DAVID BARBARA ENGELSTEIN
GABRIEL ZUCCHI DE SOUZA
HENRIQUE CORRÊA VASCONCELLOS

MOEDA USP: FOMENTANDO COLABORAÇÃO ENTRE STARTUPS E ACADEMIA POR MEIO DA TOKENIZAÇÃO DE ATIVOS

São Paulo
2021

DAVID BARBARA ENGELSTEIN
GABRIEL ZUCCHI DE SOUZA
HENRIQUE CORRÊA VASCONCELLOS

MOEDA USP: FOMENTANDO COLABORAÇÃO ENTRE STARTUPS E ACADEMIA POR MEIO DA TOKENIZAÇÃO DE ATIVOS

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Elétrico com Ênfase em Computação.

São Paulo
2021

DAVID BARBARA ENGELSTEIN
GABRIEL ZUCCHI DE SOUZA
HENRIQUE CORRÊA VASCONCELLOS

MOEDA USP: FOMENTANDO COLABORAÇÃO ENTRE STARTUPS E ACADEMIA POR MEIO DA TOKENIZAÇÃO DE ATIVOS

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Elétrico com Ênfase em Computação.

Área de Concentração:
Desenvolvimento de Sistemas

Orientador:
Marcos Antonio Simplicio Junior

São Paulo
2021

RESUMO

Nos últimos anos, o papel das *startups* vem crescendo nos mais diversos setores da sociedade, como economia e avanços científicos, fazendo com que a Universidade de São Paulo iniciasse diversos programas de incentivo a parcerias com estas empresas. Contudo, muitas vezes os serviços que a USP tem a oferecer não podem ser adquiridos pelas *startups* por estas ainda não possuírem os recursos necessários para arcarem com os custos. Neste cenário, surgiu por iniciativa do instituto InovaUSP um projeto de criação de uma moeda virtual, chamada de Moeda USP, a ser utilizada dentro do ambiente da Universidade, que poderia ser adquirida pelas *startups* em troca de parte de seu capital e utilizada para contratar os serviços almejados. A moeda poderia ainda ser utilizada como meio de troca entre laboratórios e para o pagamento de bolsistas. Para servir de prova conceito deste sistema, o projeto aqui apresentado se propôs a desenvolver um protótipo que implementa uma rede blockchain federada, que utiliza o protocolo Ethereum com o algoritmo de consenso *Proof of Authority*; nesta rede foi emitido um *smart contract* que define um token representativo da moeda virtual em questão; também foi desenvolvida uma aplicação Web a ser utilizada pelos usuários e administradores do sistema, permitindo o controle e transacionamento dos tokens de maneira segura. Como resultado, foi obtida com sucesso uma prova de conceito que atesta a viabilidade técnica deste projeto, que pode vir a ser implementado pela equipe de tecnologia da USP em parceria com a equipe econômica.

Palavras-Chave – Blockchain, Tokenização, Moeda Virtual, Ethereum, Startup, Smart Contract, Segurança, Chave Pública, Chave privada, Transação.

ABSTRACT

In the past years, the role of startups has grown across many sectors of society, such as economics and scientific advances, causing the University of Sao Paulo to initiate several programs to encourage partnerships with these companies. However, the services that USP has to offer are often impossible to be acquired by startups because they do not yet have the necessary resources to cover the costs. In this scenario, the *InovaUSP* Institute came up with a project to create a virtual currency, called *Moeda USP*, to be used within the University's environment, which could be acquired by startups in exchange for part of their capital and used to hire the desired services. The currency could also be used as a means of exchange between laboratories and for the payment of scholarship holders. To serve as a proof of concept for this system, the project presented in this document had the purpose to develop a prototype that implements a federated blockchain, using the Ethereum protocol and the Proof of Authority consensus algorithm; a smart contract that defines a token, which represents the virtual currency, was deployed in this network; it was also developed a web application to be used by users and system administrators, in order to allow the management and transaction of tokens in a secure manner. As a result, a proof of concept was successfully obtained that attests to the technical feasibility of this project, which may be implemented by the USP technology team in partnership with the economic team.

Keywords – Blockchain, Tokenization, Virtual Currency, Ethereum, Startup, Smart Contract, Security, Public Key, Private Key

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

API Application Programming Interface - Interface de programação de aplicações. 29

AWS Amazon Web Service. 33

CLI Command-line Interface - Interface de linha de comando. 31

CNPJ Cadastro Nacional de Pessoa Jurídica. 45

CSPRNG Cryptographically secure pseudorandom number generator - Gerador de Número Pseudo-Aleatório Criptograficamente Seguro. 38

EIP Ethereum Improvement Proposal. 34

ERC Ethereum Request for Comment. 30

EVM Ethereum Virtual Machine - Máquina Virtual Ethereum. 16

FEA-USP Faculdade de Economia e Administração da Universidade de São Paulo. 11

Geth Go Ethereum. 17

HTTP Hypertext Transfer Protocol. 28

IDE Integrated Development Environment - Ambiente de Desenvolvimento Integrado. 31

InovaUSP Centro de Pesquisa e Inovação. 10

IPC Inter-Process Communication - Comunicação entre processos. 30

JWT JSON Web Token. 39

LARC Laboratório de Arquitetura e Redes de Computadores. 33

NAS Nothing-at-Stake. 19

ORM Object–relational mapping - Mapeamento objeto-relacional. 31

PoA Proof-of-Authority. 20

PoS Proof-of-Stake. 18

PoW Proof-of-Work. 18

RAM Random Access Memory - Memória de acesso aleatório. 33

RDS Relational database service - Serviço de banco de dados relacional. 33

SQL Structured Query Language - Linguagem de Consulta Estruturada. 31

TSA Timestamp Authority - Autoridade de carimbo de tempo. 15

URL Uniform Resource Locator - localizador uniforme de recursos. 53

USP Universidade de São Paulo. 10

XSS Cross Site Scripting. 39

SUMÁRIO

1 INTRODUÇÃO

O presente projeto insere-se no contexto de inovação promovida por parcerias entre a iniciativa privada e instituições de pesquisa como a Universidade de São Paulo (USP). Especificamente, a ideia principal do projeto veio da intenção de aproximar a USP do mundo das *startups*, facilitando o caminho para que estas possam tirar proveito da expertise acumulada na universidade para a construção de produtos e serviços inovadores, melhorando sua competitividade. Este capítulo detalha esse cenário e os objetivos que se pretende alcançar com este projeto, bem como a forma planejada de fazê-lo.

1.1 Motivação

Atualmente, as startups desempenham um papel crucial na economia brasileira, promovendo inovações, gerando riquezas e criando empregos em diversos setores da economia. Atualmente, o Brasil possui 13.581 startups [1]. De acordo com o estudo "Inside Venture Capital Brasil", realizado pelo hub de inovação Distrito, o setor no ano de 2020 captou cerca de $ 3,5 bilhões em 469 rodadas de investimento, 337 deles em empresas com estágios iniciais de crescimento (anjo, pré-seed e seed). Apesar de corresponderem a mais de 70% das rodadas, as empresas em estágios iniciais correspondem a menos de 10% do valor total de $3,5 bilhões investidos no setor (Séries de A a G e Private Equity correspondem a cerca de $3,2 bilhões) [2]. Em meio a este cenário, nos últimos anos a Universidade de São Paulo tem apoiado startups que buscam melhorar sua competitividade por meio de parcerias com a comunidade de pesquisa, utilizando tanto a infraestrutura da universidade como seus recursos humanos especializados e laboratórios. Isso comumente é feito por intermédio do Centro de Pesquisa e Inovação (InovaUSP) [3].

Apesar desse tipo de colaboração ter sua importância reconhecida tanto por empresas como pela própria universidade, nem sempre é fácil viabilizá-la. Uma das principais razões é que, devido ao ciclo natural de desenvolvimento de empresas, frequentemente *startups* não possuem capital suficiente para arcar com os custos de uso de laboratórios

da Universidade e outros serviços, criando-se um obstáculo para que a Universidade possa contribuir ativamente no desenvolvimento das mesmas. Remover este obstáculo é, portanto, essencial para melhorar a quantidade e a qualidade dessas parcerias. Uma forma de fazê-lo consiste em tirar proveito de um modelo econômico que tem ganhado bastante visibilidade e relevância atualmente: o conceito de economia decentralizada e tokenização de ativos, que faz uso intenso de tecnologias como Blockchains e contratos inteligentes, garantindo, assim, confiabilidade, auditabilidade e transparência no sistema.

1.2 Objetivo

O objetivo deste projeto é a criação de uma moeda digital, denominada simplesmente "Moeda USP", para uso dentro do ambiente controlado da Universidade de São Paulo. Esta moeda pode ser adquirida por uma determinada startup em troca de *equity*, i.e., participação no patrimônio da empresa. Uma vez adquiridas, as moedas podem ser usadas para acesso aos serviços USP almejados: uso de espaço de laboratório para ensaios e experimentos, pagamento de insumos eventualmente utilizados (e.g., em análises químicas), remuneração de alunos bolsistas cujo tema de pesquisa seja de interesse para a empresa, etc. Outra utilização, uma vez que a moeda ingresse no ambiente interno da USP após pagamento realizado pela empresa, consiste em promover liquidez às próprias atividades internas da universidade. Por exemplo, essas moedas podem ser utilizadas na troca de experiências e serviços entre laboratórios de forma mais organizada, em vez de continuar sendo feita de modo informal, por meio de escambo de recursos humanos ou insumos.

O projeto em si é multidisciplinar, e está sendo desenvolvido em uma parceria entre a Escola Politécnica, o Centro de Pesquisa e Inovação (InovaUSP) e a Faculdade de Economia e Administração da USP (FEA-USP).

A razão para isso é que o projeto possui questões econômicas a serem tratadas, como a garantia de liquidez por parte da Universidade e riscos cambiais, e também desafios do ponto de vista da tecnologia. São estes últimos que serão abordados neste projeto de formatura, incluindo o projeto arquitetural e a construção de um protótipo funcional para guiar a eventual implantação da solução dentro do ambiente da USP. Como proposta base, é considerado o uso de plataformas de Blockchain e contratos inteligentes, tecnologias comumente utilizadas para habilitar a construção de sistemas de segurança para a troca de ativos digitais (no caso, Moedas USP).

1.3 Justificativa

A própria origem do conceito de moeda remonta à necessidade de facilitar a troca de ativos tangíveis e serviços entre diferentes entidades. Assim, justifica-se a proposta de construção de uma moeda digital para sanar esse tipo de dificuldade dentro do cenário de colaboração entre a iniciativa privada, em busca de prover produtos e serviços inovadores, e instituições de pesquisa como a USP, capaz de auxiliar nessa tarefa. Adicionalmente, a forma como o sistema está sendo elaborado promove ao menos dois tipos de benefícios para a USP em si, sempre que as startups em questão tiverem sucesso: um deles refere-se ao retorno financeiro para as atividades da universidade, incluindo manutenção de infraestrutura e pagamento de bolsas de pesquisa e permanência a alunos, em razão da participação da universidade na empresa; o outro refere-se ao potencial de se reforçar positivamente a imagem externa da USP na promoção, desenvolvimento e implantação de novas tecnologias para o benefício da sociedade.

Já o uso de Blockchain e contratos inteligentes como tecnologias base da proposta justifica-se pela efetividade dessas tecnologias para a construção de sistemas financeiros decentralizados. Isso pode ser observado pelo crescimento acelerado do uso dessas tecnologias para a construção de criptomoedas no mundo todo [4]. Em parte, essa tendência está relacionada com o fato de Blockchains terem por objetivo a descentralização como medida de segurança, promovendo transparência e auditabilidade dos sistemas resultantes [5]. É por esta razão que em alguns meios a tecnologia de blockchain é denominada "Protocolo de Confiança"[6].

1.4 Trabalhos relacionados

Já foram implementadas outras soluções de tokenização de ativos para uso interno em universidades. Morey [7] projetaram um sistema de token a ser implementado na Universidade Nacional de Taiwan de Ciência e Tecnologia, criando uma micro-economia de recompensas para reportar problemas de mal funcionamento de equipamentos, dentro de um contexto de *smart-campus*. Já a empresa *Student Coin* [8] se propõe a fornecer uma plataforma de criação de *tokens* para uso em universidades, apresentando também soluções de finanças decentralizadas. A diferença do trabalho aqui apresentado, entretanto, é a integração com o ambiente de desenvolvimento de *startups* na Universidade de São Paulo, assim como o lastro do valor da moeda no *equity* das empresas.

1.5 Método

Considerando as características do cenário alvo, composto por um conjunto dinâmico mas bem definido de entidades participantes no sistema, é interessante considerar a construção de uma arquitetura de segurança utilizando uma rede blockchain federada que permita a emissão de tokens e a realização de operações com a Moeda USP.

Dada a complexidade do sistema em questão, são exploradas algumas plataformas de Blockchain que possam servir como base para a construção do sistema, em oposição a uma abordagem na qual construir tais plataformas seria também parte do escopo do projeto. A despeito disso, todas as etapas envolvidas no desenvolvimento de sistemas devem ser exploradas, incluindo: modelar os atores do sistema e suas funções; definir requisitos funcionais e não funcionais; elaborar uma arquitetura capaz de satisfazer esses requisitos; implementar o sistema, usando técnicas de desenvolvimento ágil; testar o sistema resultante e avaliar seu desempenho e eficácia em prover os serviços necessários. Um aspecto de particular importância nesse contexto consiste em garantir, no maior grau possível, a segurança das informações que ficarão armazenadas ou que transitarão pelo sistema elaborado.

1.6 Organização do Trabalho

O restante deste trabalho está dividido nos seguintes capítulos:

- Capítulo 2 (Aspectos Conceituais): Introdução aos principais conceitos abordados neste trabalho;
- Capítulo 3 (Método de Trabalho): Método a ser empregado no desenvolvimento deste trabalho, incluindo as etapas planejadas;
- Capítulo 4 (Especificações e Requisitos do Sistema): Especificação estrutural do sistema e os requisitos funcionais e não funcionais que o sistema deve atender;
- Capítulo 5 (Projeto e Implementação): Resultados do projeto e da implementação do sistema;
- Capítulo 6 (Testes e Avaliação): Procedimento e resultados de testes de desempenho do sistema;

- Capítulo 7 (Considerações finais): Considerações sobre o desenvolvimento do projeto, dificuldades encontradas e informações para futuros projetos de continuidade; conclusões a respeito do projeto quanto ao sucesso do protótipo e viabilidade de futura implementação.

2 ASPECTOS CONCEITUAIS

Este capítulo apresenta brevemente os principais conceitos necessários para o entendimento do sistema proposto, considerando tanto as tecnologias subjacentes como princípios de desenvolvimento de sistemas.

2.1 Blockchain

Uma Blockchain pode ser definida como uma estrutura de dados na qual blocos contendo informações relevantes para o sistema (e.g., transações) formam uma cadeia cronológica, de modo que cada bloco é ligado criptograficamente ao bloco anterior [9]. Quando combinado com um mecanismo de consenso, por meio do qual todos os nós da rede convergem para uma visão única dessa ordem cronológica, essa estrutura de dados pode ser utilizada para implementar uma autoridade de carimbo de tempo (Timestamp Authority - TSA) descentralizada [10].

Desta forma, processos como aplicações e transações que normalmente fariam uso de uma arquitetura centralizada, ou dependeriam da verificação de transações por terceiros confiáveis (e.g., instituições bancárias), podem operar de forma descentralizada com um grau similar de confiança sobre a validade das transações realizadas. Desta forma, blockchains promovem a construção de sistemas distribuídos com características bastante interessantes, como transparência, robustez, auditabilidade e segurança [11, 12, 5]

Estas características têm levado a uma relevância crescente de sistemas baseados em blockchain [13, 5], os quais têm sido adotados por empresas diversas [14, 5], em diferentes campos de aplicação, como finanças, logística e até mesmo na indústria biomédica [15].

Existem essencialmente dois tipos de redes de blockchain, que são empregadas de acordo com o domínio das aplicações. Como define Ferdous et al. [16]:

- **Blockchain pública:** também conhecidas como blockchain não permissionada, permite que qualquer um participe da rede e possa validar e criar blocos novos,

assim como modificar o estado da rede armazenando e atualizando dados através de transações entre participantes;

- **Blockchain privada:** também conhecida como blockchain permissionada ou federada, é mais restritiva em comparação com uma blockchain pública, permitindo somente que entidades autorizadas e consideradas confiáveis possam participar das atividades na blockchain.

2.2 Ethereum Blockchain

O Ethereum, solução adotada como base para a construção da Moeda USP aqui apresentada, é um sistema baseado em blockchain que pode ser visto como uma máquina de estados baseada em transações [17]. Segundo *Ethereum Development Documentation* [18], o sistema abstrai a existência de um único computador (chamado de Ethereum Virtual Machine, EVM) cujo estado é conhecido por todos os nós na rede, e com o qual todos esses nós concordam. Todos os participantes da rede mantêm uma cópia do estado desse computador. Qualquer participante pode enviar uma requisição em forma de broadcast a esse computador para realizar uma tarefa de computação arbitrária, sendo esta requisição denominada uma transação (*transaction*). A cada requisição, todos os demais participantes da rede devem verificar, validar e executar a computação. Este processo causa uma mudança no estado da EVM, que é registrado e propagado pela rede. Todas as transações e o estado atual da EVM são então salvos na blockchain, criando-se uma cadeia cronológica dos eventos no sistema.

A rede Ethereum possui o Ether como sua principal criptomoeda. O Ether existe para permitir um "mercado de computação", que provê um incentivo econômico para participantes verificarem e executarem transações e fornecerem recursos computacionais para a rede.

2.2.1 Contas

Contas na rede Ethereum são entidades que possuem balanço de Ether e são capazes de enviar transações na rede [18]. Contas podem ser controladas por usuários ou implantadas de forma autônoma, por meio de *smart contracts*.

2.2.2 Smart Contracts

Smart contracts, ou contratos inteligentes, podem ser definidos como trechos de software que regulam a troca de recursos entre participantes da rede [19]. Eles utilizam protocolos e interface com usuários para formalizar e assegurar relações entre computadores, fazendo uso de criptografia e outros mecanismos de segurança, como explica Szabo [20].

No contexto da rede Ethereum, *smart contracts* são programas simples que são executados na rede [18], compostos por códigos (suas funções) e dados (seus estados) que residem em um endereço específico da blockchain. São vistos pela rede como um tipo de conta, possuindo assim balanço de Ether e podem enviar transações pela rede. Entretanto, não são controlados por usuários, mas são implantados na rede e executados como programados. Contas de usuários podem então interagir com os mesmos, submetendo transações que executam funções definidas no *smart contract*.

2.2.3 Clientes

Clientes de Ethereum são aplicações que "rodam" um nó na rede, verificando todas as transações em cada bloco, mantendo a rede segura e os dados corretos. Os clientes da rede costumam ser disponibilizados na forma de código aberto, e são mantidos por múltiplos grupos de desenvolvimento, gerando grande diversidade com o intuito de reduzir pontos únicos de falha. [18] Dois dos principais clientes existentes são Geth e OpenEthereum. Neste projeto é utilizado o cliente Geth, devido à maior facilidade de uso observada durante o desenvolvimento do projeto.

2.3 Consenso na rede

Segundo Ferdous et al. [16], algoritmos de consenso garantem concordância entre os nós distribuídos da rede sobre o estado de certo conjunto de dados. O algoritmo de consenso é o componente central em sistemas baseados em blockchain, pois influencia de forma crítica como um sistema se comporta e o desempenho que ele alcança.

Os algoritmos de consenso usados pelos diversos sistemas de blockchain podem ser classificados baseado em seu mecanismo de recompensa [16]. Assim, podem ser definidas duas grandes categorias: Consenso Incentivado e Consenso Não Incentivado. Algoritmos de Consenso Incentivados recompensam os nós por criarem e adicionarem blocos à block-

chain. Estes são utilizados principalmente em redes blockchain públicas e a recompensa serve como uma forma de incentivar os nós participantes a se comportarem de acordo com as regras estabelecidas pelo algoritmo rigorosamente. Já algoritmos de Consenso Não Incentivados não possuem qualquer mecanismo de recompensa para os nós participantes criarem e adicionarem novos blocos. Assim, são utilizados em redes privadas, nas quais os nós são considerados confiáveis (e, potencialmente, podem ser punidos caso haja desvios de comportamento). Nesse cenário, somente os autorizados podem participar do processo de criação de blocos do sistema.

2.3.1 Proof-of-Work

Algoritmos de consenso do tipo Proof-of-Work (PoW) estão entre os mais utilizados atualmente [16]. Esse é o tipo de algoritmo incentivado utilizado pelo Bitcoin [10] e pelo Ethereum [13], por exemplo. Esses algoritmos possuem duas partes, o provador e o verificador. O provador realiza tarefas computacionais envolvendo uso intensivo de recursos para calcular a resposta a um desafio definido pelo sistema; esse desafio pode ser, por exemplo, criptográfico, como o cálculo de uma pré-imagem parcial de uma função de hash [10].

O valor da resposta ao desafio é então validado pelo verificador, tarefa computacionalmente menos exigente. A assimetria de recursos exigidos entre o cálculo da resposta ao desafio e a verificação de sua validade age no sentido de coibir abusos do sistema, como a geração de diversas respostas falsas com o objetivo de consumir recursos dos verificadores [16].

O incentivo para que o provador invista recursos computacionais na tentativa de responder ao desafio pode variar, mas em geral envolve algum tipo de recompensa financeira. No algoritmo do Bitcoin, por exemplo, o provador (também chamado de minerador) é recompensado com um número de Bitcoins que é gerado pelo sistema a cada bloco válido inserido com sucesso na blockchain.

2.3.2 Proof-of-Stake

Os algoritmos de consenso do Proof-of-Stake (PoS) foram propostos como uma alternativa também incentivada ao PoW. A ideia principal dos algoritmos PoS é que os participantes que desejam criar novos blocos devem primeiramente provar que possuem uma certa quantidade de moedas no sistema, e então “travar” uma parte de suas moedas

(o *stake*) em uma conta de garantia. Um nó é sorteado para validar o próximo nó, e essas moedas servem para garantir que o nó siga as regras do protocolo, pois não fazê-lo pode levar à perda das moedas dadas como garantia. Por outro lado, caso se comporte adequadamente, o nó é premiado com uma quantidade de novas moedas ou com taxas pagas pelos demais usuários em suas transações, obtendo lucro sobre as moedas investidas como em garantia.

Em princípio, esse incentivo, junto com mecanismos de punição para o caso de não cumprimento das regras, podem fornecer um nível de segurança similar ao PoW. Adicionalmente, existem outras vantagens como o menor consumo energético derivado da não necessidade da realização de cálculos dos desafios criptográficos. [16]

Há, entretanto, algumas desvantagens envolvidas na implementação desse tipo de algoritmo de consenso. Essa classe de algoritmo está sujeita a alguns diferentes tipos de ataque devido a forma como é garantida a validação do bloco, como descreve Ferdous et al. [16]. Um ataque conhecido é denominado *Nothing-at-Stake* (NAS). Neste tipo de ataque o atacante se aproveita de momentos de *fork* da rede (quando a rede apresenta divergências entre duas "ramificações", isto é, uma parte da rede acredita que uma certa cadeia é a válida e outra acredita que é uma cadeia diferente, demorando um tempo até que as duas partes entrem em consenso) para inserir um bloco recém criado em todas as ramificações, assim aumentando a probabilidade de seu bloco ser aceito como o bloco válido. Esse tipo de ataque é menos provável em uma rede PoW, pois o minerador teria que compartilhar seus recursos de computação em múltiplas ramificações, diminuindo a probabilidade de sucesso na realização do desafio criptográfico. Em uma rede PoS, entretanto, não há custo para um nó adicionar blocos em ramificações paralelas. Este tipo de ataque pode servir de base para outros ataques, como os ataques de *Bribing* (suborno), ou *short-range* (curta distância), que consiste em tentar gastar duplamente a moeda (*double spending*) criando ramificações da rede. Primeiramente o atacante realiza um gasto seguindo o fluxo "correto", isso é, faz a transferência, a contraparte do acordo espera um determinado número de blocos e então entrega o bem adquirido. Após receber o bem, o atacante cria uma ramificação da rede um bloco antes do pagamento (por isso se chama de "curta distância", pois são poucos blocos) e suborna outros nós para aceitar esta nova ramificação. O ataque pode valer a pena enquanto o valor do bem adquirido for maior que o valor gasto com o suborno, e para os que aceitaram o suborno, se realizarem o ataque do tipo NAS simultaneamente, não têm nada a perder, somente a ganhar com o valor do suborno. Esses tipos de ataques devem ser mitigados através da penalização dos nós que realizam essas ações.

2.3.3 Proof-of-Authority

Os algoritmos de consenso Proof-of-Authority (PoA) são do tipo não incentivados, utilizados principalmente em redes privadas. Baseado na reputação dos usuários, os nós da rede fornecem como garantia sua identidade, e não suas moedas. Isso é possível assumindo que os nós que participam do consenso possuem certificados digitais que atestam a sua identidade (e.g., emitidos por uma autoridade certificadora considerada confiável por todos os nós do sistema).

No caso de atividades maliciosas de um nó validador, é possível rastrear seu dono e puni-lo de acordo, seja internamente ao sistema (e.g., por meio do confisco de suas moedas) ou fora dele (e.g., via processos judiciais). Esses nós validadores são denominados autoridades, e possuem um identificador único na rede. Assumindo que a maioria da rede seja honesta, pode-se fazer um simples revezamento entre as autoridades para decidir quem é o líder em cada rodada, i.e., quem irá determinar o bloco a ser inserido na blockchain durante aquela sua rodada. Para maior robustez, o sistema pode incluir mecanismos de verificação de disponibilidade, com trocas de mensagens periódicas para evitar que a falha em um nó eleito como líder interrompa a validação de nós por um longo período de tempo.

Contanto que o número de nós validadores seja limitado a um número reduzido, sistemas baseados em PoA costumam ser altamente escaláveis, i.e., podem abarcar um número muito grande de nós não-validadores enviando transações para validação por essa rede de tamanho reduzido. [21]

O algoritmo de consenso Proof-of-Authority foi o escolhido para ser implementado neste projeto. A escolha foi feita levando em consideração os benefícios apresentados em uma rede federada, uma vez que os nós são conhecidos, atendendo também requisitos não-funcionais relativos ao gasto energético, melhor explicado na seção 4.3.4. Além disso, há também a possibilidade de ser implementado com facilidade em uma rede Ethereum, como apresentado na seção 4.4.10.

2.4 Orientação a Objetos e princípios SOLID

Para permitir uma boa modularidade do sistema, decidiu-se pela sua construção utilizando o paradigma de Orientação a Objetos. Assim, as classes serão as responsáveis por modelar as entidades de negócio e fornecer os métodos que permitem que o sistema realize as operações necessárias.

A fim de se obter um produto que seja facilmente compreendido por terceiros, de fácil manutenção e que permita a adição de funcionalidades sem interferir nas funcionalidades já existentes, serão obedecidos os princípios SOLID. SOLID é um acrônimo para 5 princípios no desenvolvimento de sistemas orientados a objetos, que podem ser definidos como [22, 23]:

- Princípio da Responsabilidade Única (**S**ingle Responsability Principle): cada classe deve ter apenas um motivo por trás de sua construção e eventual modificação, ou seja, cada classe (ou grupo de classes, que compõem um módulo) deve corresponder a apenas uma função do sistema completo.

- Princípio do Aberto-Fechado (**O**pen-Close Principle): Toda classe deve estar aberta a extensões e fechada para modificações, de maneira que qualquer nova funcionalidade que precise ser inserida não cause modificações no código já existente;

- Princípio da Substituição de Liskov (**L**iskov Substitution Principle): Uma classe que utiliza um objeto O_B de determinada classe base B deve continuar funcionando caso um objeto $O_{B'}$ de uma classe B' derivada da classe base B seja passado como argumento;

- Princípio da Segregação de Interface (**I**nterface Segregation Principle): Consiste em dividir as interfaces grandes em interfaces menores e mais específicas, de modo que os clientes só precisem saber sobre os métodos de seu interesse. Ou seja, esse princípio mantém o sistema desacoplado, e mais fácil de modificar ou reestruturar.

- Princípio da Inversão de Dependência (**D**ependency Inversion Principle): Módulos ou classes de alto nível não devem depender de classes ou módulos de baixo nível, mas sim de suas interfaces; as abstrações (interfaces) não devem depender das implementações, mas sim o contrário.

3 MÉTODO DE TRABALHO

A abordagem utilizada neste projeto consistiu em um desenvolvimento iterativo e incremental. Após as etapas de pesquisa e escolhas das tecnologias a serem utilizadas, foi desenvolvido um Sistema Base inicial, incrementado a cada iteração para implementar e testar os casos de uso em questão.

3.1 Etapas de Trabalho

O processo de desenvolvimento do trabalho se pautou pelas seguintes etapas:

1. **Alinhamento da proposta do projeto:** Reuniões com os grupos no Instituto InovaUSP e da Faculdade de Economia e Administração da USP para entender os detalhes da proposta do projeto, incluindo os principais requisitos funcionais e não funcionais a serem satisfeitos

2. **Pesquisa Extensiva sobre os assuntos e tecnologias existentes:** Realização de pesquisas extensivas de modo a listar todas as tecnologias que podem ser utilizadas no projeto (cliente, servidor, blockchain, segurança da informação), seus pontos positivos e pontos negativos.

3. **Criação dos critérios de seleção e escolha das melhores tecnologias:** Escolha das tecnologias a serem utilizadas para as diversas partes necessárias para o projeto, com base nos critérios escolhidos

4. **Verificação da viabilidade da tecnologia escolhida:** Realização de testes individuais localmente de forma a verificar a viabilidade das tecnologias escolhidas e se aprofundar mais em suas funcionalidades.

5. **Testes das funcionalidades do Smart Contract a ser desenvolvido:** Testes locais das bibliotecas escolhidas para a criação dos smart contracts e verificação da viabilidade destes para os casos de uso a serem respeitados.

6. **Subida da blockchain no servidor escolhido:** Uma vez realizados os testes locais, realizar a subida da blockchain no servidor escolhido.

7. **Desenvolvimento do Sistema-Base (cliente, servidor, blockchain):** Escolhidas as tecnologias a serem utilizadas para o desenvolvimento (Node, Angular) e verificadas as funcionalidades da blockchain escolhida, desenvolver um sistema completo inicial, para que seja possível iniciar o desenvolvimento iterativo dos casos de uso

8. **Desenvolvimento e teste dos casos de uso:** Desenvolver e testar cada um dos casos de uso. A etapa 8 será realizada de forma iterativa.

4 ESPECIFICAÇÃO DE REQUISITOS DO SISTEMA

O sistema que será desenvolvido neste projeto, assim como qualquer sistema a ser projetado, deve atender a determinados requisitos funcionais e não-funcionais. A seguir, será apresentada a arquitetura geral pensada para o sistema, bem como os requisitos que devem ser atendidos. Além disso, introduz-se as tecnologias que deverão ser utilizadas no desenvolvimento do projeto.

4.1 Especificação Arquitetural

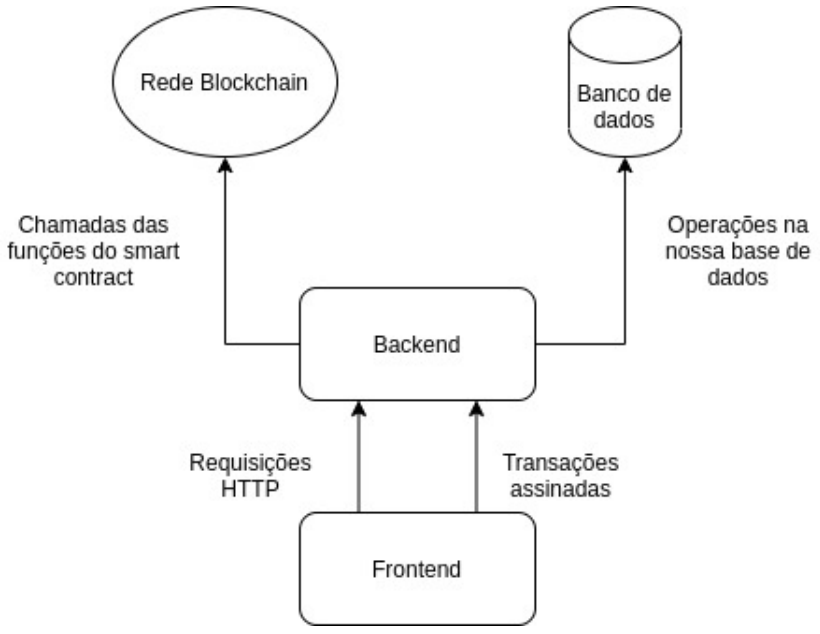

Figura 1: Diagrama - Arquitetura do Sistema

Fonte: Autoria Própria

A Figura 1 apresenta a arquitetura geral do sistema. O cliente (frontend) é o responsável pelas interações com o usuário e por fazer a assinatura localmente das transações que requerem assinatura (como transferências). Desta forma, as chaves privadas do

usuário não circulam pela rede e servidores. O backend implementa a lógica de negócio e é responsável por fazer a comunicação tanto com a rede blockchain, que processará todas as transações relativas aos tokens da Moeda USP, quanto com o banco de dados, que guardará informações específicas do nosso sistema (como dados dos usuários).

4.2 Requisitos Funcionais

Os requisitos funcionais se referem diretamente às funcionalidades do sistema, que devem atender aos casos de uso planejados. A seguir, uma explicação sobre quais são os grupos envolvidos, os casos de uso e, finalmente, as funcionalidades mínimas que o sistema deve possuir.

4.2.1 Grupos envolvidos

- Usuários: usuários do sistema que podem transacionar Moedas USP, como bolsistas e laboratórios. Existe a possibilidade de poder converter suas moedas por moeda fiduciária junto ao agente conversor em uma futura implementação do projeto.
- Administrador: responsável pelo controle do sistema e verificação de irregularidades.
- Consumidor de Serviços: Startups, por exemplo.
- Agente conversor: Responsável por fazer a conversão da moeda (USP).
- Agente emissor: Responsável por emitir Moedas USP no sistema (USP).
- Dono da moeda: É aquele que possui a moeda.

4.2.2 Casos de uso

- **Financiamento de *startup***

  Atores envolvidos: Administrador, emissor e Consumidor de serviços.

  Ações: Emissão da moeda na blockchain para Consumidor de Serviços, criação do lastro no banco de dados.

  Pré-condições estabelecidas: Consumidor de serviços já valorado por entidade responsável (USP), definindo número de moedas a serem emitidas.

- **Troca ou repasse de moedas entre usuários**

  Atores envolvidos: Usuários e Consumidor de Serviços.

  Ações: Repassar moedas.

  Pré-condições estabelecidas: Moedas já recebidas pelo Usuário ou Consumidor de Serviços que deseja repassá-las.

  Comentários: Acordo subjacente entre as partes que motiva a troca de moedas não é parte do escopo do sistema.

- **Conversão das Moedas USP**

  Atores envolvidos: Usuários e Agente Conversor.

  Ações: Enviar moedas para Agente Conversor que realiza a conversão para moeda fiduciária (similar a uma casa de câmbio).

  Pré-condições estabelecidas: Usuários com moedas e conversibilidade autorizada pela USP no momento.

  Comentários: Este caso de uso se aplica a usuários que têm a possibilidade de converter as Moedas USP em qualquer momento e os que podem realizar a conversão somente limitados por regras definidas pela USP. Os usuários podem realizar a conversão de suas Moedas USP junto ao Agente Conversor. É responsabilidade do agente conversor garantir a conversibilidade da moeda. As regras de conversão, como quanto tempo um usuário teria que esperar para poder realizar a troca, se alguns usuários, como bolsistas, poderiam realizar a troca a qualquer momento e como a USP deverá garantir a conversibilidade ainda estão sendo definidas pela equipe econômica. Entretanto, este caso de uso somente passaria a ser aplicado em um estágio avançado do projeto depois que já estiver em produção. Desta forma, foi definido que não fará parte do protótipo aqui apresentado.

4.2.3 Funcionalidades requeridas

- Possibilitar cadastro e login de usuários;
- Criar chaves na rede blockchain;
- Permitir que o usuário consulte seu saldo em Moedas USP;
- Permitir que o usuário realize transferências para outros usuários;

- Permitir que o administrador do sistema realize o bloqueio de transferências de e para determinada conta;
- Bloquear transferências para contas que não estejam cadastradas no sistema;
- Acompanhar a cotação das Moedas USP com base no lastro definido.

4.3 Requisitos Não-Funcionais

Os requisitos não funcionais não se referem diretamente às funcionalidades do sistema, mas sim a como estas funcionalidades serão implementadas. A seguir, serão apresentados os requisitos não-funcionais mais críticos ao projeto.

4.3.1 Segurança

O sistema deve ser capaz de permitir que usuários acessem sua conta e assinem transações na blockchain de forma segura, fornecendo-lhes total controle sobre seus recursos e impedindo que um terceiro realize uma transação em seu nome, mesmo em uma eventual invasão do sistema ou vazamento do banco de dados. Para isso será implementado um sistema de usuário e senha que permita acessar os recursos da plataforma; para salvar esta senha em nosso banco de dados, será utilizado um algoritmo de *Password Hashing* com alto custo computacional no lado do cliente (frontend), cuja saída será enviada ao servidor, que a usará como entrada para um outro algoritmo de Password Hashing com custo computacional reduzido, e a saída gerada será salva em nosso banco de dados, que será comparada com a gerada a partir da senha inserida pelo usuário.

Haverá ainda a chave privada de acesso a carteira da blockchain do usuário, esta chave permite a assinatura das transações e será salva cifrada utilizando um algoritmo de criptografia simétrico (AES-128-CTR) em um arquivo no dispositivo do usuário. A chave criptográfica utilizada na cifração será uma outra senha definida pelo usuário (que denominamos "senha de transação"), a qual não salvaremos em nossa base de dados. No caso de esquecimento da senha de acesso ao sistema, esta pode ser facilmente substituída por uma nova, porém no caso de esquecimento da senha que decifra a chave privada o sistema deverá então criar uma nova chave e bloquear a anterior. Quando o usuário realizar uma transação, o mesmo deverá fazer o *upload* do arquivo de sua chave privada e inserir sua senha de transações, usada para decifrar esta chave. A mesma será usada para assinar localmente a transação, esta por sua vez será enviada para o servidor para

ser submetida na blockchain, de modo que a chave privada e a senha de transação não trafegam pela rede.

4.3.2 Fácil manutenção e alterações

A modelagem do sistema deve permitir fácil manutenção e adição de novas funcionalidades. Para isso, o backend (componente que carrega a lógica de negócio e, portanto, possui código mais extenso) foi modelado conforme a Figura 2.

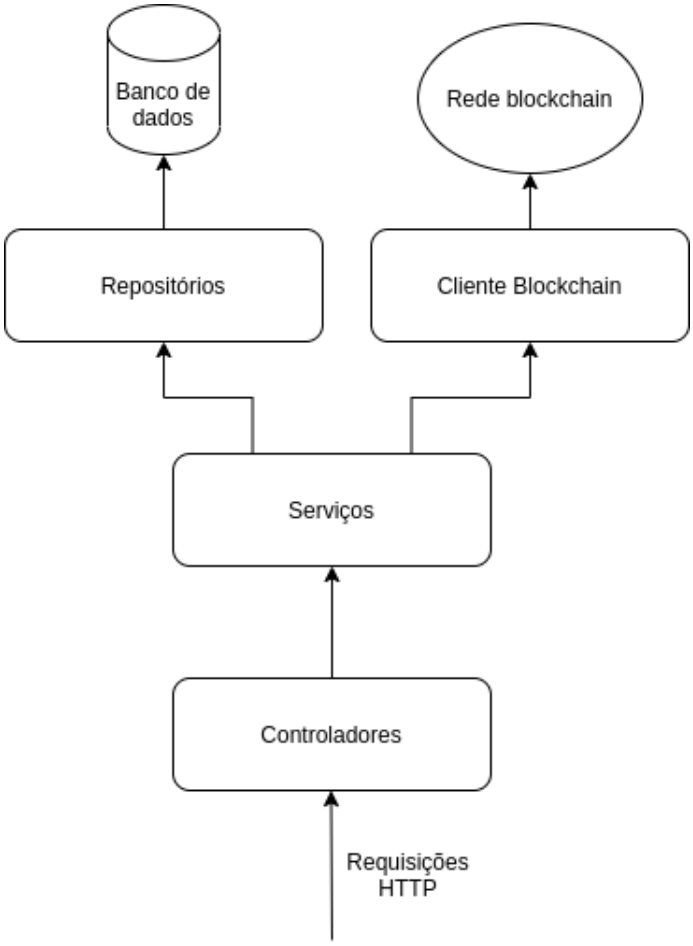

Figura 2: Diagrama - Backend do Sistema

Fonte: Autoria Própria

As classes Controladoras são as responsáveis por receberem e responderem as requisições HTTP que chegam às *routers* do servidor, chamando métodos das classes de Serviço. As classes de Serviço, por sua vez, implementam métodos específicos de cada entidade ou funcionalidade do sistema (por exemplo, a classe *UserService* implementa métodos específicos do usuário). As classes de Repositório são a camada de comunicação do sistema com o banco de dados, seguindo um padrão de design chamado *Repository*

Pattern [24]. Por fim, a classe do cliente blockchain realiza as chamadas que interagem com a rede blockchain. Com esta modelagem, as classes tendem a ficar mais simples e com poucas responsabilidades bem definidas, seguindo os princípios SOLID, facilitando o desenvolvimento, manutenção e adição de funcionalidades ao sistema.

4.3.3 Transparência e auditabilidade

O uso da blockchain para validar e registrar as transações tem como objetivo garantir transparência e fácil auditabilidade, uma vez que transações submetidas na rede e salvas nos blocos não podem ser alteradas nem apagadas [17, 11, 12, 5] sem que os nós estejam cientes da mudança.

4.3.4 Baixo consumo energético

O uso de tecnologias de blockchain levanta questões quanto ao consumo energético da rede, o que está intrinsecamente ligado ao algoritmo de consenso escolhido. Algoritmos do tipo Proof-of-Work, como o implementado na rede Bitcoin e Ethereum, necessitam de um alto consumo energético para o cálculo do desafio criptográfico [16]. Segundo o website *Digiconomist* [25], que acompanha o consumo energético do Bitcoin e Ethereum, o consumo energético médio anual do Bitcoin é de 132.25 TWh, comparável ao consumo da Argentina [26], e do Ethereum é de 53.17 TWh, comparável a Singapura [27]. Tendo isso em mente, o algoritmo de consenso utilizado será de Proof-of-Authority, como descrito na seção 2.3.3.

4.4 Tecnologias Utilizadas

Nesta seção são descritas as diversas tecnologias utilizadas neste projeto.

4.4.1 Node.js

Node.js é um ambiente de execução de javascript do lado do servidor [28], ou seja, não necessitando de um navegador para executar o código. Para o desenvolvimento do servidor Web, foi escolhida a biblioteca Express, framework de código aberto que permite a definição de rotas e APIs na aplicação.

4.4.2 Typescript

Typescript é uma linguagem de código aberto que é transformada em javascript através do compilador Typescript, podendo, portanto, ser executada em qualquer ambiente que possua compatibilidade com javascript (como navegadores de internet e Node JS), tendo o diferencial de permitir tipagem estática das variáveis [29]. Essa ferramenta permite a declaração de interfaces, facilitando o respeito ao quinto princípio SOLID, além de permitir que inconsistências no código sejam encontradas com maior facilidade.

4.4.3 OpenZeppelin

OpenZeppelin é uma biblioteca de código aberto de smart contracts (explicados na seção 2.2.2), suportada e validada pela comunidade que a utiliza. Seus smart contracts são construídos na linguagem solidity e seguem os padrões definidos nos *Ethereum Request for Comment (ERC)*, fornecendo a base para a construção de seus próprios contratos através do mecanismo de herança suportado pela linguagem [30].

4.4.4 Truffle

Uma das ferramentas da Truffle Suite, o Truffle é uma framework para desenvolvimento e testes de smart contracts em redes Ethereum [31]. Esta ferramenta facilita o processo de compilação e submissão do contrato na rede blockchain.

4.4.5 Ganache

Outra ferramenta da Truffle Suite, o Ganache permite que se suba uma rede Ethereum localmente com poucos cliques (no caso do GanacheUI) ou linhas de comando (no caso do GanacheCLI) [32]. Isto permite uma maior facilidade no processo de desenvolvimento, não tendo que se preocupar com a configuração de uma rede privada complexa ou conexão a um nó de uma das redes de teste do Ethereum.

4.4.6 Web3.js

Web3.js é uma coleção de bibliotecas que permite a interação com um nó local ou remoto de uma rede Ethereum usando HTTP, IPC ou WebSocket [33]. Ela é usada para chamar as funções genéricas da rede Ethereum e as definidas no *smart contract*.

4.4.7 Sequelize

Sequelize é uma ORM (Mapeador Objeto-Relacional) feita para Node.js, que permite o mapeamento entre os objetos instâncias das classes do código e as entidades do banco de dados. Desta forma, os desenvolvedores não precisam se preocupar em escrever consultas SQL manualmente, além da ferramenta já ter recursos próprios contra ataques do tipo injeção de SQL [34].

4.4.8 Remix

Remix é um IDE (Ambiente de Desenvolvimento Integrado) online que permite aos desenvolvedores criarem e implementarem facilmente contratos inteligentes na blockchain. Podem ser utilizadas diversas redes de desenvolvimento disponíveis. A ferramenta foi utilizada para fazer os primeiros exercícios e testes de interações entre contas no desenvolvimento do *smart contract*.

4.4.9 Angular

O Angular é um *framework* JavaScript que simplifica a construção da interface de usuário, e também o desenvolvimento de aplicações *client side*, sejam elas para a web, mobile ou desktop. A principal vantagem é que o Angular utiliza a linguagem Typescript, explicitada na seção 4.4.2. Neste projeto, foi utilizado para o desenvolvimento do frontend de uma aplicação Web.

4.4.10 Clique e Geth

Como explicado na seção 2.2.3, o protocolo Ethereum é implementado através de clientes como o Geth e o OpenEthereum. O algoritmo de consenso escolhido, *Proof-of-Authority* (seção 2.3.3), pode ser utilizado por esses dois clientes através das implementações Clique e Aura [35]. O projeto da rede *blockchain* começou a ser implementado utilizando primeiramente o Aura, uma vez que a documentação era mais detalhada, porém foram encontradas dificuldades técnicas referentes a atualizações que removeram formas de acessar contas na rede pelos nós, e para as quais não foram fornecidas soluções nativas ao ambiente do OpenEthereum, sendo necessário utilizar recursos do Geth [36]. Assim, decidiu-se realizar a implementação utilizando o cliente Geth com o protocolo Clique do algoritmo de consenso. O cliente Geth é acompanhado de um CLI (Interface de Linha de

Comando) que auxilia o desenvolvimento e criação dos elementos da rede, além de outras ferramentas para automatizar as configurações. A utilização destas ferramentas é descrita em maior detalhes na seção 5.2.

5 PROJETO E IMPLEMENTAÇÃO

O projeto foi dividido em três principais frentes de implementação, que serão detalhadas ao longo deste capítulo. A primeira é a criação de uma rede blockchain Ethereum privada para ser utilizada como base para o projeto. A segunda é a definição do smart contract que implementa o token. Por fim, a terceira é o desenvolvimento do sistema responsável pela comunicação entre o blockchain e os usuários, e que faz o gerenciamento e controle dos registros dos usuários.

A implementação do projeto pode ser verificada no repositório a seguir:

gitlab.uspdigital.usp.br/USPcoin

5.1 Ambiente de hospedagem

O projeto foi desenvolvido com o intuito de ser uma prova conceito de um sistema que cumpra os requisitos definidos. Por isso, o projeto foi implementado em um ambiente para desenvolvimento, que não representa um ambiente viável para ser posto em produção e atender a comunidade USP. A rede blockchain e o servidor foram hospedados em uma máquina virtual no servidor do LARC (Laboratório de Arquitetura e Redes de Computadores) com 4 GB de memória RAM, um núcleo, 15 GB de armazenamento em disco e rede de 1 Gbps. A máquina física onde foi hospedada possui uma plcaca mãe Intel S1400SP, processador Intel(R) Xeon(R) CPU E5-2430 0, de até 2.20GHz, com 6 núcleos e 12 *threads*, memória total de 48GB, disco total de 1 TB e possui onze máquinas virtuais em execução. O banco de dados foi hospedado na AWS (Amazon Web Services) na região de North Virginia utilizando a ferramenta RDS (Serviço de Banco de Dados Relacional), em uma máquina db.t2.micro, com um núcleo e processador Intel AVX, de até 3.3 GHz, e 1 Gigabyte de RAM.

5.2 Blockchain

A rede blockchain utilizada no projeto é uma rede privada, como definido em 2.1, que usará como base a rede Ethereum, sendo implementada pelo cliente Geth, utilizando o algoritmo de *Proof-of-Authority* implementado pelo protocolo Clique, como explicado em 4.4.10.

A criação da rede blockchain foi feita com o auxílio da ferramenta Puppeth, um gerenciador de redes privadas pertencente ao pacote Geth que auxilia a criação do *Genesis Block* da rede. Como definido na documentação [37], o *Genesis Block* é um arquivo json que contém os principais parâmetros da rede, entre eles:

- **config**: Features da plataforma Ethereum que exigiriam o agendamento de um *hard fork* em caso de mudanças, como o id da cadeia (*chainId*) e a configuração de regras que foram incluídas através de EIPs (*Ethereum Improvement Proposal)*, como por exemplo a formas de sincronização dos nós e regras de alteração do custo de transações para evitar ataques de negação de serviço. Além disso há também as configurações referentes ao algoritmo de consenso utilizado, como o tempo de intervalo entre emissões de novos blocos. Para o ambiente de desenvolvimento projetado foi definido como quinze segundos, que é o valor *default* sugerido pelo Puppeth e que atende as necessidades do protótipo;

- **gasLimit**: o valor inicial determina a quantidade de computações que podem ser efetuadas em um único bloco. Esta configuração pode ser alterada com a rede em funcionamento;

- **alloc**: alocação inicial de *Ethers* para as contas listadas no *Genesis Block*. Esses *Ethers* são para cubrir o *gas* das transações, e manter o bom funcionamento da rede. A quantidade criada é grande o suficiente para que não seja necessário criar mais ao longo de todo o tempo de funcionamento do sistema. Os *Ethers* são distribuídos pelas contas como explicado na seção 5.4.1.3;

- **difficulty**: dificuldade para se minerar um bloco. No caso de uma rede tradicional *Proof-of-Work* determina o quão rápido os blocos serão criados, porém, como no sistema implementado o algoritmo utilizado é *Proof-of-Authority*, este valor é irrelevante, assim, é definido como 1;

- **hardforks scheduling**: As modificações que foram feitas através de hard-forks da rede são listadas para que façam efeito a partir de determinado bloco. No caso

da rede implementada adotou-se a versão mais recente da rede, assim, todos os *hardforks* são programados para iniciar no bloco 0.

Uma vez criado o *genesis block* com as regras básicas de funcionamento da rede são criados os primeiros nós, que serão capazes de assinar transações. Os nós são criados utilizando o CLI do Geth, utilizando o *genesis block* criado. Como em uma rede Ethereum os nós não precisam necessariamente possuir endereço IP fixo, faz-se necessário a existência dos denominados *bootnodes*, que possuem um endereço fixo, a partir da conexão com este nó os nós comuns se juntam a rede. No caso da rede local os *bootnodes* existentes e mantidos na rede principal Ethereum não podem ser utilizados, assim, o cliente Geth fornece também a ferramenta para a criação de um *bootnode* local, processo que deve ser feito junto com a criação da chave única do nó.

Após a criação, os nós podem ser iniciados. A inicialização dos nós envolve a definição dos protocolos a partir dos quais a comunicação com os mesmos será possível, como RPC, HTTP e websocket. Além disso define-se também as APIs que poderão interagir com o nó, como por exemplo a web3.

Deste modo, obtém-se uma rede Ethereum local que utiliza o protocolo Clique para a implementação do algoritmo de consenso Proof-of-Authority. Esta rede pode ser já utilizada para fazer a implantação do contrato e realizar transações.

A Figura 3 mostra a conexão entre os dois nós criados e o bootnode usado para o processo de descoberta entre os nós. A conexão com o servidor do sistema é feita via RPC utilizando HTTP.

5.3 Smart Contract

Como já informado na seção 4.4.3, para o desenvolvimento do *smart contract*, responsável pela criação do Token da USP, foi utilizada a biblioteca de código aberto OpenZeppelin. Isto permitiu que fossem herdadas diversas funções já definidas em contratos "padrões" utilizados como base, facilitando a implementação e evitando problemas relacionados à segurança.

Primeiramente, foi necessário mapear quem seriam os atores daquele sistema, restringindo quais funções cada grupo possui acesso ou não. Para isso, o OpenZeppelin possui a estrutura de *Access Control*, na qual cada função contém um conjunto de endereços que representam as contas que possuem acesso a ela.

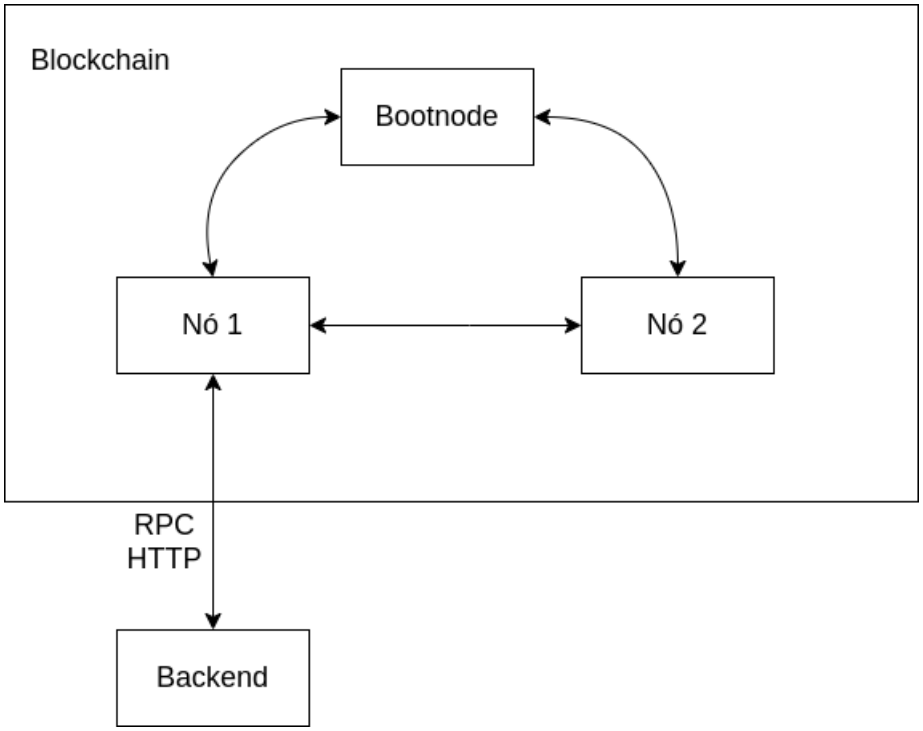

Figura 3: Conexão entre nós da blockchain e o bootnode, usado para os nós se descobrirem. O backend do sistema se comunica com o nó 1 utilizando uma conexão HTTP RPC

Fonte: Autoria Própria

Além das funcionalidades descritas acima, o *Access Control* também possui funções específicas com o objetivo de garantir e revogar permissões, e definir Grupos chamados de "Roles", aos quais cada uma das contas presentes na rede pode ser associada. Foram definidas 3 principais Roles: Administrador do Sistema (USP), Conta Ativa e Conta Bloqueada.

- **Administrador do Sistema (USP)**: Essa Role é concedida automaticamente à conta que realiza a emissão do *smart contract* na rede e possui acesso às funções de administrador do sistema, isto é: é capaz de minerar Moedas; é capaz de conceder e revogar Roles para outras contas; é capaz de parar todas as transações da rede. Uma conta administradora pode conceder a Role de administradora para outras contas.

- **Conta Ativa**: Tem de ser concedida pelo Administrador do Sistema. Possui permissões de realizar transferências, verificar saldos e interagir com outras contas da blockchain.

- **Conta Bloqueada**: Tem de ser concedida pelo administrador do sistema. Esta conta representa contas inativas ou bloqueadas na rede. Portanto, não pode realizar

nenhum tipo de transferência ou recebimento de moedas.

Tendo em vista os requisitos e Roles definidos acima, optou-se por utilizar como base o contrato "ERC20PresetMinterPauser" presente na biblioteca OpenZeppelin. Isso porque o contrato atribui automaticamente funções de administrador (DEFAULT_ADMIN_ROLE), pausador (PAUSER_ROLE) e minerador (MINTER_ROLE) à conta que realizou o deploy do contrato. O contrato também herda funções de AccessControl, de maneira a restringir o acesso das Roles definidas para o sistema.

Para cumprir os requisitos do sistema, foram criadas as funções a seguir:

- **mint(address to, uint256 amount)**: Função para minerar novos tokens. Necessária permissão de Administrador (DEFAULT_ADMIN_ROLE).
- **transfer(address recipient, uint256 amount)**: Função para realizar transferências entre contas. Necessária a permissão de Administrador ou de conta ativa (DEFAULT_ADMIN_ROLE, ACTIVE_ACCOUNT_ROLE). Para cada transferência, será cobrada uma taxa recebida pela USP, definida no momento da implantação do smart contract;
- **blockAccount(address account)**: Função para garantir permissões de conta bloqueada. Necessária permissão de Administrador (DEFAULT_ADMIN_ROLE). Quando uma conta ativa é bloqueada, as permissões de conta ativa são automaticamente revogadas;
- **activateAccount(address account)**: Função para conceder permissões de conta Ativa. Necessária permissão de Administrador (DEFAULT_ADMIN_ROLE). Além disso, quando uma conta bloqueada é ativada, as permissões de conta ativa são concedidas novamente;
- **grantAdminRole(address account)**: Função para conceder permissões de Administrador. É necessário que a conta que realiza esta ação seja uma conta Administradora.
- **pause()**: Função para Pausar todas as transações do sistema. Necessária permissão de Administrador (DEFAULT_ADMIN_ROLE e PAUSER_ROLE);
- **unpause()**: Função para Despausar todas as transações. Necessária permissão de Administrador (DEFAULT_ADMIN_ROLE e PAUSER_ROLE).

De maneira a acelerar o primeiro protótipo, primeiramente foi utilizado o Remix, um ambiente de desenvolvimento integrado online que permite que o desenvolvedor crie, compile e realize a emissão do seu smart contract em uma rede Ethereum de desenvolvimento, de maneira fácil e ágil (explicitado na seção 4.4.8). Com a ferramenta, foi possível realizar interações entre contas, e testar diversas funções implementadas. Após definido e testado utilizando o Remix, o contrato foi testado em rede local, com a utilização do Truffle (explicitado na seção 4.4.4) e do Ganache (explicitado na seção 4.4.5). Após finalizados os testes localmente, foi feita a emissão na rede blockchain desenvolvida para este projeto, hospedada em um servidor do LARC.

5.4 Sistema

O sistema desenvolvido é o responsável pelo gerenciamento dos usuários e a interação destes com a blockchain. Como descrito na seção 4.4, a linguagem utilizada para o desenvolvimento do sistema é Typescript, utilizando um ambiente Node.js no backend com a biblioteca Express e a *framework* Angular no frontend. Os principais pontos do desenvolvimento do sistema serão expostos nas seções seguintes.

5.4.1 Recursos

O sistema desenvolvido implementa os recursos e fluxos apresentados nos itens desta seção.

5.4.1.1 Autenticação

A autenticação de usuários envolveu a implementação do método de proteção de senha introduzido por Contini [38]. Neste método são salvos no banco de dados o *username* dos usuários junto com uma hash de senha e uma semente de *salt* gerada por CSPRNG (Gerador de Número Pseudo-Aleatório Criptograficamente Seguro) com ao menos 128 bits. Esta semente é trocada se o usuário trocar sua senha. Ao realizar o login, o usuário fornece ao cliente seu *username* e sua senha. O cliente envia para o servidor o *username* e solicita o *salt* para calcular a hash. O servidor concatena a semente salva na tabela *tb_users* relativa ao usuário, o *username* e o *domain-name* da aplicação, calcula a hash desse valor (utilizando SHA256) e fornece como *salt* para o cliente. Caso o usuário não conste no banco de dados, o servidor realiza o mesmo processo porém com uma semente secreta, para que nenhum atacante saiba os usuários válidos apenas requisitando o *salt*. O

cliente então realiza o cálculo do hash com a senha fornecida pelo usuário e o *salt* enviado. O cálculo do hash é feito utilizando o algoritmo Argon2 com alto custo computacional, impossibilitando que um ataque de força bruta seja feito caso haja vazamento do banco de dados. A hash gerada é então enviada para o servidor junto com o usuário para realizar o login. O servidor realiza novamente um hash utilizando um algoritmo com baixo custo computacional (SHA256), para evitar ataques de negação de serviço, e compara com a hash salva no banco. Se as hashes forem correspondentes o servidor envia um token JWT para o cliente para ser armazenado no navegador em um cookie http-only, dificultando ataques do tipo *Cross Site Scripting (XSS)*. Este token JWT é assinado utilizando o algoritmo RS256 (assinatura RSA com hash SHA256 [39]) com uma chave privada e, quando enviado para o servidor junto a alguma requisição, verificado com uma chave pública. Desta forma, torna-se possível separar a autenticação das demais funções do servidor, possibilitando um eventual uso de *single sign on*. O fluxo de autenticação pode ser verificado na Figura 4.

A autenticação de *startups* ocorre de forma semelhante, porém o CNPJ da empresa é utilizado no lugar do *username*.

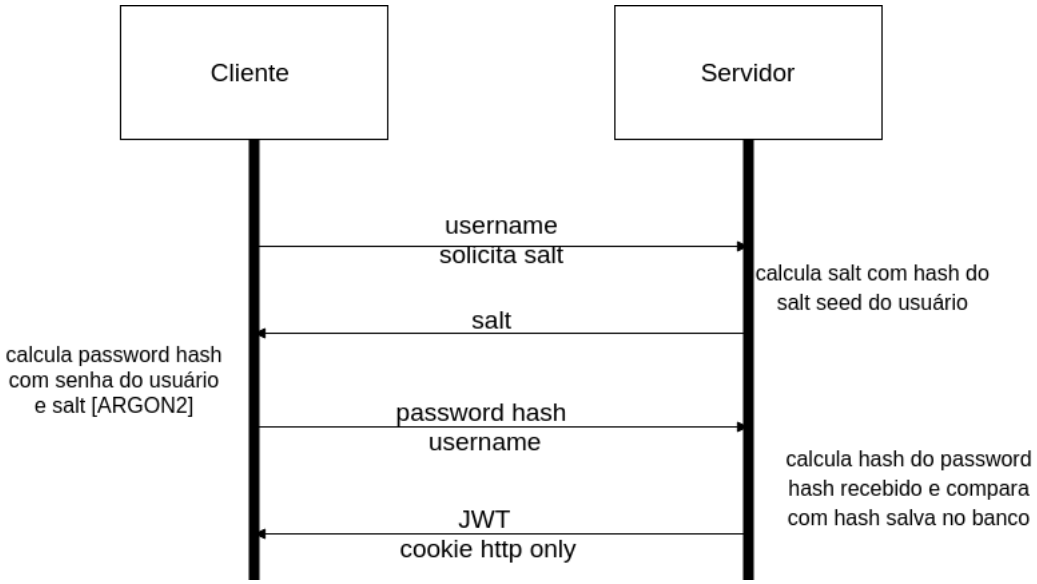

Figura 4: Fluxo de mensagens na autenticação de usuário

Fonte: Autoria Própria

5.4.1.2 Criação de conta

A criação de contas, tanto de usuário quanto de *startups*, e seu registro no sistema é feito seguindo um fluxo parecido com o de autenticação, para evitar os tipos de ataques

descritos, principalmente de negação de serviço. Primeiramente o usuário fornecerá o nome de usuário desejado ao cliente, este se comunicará com o servidor para conferir se o nome de usuário já não é utilizado. O servidor fará a conferência se o nome já não existe entre os usuários cadastrados e, caso negativo, calculará a semente do *salt* e o *salt* e enviará esses dois dados para o cliente junto com a confirmação da disponibilidade do nome de usuário. Neste momento nenhuma das informações será salva no banco de dados. O cliente então pedirá a senha para o usuário e calculará o hash junto com o *salt* recebido utilizando o algoritmo Argon2 com alto custo computacional.

O cliente cria então a conta do usuário na blockchain, que consiste na chave pública, correspondente ao seu endereço, e a chave privada, com a qual se assina transações. A criação da conta é feita utilizando a biblioteca *web3*. É solicitado ao usuário que forneça uma nova senha, denominada senha de transação, que será usada para cifrar a chave privada. A chave cifrada é então salva em arquivo no dispositivo do usuário, que se torna responsável por protegê-la, assim como a sua senha de transação. Desta forma essas duas informações extremamente sensíveis, já que possibilitam realizar transações na blockchain em nome do usuário, não são salvas em momento algum no sistema.

O nome de usuário, a hash calculada, a semente do *salt* e a chave pública da blockchain serão então devolvidas ao servidor utilizando uma comunicação TLS, junto com demais informações como nome completo, endereço e informações da empresa, no caso de uma startup. Ao receber essas informações, o servidor calculará o hash novamente utilizando um algoritmo com baixo custo computacional (SHA256) e então salvará as informações relativas ao usuário no banco de dados na tabela *tb_users* e a chave pública na tabela *tb_blockchain_credentials*, utilizando os modelos criados com a ORM Sequelize. A chave pública é salva com *status* "Pendente de aprovação"no banco de dados, e seu endereço ainda não possui a role de conta ativa do contrato, assim ainda não pode receber *tokens* (desta forma, há dois níveis de proteção para impedir que atores não autorizados realizem ações). O usuário deve então aguardar que sua chave seja ativada por um administrador.

O fluxo apresentado foi definido para que, caso haja uma falha de comunicação e perdas nas requisições, não haja informações incompletas escritas no banco, como um usuário sem senha. O fluxo de cadastro de usuários pode ser verificado na Figura 5.

O cadastro de *startups* ocorre de forma semelhante, porém o CNPJ da empresa é utilizado no lugar do *username*.

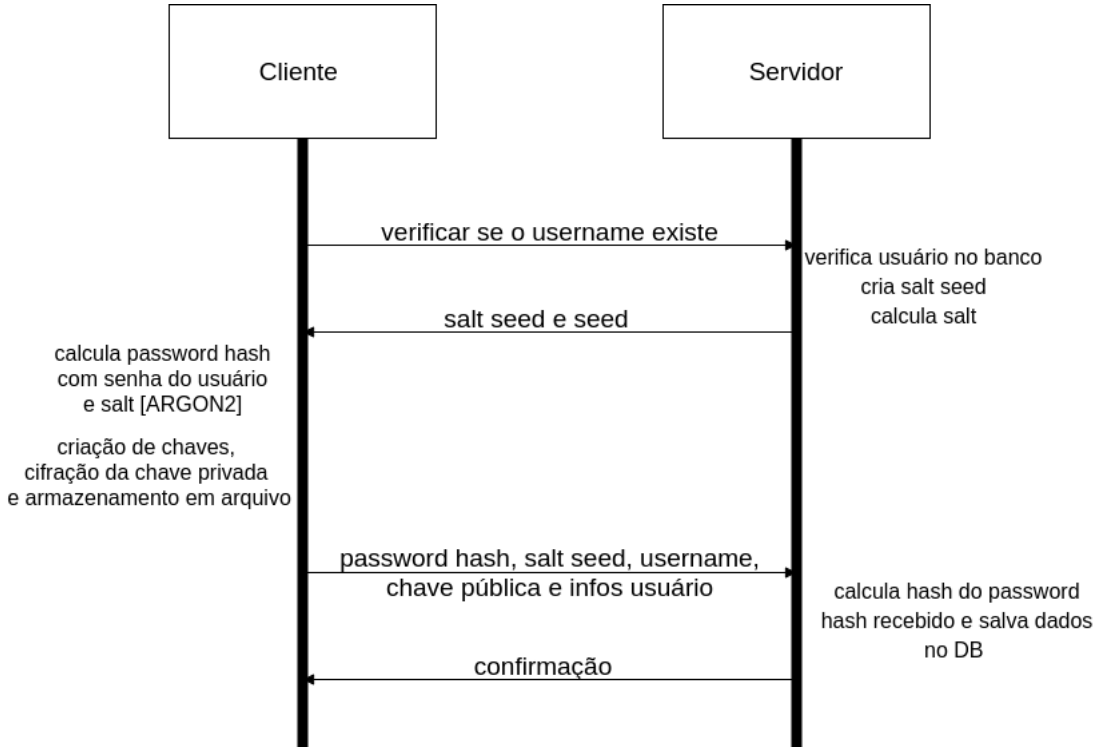

Figura 5: Fluxo de mensagens no cadastro de usuário

Fonte: Autoria Própria

5.4.1.3 Ativação e Bloqueio de chave de usuários e administradores

Como explicado na seção anterior, o simples cadastro de usuários no sistema não depende da ação de nenhum administrador, dado que uma vez cadastrados, as chaves públicas recém-criadas ficam pendentes de aprovação no sistema e sem a role de "Conta Ativa"na blockchain. Para que os mesmos possam usufruir das funcionalidades do sistema, é necessário que um administrador faça a ativação das chaves e, para isso, são necessários três passos: conferir a role de "Conta Ativa"na blockchain, transferir *Ethers* para a chave recém-criada (lembrando que eles não têm qualquer valor em nosso sistema, servem apenas para que a rede funcione adequadamente e, por isso, todas as chaves possuem uma quantidade grande o suficiente de *Ether* para que possa ser considerada ilimitada) e alterar o status da chave na tabela *tb_blockchain_credentials* para "Ativa". O principal motivo de as transações serem feitas nesta ordem é que a chance da transação com a rede blockchain falhar é maior do que com o banco de dados, e caso falhe esta transação, a com o banco não chega a ocorrer e as informações não ficam inconsistentes.

Para efetuar os passos descritos, um usuário com papel de administrador do sistema deve estar devidamente logado e com o arquivo que contém sua chave privada cifrada em mãos. Ele então deve selecionar a conta a ser ativada (que consta como "Pendente de

ativação"), o sistema vai solicitar que seja feito o *upload* do arquivo e a entrada da sua senha de transações, que o cliente irá utilizar para decifrar o conteúdo do arquivo, obter a chave privada e assinar as transações de conferir a role de "Conta Ativa"à chave pública selecionada e de transferir para a mesma uma grande quantidade fixa de *Ethers* (tanto para realizar a decifração do arquivo da chave privada quanto para assinar as transações é utilizada a biblioteca *web3*). Uma vez que as transações tenham sido assinadas, elas são enviadas uma por vez ao servidor para que sejam submetidas na blockchain (esta submissão também é feita utilizando-se a *web3*) e, ao retornar para o cliente que a última tarefa foi realizada com sucesso, o mesmo envia um último pedido ao servidor para ativar no sistema esta mesma chave, e o servidor por sua vez a coloca como ativa no banco de dados. No caso de ativar uma chave de administrador, os passos são praticamente os mesmos para usuários, com a diferença de que a role dada à chave pública na blockchain é a de "Administrador"ao invés de "Conta Ativa"e, ao ativar a chave, o servidor também atualiza o usuário no banco de dados para "Administrador"; além disso, a quantidade de *Ethers* a ser transferida à chave não é fixa, mas sim a metade do saldo do administrador que está realizando a ativação, pois foi considerada como uma maneira eficiente de manter a quantidade de *Ethers* similarmente dividida entre os administradores, considerando que todos cadastrem um número parecido de usuários, *startups* e administradores.

O fluxo dessas operações pode ser visto, sem as mensagens de confirmação, na Figura 6, porém sem as mensagens de mineração de tokens, em cinza.

O bloqueio de contas segue um fluxo similar ao de ativação, com a diferença de que a role dada à chave é "Conta Bloqueada"e o *status* no sistema é atualizado para "Bloqueada". Além disso, não são transferidos *Ethers* para a chave bloqueada.

Cada passo contém validações: um usuário cuja chave pública não tenha a role de "Administrador"na blockchain não consegue conferir roles a outras chaves, e o servidor do sistema valida o *token* JWT enviado na requisição para verificar se aquele usuário está cadastrado como um administrador.

5.4.1.4 Ativação de chave de *startups* e atualização do *Valuation*

O processo de ativar a chave de uma *startup* é similar ao de ativar a chave de um usuário, mas contém um importante passo adicional: a mineração de Moedas USP. Quando um administrador realiza a ação de ativar a chave pública de uma *startup*, o sistema requer que ele insira o *Valuation* da mesma para que, desta forma, seja possível calcular a quantidade de *tokens* a ser minerado com base no lastro da Moeda USP. Por-

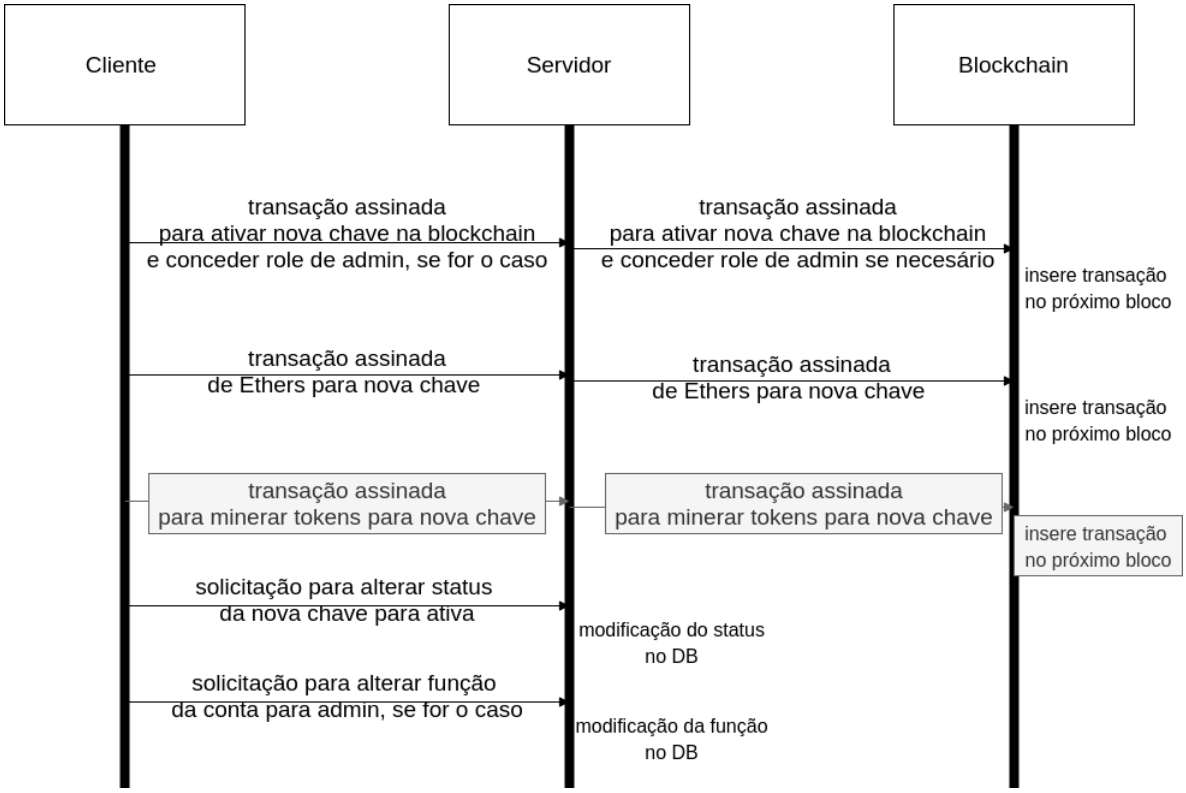

Figura 6: Fluxo de mensagens para ativação de chaves, sem mensagens de confirmação. As mensagens em cinza acontecem somente na ativação de chaves de *startups*.

Fonte: Autoria Própria

tanto, além de assinar as transações de conferir a role de "Conta Ativa"e de transferência de Ethers, o cliente também deve assinar a transação de mineração, que chama função *mint* do *smart contract*, fazendo com que ao submeter esta transação, o número calculado de Moedas USP seja gerado e depositado no endereço da chave da *startup*, que pode utilizá-las para contratar serviços internos da USP, como utilização de laboratórios. É importante ressaltar que esta operação não altera o lastro da moeda virtual em questão. Já o processo de atualização do *Valuation* tem um efeito diferente. Ele não se comunica de nenhuma forma com a rede blockchain, mas é o responsável pela alteração do lastro e consequente valorização ou desvalorização da Moeda USP. Nele, um administrador pode alterar o valor de uma *startup* já cadastrada e com conta ativa no sistema, portanto não alterando em nada o estado da blockchain, mas dado que o lastro da moeda virtual é justamente o valor das *startups*, o ato de atualizar o valor das mesmas faz com que o valor da conversão, calculado a partir da divisão do valor total das *startups* pelo número total de *tokens*, também se altere. O fluxo de mensagens pode ser visto na Figura 6, incluindo as mensagens de mineração de *tokens* em cinza.

5.4.1.5 Perda de chave privada

Caso um usuário perca o arquivo de sua chave privada ou esqueça sua senha de transação deve ser possível para este solicitar uma nova chave e ter acesso aos *tokens* que possuía em sua conta. Como a chave privada e a senha de transferência não são salvos no sistema, optou-se por realizar uma operação que envolve a criação de uma chave nova, o bloqueio da antiga, e a mineração de novos *tokens* na conta nova. Assim, quando o usuário informa que não possui mais acesso a sua chave privada, o cliente cria um novo par de chave pública (o endereço da blockchain) e privada utilizando a biblioteca *web3*. É solicitado uma nova senha de transferência ao usuário para cifrar a chave privada e salvá-la em arquivo no dispositivo do usuário. A nova chave pública é enviada para o servidor, que a salva na tabela *tb_blockchain_credentials* com *status* "Pendente de ativação". O servidor nesse momento altera o *status* da conta antiga do usuário para "Pendente de bloqueio".

Após a solicitação de uma nova chave privada o usuário perde acesso a tela de transferências e deve aguardar que sua nova chave seja ativada por um administrador do sistema. O administrador do sistema irá então ativar a nova chave do usuário de forma similar a primeira ativação da chaves, porém com alguns passos a mais a nível de sistema, embora para o administrador baste carregar sua chave e apertar o botão de ativar a chave "Pendente de ativação". Quando o administrador fizer isso, primeiramente o cliente solicitará ao servidor o saldo do endereço da chave "Pendente de bloqueio". Este por sua vez fará a mesma solicitação ao contrato na blockchain, e enviará esta informação novamente para o cliente. O cliente então solicitará o bloqueio desta chave perante o contrato ao servidor, assinando a transação de bloqueio com a chave privada do administrador utilizando a biblioteca *web3*. O servidor então submeterá esta transação na blockchain utilizando a mesma biblioteca, assim conferindo a *role* de "Conta bloqueada". Após confirmação do sucesso desta operação, o cliente fará a solicitação para que o servidor atualize o *status* desta chave na tabela *tb_blockchain_credentials* para bloqueado. Ao receber a confirmação da execução dessas operações, o cliente solicitará então a ativação da chave nova do usuário perante o contrato, novamente assinando a transação. O servidor novamente submeterá a transação na blockchain, assim a nova chave possuirá a *role* de "Conta Ativa". O cliente então fará a transferência de *Ethers* do endereço da chave do administrador para a nova chave, para que esta possa ser utilizada na rede. Esta transferência é novamente assinada pelo cliente e submetida na blockchain pelo servidor. O cliente fará então a solicitação da mineração do mesmo número de tokens que havia na chave recém bloqueada do usuário em sua nova chave, a solicitação é assinada e submetida da mesma maneira. Por fim,

após receber a confirmação de todas estas operações, o cliente solicitará ao servidor que atualize o *status* da conta nova do usuário na tabela *tb_blockchain_credentials* para ativo. O fluxo do bloqueio da chave anterior do usuário e a ativação da nova, sem as mensagens de confirmação, pode ser visto na Figura 7 .

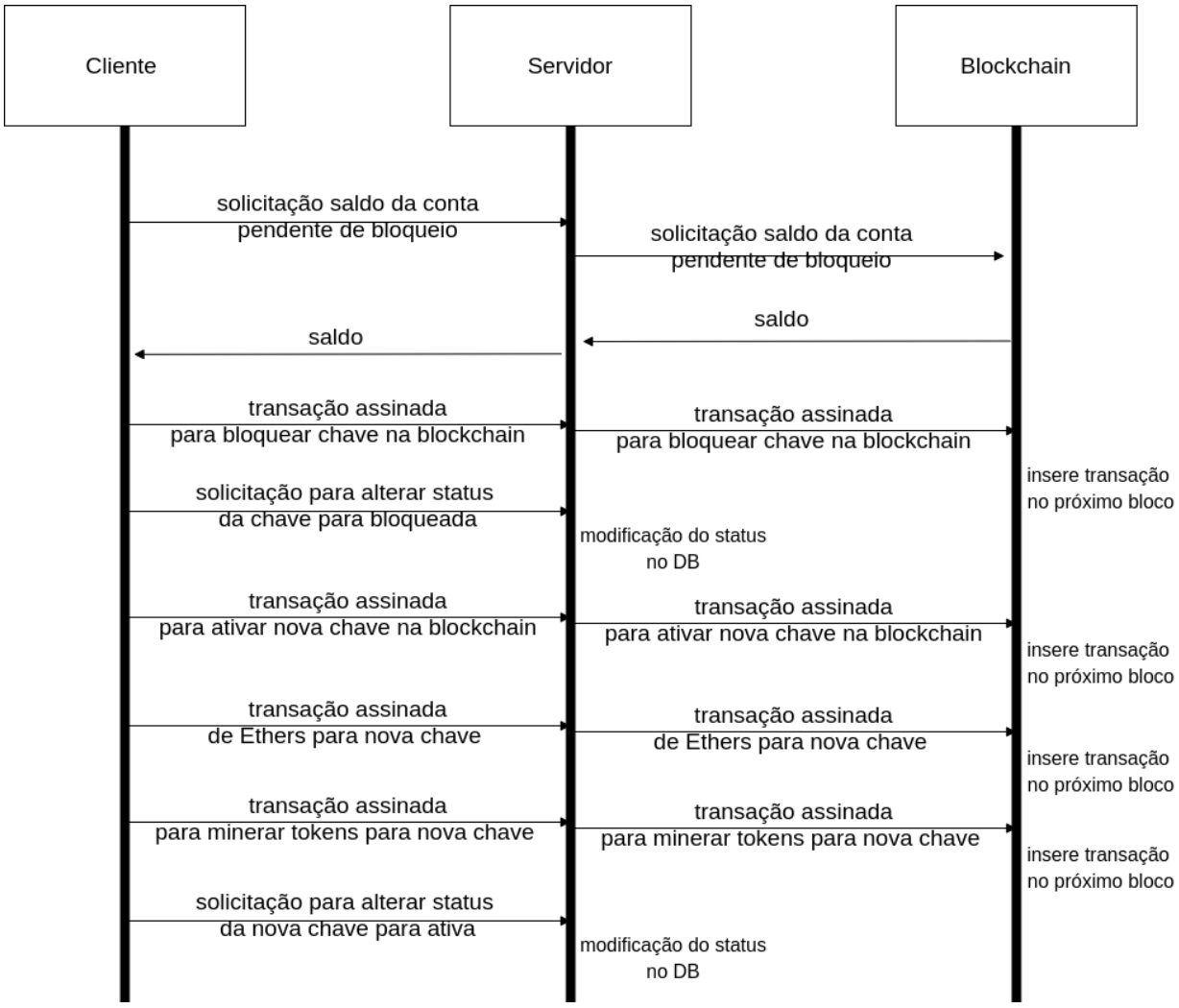

Figura 7: Fluxo de mensagens no bloqueio de uma chave e ativação de uma nova sem mensagens de confirmação

Fonte: Autoria Própria

5.4.1.6 Transferência

A transferência de Moedas USP entre os atores do sistema se dá através de um identificador único dependendo do receptor. Então se o objetivo for transferir para um usuário, basta digitar o *username* de quem deve receber as moedas, e se for para uma *startup* deve-se digitar o CNPJ cadastrado, além do valor. Com estas informações, o sistema busca dados adicionais sobre o receptor (para que seja possível conferir a transação) e a

chave ativa, que irá de fato receber as moedas. Quem está realizando a transferência deve então fazer o *upload* do arquivo de sua chave privada e inserir sua senha de transações, para que o cliente possa decifrar o que está no arquivo, obter a chave privada, assinar a transação (chamando a função *transfer* do *smart contract*) e enviá-la ao servidor para que seja submetida na rede blockchain. Importante ressaltar que para que o usuário possa realizar a transferência, tanto quem envia quanto quem recebe devem ter uma chave ativa no sistema e na blockchain. O sistema gera uma mensagem de erro customizada dependendo de qual for o tipo de problema: falta de chave ativa por parte do usuário origem ou destino, senha de transação errada e consequente falha na decifração do arquivo de chave privada, ou chave privada não condizente com a chave pública cadastrada no banco de dados (o que indica que um usuário está logado numa conta, mas usando o arquivo de chave privada de outro e, de alguma forma, teve acesso à senha de transação). O fluxo de mensagens na realização de transações pode ser visto na Figura 8.

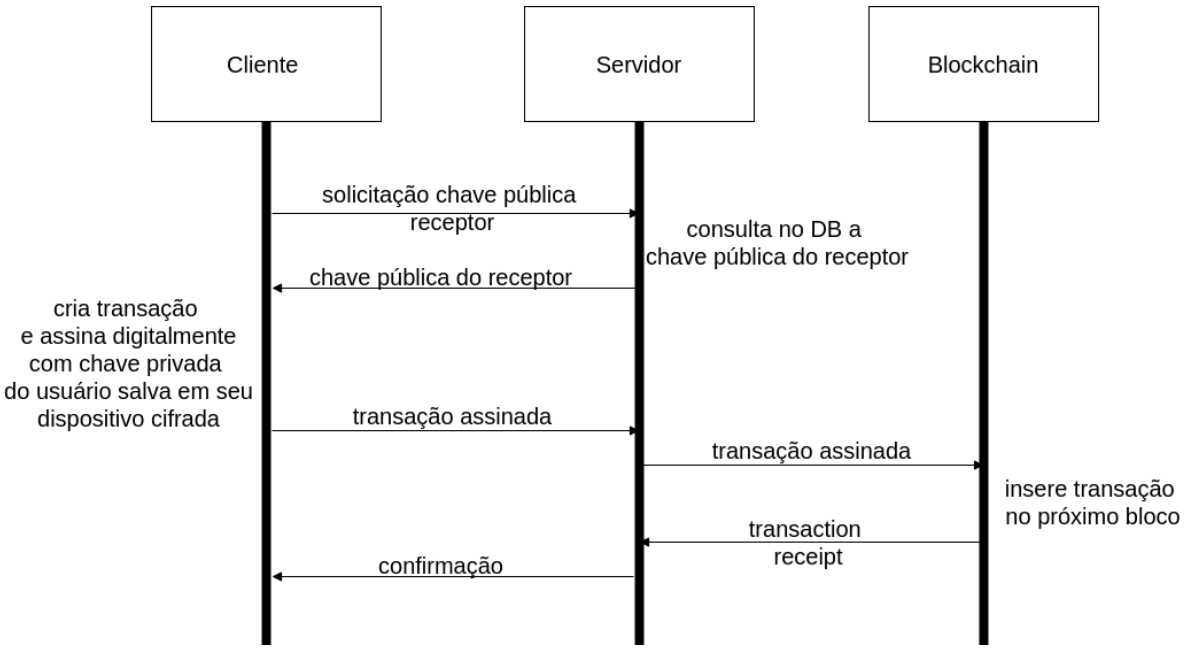

Figura 8: Fluxo de mensagens na realização de transações

Fonte: Autoria Própria

5.4.2 Estrutura arquitetural

A arquitetura do sistema pode ser dividida em dois componentes principais, o cliente, que serve de interface para o usuário, e o servidor, que se comunica com o banco de dados e com a blockchain.

5.4.2.1 Cliente

- **Login** (Acessível a todos)

  A página de Login (Figura 9) inicia o processo de autenticação do usuário 5.4.1.1. Existem duas opções na tela: preencher o Login, no qual o usuário informa o nome de usuário e a senha para se autenticar no sistema, ou "Não possui cadastro? Clique aqui", que redireciona para a página de cadastro. O processo de Login é o mesmo tanto para *Startups* quanto para os Usuários e Administradores.

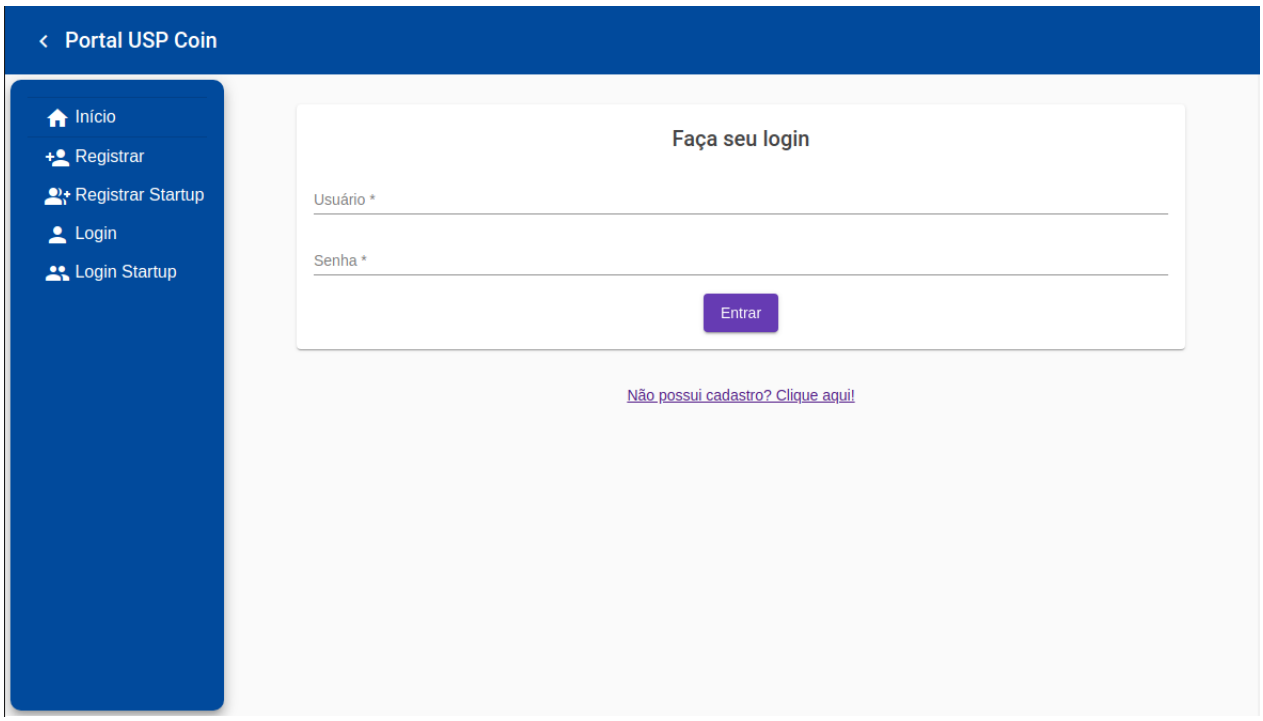

Figura 9: Tela de Login de Usuário

Fonte: Autoria Própria

- **Registro de contas** (Acessível a todos)

  Ao selecionar "Registrar" ou "Registrar *Startup*"no menu lateral, o usuário é direcionado para a tela de Registro de usuário ou Registro de *Startup* (Figura 10) , na qual se inicia o fluxo de criação de conta descrito na seção (5.4.1.2). No caso de uma criação de conta de usuário, este informa o nome de usuário desejado, e após a confirmação de sua unicidade, são renderizados os componentes para o preenchimento do "Nome Completo", "Senha do Sistema" e "Senha para Transferência", utilizada pra criptografar a chave privada. Ao preencher as informações e confirmar, o sistema instrui o usuário a realizar o download de sua chave privada cifrada. Já para registrar *Startups*, o processo é exatamente o mesmo, mas ao invés de "Nome de

Usuário", deve ser informado o CNPJ da instituição.

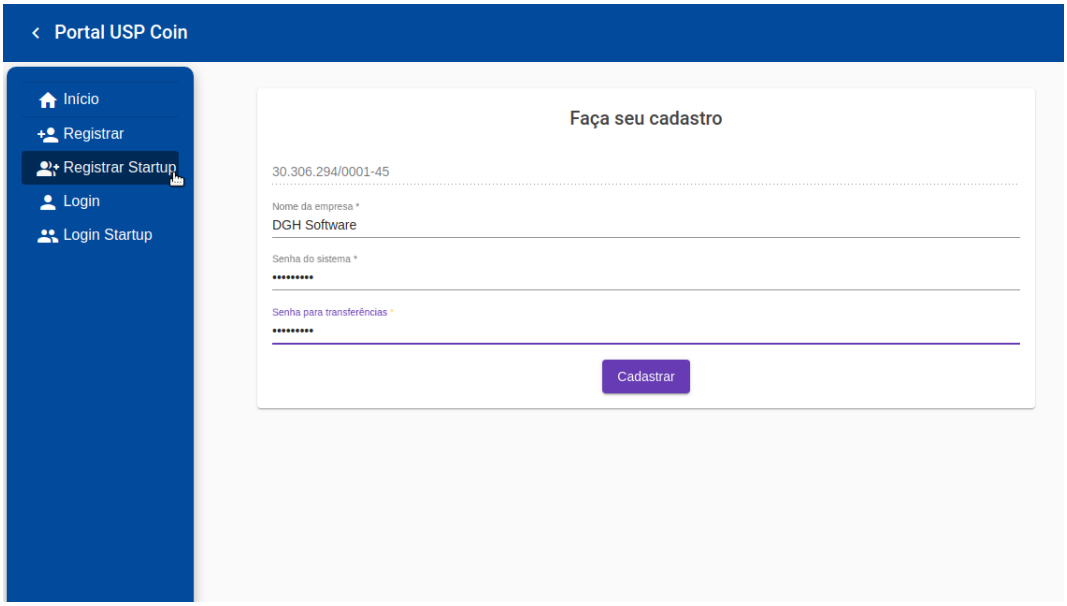

Figura 10: Tela de Cadastro de Startup

Fonte: Autoria Própria

- **Tabela Usuários e Tabela *Startups*** (Acessível apenas a administradores)

  As tabelas de Usuários (Figura 11) e *Startups* podem ser acessadas ao selecionar "Usuários"ou "*Startups*"no Menu Lateral, exclusivamente por Contas Administradoras do sistema. Muito semelhantes entre si, as tabelas disponibilizam de uma maneira interativa todas as informações de usuários e *Startups* cadastrados no sistema, incluindo sua chave ativa. Estas também permitem que sejam realizadas buscas específicas através da barra de pesquisa. Na tabela dos usuários, além de consultas, é possível realizar a ativação de conta de administrador descrita na seção 5.4.1.3, através do botão "Tornar Admin". Também é possível abrir a Tabela de credenciais de cada um dos Usuários, explicitada no item seguinte. Já na tabela de *Startups*, é possível apenas abrir a tabela de credenciais, não sendo possível tornar uma *Startup* administradora do sistema.

- **Tabela Chaves** (Acessível apenas a administradores)

  Uma vez selecionadas as tabelas dos usuário ou das *Startups*, o administrador do sistema pode clicar em "...", renderizando a Tabela de Chaves daquela conta (Figura 12). Nesta tabela é possível ativar chaves inativas ou bloquear chaves ativas, sendo que cada conta do sistema pode ter apenas uma chave ativa atribuída a ela. No caso das *Startups*, no momento em que o administrador ativa a conta, é solicitado que este forneça o *Valuation* inicial da empresa, para que sejam minerados as

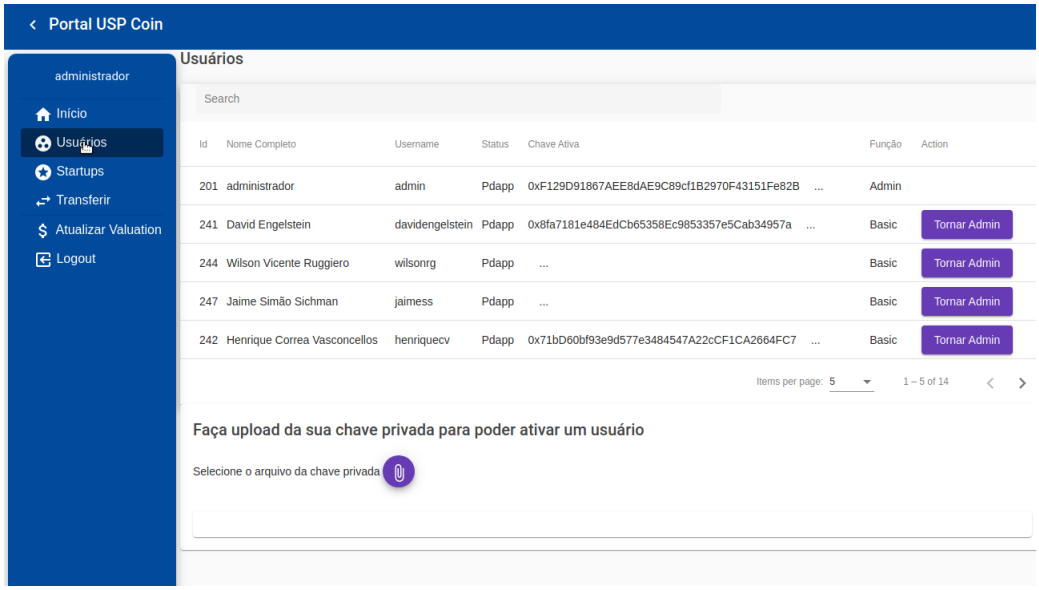

Figura 11: Tabela de Usuários

Fonte: Autoria Própria

Moedas USP relativas ao seu cadastro, processo descrito na seção. 5.4.1.4. Caso a *Startup* valorize, é possível atualizar o *Valuation* através da opção "Atualizar *Valuation*"disponível no menu lateral. A Tabela de Chaves permite ainda que o administrador realize a ativação de novas chaves quando o usuário perde seu acesso a chave privada, como descrito no fluxo na seção 5.4.1.5 (Figura 13). Para que um administrador consiga realizar as ações de Ativar Chave, Tornar Admin e Bloquear Chave, é necessário que este forneça sua chave privada e senha de transferência.

- **Atualização do *Valuation* de *Startup*** (Acessível apenas a administradores)

  Caso uma *Startup* ativa no sistema valorize, seu valor precisa ser atualizado por um dos administradores do sistema. Para isso, este deve selecionar a opção "Atualizar Valuation"presente no Menu Lateral, iniciando o fluxo descrito na seção 5.4.1.4, no qual o administrador informa o CPNJ da Startup e o novo Valuation. Para que a atualização seja completa, é necessário que o Administrador forneça sua chave privada e senha de transferência.

- **Transferência** (Acessível apenas a usuários/*startups* com contas ativas)

  A página de transferências (Figura 14) permite que qualquer usuário/*startup* com chave ativa realize transferências de moedas, como descrito no fluxo em 5.4.1.6. Para isso, basta preencher o valor, usuário de destino, sua senha de transferência e carregar o arquivo com sua chave privada. Caso o usuário tenha esquecido sua chave privada, existe também o botão de "Perdi Minha chave privada", que inicia o

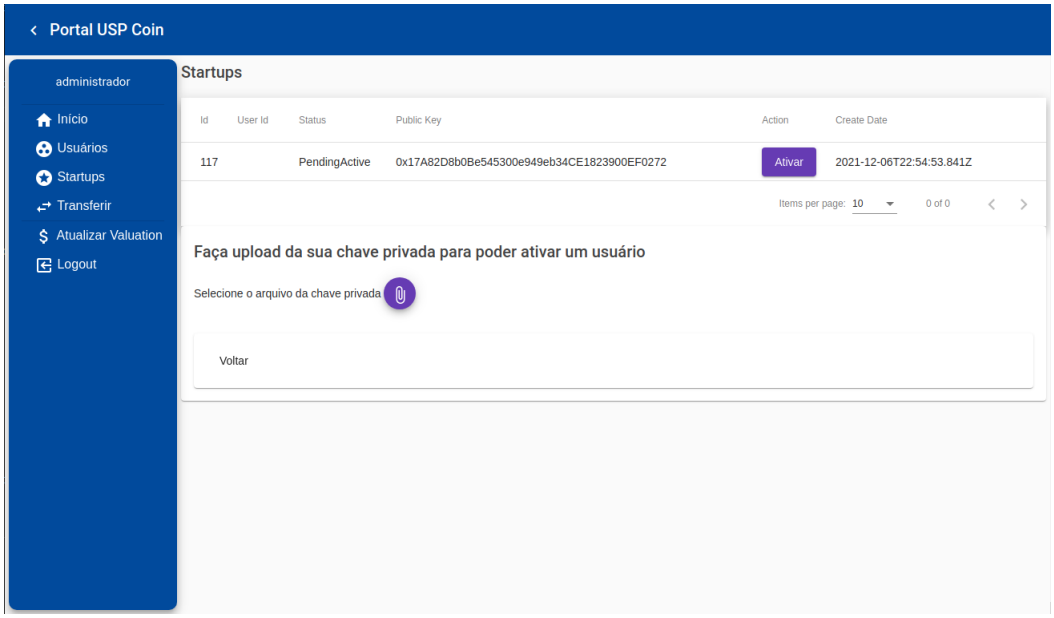

Figura 12: Tabela de Chaves ao ativar chave

Fonte: Autoria Própria

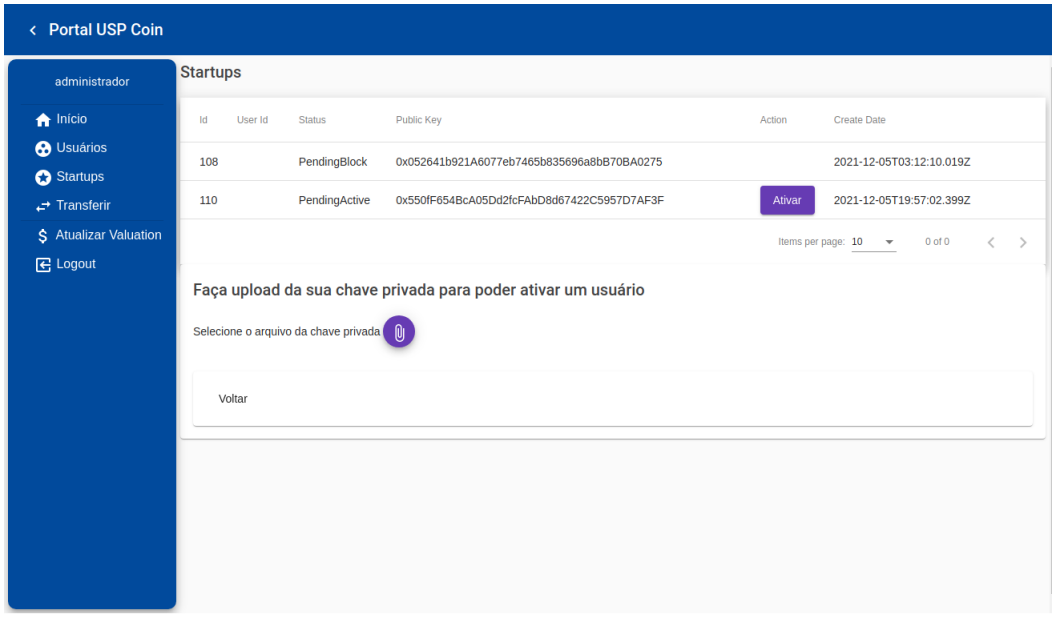

Figura 13: Tabela de Chaves ao substituir chave

Fonte: Autoria Própria

fluxo de Perda da Chave Privada explicitado na seção 5.4.1.5, bloqueando a conta ativa, criando uma nova e emitindo um novo arquivo com a chave privada da nova conta atrelada ao usuário logado.

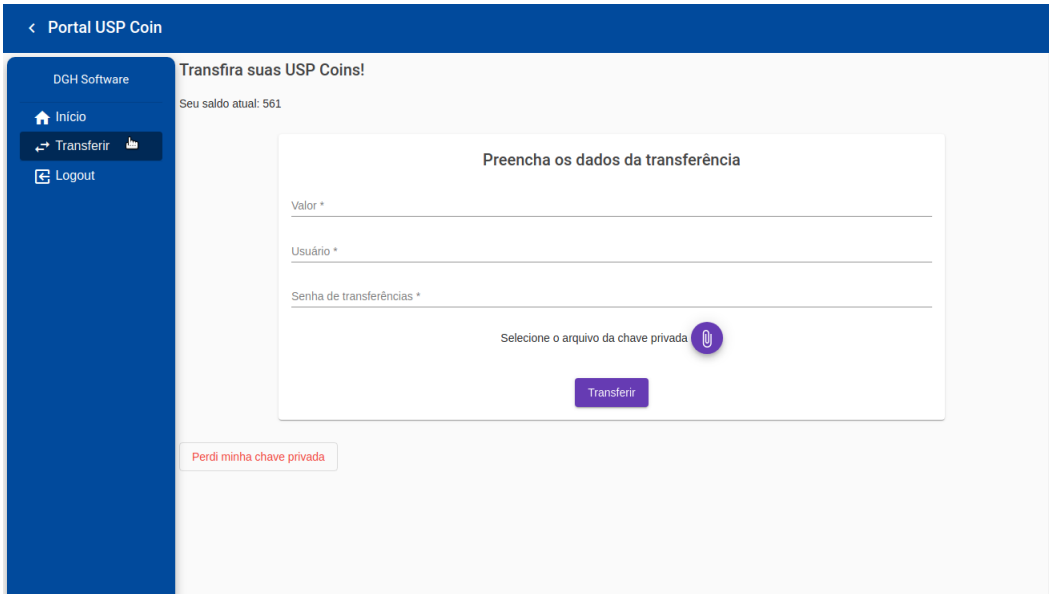

Figura 14: Tela de Transferência

Fonte: Autoria Própria

5.4.2.2 Servidor

O servidor desenvolvido para o sistema, como mencionado anteriormente, é o responsável por implementar a lógica de negócio e realizar a conexão com o banco de dados e a rede blockchain. Seu projeto foi pensado para que obedecesse aos princípios SOLID e, desta forma, o sistema fosse de fácil manutenção e extensão de suas funcionalidades.

- **Estrutura**

  A estrutura do sistema pode ser vista na figura 15, juntamente com as principais tecnologias utilizadas em algumas das entidades do sistema. As classes controladoras são as responsáveis por atender as requisições que chegam nas rotas do servidor e invocar métodos das classes de serviço. Estas, por sua vez, são responsáveis por efetuarem todas as lógicas das classes de domínio, como por exemplo criação e atualização das entidades (e.g. a classe de serviço de usuários é responsável por criar e atualizar objetos da classe Usuário). Uma vez que os serviços realizam a lógica de negócio, é necessário persistir estas informações ou no banco de dados (invocando métodos das classes de repositório) ou na blockchain (invocando métodos da classe Cliente Blockchain).

  Desta forma, as classes ficam enxutas e altamente especializadas, promovendo uma alta modularização e um baixo acoplamento. As vantagens de um sistema organizado desta forma, onde uma classe não depende de como o método de outra classe é

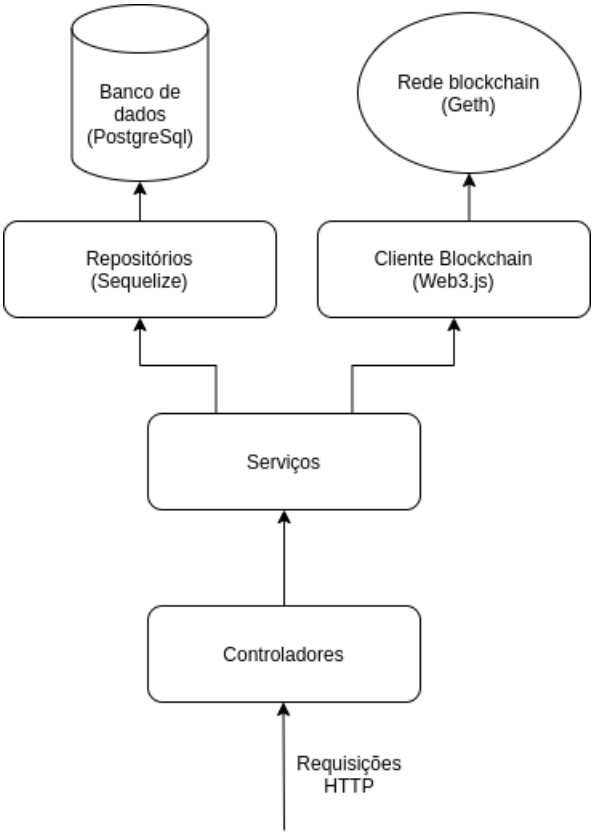

Figura 15: Diagrama do servidor e principais tecnologias

Fonte: Autoria Própria

implementado, são muitas. Por exemplo, se optar-se por não se utilizar a biblioteca *web3*, apenas a classe de Cliente Blockchain precisa ser modificada, pois as classes que invocam seus métodos não são afetadas por como os respectivos métodos são implementados. O mesmo pode ser dito para o caso de optar-se por utilizar uma ORM que não a *Sequelize*, ou mesmo não utilizar nenhuma ORM, caso no qual apenas as classes de repositório sofreriam alterações, enquanto as classes que as invocam ficariam intactas.

- **Proteção das rotas**

  O sistema conta com autenticação por token do tipo JWT, como explicado na seção 5.4.1.1. Para garantir que apenas usuários autorizados pudessem acessar os recursos das rotas definidas para este servidor, foi utilizado o conceito de *middlewares* da biblioteca *Express.js*. Os *middlewares* nada mais são do que funções que são chamadas antes do código definido para aquela rota ser executado.

  Para este sistema, foram definidas duas funções que foram usadas como *middlewares*. A primeira delas é responsável por verificar a presença de um token JWT no header

do *request* HTTP e verificar sua validade, lançando uma exceção caso o token esteja inválido ou ausente, garantindo, assim, que apenas usuários devidamente logados tenham acesso ao recurso das rotas nas quais este *middleware* é chamado. A segunda função faz a mesma validação que a primeira, mas com uma exigência maior: o token não deve apenas estar válido, mas deve pertencer a um usuário cadastrado como administrador do sistema. Desta forma, os recursos destas rotas apenas são acessíveis por usuários com este nível de permissão.

- **Documentação das APIs**

  A fim de ter-se todas as rotas devidamente documentadas e fáceis de testar, foi utilizada a ferramenta *tsoa* [40], que automaticamente gera a documentação das rotas do seu servidor com base nos tipos de retorno (uma das vantagens de se utilizar Typescript ao invés de Javascript). A documentação é gerada de acordo com o protocolo OpenAPI [41], tornando possível utilizar o *Swagger* [42] (Figuras 16 e 17), ferramenta capaz de subir um pequeno servidor a partir de uma documentação que segue o OpenAPI, e permite que todas as rotas sejam acessadas por meio de uma URL, possibilitando a visualização da rota, do tipo de *request* HTTP, dos seus parâmetros e do tipo de retorno.

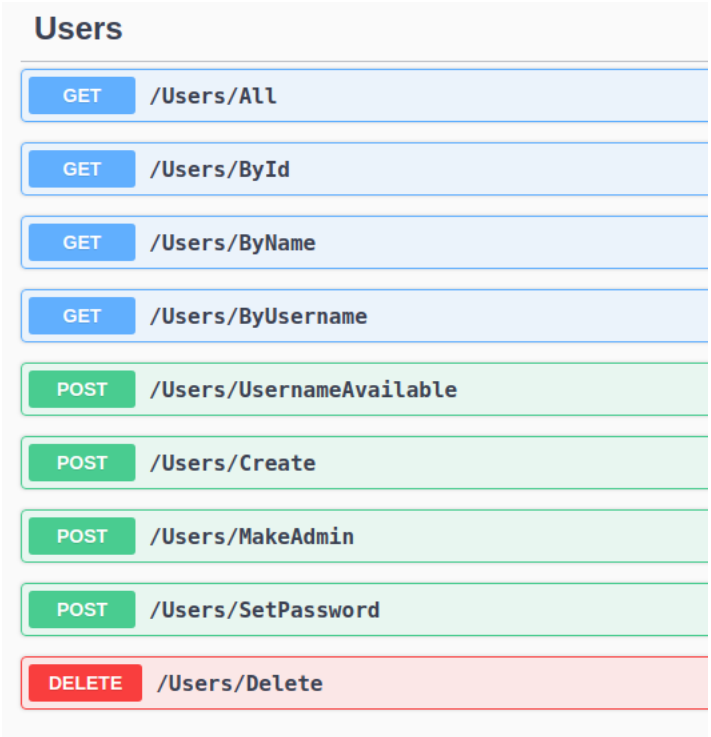

Figura 16: Página do swagger com rotas relativas aos usuários

Fonte: Autoria Própria

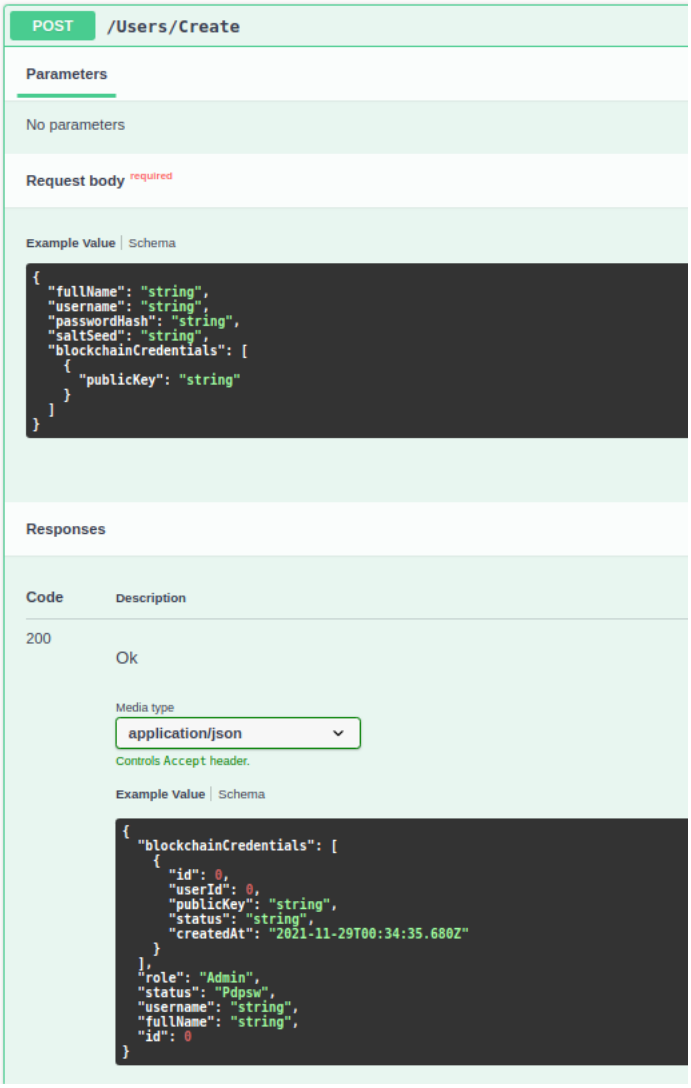

Figura 17: Tela do Swagger para fazer uma requisição de criação de usuário ao servidor

Fonte: Autoria Própria

6 TESTES E AVALIAÇÃO

O trabalho aqui apresentado teve como objetivo principal servir de prova de conceito para um futuro projeto da USP que pode ser implementado pela equipe técnica da universidade, propondo-se mostrar que é possível desenvolver um sistema que cumpra os requisitos funcionais propostos, utilizando a tecnologia de blockchain. Desta forma, não houve foco durante o desenvolvimento na performance do sistema. Entretanto, pode-se ainda fazer testes para avaliar o desempenho da aplicação em algumas tarefas que esta se propõe em realizar. Os testes realizados serão descritos nos itens a seguir.

6.1 Teste de bloqueio de chave perdida e ativação de nova chave

Este teste foi escolhido pois trata-se do fluxo com maior número de requisições ao banco de dados e a rede blockchain. Para realizar este teste, separamos 10 usuários do sistema, logamos em suas contas e solicitamos a criação de uma nova chave clicando no botão "Perdi minha chave privada". Após concluído este processo para todas as 10 contas, logamos em uma conta administradora e realizamos a ativação da nova chave gerada, processo que envolve escrita no banco de dados para atualizar o *status* da chave e submissão de transações na blockchain, tanto para bloquear a chave antiga quanto para dar a role de "Conta Ativa"à nova. Para obtermos os tempos de cada uma das 10 operações, utilizamos a ferramenta de desenvolvedor presente no navegador *Google Chrome*.

Foi observado que, em média, esta operação leva cerca de 60,64 segundos. Este tempo pode ser dividido nas seguintes requisições para o servidor: 4 requisições para submeter transações na blockchain (dar à chave antiga a role de "Conta Bloqueada", à chave nova a de "Conta Ativa", transferir *Ethers* à chave nova e minerar a mesma quantidade de *tokens* que havia na antiga para a nova), e 2 requisições para escrita no banco de dados (atualizar os *status* das respectivas chaves). Observando o tempo de cada uma destas

requisições, pôde-se observar que as de submeter transações na blockchain levavam em média 14,6 segundos, totalizando 58,4 segundos, mais de 96% do tempo médio total da operação. Por este motivo, optamos por realizar dois testes separados, um de apenas submissão de transações na blockchain e outro apenas de escrita no banco de dados, que serão detalhados nas seções a seguir.

6.2 Teste de transferência de *tokens*

Para realizar este teste, selecionamos uma chave aberta em nossa rede blockchain (aqui, entende-se por aberta uma chave cujas transações não precisam ser assinadas para serem submetidas na rede) e, com ela, realizamos diversas transferências de *tokens* para uma outra conta. Justamente por ela ser aberta, foi possível realizar a operação via servidor, não sendo necessário o cliente assinar a transação. A realização do teste foi feita utilizando-se o programa *Postman*, mais especificamente a sua ferramenta *Runner*, que permite efetuar diversas requisições a uma API de maneira sequencial, com um intervalo de tempo definido entre as mesmas.

A Figura 18 apresenta o tempo levado por 10 requisições feitas para submissão de uma transação de transferência, com um intervalo de 0 segundos entre elas. É possível perceber que o tempo gasto é de aproximadamente 15 segundos, o que é condizente com a configuração que foi feita da rede, pois ela leva 15 segundos para minerar um novo bloco e, portanto, para confirmar a submissão de uma nova transação. A primeira requisição foge do padrão justamente por ser a primeira e, portanto, foi submetida no meio deste intervalo de 15 segundos.

6.3 Teste de mudança do *status* de chave no Banco de Dados

Neste teste, queríamos verificar o tempo médio de uma requisição de escrita no banco de dados. Para isso, selecionamos uma chave pública e acionamos as rotas de bloqueio e ativação de chave do servidor, de maneira alternada, 5 vezes cada uma. Isto foi feito para que o sistema não lançasse uma exceção por tentativa de ativar uma conta já ativa ou bloquear uma já bloqueada e, assim, interrompesse o teste. A ferramenta utilizada foi a mesma do teste anterior.

A figura 19 mostra o tempo gasto por estas requisições. O tempo médio gasto foi

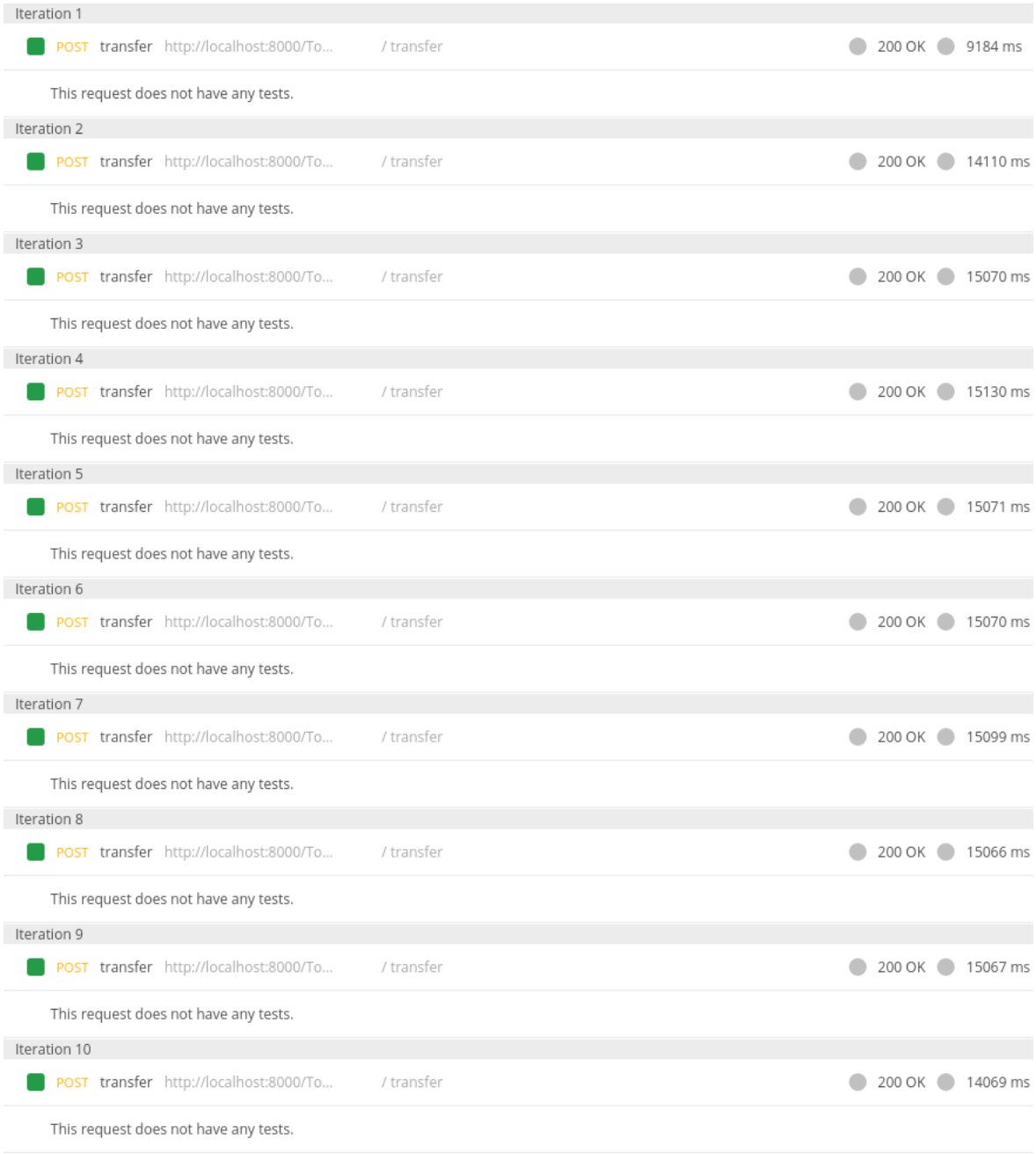

Figura 18: Testes de transferência realizados na ferramenta Postman

Fonte: Autoria Própria

de 0,21 segundos, o que demonstra que as operações de escrita em nosso banco de dados estão ocorrendo rapidamente, mesmo com o servidor de banco de dados estando em *North Virginia*, nos Estados Unidos. Pode-se concluir, portanto, que o maior gargalo do sistema são as operações de submissão de transações na rede blockchain.

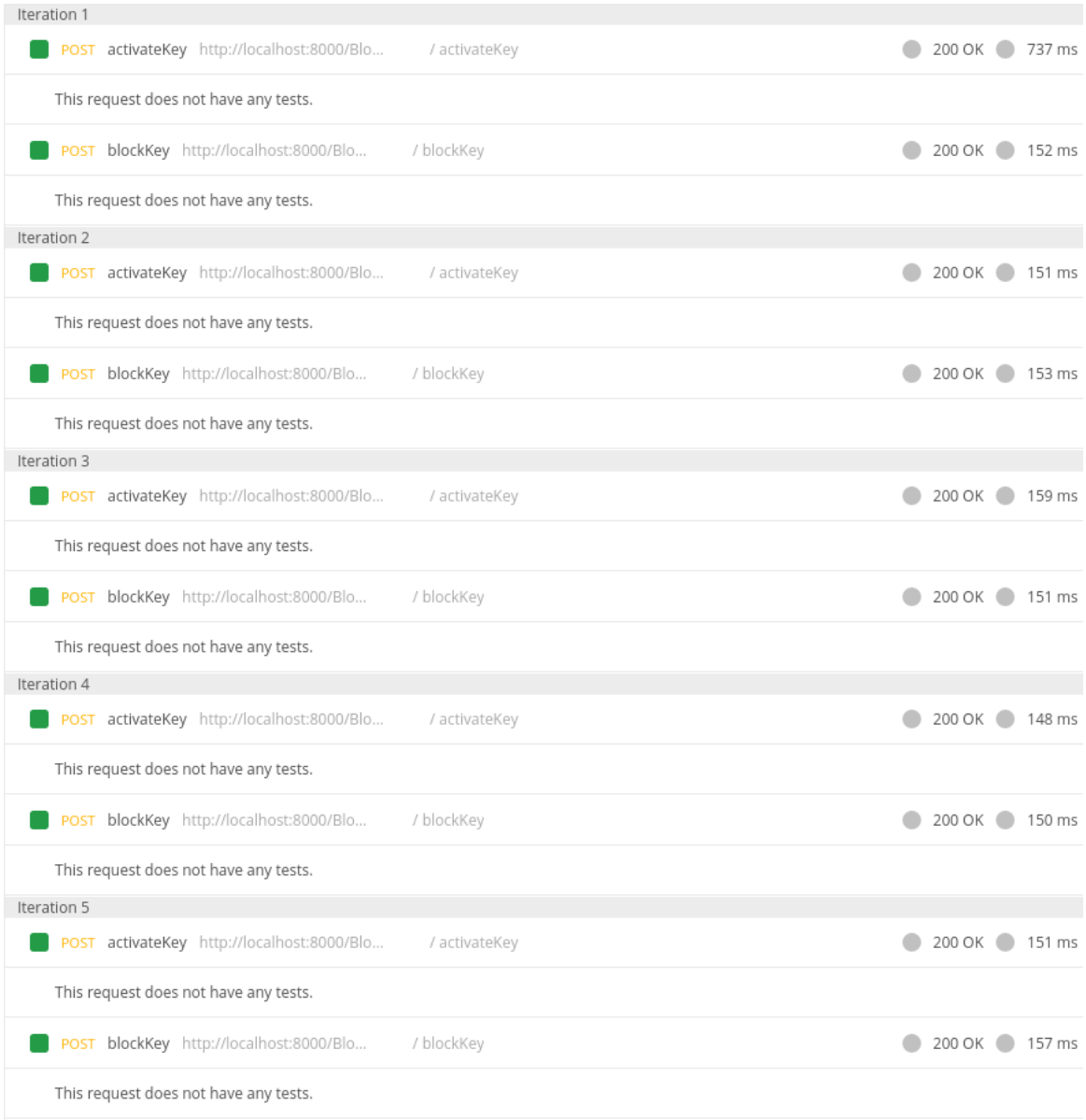

Figura 19: Testes de Ativação e bloqueio de chave no banco de dados

Fonte: Autoria Própria

7 CONSIDERAÇÕES FINAIS

O papel das *startups* está se tornando cada vez mais importante em diversas áreas da sociedade, como economia e avanços tecnológicos, e é de suma importância que a Universidade de São Paulo tenha projetos que acompanhem o avanço deste setor. Projetos que aproximem a universidade de empresas emergentes são o primeiro passo para que a contribuição da USP para a evolução da sociedade ganhe uma nova e importante frente. Além disso, estar presente nos ciclos iniciais do desenvolvimento de uma empresa pode trazer diversos benefícios, tanto em questões de ganhos econômicos, como aumento do prestígio da instituição acadêmica.

Este projeto teve como principal objetivo construir uma prova de conceito para uma das iniciativas de aproximação da USP com as *startups*, através da tokenização de ativos. O propósito foi demonstrar que é possível construir um sistema seguro, baseado na tecnologia blockchain, que permita à USP adquirir parte de empresas nos estágios iniciais de seu desenvolvimento em troca de uma determinada quantidade de uma moeda virtual; estas empresas, por sua vez, podem utilizar estas moedas recebidas para contratar serviços da Universidade, o que muitas vezes não é possível atualmente pois falta-lhes recursos financeiros devido ao próprio estágio de desenvolvimento que se encontram. Esta moeda virtual pode ser ainda utilizada dentro da universidade, como em troca de serviços entre laboratórios, pagamento de bolsistas, e para acesso a serviços diversos dentro do campus da universidade (e.g., nos restaurantes universitários).

Por não se propor a ser um projeto pronto para ser utilizado em produção, alguns pontos não foram priorizados, como o desempenho da rede blockchain e do servidor, como é possível perceber nos testes apresentados no capítulo 6. Portanto, o sistema é passível de melhorias, como a inclusão de novas funcionalidades para atender os casos de uso especificados e outros que possam vir a surgir, ou o estudo de uma implementação arquitetural mais eficiente, além da utilização de máquinas mais modernas e potentes.

Quanto ao desenvolvimento, as ferramentas utilizadas se mostraram de extrema utilidade. A biblioteca *web3* tornou a integração do sistema com a blockchain e o *smart*

contract possível de forma muito simples, e acreditamos ser recomendável a sua utilização em futuras etapas do projeto. Um ponto que vale ressaltar, porém, é que não conseguimos fazer com que a função de assinatura do contrato fosse feita de forma completamente independente da blockchain, o que fazia parte da proposta original desta função, para evitar que um dispositivo qualquer precise entrar em contato com a rede. Ao assinar a transação são feitas três requisições para verificar o estado da rede. Uma possível solução seria redirecionar estas requisições para o servidor, e este faria as chamadas e retornaria a resposta para o cliente. Entretanto, embora a comunicação do cliente com a blockchain ainda precise existir, a assinatura acontece no cliente, de modo que a chave privada do usuário não precisa trafegar pela rede ou pelo sistema, o que era o requisito principal desta funcionalidade.

A única ferramenta que tivemos grande dificuldade e que não recomendamos para eventuais continuações do projeto é a ORM escolhida, Sequelize, devido a qualidade de sua documentação. Durante o processo de desenvolvimento foi necessário remodelar as classes correspondentes as tabelas do banco de dados, uma vez que as funcionalidades desta ferramenta criavam dificuldades que levaram ao atraso de algumas partes do projeto. Uma alternativa possível é a TypeORM [43], também utilizável em ambiente Node e para bancos de dados relacionais.

Quanto a funcionalidades que podem ainda ser implementas em uma versão de produção do sistema destaca-se a conversão de Moedas USP em moeda fiduciária e a validação da identidade dos usuários. Como explicado na seção 4.2.2, a conversão da moeda é um caso de uso que ainda está em definição por parte da equipe econômica, e pode ser implementado uma vez que o sistema atinja uma maior maturação. Já a validação dos usuários pode ser feita uma vez que o sistema esteja integrado ao restante dos sistemas universitários. Na etapa de ativação das chaves, o administrador do sistema pode realizar a validação a partir de informações coletadas no cadastro de usuários e cruzando-as com dados da universidade.

Embora o sistema implementado, em seu formato atual, não esteja plenamente pronto para ser implantado dentro da universidade, ele cumpre seu objetivo como prova de conceito, mostrando que é viável desenvolver um sistema que cumpra os requisitos propostos: emitir uma moeda lastreada em ativos reais, participação em *startups*, que pode ser trocada entre membros do sistema de forma segura. A especificação de tais funcionalidades, bem como os módulos essenciais utilizados para sua construção podem, assim, ser usados como base para a construção de uma solução integrada aos sistemas da universidade. Desta forma, aliada a processos definidos para sua utilização pelos diversos atores do am-

biente universitário, tem-se grande potencial para que o sistema criado traga benefícios tangíveis à comunidade USP, podendo ainda ser expandido para abarcar outras universidades parceiras caso desejado. De fato, encontra-se em andamento a discussão entre Reitoria da USP e InovaUSP para fazer com que o sistema aqui proposto se torne uma solução institucional, iniciativa esta que tem sido bem recebida pelos órgãos administrativos da universidade. Colaborar com essa empreitada é, portanto, o próximo passo natural deste projeto.

REFERÊNCIAS

[1] Startbase. *Estatísticas*. URL: https://startupbase.com.br/home/stats (acesso em 26/06/2021).

[2] DISTRITO. *Inside Venture Capital Brasil*. Jun. de 2021. URL: https://materiais.distrito.me/dataminer-inside-venture-capital-brasil (acesso em 27/06/2021).

[3] *O Inova*. URL: https://inova.usp.br/o-inova/ (acesso em 27/06/2021).

[4] Alex Nascimento. *As novas tendências em blockchain e criptoativos para 2021*. Jan. de 2021. URL: https://exame.com/blog/alex-nascimento/as-novas-tendencias-em-blockchain-e-criptoativos-para-2021/ (acesso em 27/06/2021).

[5] Fran Casino, Thomas K. Dasaklis e Constantinos Patsakis. "A systematic literature review of blockchain-based applications: Current status, classification and open issues". Em: *Telematics and Informatics* 36 (2019), pp. 55–81. ISSN: 0736-5853. URL: https://www.sciencedirect.com/science/article/pii/S0736585318306324.

[6] Patrick Schueffel, Nikolaj Groeneweg e Baldegger Rico. "The Crypto Encyclopedia: Coins, Tokens and Digital Assets from A to Z". Em: Growth publisher, ago. de 2019, p. 63. ISBN: 978-2-940384-47-1. URL: http://hesso.tind.io/record/3815.

[7] Antoni Pérez-Villegas Morey. "TSmart Token for the university using Blockchain". Diss. de mestr. National Taiwan University of Science e Technology, Escola Tècnica Superior d'Enginyeria Industrial de Barcelona, 2017.

[8] *Student Coin*. URL: https://www.studentcoin.org/ (acesso em 29/11/2021).

[9] MIT. *A glossary of blockchain jargon*. MIT Technology Review. 2018.

[10] Satoshi Nakamoto. "Bitcoin: A peer-to-peer electronic cash system". Em: *Decentralized Business Review* (2008), p. 21260.

[11] G. Greenspan. *Ending the bitcoin vs blockchain debate*. 2015a. URL: http://www.multichain.com/blog/2015/07/bitcoin-vs-blockchain-debate.

[12] Konstantinos Christidis e Michael Devetsikiotis. "Blockchains and smart contracts for the internet of things". Em: *IEEE Access* 4 (2016), pp. 2292–2303.

[13] J. Zhao, Shaokun Fan e Jiaqi Yan. "Overview of business innovations and research opportunities in blockchain and introduction to the special issue". Em: *Financial Innovation* 2 (dez. de 2016). DOI: 10.1186/s40854-016-0049-2.

[14] IBM. *Three ways blockchain Explorers chart a new direction*. 2017. URL: https://www-935.ibm.com/services/studies/csuite/pdf/GBE03835USEN-00.pdf (acesso em 27/06/2021).

[15] Jameela Al-Jaroodi e Nader Mohamed. "Blockchain in Industries: A Survey". Em: *IEEE Access* 7 (2019), pp. 36500–36515. DOI: 10.1109/ACCESS.2019.2903554.

[16] Md. Sadek Ferdous et al. "Blockchain Consensus Algorithms: A Survey". Em: *arXiv: Distributed, Parallel, and Cluster Computing* (jan. de 2020).

[17] Daniel Davis Wood. "Ethereum: A secure decentralised generalised transaction ledger". Em: *Ethereum project yellow paper* 151.2014 (2014), pp. 1–32.

[18] *Ethereum Development Documentation*. URL: https://ethereum.org/en/developers/docs/ (acesso em 27/06/2021).

[19] Massimo Bartoletti. "Smart Contracts Contracts". Em: *Frontiers in Blockchain* 3 (2020), p. 27. ISSN: 2624-7852. DOI: 10.3389/fbloc.2020.00027.

[20] Nick Szabo. "Formalizing and Securing Relationships on Public Networks". Em: *First Monday* 2.9 (set. de 1997). DOI: 10.5210/fm.v2i9.548.

[21] Smriti Verma. *Proof-of-Authority*. URL: https://forum.openzeppelin.com/t/proof-of-authority/3577 (acesso em 27/06/2021).

[22] Robert C Martin. "Design principles and design patterns". Em: *Object Mentor* 1.34 (2000), p. 597.

[23] R.C. Martin et al. "Agile Software Development: Principles, Patterns, and Practices". Em: Alan Apt series. Pearson Education, 2003, pp. 95–131. ISBN: 9780135974445.

[24] *Design the infrastructure persistence layer - The Repository pattern*. URL: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design (acesso em 27/06/2021).

[25] *Digiconomist*. URL: https://digiconomist.net/ (acesso em 27/06/2021).

[26] *Bitcoin Energy Consumption Index*. URL: https://digiconomist.net/bitcoin-energy-consumption (acesso em 27/06/2021).

[27] *Ethereum Energy Consumption Index*. URL: https://digiconomist.net/ethereum-energy-consumption (acesso em 27/06/2021).

[28] *About Node.js*. URL: https://nodejs.org/en/about/ (acesso em 27/06/2021).

[29] *What is TypeScript?* URL: https://www.typescriptlang.org/ (acesso em 27/06/2021).

[30] *Documentation*. URL: https://www.typescriptlang.org/ (acesso em 27/06/2021).

[31] *Truffle Smart Contracts made weeter*. URL: https://www.trufflesuite.com/truffle (acesso em 27/06/2021).

[32] *Ganache overview*. URL: https://www.trufflesuite.com/docs/ganache/overview (acesso em 27/06/2021).

[33] *web3.js - Ethereum JavaScript API*. URL: https://web3js.readthedocs.io/en/v1.3.4/ (acesso em 27/06/2021).

[34] *Sequelize*. URL: https://sequelize.org/master/index.html (acesso em 27/11/2021).

[35] Stefano De Angelis et al. "PBFT vs proof-of-authority: applying the CAP theorem to permissioned blockchain". Em: Italian Conference on Cybersecurity, jan. de 2017.

[36] *issue 9997: Remove/split out account management*. URL: https://github.com/openethereum/parity-ethereum/issues/9997 (acesso em 14/10/2021).

[37] *Private Networks*. URL: https://geth.ethereum.org/docs/interface/private-network (acesso em 15/10/2021).

[38] Scott Contini. "Method to Protect Passwords in Databases for Web Applications". Em: *IACR Cryptol. ePrint Arch.* 2015 (2015), p. 387.

[39] Michael Jones. "JSON Web Algorithms (JWA)". Em: Request for Comments 7518 (mai. de 2015). DOI: 10.17487/RFC7518. URL: https://rfc-editor.org/rfc/rfc7518.txt.

[40] *tsoa - Getting started*. URL: https://tsoa-community.github.io/docs/getting-started.html (acesso em 28/11/2021).

[41] *OpenAPI Initiative*. URL: https://www.openapis.org/ (acesso em 28/11/2021).

[42] *Swagger - API Development for Everyone*. URL: https://swagger.io/ (acesso em 28/11/2021).

[43] *TypeORM*. URL: https://typeorm.io/#/ (acesso em 02/12/2021).