trâmites legais. Com efeito, em março de 2022, ocorre a operação com o sugestivo nome de "Quadro Negro". A atividade policial tem como objetivo desarticular organização criminosa envolvida em desvio de verbas destinadas à educação pública na Universidade Federal Fluminense (UFF) [Mirelle Pinheiro 2022]. A investigação identificou suposto desvio de R$ 3,7 milhões em contratações emergenciais e ordinárias entre a UFF e uma grande empresa de terceirização de mão de obra, no período de 2011 a 2015.

Diante disto, propõe-se uma solução utilizando tecnologia *blockchain* para lidar com os problemas mencionados, dificultando ações ilícitas. Esta tecnologia inventada pelo lendário Satoshi Nakamoto [Nakamoto 2009], possui propriedades intrínsecas de transparência e imutabilidade dos dados. Com ela, pode-se facilitar a constatação de utilização inadequada de recursos, uma vez que qualquer transação terá visibilidade pública e sua autenticidade será inegável. Neste trabalho, apresenta-se um esquema que utiliza contratos inteligentes e a implementação *Open-Source* de *blockchain Hyperledger Fabric* para implementar uma solução ao problema descrito, resultando na criação de um *token* criptográfico.

2. Objetivos

Este trabalho procura criar um protótipo de *token* criptográfico para uma rede *blockchain* para servir de intermédio de pagamento entre instituições e seus beneficiários de modo a dificultar transações ilícitas e desvios de recursos, mas ainda servir ao público eficientemente e naturalmente.

2.1. Objetivos Específicos

- Desenvolver um *token* criptográfico que permita a sua troca entre usuários cadastrados na rede para pagamento de serviços;
- Implementar um sistema que permita a emissão e distribuição dos *tokens* por parte da autoridade responsável;
- Implementar uma interface pela qual os usuários possam interagir com a rede e administrar seus *tokens*;

3. Fundamentação Teórica

3.1. Função de Hash

Função de hash H é uma função que aceita uma entrada M de qualquer tamanho e produz uma saída h de tamanho fixo. Em geral, uma função deste tipo é dita boa caso conjuntos grandes de entradas produzam saídas aparentemente aleatórias [Stallings 2013]. A função é chamada de função de hash criptográfica quando é computacionalmente inviável encontrar:

- a entrada M a partir de uma saída conhecida h Ou
- duas entradas M, M' que possuam a mesma saída $h = h'$

Em razão destas características, funções deste tipo são bastante utilizadas para determinar se algum dado mudou ou não. A imagem 3.1 representa o funcionamento de uma função de *Hash*.

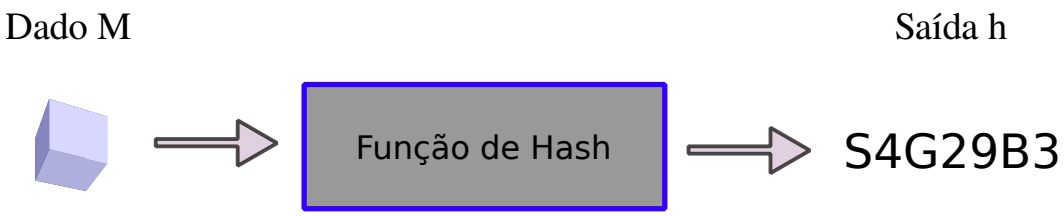


Figura 1. Representação de uma função de hash.

3.2. Blockchain

Blockchain, como o nome sugere, significa uma lista encadeada de blocos. É útil entender *blockchain* como sendo um banco de dados distribuído, ou seja, uma coleção de informações armazenada digitalmente. Ao invés, porém, de uma estrutura centralizada que contém todos os dados e serve pedidos, trata-se de uma lista encadeada e, geralmente, descentralizada com os pedaços de informação, geralmente transações, armazenados e lacrados em blocos. Estes blocos são encadeados utilizando ponteiros *hash* para o bloco anterior da lista. Este tipo ponteiro nada mais é que um par composto por um ponteiro ao bloco, e o *hash* deste bloco. Desta maneira, além do bloco conter um ponteiro para o elemento anterior da lista, este ponteiro também contém o *hash* daquele bloco [Narayanan et al. 2016], conforme a imagem 3.2. Definindo desta maneira, é fácil de perceber que uma *blockchain* é resistente a alterações em qualquer parte, uma vez que ao alterar os dados de qualquer bloco, todos os *hashes* daquele ponto em diante ficarão inconsistentes, começando com o *hash* do próprio bloco alterado.

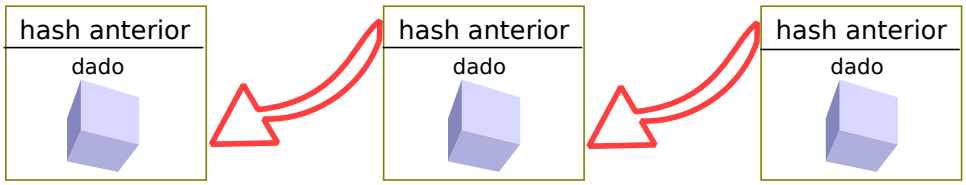


Figura 2. Representação de uma *hash list*.

Em essência, *Blockchain* é composta por de 6 grandes propriedades [Lin and Liao 2017], são eles:

1. **Descentralização**: Talvez a funcionalidade central de qualquer *blockchain* é não ser controlado por uma autoridade central. Uma rede, tipicamente, serve a todos os participantes e realiza as modificações na *blockchain* baseadas em algum algoritmo de consenso.

2. **Transparência**: É comum que as transações que ocorrem dentro de uma rede *blockchain* sejam públicas e, por este motivo, que existe confiança entre as partes. Ou seja, os usuários podendo verificar as transações dentro da rede, percebem que não houve geração espontânea da moeda na rede, ou que alguém está ferindo o protocolo.
3. **Código aberto**: Possuir um repositório público com o código. Outro elemento que toca no quesito confiança, permite diferenciar entre *forks* dentro da mesma moeda.
4. **Autonomia**: Autonomia diz respeito à capacidade de qualquer nodo agir para atualizar a blockchain sem interferência de outros nodos.
5. **Imutabilidade**: Dados escritos na *blockchain* são permanentes. Isso só pode ser mudado por meio de ataques específicos. Por exemplo, na rede do Bitcoin onde, caso de 51% dos nodos colaborarem, é possível alterar o histórico da *chain*.
6. **Anonimidade**: Não existe uma associação direta entre o usuário com seus recursos na *blockchain* e o nome dele, apenas o endereço dele.

3.3. Permissioned vs Permissionless blockchain

Blockchains como a do *Bitcoin* são *permissionless* (não permissivo). Isto significa que não existe uma entidade que concede permissões para novos usuários entrarem na rede e fazerem modificações. A rede é aberta ao público e qualquer um pode interagir com ela. Isto é favorável ao *Bitcoin* pois cria uma descentralização maior e permite com que mais usuários utilizem a rede. Ademais, participantes da rede são livres de censura, por não depender de uma autoridade centralizada decidir sobre a validade ou não de transações.

Por outro lado, redes *Permissioned* (permissivas), ou *blockchains* privadas, são fechadas a um grupo de agentes. Geralmente, estes agentes são previamente determinados, e por vezes, fazem parte de algumas organizações que desejam realizar negócios entre si, mas necessitam da privacidade e segurança provida por uma *blockchain*. Participantes de redes deste tipo abrem mão de anonimidade, necessitando autenticar-se. No caso da proposta atual, este seria o modelo ideal, pois nem todos devem ter acesso ao *token*, visto que nem todas as pessoas devem ter acesso a um restaurante universitário, por exemplo.

Redes privadas também contam com um algoritmo de consenso. Entretanto, algoritmos de redes privadas não utilizam algoritmos de mineração que consomem recursos computacionais, justamente porque há um nível maior de confiança entre os participantes. Usuários maliciosos podem ser facilmente identificados e excluídos da rede, visto que, em caso de uso malicioso dentro da rede, sabe-se exatamente qual usuário fez a ação, e inclusive os dados pessoais dele, em função da transparência e necessidade de autenticação provida pela blockchain privada. Em geral estes algoritmos de consenso são mais rápidos e mais computacionalmente eficientes [Yaga et al. 2019]. Algumas implementações de *blockchain*, como o *hyperledger fabric*, utilizam um serviço de ordenação que garante que todas as mensagens são vistas na mesma ordem por todos os participantes, definindo para a rede, qual a próxima ação a ser minerada em um bloco.

3.4. Token

Tokens podem ser definidos como entidades que representam valor econômico. Originalmente, conchas e miçangas foram as primeiras formas de *token* usadas. Outros exemplos de *tokens* surgiram ao longo da história da humanidade, como por exemplo, fichas de

cassino, vouchers, cartas de baralho, títulos de dívida, cartões de identificação, cartões de fidelidade e bilhetes de loteria. Em maioria, os *tokens* possuem alguma característica que dificulta ou impede a falsificação. Cédulas e moedas também podem ser considerados como exemplos de *tokens* [Voshmgir 2020]. Dentro da computação, *tokens* podem ser utilizados para representar o direito de realizar um conjunto de ações, como é o caso do *Json Web Token* (JWT) que permite um mecanismo de autenticação *stateless* (sem estado) [Jones et al. 2015].

3.4.1. Token criptográfico

Tokens criptográficos são aqueles que existem em uma blockchain. Estes *tokens* representam um conjunto de regras, definidas no contrato que os emite, o contrato de *tokens*. *Tokens* criptográficos são acessíveis apenas por meio da carteira que as possui [Voshmgir 2020].

Os primeiros *tokens* deste tipo foram os nativos às redes blockchain que fazem parte do esquema de incentivo da rede, também chamados de *tokens* de protocolo. Na rede do Bitcoin, por exemplo, o *token* nativo chama-se Bitcoin. Nesta rede, o Bitcoin é criado e transacionado com base em um conjunto de regras cripto-econômicas que determinam as circunstâncias nas quais as transações Bitcoin são válidas e quando novos blocos são criados [Voshmgir 2020].

Com o advento do *Ethereum*, por outro lado, *Tokens* criptográficos passaram a ser criados por meio de contratos que podem ser escritos pelos próprios usuários. Isto levou à criação de uma série de padrões na rede *Ethereum* dispondo sobre as interfaces de diversos tipos de *tokens* capazes de existir na mesma rede do *Ethereum*. Estas interfaces foram definidas por meio de *Ethereum request for comment* (ERC), similarmente aos *Request For Comments* (RFC) criados para padronizar a internet.

Por exemplo, o surgimento do padrão ERC-721 [Entriken et al. 2018] dentro do *Ethereum* que dispõe sobre a criação de *Tokens* Não Fungíveis (NFT), permite que usuários da rede criem com facilidade *tokens* que representem quaisquer tipos de colecionáveis, obras de arte, propriedade digital, entre outros. Para isso, o ERC-721, possui propriedades especiais que permitem com que um *token* da rede seja único. Ou seja pode-se interpretar um *token* criptográfico como sendo uma interface para um objeto num contrato inteligente.

3.5. Contratos Inteligentes

A noção de um contrato não é nova no histórico da humanidade. Desde contratos firmados entre duas potências europeias sobre a delimitação de território sul americano, até um simples contrato de aluguel de imóvel, eles são indispensáveis e hoje são pilar das relações econômicas. Nick Szabo, porém, em 1997 idealizou uma evolução natural do contrato: embutir as clausulas do contrato em software e hardware, de modo a tornar caro ou proibitivamente caro o rompimento delas [Szabo 1997].

O protocolo do *Bitcoin*, segundo o *whitepaper*, permite utilização de *scripts* limitados nas transações. Isto permite a usuários certo nível de expressividade, permitindo algo como disponibilização dos fundos de uma transação, somente após uma certa data.

A linguagem de *script* do Bitcoin é propositalmente limitada, pois um ator malicioso poderia utilizar desta característica para ocupar a rede com processamento intenso. Portanto, não existem laços ou estruturas de controle complexas, exceto o controle de fluxo condicional na linguagem. Isto garante a Turing-incompletude dela, e dificulta a utilização de bombas lógicas [Antonopoulos 2015]. Em 2014, porém, O *Ethereum* foi lançado com suporte a contratos inteligentes Turing completos.

3.6. Ethereum

Ethereum não é simplesmente uma criptomoeda, é uma plataforma distribuída de desenvolvimento, concebida por Vitalik Buterin, cujo intuito é, entre outras coisas, permitir a desenvolvedores criarem aplicações e definirem contratos inteligentes de acordo com suas necessidades de negócio [Buterin et al. 2014]. Utilizando contratos inteligentes do *Ethereum*, é possível, por exemplo, realizar uma venda de produto a prazo, e caso ocorra perda do prazo de pagamento, fazer com que o contrato cobre os juros automaticamente.

Atualmente, o algoritmo de consenso utilizado pela rede é o *Proof of Work* (PoW). Está prevista uma alteração no algoritmo de consenso da rede *Ethereum* no futuro, com vistas de contornar os gastos energéticos envolvidos na mineração. Quando ocorrer a mudança, a rede passará a utilizar o algoritmo Proof of Stake (PoS).

4. Desenvolvimento

Para criar um *token* criptográfico em acordo com a proposta, é necessário escolher uma *blockchain* para hospedar o *token*. Como discutido anteriormente, a rede *Bitcoin* não é adequada, visto que sua linguagem de *scripting* não possui a complexidade necessária para poder expressar as regras de negócio inerentes à proposta. *Ethereum*, por outro lado, possui suporte a contratos inteligentes e é utilizado em larga escala. Porém, *Ethereum* é uma rede *Permissionless*. Como mecionado, estaria fora da proposta permitir com que qualquer usuário tivesse acesso aos *tokens*. Além disso, a rede do *Ethereum* atualmente apresenta taxa de transação muito alta, o que tornaria inviável realizar uma simples transferência entre dois usuários.

Diante disto, optou-se por criar uma *Blockchain*, utilizando o *framework Hyperledger* ***Fabric***. Hyperledger é um framework de código aberto, e foi desenvolvido pela Fundação Linux em 2015. O *Hyperledger* ***Fabric*** é uma plataforma de *Distributed Ledger Technology* que foi desenvolvida pela IBM para uso corporativo. Usar *Hyperledger* ***Fabric*** é bastante conveniente quando se quer criar uma rede com alta modularidade. Isto significa que, caso se queira restringir identidades a certas atividades, ou criar canais de comunicação ocultos entre certos participantes da rede, é conveniente de se fazer utilizando ***Fabric*** [Fabric 2020].

4.1. Arquitetura

O *Hyperledger* ***Fabric*** permite a criação de organizações para compor uma solução e melhor representar o modelo de negócio. Uma organização neste contexto representa uma parte interessada, capaz de acessar canais e emitir identidades para participantes desta organização. Decidiu-se por uma arquitetura de *blockchain* onde estão presentes 3 organizações: o MEC, a UFSC e os estudantes. A ideia por trás disto é permitir com que usuários da organização MEC sejam capazes de emitir *tokens*, enquanto a organização dos

estudantes não, tampouco a UFSC. Além disso, deve estar presente um nodo ordenador, que é responsável por fazer o consenso na blockchain, mas que em nada interfere na implementação. Fora a emissão dos *tokens*, todas as organizações possuem acesso a todas as funcionalidades. A organização que representa a UFSC, não possui utilidade aparente, porém ela pode ser utilizada como intermediária para envio de *tokens*, uma vez que a escolha dos beneficiários é por parte dela. Da forma como foi feito, porém, a UFSC envia ao MEC a lista de beneficiários e o MEC realiza a transferência para eles. Ambas alternativas estão de acordo com a lei e resolvem o problema.

O esquema proposto para o ciclo de vida dos *tokens* pode ser visualizado na figura 4.1. De acordo com ele, o MEC emite um número de *tokens*, e os transfere para os beneficiários cadastrados. Estes, por sua vez, podem utilizar o *token* em serviços aprovados pela UFSC, tais como RU, pagamento de aluguel, e outros. Ao final de cada mês, a UFSC concentrará um número de *tokens* os quais serão enviados de volta para o MEC, fechando o ciclo. Uma vez finalizado o *token*, o MEC envia o dinheiro na mesma quantia que o total em *tokens* recebido pela UFSC naquele mês.

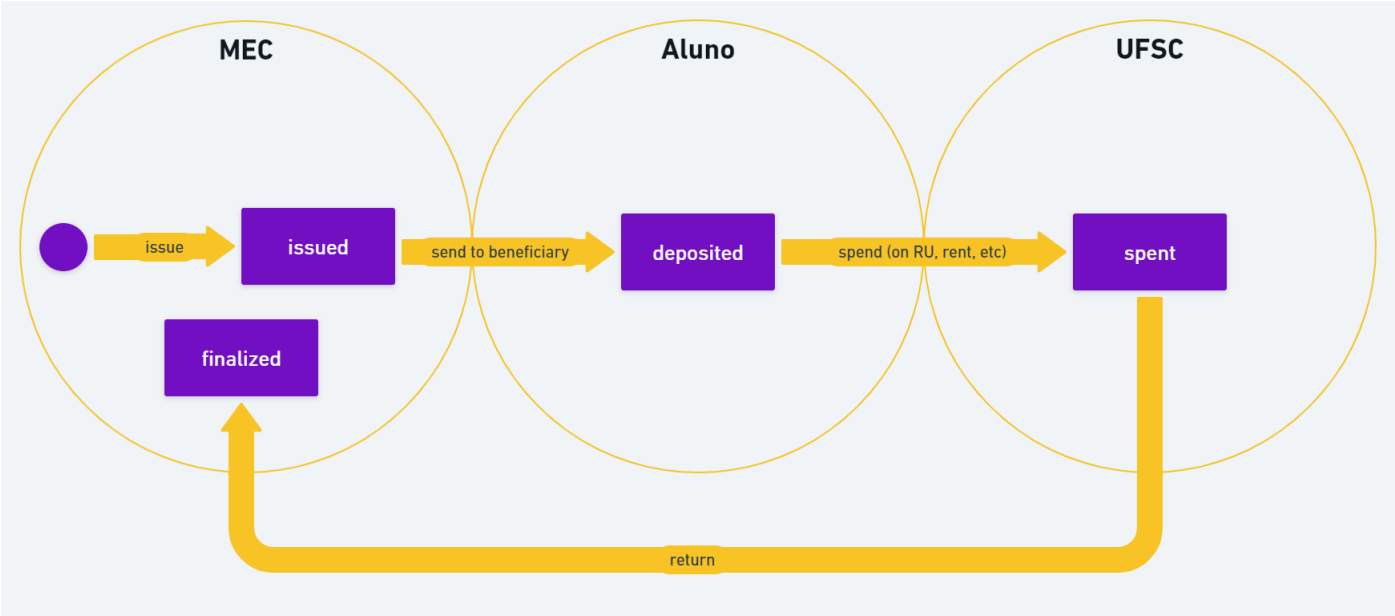


Figura 3. Ciclo de vide de um *token*, como definido no contrato.

5. O contrato inteligente

Os contratos foram escritos em *Typescript*, e depois transpilados [1] para *Javascript*, linguagem para a qual existe uma interface com o *Hyperledger* ***Fabric***. O *token* e o seu funcionamento foram escritos em 5 classes que compõem o mesmo contrato batizado de *mycc*.

Em essência o *token* se comporta como uma criptomoeda, pois representa valor monetário no mundo físico. Por isso, este projeto baseou-se na implementação do *Bitcoin*. Assim como no Bitcoin, os usuários da rede não possuem propriamente uma estrutura

[1] Transpilar significa transformar código fonte de uma linguagem em um código fonte equivalente em outra linguagem de programação.

de carteira onde estão armazenados os ativos [Antonopoulos 2015]. Esta é apenas uma abstração que facilita o entendimento. Na realidade, cada fração de cripto-ativo possui um dono, e portanto, a maneira de se saber a quantidade de ativos que certa identidade possui, é contando a quantidade de ativos não gastos no nome dela na lista de *Unspent Transaction Outputs* (UTXO).

Quando um *token* é gasto, ou seja, quando um usuário envia uma certa quantidade para outro, o contrato não muda o dono dos *tokens* que foram utilizados nesta transação, até porque nem sempre existe um só *token* com exatamente o valor a ser enviado. Em realidade, os *tokens* que foram escolhidos são destruídos, e em seu lugar é criado um novo *token* com o valor que se quer enviar em nome do destinatário, e, caso necessário, cria-se também um *token* com o valor do troco em nome do remetente. Assim como no Bitcoin, a lista de UTXO é pública e pode ser consultada por qualquer participante de qualquer organização na rede. O que impede usuários de consultarem os fundos de outros usuários é o *back-end*.

O contrato desenvolvido também permite ao Ministério da Educação (MEC), e somente o MEC, emitir os *tokens* à vontade. Isto é garantido, pois a função de emissão do contrato checa se a identidade que iniciou a transação pertence à organização MEC. *Tokens* criados desta forma entram na carteira do MEC, a partir do qual podem ser enviados para os beneficiários utilizando a função de transação.

6. Aplicação web

Durante o desenvolvimento, a interação com o contrato inteligente ocorreu por meio de *Command-line Interface* (CLI). Tendo em vista a principal função da moeda ser o pagamento de serviços, optou-se por uma interface mais parecida com um banco. Para implementar esta parte do trabalho, utilizou-se a biblioteca *express* para o *Back-end* e *NEXTJS* para o *Front-end*, ambos em *typescript*. Um *back-end* ou servidor, é necessário pois cada usuário dentro da rede precisa criar uma carteira em disco, contendo a chave privada e a organização à qual pertence. Não é possível realizar alterações no sistema de arquivos apenas com a interface.

As visões de usuário contam com design responsivo, ou seja, são compatíveis com diferentes tipos de dispositivos, portanto um usuário poderia utilizar a interface web tanto no seu celular, quanto em seu computador pessoal. Foram implementadas 6 visões de usuário, com uma delas, a de emissão de *tokens*, visível apenas pelo MEC. Para se referir ao *token* implementado, escolheu-se o nome Cripto RU (CRU), ou seja, usuários receberão *tokens* CRU como pagamento de auxílios.

6.1. Visões

Aqui são mostradas algumas capturas da visão do usuário, bem como do administrador, criadas para atender à proposta. Ao se autenticar com sucesso, a página principal apresenta para o usuário seu saldo de *tokens*, e as cinco ações: Logout, Atualizar Cadastro, Transferir, emitir e Ver Histórico conforme mostra a captura à direita na imagem 7.2. Caso o usuário não for administrador (MEC), a interface não mostrará a opção de emitir *tokens*, como na imagem 7.2. Por padrão, o contrato já inicializa a identidade do MEC possuindo 1000000 de *tokens*. A outra visão apresentada é a de histórico, permitindo ao usuário verificar remetente, destinatário, data e montante de todas as transações que ele fez ou já recebeu, conforme a imagem 7.2.

7. Considerações Finais e Trabalhos Futuros

7.1. Considerações Finais

A proposta inicial de conceber um *token* criptográfico e uma interface que permita sua manipulação foi desenvolvida com sucesso. Vale a pena discutir as consequências da maneira como foi desenvolvida a solução e compará-la com as soluções utilizadas por instituições públicas atualmente.

Existem alguns benefícios de se ter um esquema assim, ou seja um blockchain e não simplesmente um banco de dados centralizado na mão do governo. A vantagem principal é permitir a implantação de contratos. Isto é, é possível descrever situações altamente complexas que reflitam a realidade e fazer com que a rede realize toda a operação por meio do contrato. Isto é bastante interessante para o sistema, pois assegura maior transparência e segurança aos participantes. Por exemplo, um dos auxílios atualmente pagos pela UFSC é o auxilio moradia. O aluno recebe o dinheiro do auxílio diretamente na conta dele, mediante comprovação por meio do contrato. Utilizando a solução desenvolvida, é possível com que as regras e cláusulas deste contrato físico sejam modeladas no contrato inteligente. Isto faria com que o dinheiro fosse automaticamente cobrado da carteira dele, bem como quaisquer taxas ou compensações que sejam aplicáveis dependendo de acontecimentos futuros. Todo o processo de aprovação do benefício e pagamento ocorreria com bem menos atrito, além de reduzir o risco de haver erros.

A próxima vantagem é a imutabilidade e rastreabilidade simplificada dos recursos. Não é simples de esconder das autoridades uma transação que crie os recursos no nome de um intruso, até porque a rede não vai permitir. Para poder fazer algo neste estilo, seria necessário atualizar o contrato com código malicioso. Além disso, os usuários, ao utilizarem a rede para transacionar no dia a dia, naturalmente vão indicar mau-uso de recursos, uma vez que atores malignos na rede concentrarão grande parte dos recursos, permitindo fácil identificação. Atualmente, para congelar fundos sob mau uso por parte de estudantes, é necessário passar por todo um processo investigativo jurídico que é lento. Por outro lado, pelo menos tecnicamente, é simples para as autoridades comandar a invalidez dos *tokens* assim que detectar mau-uso, situação que não se verifica com passes físicos ou dinheiro vivo, ou até mesmo dinheiro em conta bancária.

7.2. Trabalhos Futuros

Este trabalho pode servir de base para realizar alguns outros projetos no futuro, como por exemplo:

- Unificar com o sistema de transporte coletivo, permitindo com que usuários utilizem seus *tokens* para comprar passagens de ônibus.
- Integração com o sistema de controle de acesso do RU, permitindo com que usuários utilizem as suas carteirinhas para pagar com os seus *tokens* na entrada do restaurante.
- Ampliação/Adaptação das funcionalidades para incluir outras modalidades de bolsa e/ou outras instituições de ensino superior. Este trabalho teve como principal objetivo atender ao PNAES, porém, outros tipos de assistência, inclusive providenciados pela UFSC, poderiam também ser contemplados pelo esquema proposto.

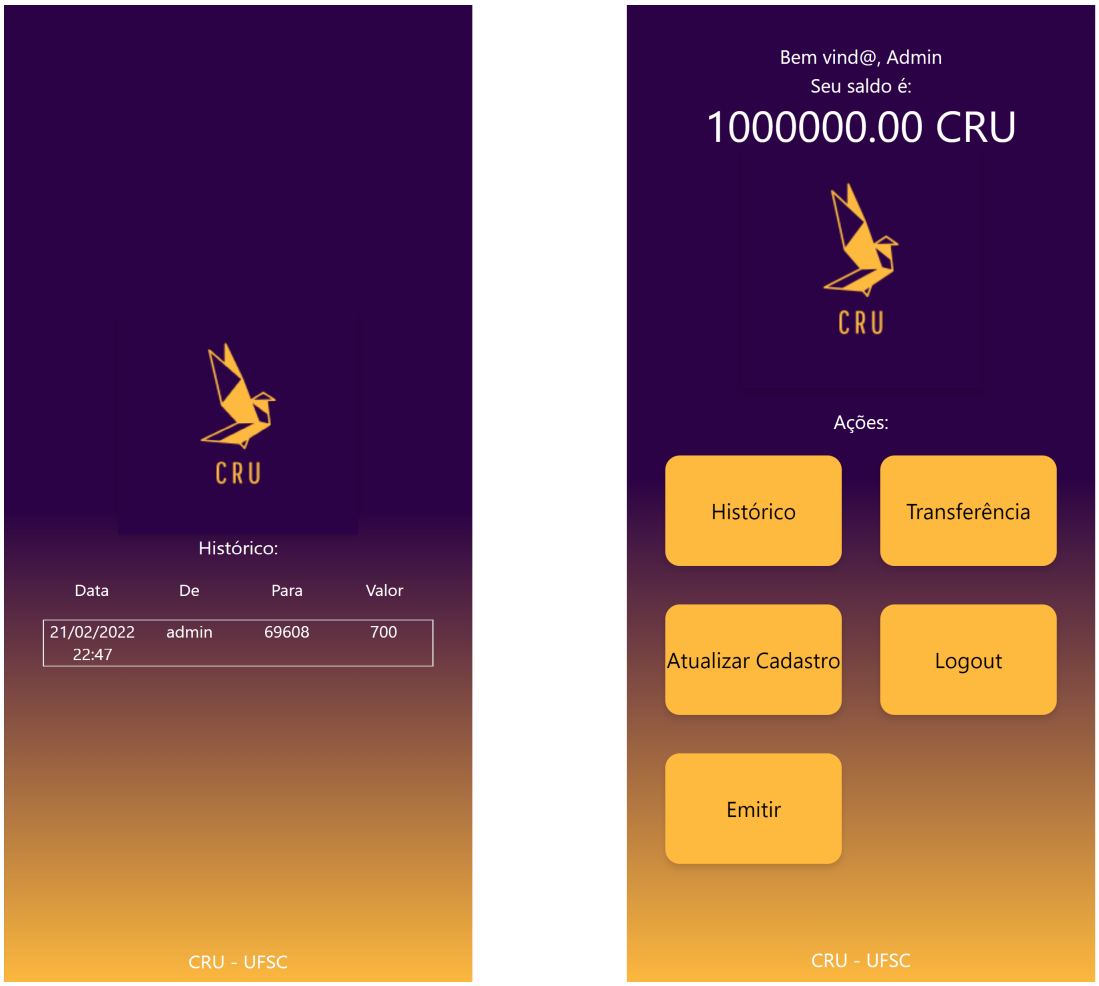


Figura 4. À esquerda, visão do histórico. À direita, página inicial do aplicativo – tela de administrador.

Referências

Antonopoulos, A. M. (2015). *Mastering Bitcoin*.

Brasil (2010). Decreto nº 7.234, de 19 de julho de 2010. dispõe sobre o programa nacional de assistência estudantil - pnaes. *Diário Oficial [da] República Federativa do Brasil*.

Buterin, V. et al. (2014). A next-generation smart contract and decentralized application platform.

Entriken, W., Shirley, D., Evans, J., and Sachs, N. (2018). Eip-721: Non-fungible token standard. EIP 721.

Fabric, H. (2020). Hyperledger fabric white paper.

Jones, M., Bradley, J., and Sakimura, N. (2015). Json web token (jwt). RFC 7519, RFC Editor. `http://www.rfc-editor.org/rfc/rfc7519.txt`.

Lin, I.-C. and Liao, T.-C. (2017). A survey of blockchain security issues and challenges. *Int. J. Netw. Secur.*, 19(5):653–659.

Macedo, F. (2016). Pf investiga fraudes até no bandejão da universidade de viçosa.

Mirelle Pinheiro, C. C. (2022). Pf investiga professor pelo desvio de r$ 3,7 milhões de universidade.

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

Narayanan, A., Bonneau, J., Felten, E., Miller, A., and Goldfeder, S. (2016). *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press, USA.

Stallings, W. (2013). *Cryptography and Network Security: Principles and Practice*. Prentice Hall Press, USA, 6th edition.

Szabo, N. (1997). Formalizing and securing relationships on public networks. *First monday*.

Voshmgir, S. (2020). *Token Economy: How the Web3 reinvents the Internet*. Shermin Voshmgir.

Yaga, D., Mell, P., Roby, N., and Scarfone, K. (2019). Blockchain technology overview. *arXiv preprint arXiv:1906.11078*.