UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
ESCOLA DE INFORMÁTICA APLICADA

SHADOW WALLET: UMA CARTEIRA DE CRIPTOMOEDAS FOCADA EM PRIVACIDADE

DANIEL LUCAS JASPONDE CALONICO
GUSTAVO EMMANUEL JASPONDE CALONICO

ORIENTADOR
JEFFERSON ELBERT SIMÕES

RIO DE JANEIRO, RJ – BRASIL
FEVEREIRO DE 2022

DANIEL LUCAS JASPONDE CALONICO
GUSTAVO EMMANUEL JASPONDE CALONICO

SHADOW WALLET: UMA CARTEIRA DE CRIPTOMOEDAS FOCADA EM PRIVACIDADE

Projeto de Graduação apresentado à Escola de Informática Aplicada da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Jefferson Elbert Simões

Rio de Janeiro, RJ – Brasil
Fevereiro de 2022

C165 Calonico, Daniel L. J.; Calonico, Gustavo E. J.

Shadow Wallet: Uma Carteira de Criptomoedas Focada Em Privacidade / Daniel Lucas Jasponde Calonico, Gustavo Emmanuel Jasponde Calonico.
-- Rio de Janeiro, 2022.

99

Orientador: Jefferson Elbert Simões.

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal do Estado do Rio de Janeiro, Graduação em Sistemas de Informação, 2022.

1. Blockchain. 2. Monero. 3. Criptomoedas. 4. Privacidade. I. Elbert Simões, Jefferson, orient. II. Título.

SHADOW WALLET: UMA CARTEIRA DE CRIPTOMOEDAS FOCADA EM PRIVACIDADE

DANIEL LUCAS JASPONDE CALONICO
GUSTAVO EMMANUEL JASPONDE CALONICO

Projeto de Graduação apresentado à Escola de Informática Aplicada da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) para obtenção do título de Bacharel em Sistemas de Informação.

Aprovado em 18 de Fevereiro de 2022 pela banca examinadora:

_____

Prof. JEFFERSON ELBERT SIMÕES (Orientador)

_____

Prof. MÁRCIO DE OLIVEIRA BARROS

_____

Prof. LEONARDO LUIZ ALENCASTRO ROCHA

Rio de Janeiro, RJ – Brasil
Fevereiro de 2022

Agradecimentos

Agradecemos à Dona Graça, nossa mãe, por estar sempre presente e apoiando esta longa jornada acadêmica de 7 anos.

RESUMO

Neste trabalho, apresentamos a *Shadow Wallet*, uma carteira de criptomoedas, *Free and Open-Source Software,* para armazenamento e uso diário de criptomoedas focadas em privacidade. Seu objetivo é facilitar a adoção das criptomoedas com ênfase na privacidade por usuários de criptomoedas. Neste texto é feita uma revisão técnica sobre a *blockchain*, tecnologia base do *Bitcoin*, relacionando seus indicadores econômicos recentes à sua relevância tecnológica e apresentando seus aspectos que tiveram impacto direto na arquitetura da Shadow Wallet. Finalmente, é descrita a implementação do *software*, detalhando o uso das linguagens, frameworks e ferramentas no seu processo de desenvolvimento.

Palavras-chave: Blockchain, Monero, Carteira de Criptomoedas, Criptomoedas, Privacidade.

ABSTRACT

In this work, we present the *Shadow Wallet,* a Free and Open-Source crypto wallet application, for daily use. Its objective is to ease the adoption of privacy-focused cryptocurrencies, by users of cryptocurrencies. In this paper, we make a technical analysis of the blockchain, Bitcoin's base technology, relating its relevance to economic indicators, and highlighting important aspects of its technology that impacted the architecture of the Shadow Wallet application. Finally, the software implementation is described, detailing the use of all tools, languages, frameworks and theoretical analysis in its development process.

Keywords: Blockchain, Monero, Cryptowallets, Cryptocurrencies, Privacy.

SUMÁRIO

LISTA DE FIGURAS

1 Introdução

A *blockchain* tornou-se uma tecnologia importante no mercado financeiro digital, por ser a tecnologia base dos criptoativos. Sua implantação surgiu como fundamento do *Bitcoin* [1], o primeiro projeto capaz de criar um sistema digital de moeda *peer-to-peer* (P2P), que não depende de uma autoridade centralizada para emiti-la, gerenciá-la ou validá-la. Tal autoridade está distribuída entre os participantes desta rede, que mantém e atualizam o estado da mesma, trabalhando e comunicando entre si para chegar a um consenso sobre este estado.

1.1 – Motivação

Com uma capitalização total de US$ 2 Trilhões [2], o mercado de criptoativos mostrou um crescimento constante na última década, tendo uma excelente performance durante 2020, mesmo em meio à pandemia de COVID-19 [3]. Isto comprova a relevância econômica da *blockchain*, mesmo ela sendo uma tecnologia com apenas 11 anos de idade. Seu protocolo básico foi amplamente adaptado e aplicado em diversos campos da tecnologia da informação, com propostas para soluções nos campos de Big Data, AI, 5G e Computação em Nuvem, entre outros [4].

Com tamanha relevância econômica, a *blockchain* se torna cada vez mais comum em projetos, com as mais diversas aplicações. Por ser uma tecnologia nova, o conhecimento pouco consolidado abre uma oportunidade para desenvolvedores nesta área, que se tornam importantes não apenas na engenharia desses projetos, mas na boa aplicação dos conceitos e fundamentos da blockchain em sua implementação.

1.2 – Objetivos

Este trabalho de graduação possui 3 objetivos primários:

- Consolidar conhecimento sobre a *blockchain*, suas tecnologias e aplicações;
- Discorrer sobre as limitações de uma *blockchain*, e suas implicações;
- Implementar uma solução em software livre e de código aberto (*free and open-source software (FOSS)*) [5], que facilite o uso de criptomoedas com foco em privacidade.

A *blockchain* é uma tecnologia recente, com conhecimento de pouca consolidação acadêmica. Discussões e trabalhos relevantes sobre *blockchain* estão documentadas em repositórios de código, fóruns sobre criptomoedas e *whitepapers*. Tais documentações possuem finalidades práticas, como propostas, investigações e sugestões para melhoria de código da *blockchain*, ou discussões sobre o estado da governança desta rede.

Os *whitepapers* são documentos que detalham o funcionamento e governança de um projeto que usa *blockchain*, mas não possuem objetivo acadêmico. Seu foco é convencer possíveis investidores na viabilidade de algum projeto, para aquisição de capital que permita o início deste projeto. Tais documentos possuem poucos detalhes técnicos, estando estes na implementação do próprio código do projeto.

No capítulo de revisão dos trabalhos relacionados, será feita uma análise sobre a *blockchain* e seus conceitos base, ilustrando seu funcionamento com exemplos em termos de *Bitcoin*. Depois, será expandida sua análise para outros casos de uso da tecnologia, abordando o estado em que se encontra seu uso.

Das limitações, será discorrido sobre o estado da distribuição do poder computacional no *Bitcoin*, e como isto afeta a governança da rede. Também será abordada a privacidade de dados em uma rede *blockchain* tradicional, que foi originalmente criada como um livro de registros transparentes, onde seus dados podem ser lidos por qualquer indivíduo que possuir acesso à sua cópia. Isto traz implicações sobre sua privacidade, pois é possível mapear o estado de cada usuário de uma *blockchain*, e saber todas as mensagens que o mesmo enviou e recebeu, determinando seu saldo. No *Bitcoin*, é possível usar os dados transparentes de sua *blockchain*, junto a outros dados externos, para descobrir quem são os indivíduos donos de alguma carteira.

Finalmente, neste trabalho é detalhado o projeto e desenvolvimento da *Shadow Wallet,* um *software* livre e de código aberto de uma carteira de criptoativos, com foco no

gerenciamento de criptomoedas privadas. Seu objetivo é facilitar a utilização de criptomoedas privadas por qualquer grupo ou indivíduo, permitindo o recebimento, envio e verificação de criptomoedas de maneira custodial. Desta forma, todas as chaves e informações ficariam sob custódia do usuário em seu dispositivos, e operações com a *blockchain* são feitas localmente, dispensando o uso de um servidor com back-end.

1.3 – Organização do Texto

Este Trabalho de Conclusão de Curso (TCC) está estruturada nos seguintes capítulos:

- Capítulo 2: Trabalhos Relacionados, com uma revisão técnica e econômica da *blockchain* e suas aplicações mais populares.
- Capítulo 3: Projeto de Aplicativo, com a motivação, objetivos e descrição geral da arquitetura da *Shadow Wallet*.
- Capítulo 4: Implementação, com o processo de construção do *software* da *Shadow Wallet* e detalhes das suas etapas.
- Capítulo 5: Conclusões, com reflexões sobre este trabalho.

2 Trabalhos Relacionados

Neste capítulo será explicada a *blockchain*, seu protocolo, atores, entidades e aplicações, de forma ampla. É importante ressaltar a diferença entre *Bitcoin* e *blockchain*: o primeiro é a moeda digital; o segundo é um protocolo, que faz parte do conjunto de protocolos do *Bitcoin*, e permite o funcionamento do mesmo. Ou seja, o *Bitcoin* usa uma *blockchain*.

Como o *Bitcoin* é precursor à *blockchain*, o termo "*blockchain*" originalmente dá nome apenas ao livro-mestre de transações, não sendo, de fato, o protocolo ou o conjunto de protocolos que constituem o sistema do *Bitcoin*. Entretanto, a *blockchain* ganhou outras aplicações, e seu uso foi estendido para além do *Bitcoin*.

O foco deste trabalho está na aplicação ampla da *blockchain.* Desta forma, qualquer menção à palavra "protocolo" e "*blockchain*" remete ao conjunto base de protocolos que permitem o funcionamento do *Bitcoin* e outras criptomoedas. A *blockchain* será explicada de forma genérica e, quando necessário, serão usados exemplos do *Bitcoin* para demonstrar seu funcionamento.

2.1 – Blockchain

A *blockchain* é um protocolo para ordenar mensagens em uma rede onde os participantes não se confiam, de forma que estes tenham um consenso sobre o estado desta rede, sem o uso de uma autoridade "confiável" de *timestamp,* que autentique ou defina a ordem dessas mensagens. Estas mensagens alteram o estado desta rede, e as pessoas que desejam participar de uma *blockchain* aceitam o contrato definido no código do próprio protocolo, sendo responsáveis pelo envio, manutenção, validação e propagação das mensagens que circulam nesta rede. É importante que cada participante se comunique de forma eficaz, mantendo uma

cópia de todas as mensagens que recebeu e verificando se sua cópia concorda com a cópia de todos os outros participantes da rede, ajudando assim a manter o estado da rede e, consequentemente, sua integridade

Para tal, um conjunto de regras é definido no código do protocolo da *blockchain*, para incentivar a participação justa entre usuários da mesma. Caso o sistema para qual se propõe uma *blockchain* tenha participação de usuários confiáveis, ou seja necessário marcar a *timestamp* precisa do momento no qual a mensagem é enviada ou recebida, é preferível usar uma autoridade de *timestamp*. Da mesma forma, se a ordem das mensagens não for importante, não é necessário utilizar *blockchain*.

Sua característica de maior importância é ter um registro único e difícil de ser alterado, que permite gerar escassez de informações em sistemas digitais e redes distribuídas, como a internet. Para o *Bitcoin*, o primeiro sistema a usar este protocolo, era importante que transações não pudessem ser livremente alteradas pelos usuários da rede, e que apenas o usuário dono de algum saldo de *bitcoins* pudesse o gastar. Assim, foi possível criar o primeiro sistema financeiro digital completamente distribuído, descentralizado e íntegro, que obteve sucesso e amplo uso na internet.

Usuários participantes de uma rede *blockchain* enviam mensagens, com transações de valores pela *internet.* Através dos blocos de mensagens é possível registrar transações em um histórico de blocos encadeados, usados na mitigação da concorrência de mensagens na *blockchain*, ajudando na busca de consenso entre os nodos sobre o estado da rede e sobre o saldo dos participantes, permitindo verificar se um nodo possui *bitcoins* para gastar ou não.

Nodos mineradores são encarregados de criar os blocos e, para isso, gastam poder computacional na criação de uma prova de trabalho, garantindo a autenticidade e validade daquelas transações. Este processo de criação de blocos é conhecido como *mineração de blocos*, que é recompensada pela emissão de novos *bitcoins* ao minerador que for capaz de criá-lo, caso o bloco seja aceito por outros nodos daquela *blockchain*, e pelo recolhimento das taxas opcionais de transação, pagas por usuários da rede. Tais mecanismos buscam custear a operação de mineração e incentivam a participação justa entre mineradores.

2.1.1 – Usuário

Usuários participantes da *blockchain* são identificados por seus endereços de *chave pública* (PbK), gerados a partir de uma *chave privada* (PvK) utilizando o algoritmo de curvas elípticas [6], um tipo de criptografia assimétrica. Chaves públicas são geradas usando

transformações algébricas a partir de um ponto gerador em uma curva elíptica, que é somado *n* vezes, de acordo com o valor binário de uma chave privada. Isto torna difícil determinar qual a chave privada foi usada para gerar uma chave pública. Chaves privadas possuem relacionamento 1 para 1 com as chaves públicas, ou seja, uma chave pública pertence apenas a uma única chave privada.

Sabendo que aquela chave pública está associada a uma chave privada apenas, o protocolo permite o uso de identificadores secretos e únicos para os participantes. Este segredo não precisa ser revelado para autenticar a identidade de um usuário, já que o algoritmo de curvas elípticas garante que os resultados da assinatura de mensagens com este segredo remete a apenas um único autor.

Um usuário remetente A, que deseja enviar uma mensagem para um usuário destinatário B, utiliza sua chave privada para assinar esta mensagem, endereçada à chave pública do usuário B. Utilizando a chave pública de A, é possível autenticar a mensagem, já que só poderia ser gerado com a chave privada de A.

No *Bitcoin*, o conteúdo dessas mensagens contém transações monetárias, onde A envia uma quantia de *bitcoins* para B, e assina esta mensagem com sua chave privada, autenticando aquela transação.

2.1.2 – Nodo

Nodos são uma classe de usuários participantes deste sistema, que possuem uma cópia de todas as mensagens enviadas por todos os endereços de chave pública, desde o início da execução do protocolo. Estes nodos também são identificados por um par de chaves público-privadas, e são os principais responsáveis por receber e propagar as mensagens na rede, verificando com seus nodos vizinhos qual é o estado atual da rede e atualizando o seu próprio estado. Qualquer usuário da rede pode guardar uma cópia de todas as mensagens, se tornando um nodo.

Um usuário pode se comunicar com um nodo, que age como porta de entrada para propagar mensagens. Assim, um usuário não precisa ter uma cópia de todas as mensagens para participar da rede, com o porém de ter que confiar nas informações deste nodo, e que o mesmo vai entregar sua mensagem. Como o protocolo permite que todos os usuários sejam nodos em si, este usuário pode consultar diferentes nodos para verificar se as informações recebidas estão de acordo com a maioria na rede.

O conjunto de participantes com poder decisório é o que define a *rede da blockchain*. Usuários não possuem uma cópia de todas as mensagens da rede, e não conseguem confirmar a validade de uma mensagem e decidir se a mesma está de acordo com o histórico. Nodos, por possuírem esse histórico, conseguem determinar se uma mensagem é válida e está de acordo com o histórico da rede. Entretanto, a participação de usuários é o que popula uma *blockchain*, e seu poder de decisão está em usar aquela *blockchain* ou não. Portanto, qualquer menção futura à palavra *"rede"* remete sempre ao coletivo de usuários e nodos participantes de uma *blockchain*, com a distinção deste papéis anteriormente mencionados.

2.1.3 – Mensagens

Mensagens são criadas por usuários e nodos da rede, e são propagadas entre nodos participantes. Uma mensagem é *autêntica* apenas se o autor assina seu conteúdo utilizando sua chave privada, associada à chave pública. Pela definição do algoritmo de curvas elípticas, nodos são capazes de autenticar estas mensagens com pouco custo computacional, utilizando a chave pública do autor. Uma mensagem é *válida* apenas se, além de autêntica, está de acordo com o histórico da rede e não contradiz o conteúdo de mensagens passadas. Cada mensagem é única, não podendo ter assinaturas iguais para conteúdos diferentes.

Nodos que comprovam a autenticidade e validade de uma mensagem podem propagá-la para nodos vizinhos. Mensagens são propagadas por um protocolo de salto múltiplo (*Multi-Hop*) [7], em que há um *best-effort* dos nodos para entregar novas mensagens aos nodos vizinhos. Esta propagação ocorre com atraso, pois depende da disponibilidade dos enlaces físicos entre os nodos naquele momento. Isto permite a concorrência entre mensagens, e que nodos enviem transações com mensagens diferentes para vizinhos, propagando mensagens contraditórias.

Por exemplo, no *Bitcoin*, mensagens sobre transações contraditórias são classificadas como um ataque de gasto duplo ("*Double Spending Coins*"). Neste ataque, um usuário A gasta o mesmo saldo em duas transações diferentes, uma para um usuário B e outra para o usuário C, enviando as mensagens para nodos diferentes e distantes. Para evitar este tipo de concorrência, mensagens são agrupadas periodicamente, e a rede escolhe apenas uma das mensagens para entrar neste grupo, geralmente a que foi recebida primeiro pela maioria dos nodos. A inclusão desta mensagem automaticamente invalida a outra, pois estará em desacordo com o histórico da rede.

2.1.4 – Blocos

Por se tratar de um sistema assíncrono, onde a propagação de mensagens ocorre com atrasos, o protocolo faz uso dos *blocos,* que agrupam e consolidam mensagens enviadas na rede em um determinado período, tornando uma mensagem "confirmada". Este bloco facilita a propagação de mensagens válidas e a atualização do estado da rede por nodos participantes, pois protocolam um formato e intervalo específico para organizar as mensagens novas. Isto facilita que a rede chegue num consenso sobre o estado de cada usuário daquela *blockchain*.

Mensagens novas, válidas e não-confirmadas são propagadas entre nodos e esperam num *pool*, até que sejam incluídas em um bloco por algum nodo. É necessário identificar se aquela mensagem é autêntica e válida, antes de confirmá-la num bloco. Isso é importante para manter a integridade do sistema, prevenindo que usuários enviem mensagens que alterem o estado da rede de modo fraudulento.

O conteúdo do bloco consiste em um *hash*, o *hash* de seu bloco ancestral e o *payload*, a lista das mensagens confirmadas por aquele bloco. Blocos também possuem limite máximo de dados que podem ser incluídos, conhecido como tamanho do bloco. Eles são criados em sequência, e marcam a data e hora das mensagens que foram incluídas em seu corpo. Sua sequência registra um histórico de mensagens, análogos a uma página em um livro de transações de um banco.

Um bloco é válido quando todas as mensagens da lista são válidas, quando seu *hash* é válido e quando seu *hash* ancestral não pertence a nenhum outro bloco que já tenha sido incluído na cadeia de blocos. Os critérios de validade para o *hash* do próprio bloco estão descritos na próxima subseção.

Um novo bloco pode ser sugerido por qualquer nodo participante da rede. Apenas um bloco feito por um nodo pode ser incluído de cada vez, e blocos sucessores atualizam o estado da rede com novas informações sobre operações dos usuários, fazendo uma referência a um bloco ancestral, o atual bloco no final da cadeia. Esta característica de sequenciamento de blocos cunhou o termo *blockchain*, que se tornou seu nome. Nodos aceitam o novo bloco ao incluir seu *hash* no conteúdo do seu próximo bloco a ser sugerido.

O último bloco da rede pode ser lido e verificado até o bloco gênesis, para se obter um estado atual de todos os usuários da rede. No *Bitcoin*, por exemplo, os blocos são transparentes, e as informações de transação mostram exatamente o valor movimentado e as chaves públicas envolvidas nessa transação. Assim, é possível determinar o saldo exato de todos os usuários a qualquer momento. Apesar disso, apenas com os dados da própria rede do

Bitcoin é impossível determinar a identidade das pessoas físicas por trás de um usuário. É necessário consultar uma fonte de dados externa, cruzando com as transações para determinar quem é o dono de uma carteira. Ou seja, a rede do *Bitcoin* em si não é anônima, mas sim pseudônima, e preserva a identidade real de seus participantes.

Para validar se determinado usuário A tem saldo para fazer alguma transação de *bitcoins*, analisa-se todos os blocos até o bloco gênesis, verificando em quais blocos A recebeu ou enviou *bitcoins*. Sabendo o estado atual de todos os usuários da rede, caso A envie uma mensagem com uma transação maior que a soma de todos os seus recebimentos anteriores, é possível concluir que a mesma é inválida e não está de acordo com o histórico da rede.

Essa validação completa do histórico de qualquer *blockchain* exige que usuários possuam uma cópia de todos os blocos e mensagens da rede, o que pode ser computacionalmente custoso, como mencionado antes. Portanto, usuários podem optar por se comunicar com nodos, que possuem esta cópia e fazem a autenticação, validação e propagação de mensagens com transações válidas. Além disso, os nodos podem optar por carregar apenas a cópia das mensagens não-gastas da rede, se tornando "nodos leves" *(Light Nodes)*. Desta forma, é possível validar rapidamente uma mensagem nova, já que basta verificar se a nova mensagem faz alterações apenas em mensagens não-gastas.

Pela natureza assíncrona da rede, é possível que dois nodos incluam blocos válidos, com mensagens diferentes, ao mesmo tempo. Isto causa uma ramificação de blocos concorrentes, e torna o histórico da rede inconsistente. Eventualmente, a rede deve escolher qual dos ramos manter e qual deles descartar, pois só é possível adicionar o um *hash* de um bloco ancestral por vez. Assim, os blocos do ramo descartado se tornam "órfãos", e suas mensagens precisam ser incluídas novamente em um bloco posterior, para serem confirmadas. No *Bitcoin*, a rede deve escolher sempre a cadeia mais longa, ou seja, esperar qual dos dois blocos terá um bloco descendente primeiro.

2.1.5 – Prova de Trabalho

Um novo bloco pode ser sugerido por qualquer nodo participante da rede. Entretanto, um sistema onde nodos podem incluir informações livremente corre o risco de ter sua integridade comprometida, pois está aberta ao *spamming*. Para evitar a inclusão de blocos inválidos por nodos, a criação de blocos inclui a *prova de trabalho* (*Proof-of-Work*).

A prova de trabalho consiste em um desafio criptográfico. Neste desafio, o nodo autor do bloco busca um *hash,* o resultado de uma computação criptográfica, que esteja de acordo com

um critério definido pelo protocolo daquela *blockchain*. Esta computação é conhecida como *hashing*, e toma como entrada o *hash* do bloco mais recente e sem descendentes da *blockchain*, a raiz da árvore *merkle*, com uma combinação de *hashes* das novas mensagens que este bloco vai confirmar, e um *nonce*, uma sequência de *bits* com valor aleatório, que pode ser ajustado para alterar o *hash* desta computação.

O resultado de um *hashing* é computacionalmente "aleatório", por definição da função *SHA-256* [8], sendo necessário testar *nonces* diferentes, na tentativa e erro, para se obter um *hash* dentro do critério protocolado. Portanto, o cálculo de um *nonce* exige alto custo computacional, não sendo trivial criar um bloco válido, tornando custoso a inserção de quaisquer mensagens, válidas ou até mesmo inválidas.

O critério de aceite para o *hash* está programado no código da própria *blockchain*, e define qual a dificuldade computacional para um bloco. Esta dificuldade é ajustada após uma quantidade fixa de blocos, de acordo com a capacidade computacional da rede naquele momento. Por exemplo, o *Bitcoin* utiliza o algoritmo *Double-SHA-256*, e seu *hash* deve começar com uma certa quantidade de zeros. Esta quantidade é ajustada a cada 2016 blocos, o que ocorre aproximadamente a cada 14 dias, em média. Na data de 16 de janeiro de 2022, 19 zeros eram necessários para que o bloco fosse válido.

A dificuldade computacional de um bloco busca balancear o tempo necessário para a criação de um novo bloco. Dependendo da capacidade computacional de todos os nodos da rede, é possível que o tempo de confirmação de um bloco diminua, o que nem sempre é desejado. No *Bitcoin*, o alvo para cada bloco é de 10 minutos, em média.

O *hash* é facilmente verificável, e se um nodo recebeu um bloco com a prova de trabalho correta. Assim que um nodo conseguir encontrar um *hash* válido, ele propaga seu bloco, que pode ser validado por outros nodos vizinhos. Estes nodos, após receberem um novo bloco, aceitam aquele bloco assim que começam a trabalhar no *hash* de um bloco novo, usando o *hash* do bloco recebido.

Blocos se encadeiam ao incluir uma referência ao *hash* do bloco anterior no cálculo do seu *hash,* fazendo com que esta prova de trabalho siga uma estrutura linear, onde blocos descendentes apontam para blocos ancestrais, o que gera mais uma camada de persistência. Isto consolida a estrutura de histórico de mensagens da rede, onde a partir do bloco mais recente, é possível subir a cadeia até o bloco gênesis, traçando todas as mensagens válidas e atingindo um conhecimento sobre o estado atual da rede. Cada nodo possui uma cópia deste histórico, e concorda com sua validade ao fazer uma prova de trabalho utilizando o *hash* do Bloco mais recente na cadeia mais longa.

Alterações nas mensagens do bloco faz com que seu *hash* se altere, o que altera os *hashes* de todos os blocos posteriores. Isso torna qualquer alteração no histórico difícil, já que a computação do *hash* de um bloco é computacionalmente custoso, e calcular o *hash* de vários blocos em sequência exige altíssimo poder computacional. Para alterar mensagens em um bloco do passado, um nodo malicioso precisa computar não só o novo *hash* de um bloco passado que foi alterado, mas também de todos os novos *hashes* de seus blocos sucessores.

A prova de trabalho de blocos órfãos é um resíduo válido de computação, que acaba desperdiçado. Para aproveitar os blocos órfãos, *blockchains* como a da *Ethereum* [9] permitem que blocos descendentes incluam ramos ancestrais que estão órfãos, desde que o órfão esteja até 6 blocos atrás e que suas mensagens não conflitam com as mensagens dos blocos atuais.

2.1.6 – Mineradores

No contexto de *blockchain*, este processo de *hashing* é conhecido como *mineração de blocos (Mining)*. *Mineradores* são nodos da rede que, além de possuírem uma cópia de todos os blocos da rede, realizam a prova de trabalho para criar blocos válidos, gastando poder computacional para encontrar o *nonce*. Em qualquer *blockchain*, os mineradores concorrem para encontrar este *hash*, numa espécie de loteria computacional. Não há critérios para novos mineradores começarem esta atividade; caso um nodo decida minerar, basta colocar seu computador para calcular o *hash* de um novo bloco, usando o *hash* do último bloco da rede.

A prova de trabalho cria uma dificuldade computacional para que nodos escrevam apenas blocos que sejam válidos, evitando o *spam* de informações inválidas. Mineradores validam e aceitam novos blocos feitos por mineradores concorrentes ao começar uma nova prova de trabalho em cima do *hash* deste novo bloco recebido. Caso vários mineradores recebam um bloco inválido, mesmo que a prova de trabalho esteja correta, eles podem apenas rejeitar este bloco, e preservar o histórico da rede.

Um nodo malicioso que quisesse atacar as informações da rede, além de gastar um alto poder computacional, precisaria ser mais rápido do que todos os outros mineradores combinados. Ou seja, precisaria ter mais 50% ou mais da computação total da rede. Isto torna economicamente custoso e computacionalmente difícil o ataque de um nodo malicioso, já que para alterar dados do passado o atacante deve ganhar uma “corrida lotérica criptográfica”, tendo que computar o *hash* do bloco no passado e todos seus os blocos sucessores. E mesmo que possua sucesso neste ataque, a rede pode simplesmente rejeitar estes blocos.

2.1.7 – Recompensas de Bloco

Mineradores são entidades com participação importante em uma rede *blockchain*. A confirmação de novas mensagens depende dos mesmos, que acabam por concentrar o poder de decidir quais mensagens serão ou não confirmadas na rede. Para incentivar a participação justa dos mineradores na rede, e evitar que os mesmos coordenem ataques à rede em seu favor, a mineração é incentivada pela *recompensa de bloco*. A recompensa de bloco adiciona um elemento econômico à mineração, permitindo que a mesma pague pelo funcionamento da rede. A recompensa depende da *blockchain*, e é programada no código de seu protocolo.

Uma das vulnerabilidades conhecidas da *blockchain* é o ataque de 51%, onde mais da metade dos mineradores da rede se organizam para criar e manter blocos com mensagens fraudulentas, alterando o histórico de mensagens que, apesar de válido, não são a história real daquela *blockchain*. Ao adicionar a recompensa na mineração, cria-se um incentivo econômico para a entrada de novos mineradores na rede, o que aumenta a quantidade de poder computacional e de mineradores necessários para coordenar um ataque.

O *problema dos generais bizantinos* [10], um problema clássico da teoria de jogos, define muito bem esta situação. Neste problema, vários generais precisam se coordenar em um ataque simultâneo a um campo inimigo para vencer a guerra. É imprescindível a participação de 100% dos generais e seus exércitos para o ataque obter êxito, pois se apenas um dos exércitos se abster do ataque, ele será mal-sucedido. Entretanto, todos esses generais estão separados, e seu meio de comunicação não é confiável, podendo haver interceptações de mensagens que podem não chegar. Assim, não é possível que cada general confie que todos os outros generais irão atacar ao mesmo tempo, e este faz o que é melhor na sua capacidade de conhecimento: se abster do ataque. Desta forma, o campo inimigo continua intacto, e é capaz de vencer os menos de 100% dos generais que atacaram.

Apesar de haver grande vantagem para os mineradores ao coordenarem um ataque à rede, não há motivo para que atacantes confiem uns nos outros. Se um membro atacante desistir do ataque, não é possível que o ataque ocorra com sucesso. E mesmo que o ataque ocorra com sucesso, a cadeia dos atacantes pode ser ignorada pelos nodos que ainda participam justamente. Isto significa que o poder computacional deles não está sendo usado na mineração de blocos válidos na rede, abrindo oportunidade para outros mineradores. Ou seja, o ataque cria o incentivo para que os atacantes traiam o ataque, pois podem se beneficiar da ausência dos outros mineradores que estão atacando a rede.

Assim, o protocolo define um conjunto de regras que reforçam a participação justa entre nodos, onde os esforços para atacá-la são fáceis de identificar e prevenir, exigem alto custo computacional, ou exigem coordenação entre diversas partes que não se confiam.

No *Bitcoin*, o minerador que encontrar o *hash* de um novo bloco tem direito a emitir novos *bitcoins*, criando uma transação especial que envia os novos *bitcoins* para si. Esta transação é conhecida como *coinbase transaction* (transação da base de moedas), e também permite a oferta inicial de moeda na rede. Tal transação também só será consolidada caso outros nodos aceitem este bloco, e usem seu *hash* na mineração de um novo bloco, tornando aquele bloco parte da cadeia. Blocos órfãos não recompensam a mineração, já que os mesmos não fazem parte da cadeia mais longa. O mesmo processo ocorre em *blockchains* focadas em transações monetárias.

Em seu início, a recompensa por bloco minerado no *Bitcoin* era de 50 *bitcoins*, e continua sendo dividida pela metade a cada 210 mil blocos, até que se torne zero *bitcoins*. A previsão é de que isso ocorra em 2140, quando a rede terá em circulação o total de aproximadamente 21 milhões de unidades. Este limite busca uma inflação controlada, permitindo que o *Bitcoin* mantenha seu valor através da sua escassez. O número 21 milhões foi convencionado e definido no código de sua *blockchain* logo em sua gênese.

2.1.8 – Taxa de Transação

Além da recompensa de bloco, em *blockchains* para valores monetários os usuários que enviam mensagens podem incluir uma *taxa de transação* (*transaction fee*). Essa taxa de transação é opcional, e está implícita na diferença entre as entradas e saídas de uma transação. Esta diferença pode ser recolhida pelo minerador que criou um bloco, e, da mesma forma que a recompensa de bloco, só será consolidada quando aquele bloco se tornar parte da cadeia. Então, caso o bloco seja órfão, o minerador não receberá as taxas de mineração.

Este é mais um incentivo para mineradores darem prioridade na inclusão daquela mensagem na formação de um novo bloco. Ao final do seu ciclo de emissão de novas moedas, mineradores do *Bitcoin* não terão mais a recompensa de mineração. Entretanto, ainda poderão recolher as taxas de transação, permitindo que a mineração continue economicamente viável, custeando as operações dos mineradores, e incentivando a participação justa.

A taxa também é um incentivo econômico para que os mineradores sempre incluam o máximo de transações possíveis em um bloco, evitando a inclusão de blocos vazios e usando ao máximo os recursos disponíveis da *blockchain*.

2.1.9 – Limitações

Das limitações conhecidas da *blockchain*, a principal é sua escalabilidade. O tamanho de bloco tem impacto direto na quantidade de mensagens confirmadas nesta rede, o que pode causar gargalos se há alto tráfego de informações. Quanto maior o tamanho limite do bloco, maior a quantidade de transações nele é possível incluir, o que possivelmente resolveria este problema.

Entretanto, neste caso os mineradores com maior conectividade de rede (*bandwidth*) teriam uma vantagem injusta com com mineradores com menor conectividade, pois blocos grandes demoram mais para serem propagados pelos enlaces físicos. Mineradores com alta conectividade receberiam blocos novos mais rapidamente, e teriam mais tempo para encontrar o *nonce*. O resultado seria uma rede concentrada na mão de mineradores com mais recursos.

A finalidade deste limite é permitir que o bloco seja propagado com maior rapidez, facilidade, e que permita a rede ter participação justa dos mineradores. Portanto, este limite é um comprometimento entre descentralização e latência, o que se torna inevitável quando há muitas mensagens a serem consolidadas, mas que permite a rede ter segurança e participação horizontal entre os nodos.

No *Bitcoin,* cada bloco possui 1 *megabyte* de dados sobre transações. Um bloco de *Bitcoin* é minerado a cada 10 minutos, aproximadamente, e é capaz de incluir entre 2000 e 6000 [11] transações. Sua taxa de confirmação de transações fica entre 3 até 7 transações por segundo. Isso é até 500 vezes menos do que a Visa, adquirente de cartão de crédito, que é capaz de processar 1.500 transações por segundo [12], em média.

Algumas propostas já foram feitas para contornar a falta de escalabilidade, inclusive abandonar o limite de bloco, o que não foi aceito pelos mineradores da *blockchain* original do *Bitcoin*. Esta discordância levou os maiores mineradores a fazerem um *hardfork;* um processo onde a *blockchain* é intencionalmente ramificada, e uma *blockchain* nova é criada a partir da ramificação da *blockchain* original. As duas *blockchains* compartilham o mesmo histórico até o bloco ramificado, e a partir do mesmo os blocos sucessores têm novas regras e

são incompatíveis, o que torna ambas *blockchains* diferentes. O *hardfork* do *Bitcoin* com maior relevância é o *Bitcoin Cash*, que tem tamanho limite de 32 *megabytes* por bloco.

Outras propostas para atacar a escalabilidade foram feitas, com graus variados de sucesso. Estas alterações são chamadas de *softforks*, onde os nodos aceitam as alterações sem a necessidade de ramificar o histórico. O *softfork* de maior sucesso atualmente no *Bitcoin* é a *SegWit* [13], que tirou a necessidade de incluir o *hash* da assinatura de cada mensagem individual no bloco, aumentando o tamanho de seu bloco, na prática.

Fora da rede, propostas *off-chain,* que não envolvem alterações no protocolo de uma *blockchain*, também são feitas para contornar o uso da *blockchain* para transações menores e contratos que ocorrem com frequência conhecida, mas que podem ter uma transação ao final deste contrato. A implementação mais importante no *Bitcoin* é a *Lightning Network* [14], que cria um *buffer* de transações em uma camada fora da *blockchain*.

2.2 – Criptomoedas e capitalização

A relevância da *blockchain* se mostra através da capitalização do seu mercado. Em fevereiro de 2021, o *Bitcoin* atingiu uma capitalização de 1 trilhão de dólares. Somado a outras criptomoedas, a capitalização total no mesmo período atingiu 2 trilhões de dólares [2]. Para efeitos de comparação, a capitalização total do mercado do ouro é estimada em 11 trilhões de dólares [15]. O valor da Apple, empresa de maior capitalização do mundo, em abril de 2021, era de 2 Trilhões de dólares [16].

Exchanges são os principais serviços para negociação de criptoativos na internet. Atualmente, o volume de operações para compra e venda de criptoativos na *Binance*, exchange com maior volume total de transações em abril de 2021, atingiu U$ 46 bilhões em 24h, entre os dias 15 e 16 de maio de 2021 [17]. O volume total de operações em exchanges no mesmo período foi de U$ 204 bilhões [18].

Os números em relação ao *Bitcoin* e outros criptoativos mostram a adoção de projetos com *blockchain* por pessoas e empresas, e sua crescente evolução ao longo do tempo. Projetos em *blockchain* mostraram sua importância em sua primeira década de existência, e o aumento de usuários e a grande capitalização demandam uma variedade de soluções em tecnologia da informação, voltadas à projetos em *blockchain*. Este é mercado atrai um grande número de projetos concorrentes, com diferentes moedas, regras, leis, implementações e arquiteturas.

2.3 – Casos de Uso da Blockchain

O caso de uso base para uma *blockchain* é em sistemas onde é preciso ordenar mensagens entre partes que não se confiam, sem a necessidade confiar numa autoridade centralizada, que valide as mensagens e as ordena em uma linha do tempo.

De forma geral, projetos utilizando *blockchain* procuram uma solução distribuída para algum problema conhecido na tecnologia da informação Cada um traz uma proposta para melhorar as limitações de *blockchains* mais antigas, identificadas principalmente no *Bitcoin*. Das várias classes de aplicação com *blockchain*, as mais relevantes são:

- contratos inteligentes (*smart contracts*);
- *tokens* fungíveis e não-fungíveis;
- computação distribuída;
- finanças descentralizadas (*DeFi*);
- *stablecoins*;
- oráculos e indexação;
- reserva de valor;
- *memecoins*;
- jogos digitais;
- privacidade e resistência à censura.

A *blockchain* pode ser encarada como um grande livro-mestre (um livro-registro de transações bancárias), que é distribuído. O conjunto deste livro contém o histórico da rede, e sua leitura até a primeira página determina seu estado. A *blockchain* só é capaz de alterar este livro em determinadas regras e condições, protocoladas em seu código. Entretanto, este código pode ser estendido para executar outros protocolos, similares às instruções de um computador tradicional.

A *Ethereum* surgiu para permitir o uso de *contratos inteligentes*, um código que é programado por um usuário é executado pela rede da *Ethereum*. Um bloco representa a execução de diversos contratos, e o histórico desta rede representa os estados anteriores em que este computador esteve, após a execução dos contratos que precisaram ser processados. A leitura do primeiro ao último bloco permite que se tenha um conhecimento sobre o estado atual de todos os contratos nesta máquina.

Assim a *blockchain* é capaz de se comportar como um computador distribuído. No caso da *Ethereum*, esta é uma máquina chamada de *Ethereum Virtual Machine (EVM)*, e a rede se comporta como uma camada de infraestrutura para a construção de projetos de criptoativos. Nesta rede, usuários implantam aplicações que executam códigos remotamente, e nodos executam estes códigos para atingir um estado de consenso de maneira distribuída. Outras duas *blockchains* de computação distribuída relevantes são a *Solana (SOL)* [19], com sua *Solana Cluster*, e a *Cardano (ADA)* [20].

Um uso comum de contratos inteligentes é na criação de *tokens*, que representam um patrimônio, como um ativo ou escritura, ou participação em uma organização, como ações em empresas. A quantidade e finalidade desses *tokens* é definida no código deste contrato inteligente, que também inclui as regras de emissão de *tokens*, a venda e a compra dos mesmos.

Tokens são exclusivamente de duas classes: *fungíveis*, quando os *tokens* de um mesmo contrato possuem as mesmas qualidades entre si, podendo ser trocados e divididos igualmente, pois possuem o mesmo valor individual; ou *não-fungíveis*, quando os *tokens* de um mesmo contrato possuem qualidades únicas, não podendo ser divididos ou trocados entre si pois possuem valores diferentes. O *Bitcoin* é um exemplo de rede de *tokens* fungíveis, onde todos os *bitcoins* são *tokens* equivalentes entre si.

Os projetos mais relevantes em *blockchains* com contratos inteligentes são da classe das *finanças descentralizadas (DeFi),* que executam serviços financeiros tradicionais de maneira *peer-to-peer* na *blockchain*. Serviços em *DeFi* incluem empréstimos, provisão de liquidez, crédito e até mesmo ativos financeiros listados em bolsas de valores. Projetos de relevância em *DeFi* são a *Uniswap (UNI)* [21], e a *Sushiswap (SUSHI)* [22]. Ambas utilizam contratos inteligentes para permitir que usuários tenham a custódia dos criptoativos, e façam o comércio das mesmas, sem uso de um terceiro que atua como mediador, como é o caso das bolsas de valores e exchanges no mercado financeiro.

A negociação de criptoativos é um mercado mundial, já que estão na internet, e utilizam diversas moedas para essas transações. Para operar com moedas fiduciárias, o mercado *DeFi* utiliza *stablecoins,* que atrelam o valor de um *token* com uma moeda emitida por algum banco central, como o real, euro ou dólar estadunidense. Este *token* tem paridade de 1-para-1 com a moeda representada, permitindo a representação de moedas tradicionais em *blockchain*, para uso na compra e venda de outros criptoativos. O projeto de maior relevância nesse meio é o *Tether USD (USDT)* [23].

Um dos problemas encontrados no mercado de *DeFi* é a falta de transparência com os dados de precificação durante a troca de criptoativos. Isso ocorre quando serviços *DeFi* não publicam os dados e indicadores utilizados para estabelecer preços de câmbio entre dois criptoativos. Estes serviços utilizam bancos de dados relacionais comuns, e não abrem estes bancos para consulta pública por motivos de segurança. Isto fez com que surgissem os *oráculos,* um serviço que registra esses dados em contratos inteligentes, em uma ou mais *blockchains*, e *indexadores*, que permitem a consulta e audição destes dados com facilidade, por partes confiáveis e de maneira descentralizada. Um projeto de indexador com relevância é a *Chainlink (LINK)* [24].

Projetos para reserva de valor usam contratos inteligentes para gerenciamento de ativos reais de maneira digital, como ouro, créditos carbono e ações. Dois exemplos com ouro são a *Pax Gold (PAXG)* [25] e o *Tether Gold (XAUT)* [26], em que seus *tokens* representam a posse de 31.1 gramas de ouro, sob a custódia da empresa *Brink's*. Estes *tokens* facilitam o comércio de ouro, que é um ativo custoso de se obter e muito arriscado de custodiar. Projetos dessa classe costumam passar por auditorias, para garantir que os ativos realmente existem e estão sendo custodiados da maneira correta.

Ainda assim, para reserva de valor, o projeto mais relevante continua sendo o *Bitcoin*, que vem sendo adotado por projetos *DeFi* e instituições financeiras, como o *Itaú* e o *BTG Pactual* [27], como criptomoeda para este objetivo. Isso se deve à crença nos seus fundamentos econômicos, de escassez e ampla aceitação entre pessoas e grandes instituições econômicas, como bancos e empresas de serviços financeiros.

Por ser um novo universo com rápida expansão na internet, vale mencionar os projetos polêmicos feitos por *meme.* O mais popular, a *DogeCoin (DOGE)* [28], foi criada como sátira às criptomoedas, com intuito de mostrar que usuários de criptomoedas comprariam qualquer criptoativo, mesmo que fosse uma piada. Tanto que seu logo é o *doge*, uma foto de um cachorro que era um *meme* muito popular na época em que a moeda foi criada.

Entretanto, os *tokens* encontraram uma aplicação melhor em jogos que os usam para representar algum item colecionável que pode ser usado, como *skins* (texturas para objetos 3D), armas, veículos e até troféus. Estes *tokens* são distribuídos por um jogo que também possui um contrato inteligente, e assim que está em posse pelo usuário, podem ser usados em diversos jogos, e até vendidos entre os jogadores. O jogo *Crypto Kitties* conseguiu congestionar a rede da *Ethereum* em dezembro de 2017 [29], devido a sua popularidade. Nele, usuários criam e trocam figurinhas virtuais de mascotes peludos, alguns deles chegando a ser negociados por milhares de dólares.

Finalmente, projetos voltados para a privacidade e resistência à censura procuram manter seguro e privado todo o ecossistema de criptomoedas. A *blockchain* é capaz de preservar a identidade dos usuários que participam da rede, e não discrimina os participantes em seu protocolo. Ou seja, a *blockchain* permite a maior horizontalidade possível dentro de sua rede. Entretanto, há interesses de entidades governamentais e empresas em bloquear o uso de alguns criptoativos, que punem pessoas e empresas que operam com os mesmos, caso seja descoberto seu uso, censurando o uso desses ativos.

Neste cenário de censura, em que indivíduos, empresas, e até países são sancionados por terceiros e restringidos no uso de criptoativos, é imprescindível oferecer verdadeira privacidade e anonimato dentro da *blockchain*. A natureza pseudônima do *Bitcoin* e de outras *blockchains* tradicionais permitem o cruzamento e triangulação de dados fora da rede, para descobrir os endereços individuais de pessoas e empresas, e sancionar as mesmas, seja nas *exchanges* de criptomoedas, seja através de ações legais. Empresas como a *Blockstream* [30] trabalham com a triangulação e o catalogamento de usuários do *Bitcoin*.

Os projetos voltados à privacidade da *blockchain* implementam algoritmos para ofuscar os dados das mensagens e blocos, tornando toda a rede completamente anônima. Estes algoritmos preservam as propriedades da *blockchain*, sem comprometer a autenticação de mensagens nem a validade do histórico. Entre os mais relevantes estão o *Monero (XMR)* [31], uma alternativa privada ao *Bitcoin* que será explicado em maiores detalhes na seção 2.4, e a *DERO* [32], uma alternativa privada à *Ethereum*.

2.4 – *CryptoNote* e *Monero*

O *Monero* é uma criptomoeda que utiliza *blockchain* como principal estrutura de dados. Com foco em privacidade, seu protocolo ofusca os usuários e as transações feitas pelos mesmos, se tornando completamente privada. Diferentemente do *Bitcoin,* que possui blocos públicos e transparentes, sua rede é mais resistente ao rastreio de dados, que podem levar à censura no uso de criptomoedas. Os valores e agentes envolvidos nas transações contidas em blocos da rede são ofuscados por criptografia assimétrica, e o algoritmo de mineração é desenhado para reduzir a desigualdade entre os nodos da rede.

Após o surgimento do *Bitcoin*, deficiências foram identificadas em seu protocolo, como sua falta de privacidade na sua rede. Devido ao modelo de nodos distribuídos e sem hierarquia, é difícil atingir um consenso da rede para implementar atualizações em uma *blockchain* grande, como a do *Bitcoin*, e se torna mais fácil criar uma nova criptomoeda.

Desta forma, surgiram *forks* do *Bitcoin,* com implementações que apresentam alternativas a esses problemas. O *Bytecoin* surgiu com a proposição para ofuscação de membros e transações na *blockchain*, com uma implementação baseada no *CryptoNote* [31], um protocolo que visa a privacidade, anonimidade e resistência à censura na *blockchain* de uma moeda. No entanto, a implementação com maior adoção e capitalização de mercado é o *Monero*, um *fork* do *Bytecoin*.

A *blockchain* do *Monero* protocola os métodos *ring signature*s, para ofuscar os remetentes de mensagens; as *ring confidential transactions*, para ofuscar os valores de todas as transações incluídas em um bloco; o *Dandelion++*, para ocultar o endereço de IP do remetente; e os *endereços stealth (stealth addresses)*, para dificultar o rastreamento dos destinatários. O algoritmo de mineração do *Monero*, o *Random X*, também dificulta a concentração do poder computacional por grandes agentes mineradores da rede, tornando sua mineração mais acessível e horizontal.

A *ring signature* é um método de assinatura de mensagens que ofusca apenas o remetente da transação, mas não seus valores. Assim como na definição tradicional da *blockchain*, usuários na *blockchain* do *Monero* são identificados por um par de chaves público-privada. Ao realizar uma transação, este usuário envia uma mensagem assinada com sua chave privada, mas que inclui as chaves públicas de outros usuários, registradas em transações passadas e selecionadas de maneira aleatória.

Uma *ring signature* é definida pela função *ring_sign(pvk, pubk[ ], msg) = signature*, onde *pvk* é a chave do usuário, *pubk[ ]* uma lista com as chaves públicas de outros usuários, e *msg* a mensagem enviada, que neste caso é uma transação de *Monero*. Esta função retorna uma *signature*, uma assinatura da transação que pode ser verificada por qualquer usuário da rede, através da função *ring_verify*, definida por *ring_verify(signature, pubk[ ], msg) = bool.*

O conjunto da chave privada do usuário remetente e chaves públicas aleatórias dessa *ring signature* é conhecido como o *anel*. Mesmo com essa assinatura, um observador teria uma grande dificuldade para determinar qual das chaves pertence ao usuário que a emitiu, já que todas as chaves aparentam ser válidas para aquela assinatura. Essa dificuldade aumenta com o número de chaves públicas neste anel, e permite ofuscar o verdadeiro dono daquela transação.

A rede do *Monero* adotou as *ring confidential transactions* em janeiro de 2017, forçando a ofuscação dos valores de todas as transações. Antes da implementação, era possível um agente externo monitorar as entradas e saídas de valores transacionados na rede. Numa *ring confidential transaction,* a ofuscação é feita para as entradas e saídas dos valores de uma

transação, usando dois algoritmos de criptografia: os *Pedersen Commitments* [33], que garante que uma equação do tipo $X + W = Y + Z$ é verdadeira, sem revelar quais são os valores exatos de *X, Y, W, e Z*; e os *range proofs* [34], que garantem que os valores de *X, Y, W, e Z* estão entre 0 e 2^{32}.

Os *Pedersen Commitments* são uma coleção de algoritmos criptográficos, que permitem o envio de mensagens criptografadas com um segredo, onde os remetentes assinam mensagens sem revelá-la ou ser capaz de modificá-la. Estas mensagens são concatenadas com um segredo, e criam um *commitment,* o *hash* de uma mensagem que terá seu segredo revelado posteriormente. Este tipo de mensagem é conhecido como mensagem de *Prova de Conhecimento-Zero* (*Zero-Knowledge Proof Message)*, onde é possível constatar que uma mensagem é verdadeira, sem a necessidade de revelar esta mensagem.

Remetentes de *Monero* enviam *commitments* com transações de *Monero*, sem revelar o valor exato da transação, mas garantindo que a quantidade de entrada no *input* será igual a quantidade de saída no *output*. Qualquer outra parte alheia à transação pode verificar que nenhuma quantidade de *Monero* está sendo criada ou destruída durante as transações, mas apenas o destinatário da transação é capaz de decriptar este *commitment*, e verificar o valor exato que lhe foi enviado.

Como a rede não consegue ver a transação em si, para garantir que o destinatário de uma transação não gaste mais do que foi recebido, os *outputs* das transações em um *commitment* dele são obrigatoriamente utilizados como *inputs* para as transações futuras deste destinatário, sendo todas as transações contidas na rede do *Monero* feitas utilizando *outputs* de transações passadas. De forma parecida com os blocos da *blockchain*, um *commitment* novo usa o *commitment* anterior, e seu resultado e validade depende de uma cadeia de *commitments*. Caso o valor seja alterado, o *commitment* será inválido, e é possível verificar que a transação é fraudulenta.

Os blocos também usam os *range proofs* para validar que os valores também são números positivos, prevenindo a destruição ou criação de moedas e preservando a quantidade em circulação na rede. Esses algoritmos garantem que as funções de ofuscação ainda preservam as características da *blockchain*, permitindo que usuários provem a integridade da rede, sem informar diretamente quem são as partes ou quais os valores.

O algoritmo *Dandelion++* transmite as transações para outros nós através de canais encriptados, similar à rede *TOR* [35]. Ao atingir um nodo aleatório, após percorrer um número *n* de nodos, este propaga a transação publicamente para os outros nodos. Este número

n é definido através de um consenso da rede. Com isso, se torna difícil determinar o IP dos remetentes das transações da rede, e a localização do indivíduo dono desta mensagem.

Os endereços *stealth* forçam o destinatário a criar endereços aleatórios de uso único para cada transação, a partir da sua chave pública, em nome do remetente. O destinatário, além de possuir uma chave pública, possui endereços *stealth* diferentes para faturação de transações individuais, sendo estes publicados na *blockchain*. Estes endereços não podem ser ligados à chave pública pessoal do destinatário, ou qualquer outro endereço *stealth* utilizado em transações passadas. Com o endereço *stealth*, a *blockchain* terá um registro público deste endereço, mas apenas o destinatário é capaz de determinar que é o dono do mesmo. O remetente também será capaz de determinar quem é o dono. Não pela *blockchain* em si, mas porque o destinatário, para receber o pagamento, enviou aquele endereço por meios externos.

Os endereços *stealth* são definidos e gerados pela função *stealth(address) = address',* onde *address ≠ address'*. Esta função se comporta como um *hash*, e é computacionalmente difícil para um observador determinar qual a chave pública original é associada àquele endereço *stealth*, preservando a anonimidade do usuário. Dessa forma, ambos endereços são livres de associação direta, podendo ser associados apenas pelo uso da chave privada do dono daqueles endereços.

O *Random X* é um algoritmo de mineração criado para dificultar a mineração em larga escala. Foi adotado pela rede do *Monero* em novembro de 2019, e é baseado na geração aleatória de programas de computador, com alto consumo de memória, para minimizar a vantagem dos *hardwares* especializados. Nele, utiliza-se uma máquina virtual com instruções específicas para a geração e execução desses programas aleatórios. A saída desses programas é consolidada em um *hash*, com tamanho de *256-bits*, como prova de trabalho.

O algoritmo de geração aleatório de programas é trocado em intervalos periódicos de blocos. Esse intervalo é determinado através da governança do projeto: uma decisão coletiva dos desenvolvedores, usuários e mineradores. Em novembro de 2021, o intervalo era de 2048 blocos, aproximadamente 3 dias. O novo algoritmo é gerado utilizando uma chave, que é o *hash* do último bloco publicado nesse intervalo.

Dado que o algoritmo de mineração sempre muda com frequência, esse intervalo e sua aleatoriedade são suficientes para tornar difícil a fabricação de computadores especializados, como *ASICs (Application-Specific Integrated Circuits)* [36], para mineração do *Random X*. Assim, torna-se economicamente impraticável a construção de um circuito específico para o *Monero*, o que permite a entrada e permanência de mineradores menores na rede, tornando mais difícil a centralização do poder computacional na rede do *Monero*.

Vale mencionar alguns *forks* relevantes com foco em privacidade, como o *Haven* [37], *DERO* [28], *Equilibria* [38] e *Wownero* [39]. O *Haven* e a *Dero* são *forks* do Monero que incluem a implementação de *smart-contracts* em sua *blockchain,* entre outras funcionalidades similares à *Ethereum*. A *Equilibria* é um projeto de oráculo, que indexa valores de *smart-contracts,* similar à *Chainlink,* mas para *blockchains* com foco em privacidade. E finalmente a *Wownero*, que é uma *memecoin* cópia do *Monero*, do mesmo modo que a *Dogecoin* é uma *memecoin* do *Bitcoin*. Entretanto, a *Wownero* usa uma *blockchain* com transações privadas, enquanto a *Dogecoin* tem *blockchain* transparente.

3 Projeto de Aplicativo

O *Bitcoin*, por publicar todas suas transações e agentes de forma transparente, é uma rede pseudônima. Suas transações não são completamente privadas e permitem o rastreamento de usuários através de ferramentas externas à rede, como a triangulação de dados. Estes dados permitem que governos e empresas, que possuem interesses conflitantes com o que é permitido pelo uso da *blockchain*, pratiquem sanções a estes usuários, como a proibição do uso de criptomoedas e ações legais.

Essa transparência também compromete a privacidade financeira de um indivíduo, que pode ter todo seu extrato financeiro exposto. Por ser a criptomoeda mais relevante, o *Bitcoin* atrai usuários leigos, sem conhecimento a respeito das informações publicamente disponíveis sobre transações na própria *blockchain* do *Bitcoin*.

3.1 – Modelo de negócio

A *Shadow Wallet* foi criada para facilitar a adoção de criptomoedas que dão ênfase à privacidade. Ela é uma carteira de criptoativos *free and open-source,* que facilita a utilização destes ativos por qualquer grupo ou indivíduo, permitindo que usuários manipulem seus criptoativos de maneira custodial, onde as chaves privadas estão em posse completa do usuário, em seu dispositivo, possibilitando o uso dos mesmos em seu dia-a-dia.

Alguns aplicativos de carteiras muito usadas no mercado, como as da *blockchain.com*, *Freewallet* e *Binance*, fazem uso de um servidor externo para operar com as chaves privadas do usuário. Estes aplicativos são apenas a interface gráfica do usuário, e fazem uso de uma API para chamar operações com a *blockchain*. Estas operações serão de fato operadas pelo *back-end* destas empresas, que centralizam as chaves de seus usuários. Na prática, os usuários

não estão em posse direta das próprias chaves públicas e privadas, o que torna o uso dessas carteiras não-custodiais.

Carteiras similares à nossa proposta, com uso custodial, que executam a lógica localmente e se comunicam diretamente com a *blockchain*, são a *Monerujo* e a *Cake Wallet.* A *Monerujo* possui uma interface pouco intuitiva, com várias camadas de navegação e poucos recursos de ajuda. É possível melhorar sua navegação, pois é uma ótima carteira para usuários avançados, com ferramentas completas para uso das criptomoedas. A *Cake Wallet* é um exemplo de carteira com excelente interface, de navegação simples e segura. Entretanto, apesar de possuir suporte para *Monero*, não é uma carteira voltada para criptomoedas focadas em privacidade, com suporte apenas para moedas mais populares como *Bitcoin e Ethereum*.

O projeto da *Shadow Wallet* ataca os problemas das carteiras mencionadas acima, trazendo uma navegação de, no máximo, 3 subníveis a partir da tela de visão geral da carteira, permitindo o uso completo de ferramentas para manipular criptomoedas. Além disso, foi incluído uma seção de ajuda em seu menu principal, permitindo aos usuários um fácil acesso à informação, para aprendizado prático do uso de criptomoedas e da carteira.

O escopo inicial é implementar um aplicativo com suporte a transações de *Monero*, para dispositivos com sistema operacional *Android* e *iOS*, pavimentando o caminho para a implementação de outros criptoativos derivados do mesmo. Tais derivações são conhecidas como *forks,* projetos concorrentes e similares ao *Monero*, que usam como base o protocolo *CryptoNote*, e derivam de ramificações do código original do *Monero*. Portanto, estes projetos possuem funcionalidades similares ao *Monero*, o que diminui a dificuldade de alterar as bibliotecas para inclusão e suporte destas moedas. Desta forma, a *Shadow Wallet* permite usuários utilizarem moedas privadas, que não são oferecidas pela *Cake Wallet*.

3.2 – Análise de Requisitos

A *Shadow Wallet* é um aplicativo de celular, que funciona como gerenciador de carteiras de *Monero*. Sua funcionalidade primária é permitir que usuários controlem seus ativos em *Monero*, podendo receber e enviar, criar novas carteiras e importar suas carteiras antigas. O usuário também pode configurar seu nodo de comunicação com a *blockchain*, podendo escolher sua origem de dados de preferência.

Focada em 2 aspectos primários, segurança e simplicidade, a carteira permite ao usuário configurar o bloqueio e desbloqueio da aplicação usando um PIN de 6 dígitos ou biometria, se seu dispositivo permitir. O bloqueio do aplicativo encripta as informações, para que elas

não possam ser acessadas por outras aplicações. Finalmente, o aplicativo permite optar por um modo claro e escuro da interface gráfica.

As regras de negócio se classificam em duas categorias: regras relacionadas à *blockchain* e regras da aplicação. As regras relacionadas à *blockchain* precisam seguir os protocolos do *Monero*, para que o aplicativo funcione corretamente e de acordo com o contrato programado em sua *blockchain.* As regras da aplicação são mais flexíveis, e buscam trazer segurança e simplicidade no uso de uma carteira de *Monero*.

Os requisitos funcionais principais do aplicativo permitem ao usuário:

- Gerenciar carteiras de *Monero* via interface gráfica;
- Importar/recuperar o backup de carteiras anteriores de *Monero*;
- Gerar novas carteiras, endereços e sub-endereços;
- Remover carteiras;
- Ver saldo de carteira;
- Ver e fazer backup das chaves privadas e *seed* mnemônica da carteira;
- Enviar e receber *Monero*;
- Gerenciar nodos de *Monero*;
- Autenticar/bloquear a aplicação por PIN, senha ou biometria;
- Configurar aparência da aplicação;
- Configurar rotinas automáticas de sincronismo das carteiras.

Os requisitos não-funcionais do aplicativo requerem a encriptação dos dados operados pela aplicação, garantindo a integridade e segurança das informações sensíveis, e permitindo que apenas o usuário autenticado consiga manipulá-las. Além disso, requisições síncronas devem ocorrer em até 1 segundo, com 3 segundos para operações assíncronas que dependem de alguma resposta de um nodo da *blockchain*, permitindo o usuário cancelar operações que demorem mais de 2 segundos. Processos mais longos devem ser executados em *background*.

Dos requisitos, o aplicativo não mantém estado ou sessão dos usuários, que é encerrada assim que o aplicativo é fechado. O aplicativo também não faz transações às quais o usuário não solicitou, além de não poder selecionar um nodo de *Monero* sem que o usuário solicite tal mudança. Finalmente, o aplicativo não rodará um nodo no dispositivo, pois é muito custoso computacionalmente.

Casos de uso também se classificam em duas categorias: funcionalidades de aplicativo e funcionalidades relacionadas à *blockchain*. As funcionalidades relacionadas apenas ao

aplicativo não fazem chamadas ou utilizam serviços que consultem a *blockchain*, podendo ser executadas de modo síncrono. As funcionalidades de *blockchain* são executadas de modo assíncrono, pois precisam ler ou propagar informações à *blockchain*.

Os casos de uso mais críticos de uso da carteira são: importar carteira, geração de nova carteira, geração de novo endereço *stealth*, abrir e visualizar carteira, visualizar os segredos da carteira, enviar *Monero*. Todos estes casos de uso estão relacionados ao gerenciamento das chaves privadas, chaves públicas e envio de fundos.

Alguns dos casos de uso anteriores dependem de dois casos de uso, relacionados à *blockchain*: Ler *blockchain,* e *Broadcast* para *blockchain*. Estes casos de uso também estão relacionados ao gerenciamento dos nodos de *Monero*: Adicionar, remover e visualizar nodo. Estes últimos casos de uso são necessários para que o usuário consiga ler e fazer *broadcasts* para a *blockchain*, através de um nodo intermediário, já que não terá uma cópia da *blockchain* em seu dispositivo.

Finalmente, casos de uso apenas da aplicação: Compartilhar Endereço/sub-endereço, para que possa receber Monero de outro usuário; Autenticar usuário, para poder acessar as informações da carteira; Bloquear aplicativo, para encerrar o uso da aplicação e encriptar seus dados; Configurar aparência do aplicativo, para trocar a cor das telas de acordo com sua preferência.

A descrição completa dos requisitos do projeto pode ser encontrada em detalhes no Anexo I, contendo todas as descrições de casos de uso e os modelos de processo de negócio. Nesse documento, também estão contidos os diagramas para a visualização do fluxo dos casos de uso e ação do usuário na aplicação.

3.3 – Decisões Técnicas e Arquitetura

A arquitetura do aplicativo possui três componentes: o *núcleo*, a *lógica* e a *interface gráfica*. O núcleo é encapsulado na camada lógica, e usa um *wrapper* intermediário para a comunicação entre os dois. A interface gráfica se comunica com a camada lógica por *API* simples, permitindo ao usuário fazer chamadas através de telas em seu dispositivo.

O aplicativo se aproxima a uma arquitetura monolítica, mas com componentes lógicos encapsulados por camadas. Sua vantagem em relação à arquitetura de componentes independentes é a possibilidade de aplicar protocolos de segurança de uma maneira mais simples. Por ser uma aplicação para movimentações monetárias, seria necessário aplicar

métodos de segurança em cada *API*, para cada chamada de comunicação entre componentes. Como isto seria custoso para manter, preferiu-se encapsular os componentes em camadas.

No núcleo, está implementado o projeto base do *Monero*, onde estão compilados os binários de suas bibliotecas. Foi utilizada uma biblioteca estável da *monero-project* [40], implementada em *C++*, que permite ao aplicativo se comunicar com a *blockchain* do *Monero*. Isso tira a necessidade de codificar, atualizar e manter estas funcionalidades. Esta biblioteca é utilizada também pela *Cake Wallet.*

Na camada lógica, há abstrações de componentes da aplicação, como controladores, serviços, modelos de dados e *singletons*, instâncias estáticas de classes com propriedades comuns, que podem ser acessadas por qualquer outro ponto da aplicação. Esta camada foi implementada na linguagem *Dart* [41].

O *Dart* é uma linguagem de programação orientada a objetos, fortemente tipada e criada com propriedades robustas para chamadas assíncronas, como *async-await.* Ela contém tratamentos para concorrência isolada permitindo que interfaces gráficas contenham código orientado a eventos. Além disso, a linguagem contém recursos como: *sound null safety,* para tratamento de campos que podem ser nulos; operadores de *spread*, para permitir que uma função receba 0 a n argumentos, similares ao do Ecmascript 6; e ferramentas para permitir customização de UI, focadas em plataformas específicas. O conjunto desses recursos permitem usar o mesmo código-fonte em *Dart* para compilar aplicativos *Android* e *iOS*, reduzindo o trabalho de desenvolvimento para atingir duas das maiores plataformas mobile no mercado.

Entre as camadas de núcleo e lógica foi implementado um *wrapper*, uma interface necessária para realizar as chamadas das funções do núcleo. O *wrapper* foi separado da camada lógica, e também é implementado usando *Dart*.

Na camada de interface gráfica, estão contidos os elementos interativos para o usuário final, como telas, botões, caixas de texto e definições de cores e temas. Esta camada permite ao usuário fazer chamadas à camada lógica, utilizando a tela de seu dispositivo. Este componente foi implementado utilizando *Flutter* [42], uma biblioteca implementada em *Dart*, focada no desenvolvimento de interfaces de aplicativos multiplataforma.

Toda chamada da interface gráfica para a camada lógica é interceptada pelo componente `view_model`, que encapsula os modelos e limita as operações permitidas pelo front-end. Suas funcionalidades serão detalhadas no capítulo 4.

A figura a seguir ilustra a organização da arquitetura e seus encapsulamentos:

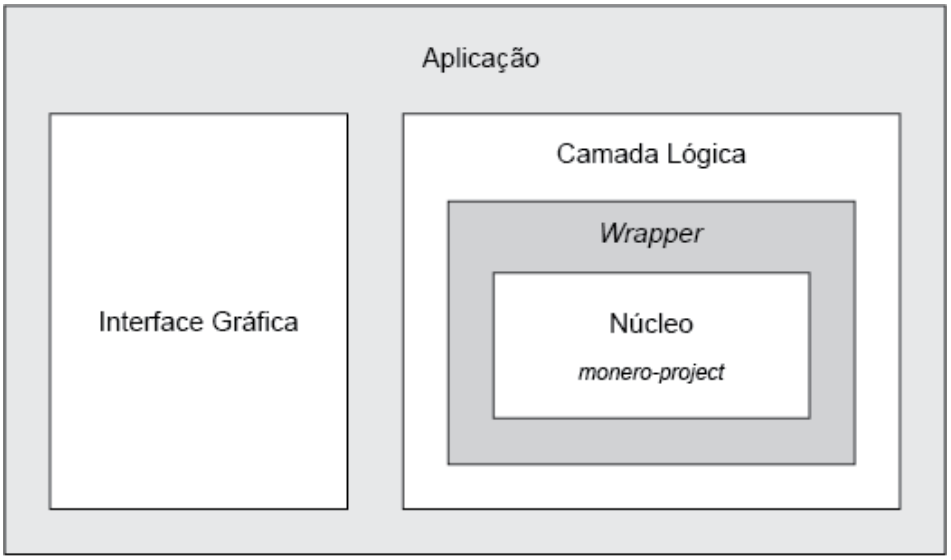


Figura 1 - Disposição dos componentes da *Shadow Wallet.*

O diagrama abaixo mostra o fluxo de comunicação entre estes componentes:

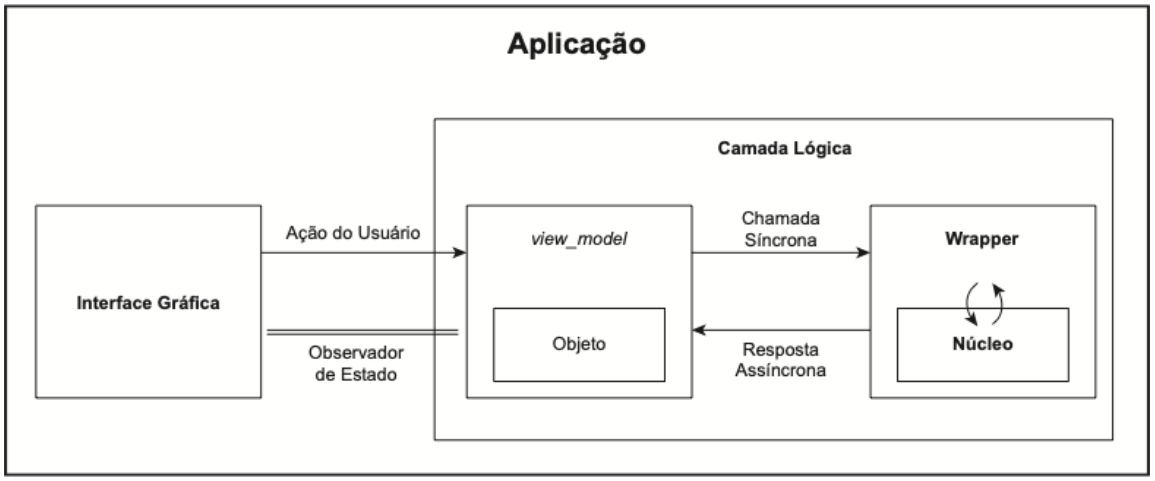


Figura 2 - Fluxo de requisições entre componentes da aplicação.

3.4 – Considerações sobre segurança

A *Shadow Wallet* é uma aplicação que opera com valores monetários, havendo uma extrema importância em garantir a segurança e integridade de suas operações. Para tal objetivo, a aplicação faz uso da *keystore* [43] e dos *cryptographic salts* [44].

A *keystore* é um serviço do sistema operacional, que guarda chaves privadas de aplicações, geradas pelos desenvolvedores delas. Essas chaves são usadas pelo aplicativo para encriptar e decriptar dados que estarão na memória *RAM* do dispositivo. O par de chaves da aplicação garante que o que será escrito em memória não será acessado indevidamente por

outras aplicações, como as chaves privadas de *Monero*, a *seed* mnemônica, que representa essas chaves ou as senhas de *backup* dessas chaves.

Os *cryptographic salts* são caracteres concatenados às chaves da aplicação, geradas aleatoriamente durante a build do aplicativo. Isso ajuda a prevenir ataques de tabelas de arco-íris, onde são executadas tentativas de adivinhar as senhas por força bruta, através de *hashes* conhecidos.

Procurando seguir uma implementação limpa e testada em campo, a *Cake Wallet* foi usada como inspiração no projeto da *Shadow Wallet*. Em ambas, a camada de núcleo exige uma autenticação através de uma chave *SSH* [45] para acessar as chaves privadas de criptoativos, que estão armazenadas de forma encriptada na memória secundária do dispositivo. A chave *SSH* é armazenada dentro de um arquivo encriptado, gerenciado por uma *secure storage*, uma classe do *Flutter* que gerencia a encriptação, decriptação e armazenamento desses arquivos na memória do dispositivo. Para acessá-la, é necessário passar por uma autenticação de PIN ou biometria, que é disponibilizada por uma *API* do sistema operacional.

A *Shadow Wallet* utiliza sete chaves aleatórias como *cryptographic salts*. Cada uma delas é utilizada em uma parte diferente do processo de encriptação e decriptação das informações sensíveis contidas na aplicação, e estão ofuscadas dentro dos binários da mesma:

- A chave *salt* é utilizada na autenticação por PIN ou biometria.
- A *keySalt* é concatenada junto com a chave *salt* no processo acima
- A *keychainSalt* é utilizada na chave privada armazenada na *keystore* do sistema operacional.
- A *walletSalt* para utilizar as chaves privadas de *SSH*, e consequentemente mexer com os criptoativos da carteira.
- A *shortSalt* é concatenada com a chave de *walletSalt* durante o processo acima.
- A *backupSalt* é utilizada para exportar e importar o arquivo de backup da carteira.
- A *backupKeychainSalt* é utilizada no processo de armazenamento de backup das chaves privadas da aplicação, dentro do domínio da mesma, sendo esse domínio encriptado e decriptado junto com a chave privada de *keystore* da aplicação.

A figura a seguir ilustra o fluxo para se obter as chaves privadas decriptadas:

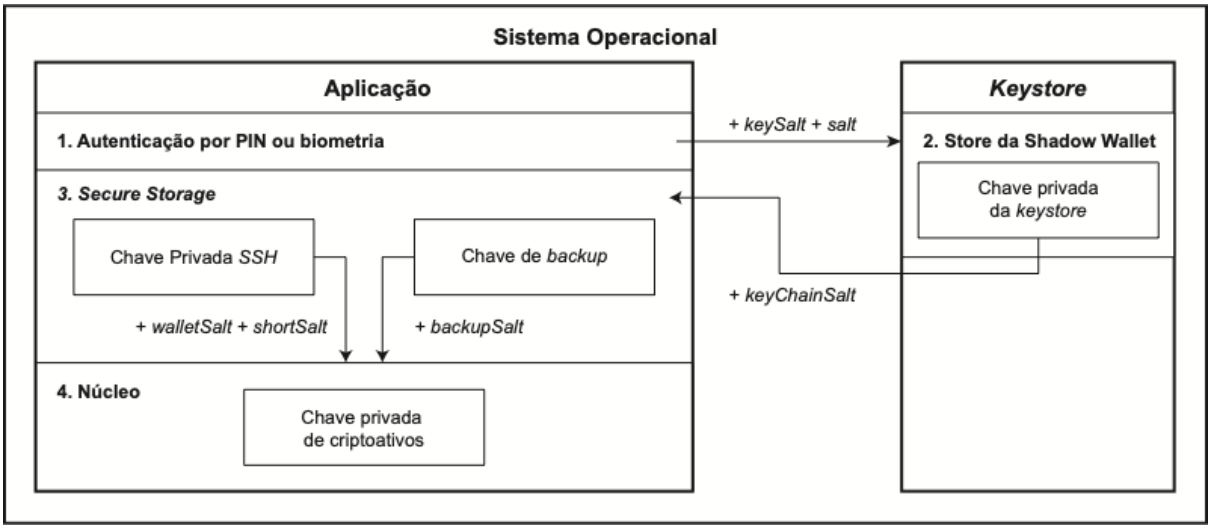


Figura 3: Fluxo de requisição e uso das chaves para autenticação na aplicação.

Um atacante que queira obter os fundos dos criptoativos terá que acessar as chaves privadas, que estão encriptadas pelo núcleo do aplicativo. Para conseguir decriptar essas chaves, é necessário autenticação nessa camada com uma chave *SSH*, armazenada em memória secundária, dentro da *secure storage*. O acesso à essa chave *SSH* exige que o atacante precise decriptar a *secure storage,* utilizando a chave privada que está na *keystore* do sistema operacional, junto ao *salt* adicionado pelo aplicativo. Essa chave é fornecida ao realizar a autenticação por PIN ou biometria, adicionando as duas chaves de *salt* fornecidas pelo aplicativo, necessárias nesta etapa. Após isso, o aplicativo fornece a chave privada armazenada no *keystore* do sistema operacional junto com o *salt* para acesso à *secure storage*. Apenas dessa forma, é possível obter a chave *SSH* e autenticar na camada de núcleo do aplicativo, e obter acesso às chaves privadas do usuário e seus fundos.

Os *cryptographic salts* tornam o ataque descrito acima difícil de ser executado, pois o atacante teria que adivinhar a senha através de força bruta, concatenando uma chave aleatória à senha, o que é computacionalmente custoso e levaria tempo. Além disso, a autenticação do aplicativo conta com um tempo de *backoff*, em que novas tentativas de desbloqueio são recusadas caso a senha incorreta tenha sido inserida sucessivamente, retardando o ataque de força bruta.

De forma direta, realizando a autenticação por PIN ou biometria, é possível acessar os dados sensíveis do aplicativo. Sem ela, é improvável que algum atacante externo decripte os arquivos onde estão contidos a chave *SSH* e, consequentemente, as chaves privadas dos criptoativos. Para que isso ocorra, é necessário que o atacante tenha o controle do dispositivo

onde o aplicativo está instalado, o que torna o aplicativo tão seguro quanto o sistema operacional onde o mesmo está sendo executado.

4 Implementação

A implementação da *Shadow Wallet* contou com um processo completo de desenvolvimento de *software*, incluindo etapas preliminares de prototipagem e validação. O projeto contou com duas etapas, para executar partes do trabalho que não causavam dependências diretas.

Na primeira etapa, foi desenvolvido em paralelo o protótipo de interface, o núcleo com *wrapper*, e a camada lógica. Neste momento, os componentes lógicos da aplicação não seriam controlados por uma interface gráfica, então foi possível codificar estes componentes separadamente.

Na segunda etapa do projeto, foi codificado o *front-end* do aplicativo, de acordo com a interface gráfica desenvolvida no protótipo. Esta etapa dependia que o protótipo estivesse maduro o suficiente antes de iniciar. Assim que este esteve pronto, as telas foram incluídas, permitindo o uso de uma interface gráfica para controle do aplicativo.

O repositório com o código da aplicação, junto com as releases do .apk compilado, pode ser encontrado no site *github.com/gustavocalonico/shadow_wallet* .

4.1 – Prototipagem

A prototipagem do aplicativo teve duas fases, uma com baixa fidelidade e outra com alta fidelidade. Isto permitiu o amadurecimento da interface gráfica antes da implementação, para testar os fluxos de navegação e evitar refazer a implantação de telas em uma etapa mais avançada no ciclo de vida do *software*.

O protótipo de baixa fidelidade foi usado para desenhar os componentes da interface gráfica, e simular a navegação inicial. Testaram-se ideias de componentes e telas, para planejar a navegação entre as telas e escolher qual se adequa melhor ao projeto do aplicativo.

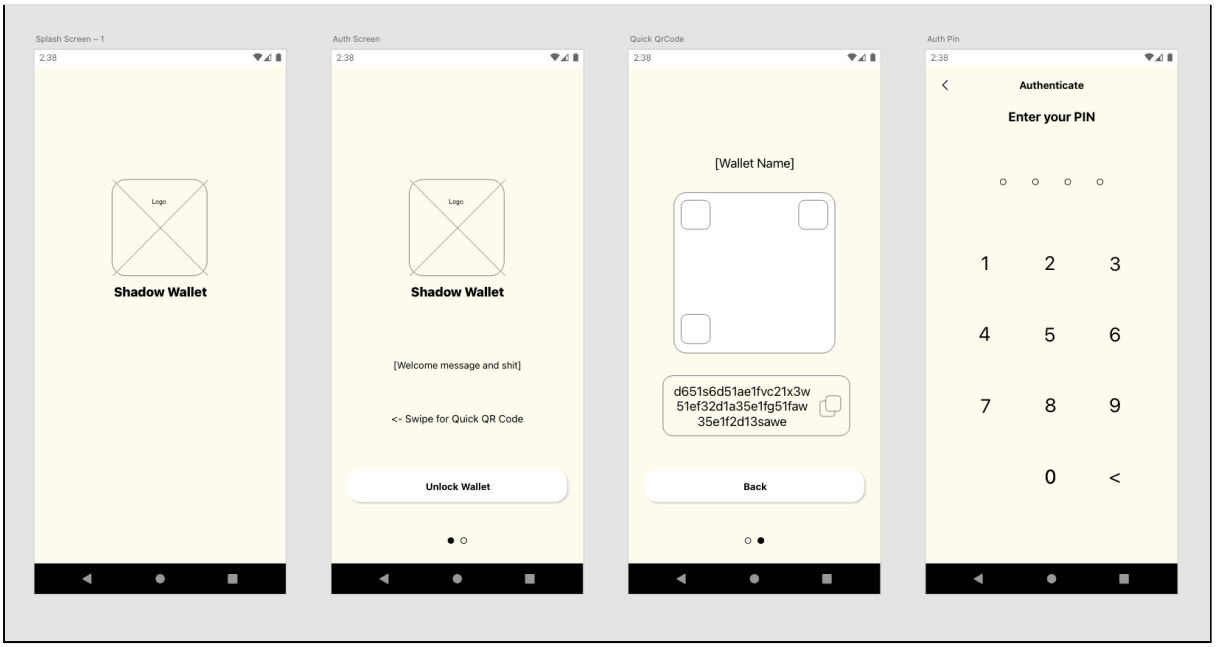


Figura 4: Protótipo de baixa fidelidade do aplicativo. Da esquerda para a direita: Tela de *Splash*; Tela de Login; Tela de *QR Code* rápido; Tela de inserção de PIN.

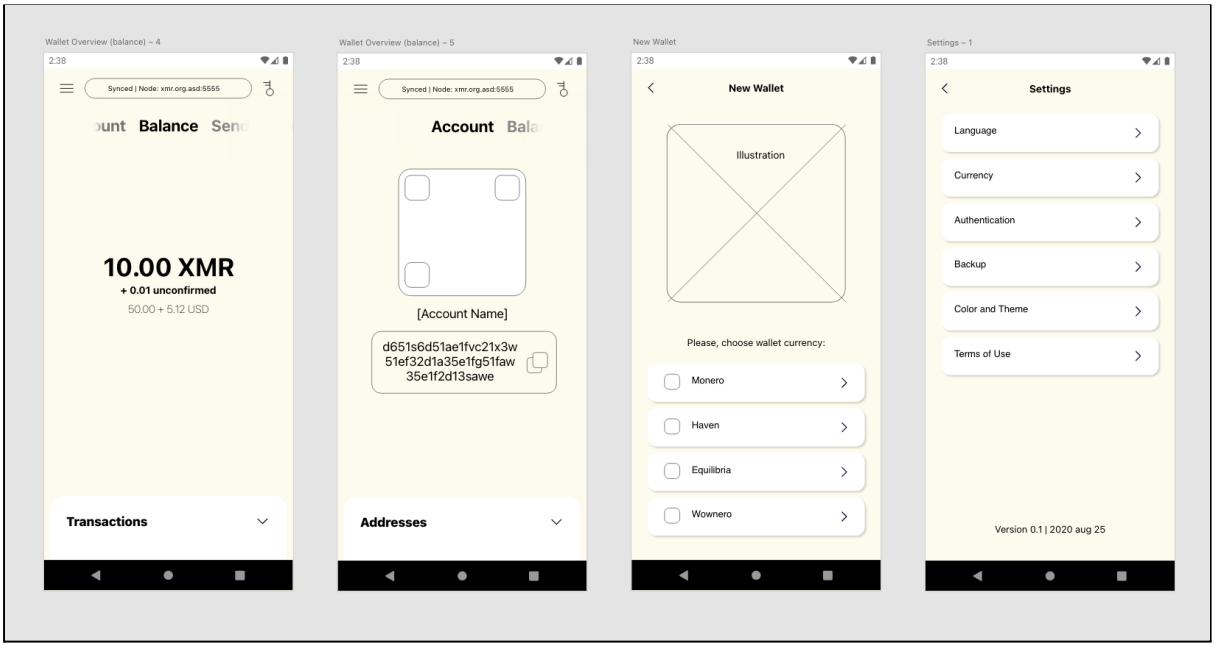


Figura 5: Protótipo de baixa fidelidade do aplicativo. Da esquerda para a direita: Tela de visão do saldo; Tela de endereços da carteira; Tela de criação de nova carteira; Tela de configurações.

O protótipo de alta fidelidade teve quatro versões, uma para cada tema do aplicativo. Este protótipo foi usado para desenho final dos componentes, escolha de cores, ícones, fontes e ajuste da navegação da interface gráfica.

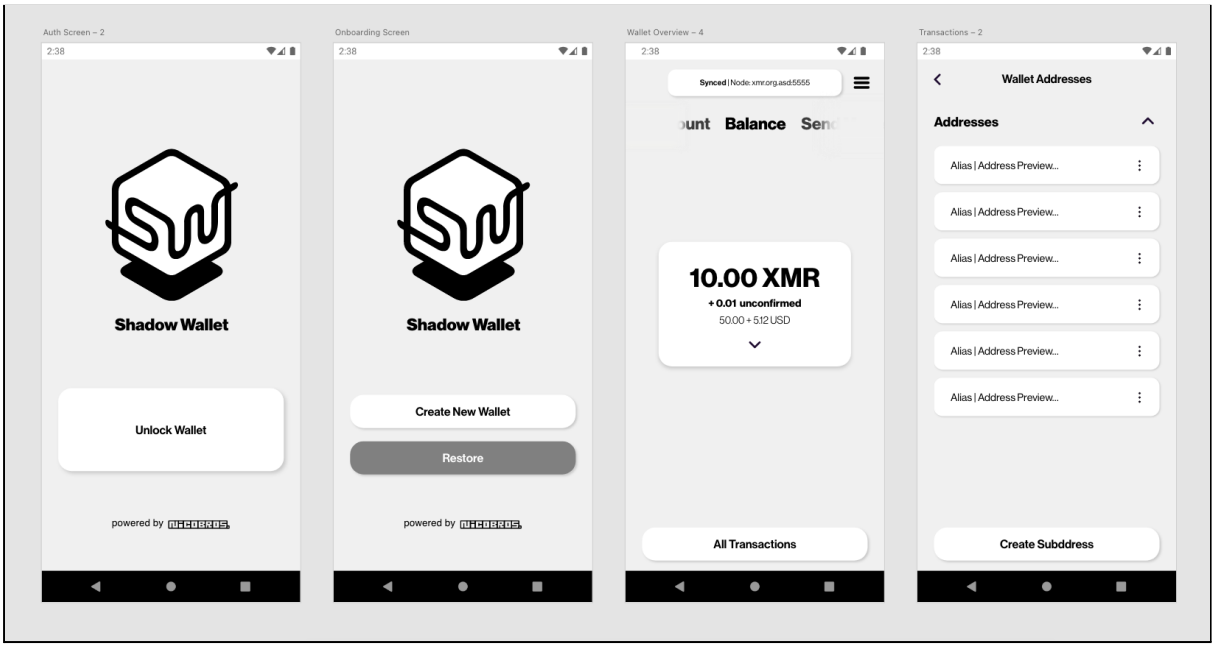


Figura 6: Protótipo de alta fidelidade do aplicativo, tema claro. Da esquerda para a direita: Tela de desbloqueio do aplicativo; Tela de boas-vindas; Tela de visão do saldo; Tela de lista de endereços da carteira.

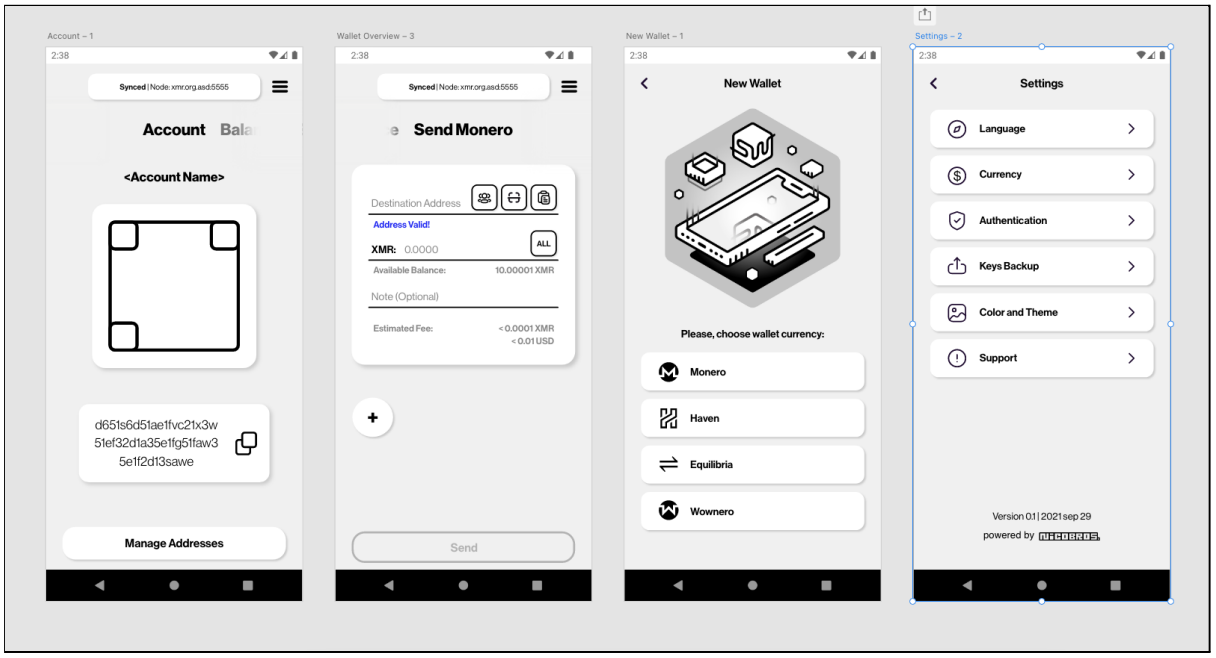


Figura 7: Protótipo de alta fidelidade do aplicativo, tema claro. Da esquerda para a direita: Tela de endereços da carteira; Tela de envio de *Monero*; Tela de escolha da moeda de nova carteira; Tela de configurações.

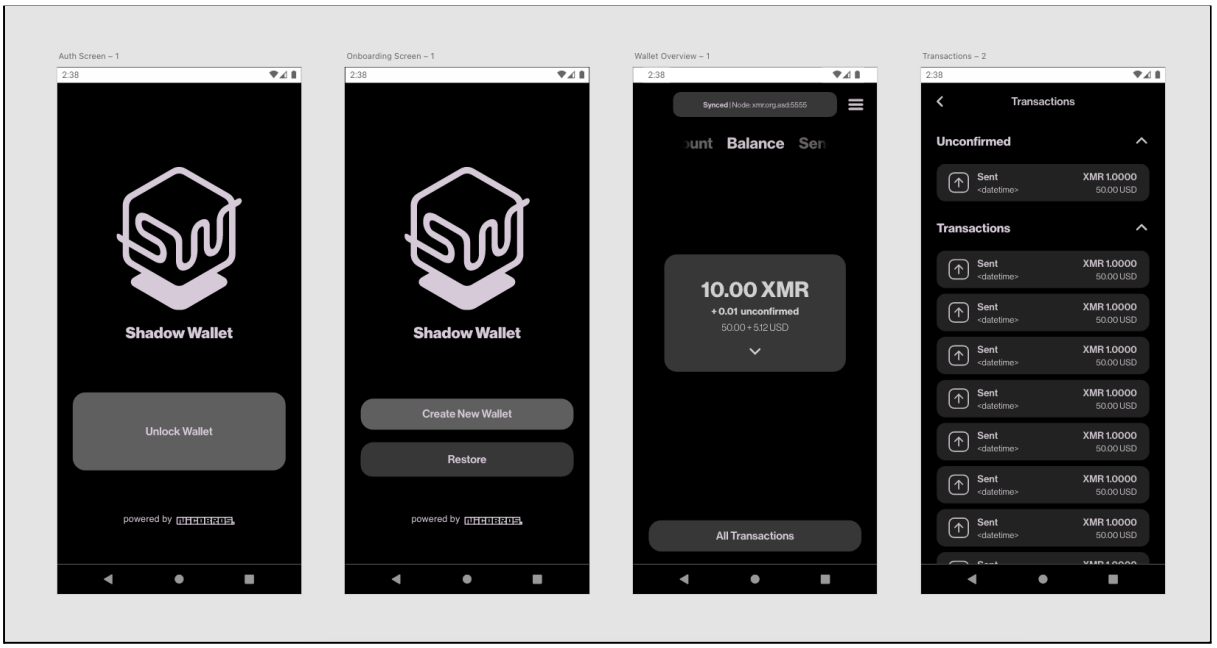


Figura 8: Protótipo de alta fidelidade do aplicativo, tema escuro. Da esquerda para a direita: Tela de desbloqueio do aplicativo; Tela de boas-vindas; Tela de visão do saldo; Tela de lista de endereços da carteira.

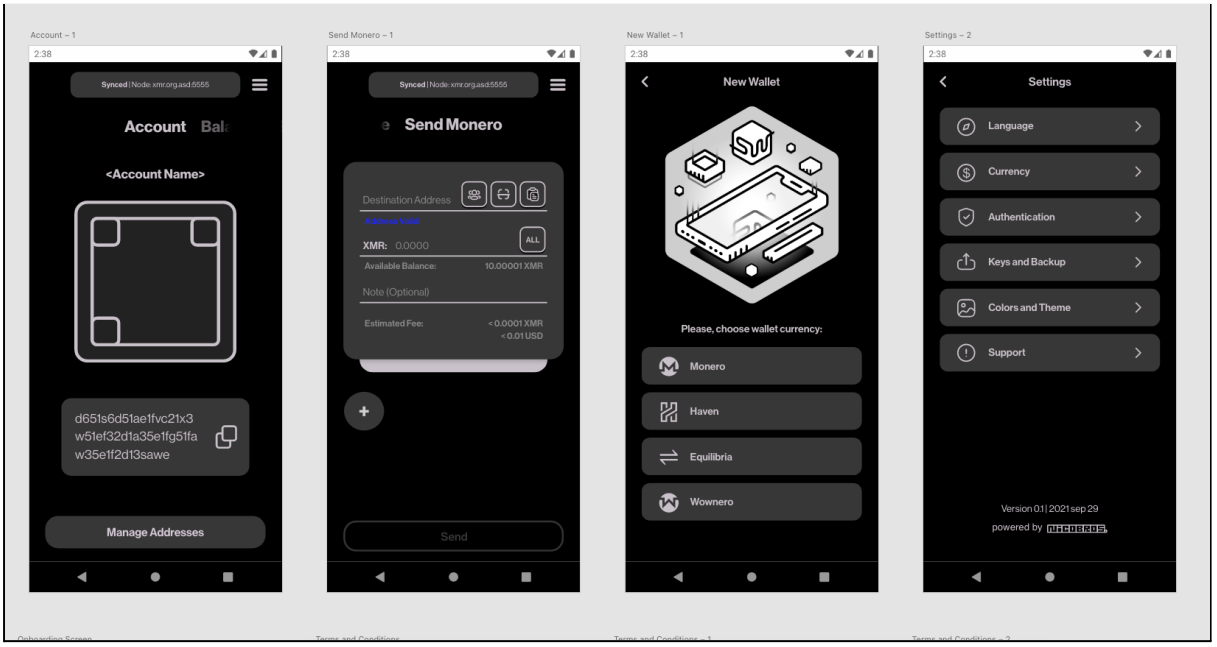


Figura 9: Protótipo de alta fidelidade do aplicativo, tema escuro. Da esquerda para a direita: Tela de endereços da carteira; Tela de envio de Monero; Tela de escolha da moeda de nova carteira; Tela de configurações.

A prototipagem auxiliou a tomada de decisão sobre a interface do aplicativo, evitando retrabalho em telas que não foram planejadas corretamente. Além disso, o mesmo serviu

como guia para a implantação do front-end, já que o visual e interação entre componentes já estava definido.

4.2 – Lógica da Aplicação

A aplicação tem uma arquitetura em camadas, onde o módulo do núcleo é encapsulado por *wrapper*, dentro da camada lógica, com a interface gráfica apenas interagindo com essa camada superior. Desta forma, camadas mais internas não conseguem acessar camadas superiores, e é necessário o uso de *observables* [46], um design-pattern utilizado para comunicação entre entidades com relacionamentos um-para-vários, para notificar mudanças de estado da entidade singular para as outras várias entidades relacionadas, que precisam reagir à essa alteração. Ele é utilizado para garantir que a camada de núcleo não terá seu estado alterado por ninguém além de suas próprias funções, e que o mesmo estará sempre com as últimas atualizações.

A classe `App`, raíz da *Shadow Wallet*, possui a função `build( )`, definida na interface `StatelessWidget` do *Flutter*. Ela retorna um *observable* que contém o componente base da aplicação, chamado de `Root`, além de configurações iniciais e os componentes base:

```
class App extends StatelessWidget {
    App() { ... }

    @override
    Widget build(BuildContext context) {
        return Observer(builder: (BuildContext context) {
            return Root(
                authenticationStore: authenticationStore,
                navigatorKey: navigatorKey,
                child: MaterialApp(...)
            );
        });
    }
}
```

Figura 10: Trecho de código da classe `App`, com exemplo do seu método `builder( )`.

Esta classe encapsula a aplicação inteira dentro de um *observable*. A alteração dos dados contidos neste *observable* é limitada, sendo permitida apenas para alguns fluxos e chamadas da aplicação. Todos os componentes contidos em camadas interiores estão fundamentados nesse componente.

A camada lógica é dividida em pacotes menores, que gerenciam o estado da aplicação. Esses pacotes foram implementados seguindo a modularidade da carteira, e são coleções de classes e funções para auxiliar na lógica da aplicação:

- `core`
- `entities`
- `monero`
- `reactions`
- `store`
- `themes`
- `utils`
- `view_model`

Em `core`, há as funcionalidades críticas para o gerenciamento de uma carteira genérica de criptoativos: criar ou restaurar uma carteira de criptoativos; criar e assinar transações; validar endereços para recebimento; criar e restaurar *backups*; e conectar a um nodo específico ou aleatório da rede. Nele estão todas as chamadas necessárias para atualizar o estado no núcleo da aplicação.

Em `entities`, há os modelos de dados utilizados na aplicação. Nele também estão contidos os modelos para moedas que possam futuramente ser suportadas pela aplicação.

Em `monero`, estão implementadas funcionalidades específicas do *Monero*, como validação de endereços e criação de endereços *stealth*.

Em `reactions`, há funções para execução assíncrona de tarefas quando uma outra ação pré-definida é completada, como a atualização da rede.

Em `store` contém classes para armazenar informações sensíveis, como as chaves *SSH*, lista de nodos conectados e *backups*.

Em `themes`, são definidas variáveis importantes para a interface gráfica do aplicativo, como cores primárias, secundárias, fontes e seus tamanhos.

Em `utils`, há classes e funções auxiliares, utilizadas em diversas partes do aplicativo, como formatação de datas, formatação de unidades monetárias e idiomas.

Em `view_model`, há uma representação dos modelos em *observables,* para apresentá-los na camada de interface gráfica. Assim, esse pacote permite a alteração dos modelos apenas pelas ações assíncronas contidas dentro do mesmo, limitando que a interface gráfica faça chamadas que não estejam definidas pelo modelo.

4.2.1 – Núcleo & *Wrapper*

Na camada de núcleo, foi utilizada a biblioteca do *Monero* na *monero-project*, que é codificada em *C++*. A biblioteca *dart:ffi* [47] foi usada para chamar funções da biblioteca de *Monero* do núcleo, implementadas em *C* e *C++*. O uso dessa biblioteca permite incluir no projeto, que foi implementado em *Dart*, a biblioteca pública mais utilizada, revisada e testada do *Monero*. Assim, dispensou-se a implementação das funcionalidades de *blockchain* em *Dart*, o que cria uma dependência mas evita possíveis erros e falhas de segurança.

A implementação dessas chamadas é abstraída pelo seguinte código:

```
int getCurrentHeight() => getCurrentHeightNative();
bool isNeededToRefresh() => isNeededToRefreshNative() =/= 0;
String getFileName() => convertUTF8ToString(pointer: getFileName());
String getSeed() => convertUTF8ToString(pointer: getSeedNative());
String getAddress({int accountIndex = 0, int addressIndex = 0}) =>
    convertUTF8ToString(pointer: getAddressNative(accountIndex,
    addressIndex));
int getFullBalance({ int accountIndex = 0 }) =>
    getFullBalanceNative(accountIndex);
int getUnlockedBalance({ int accountIndex = 0 }) =>
    getUnlockedBalanceNative(accountIndex);
bool isConnectedSync() => isConnectedNative();
void startRefreshSync() => startRefreshNative();
void setRefreshFromBlockHeight({ int height }) =>
    setRefreshFromBlockHeightNative(height);
void closeCurrentWallet() => closeCurrentWalletNative();
String getSecretViewKey() =>
    convertUTF8ToString(pointer: getSecretViewKeyNative());
String getSecretSpendKey() =>
    convertUTF8ToString(pointer: getSecretSpendKeyNative());
String getPublicViewKey() =>
    convertUTF8ToString(pointer: getPublicViewKeyNative());
String getPublicSpendKey() =>
    convertUTF8ToString(pointer, getPublicSpendKeyNative());
```

Figura 11: Trecho de código do conjunto de chamadas do *Dart*, que encapsulam métodos do *C++*.

O trecho de código acima contém as funções implementadas em *Dart* que chamam as funções em *C++*, descrevendo a assinatura e o tipo de retorno delas. Cada função está relacionada a uma operação com a *blockchain*, implementada direto no núcleo. Esta *API* permite que sejam utilizados binários implementados em *C++* dentro da aplicação.

A função `getSyncingHeight( )` obtém a altura do bloco em que a *blockchain* do *Monero* está atualmente.

A função `isNeededToRefresh( )` confere, através da altura do bloco, se os dados locais precisam ser atualizados.

A função `getFileName`() retorna o nome onde estão guardadas as chaves privadas da carteira.

A função `getSeed()` retorna a *seed* da carteira.

A função `getAddress( )` retorna endereços públicos para recebimento de *Monero*, aceitando a conta e o índice do endereço a ser resgatado.

A função `getFullBalance({int accountIndex=0, int addressIndex=0})` retorna o saldo, em *Monero*, da conta inserida como argumento.

A função `getUnlockedBalance( )` retorna o saldo, em *Monero*, que pode ser enviado em uma transação.

A função `isConnectedSync( )` retorna verdadeiro ou falso se a carteira está conectada e sincronizada com a rede.

A função `startRefreshSync( )` inicia a sincronização da carteira com a rede.

A função `setRefreshFromBlockHeight({ int height })` inicia a sincronização com a *blockchain*, a partir do bloco passado como argumento.

A função `closeCurrentWallet( )` encripta e fecha a carteira apropriadamente, e, apesar de ser uma função do núcleo, é usada no momento que o usuário fecha a aplicação.

As funções `getSecretViewKey()` e `getSecretSpendKey()` são usadas para consultar e utilizar as chaves privadas da carteira. As funções `getPublicViewKey( )` e `getPublicSpendKey( )` servem para consultar e utilizar as chaves públicas da carteira de *Monero*.

4.2.2 – Camada Lógica

A camada lógica foi codificada em *Dart*, e sua implementação seguiu o modelo de *singletons* registrados, com um domínio comum à toda a aplicação. Utilizando o *Hive* [48], uma biblioteca gerenciadora das instâncias estáticas que serão acessados por diversas outras classes, é possível fazer uso dos *singletons* de qualquer lugar na aplicação. Estes *singletons* são injetados na função `main`, através da função `registerAdapter( )` do *Hive,* que toma como entrada quaisquer classes de serviço.

O código abaixo exemplifica esta implementação:

```
Future<void> main() async {
    await Hive.close();
    Hive.init(appDir.path);
    if(!Hive.isAdapterRegistered(Contact.typeId)) {
        Hive.registerAdapter(ExampleAdapter());
    }

    ...
}
```

Figura 12: Trecho de código da função `Main( )` do aplicativo, definido pelo *Dart*, com a configuração do *Hive*.

Desta maneira, é possível evitar instâncias duplicadas de classes que alteram o estado do núcleo da aplicação, permitindo maior controle e segurança nas chamadas que acessam as funções relacionadas à *blockchain*. Neste ponto, o *software* está funcional para executar transações de *Monero*, apesar de fazê-las apenas via linha de comando.

4.3 – Interface Gráfica

Com um protótipo maduro e a lógica implementada, foi implantada a interface gráfica seguindo o modelo definido no protótipo. Dentro deste módulo, há apenas a implementação de elementos de interface gráfica, como botões, fontes, campos de texto, formulários, telas de visão, telas de configuração, menus e telas de autenticação. Sua implementação está dividida entre componentes e telas.

A interface gráfica permite ao usuário requisitar alterações para o `core`, que é responsável por atualizar os componentes da camada lógica. As chamadas que cada tela pode realizar para o `core` são definidas no pacote `view_model`, ambos da camada lógica.

Quando uma chamada da interface gráfica é feita ao `core`, uma promessa de resposta assíncrona é retornada para a interface gráfica. Durante esta promessa, o `core` chama o *wrapper* que, por sua vez, executa a funcionalidade nativa no núcleo, que foi requisitada pelo usuário. Ao término da execução, o *wrapper* retorna o resultado para o `core`, que atualiza seu estado, enviando um retorno à chamada assíncrona inicial. Através de uma função do *observable*, a interface gráfica consegue analisar o estado atual do núcleo pelo `core`, atualizando seu próprio estado de acordo.

Os componentes foram implementados utilizando `StatelessWidgets`, um componente abstrato na biblioteca do *Flutter,* que auxilia na implementação de outros componentes sem registro de estado. Isso permite criar uma classe base para cada componente, mas que será preenchido de acordo com seu contexto, já que o mesmo componente gráfico será usado múltiplas vezes em diferentes telas.

Todas as telas da aplicação estendem a classe `BasePage`, que contém propriedades comuns e necessárias a todas as páginas da interface gráfica. Entre as propriedades, a mais relevante é a `rootWrapper`, que contém uma referência ao componente base, definido na camada de aplicação. A implementação desses componentes foi padronizada para facilitar a alternância entre páginas e limpeza de memória das mesmas.

O trecho abaixo possui um exemplo para criação de uma classe de página e widget:

```
class Widget extends StatelessWidget {
    Widget() { ... }

    @override
    Widget build(BuildContext context) {
        return ButtonTheme(...);
    }
}
```

Figura 13: Trecho de código da classe `Widget`. Esta classe é definida pela biblioteca do *Flutter*, e é estendida por todos os componentes gráficos na aplicação.

```
class BasePage extends StatelessWidget {
    BasePage() { ... }

    Widget Function(BuildContext, Widget) get rootWrapper => null;

    Widget body(BuildContext context);

    @override
    Widget build(BuildContext context) {
        final root = Scaffold(body: body(context), ...);
        return rootWrapper?.call(context, root) ?? root;
    }
}
```

Figura 14: Trecho de código da classe `BasePage`. Esta classe é uma implementação customizada da *Shadow Wallet*, e é estendida por todas as páginas da aplicação.

4.4 – Produção do Aplicativo Android e iOS

Finalmente, a aplicação da *Shadow Wallet* é compilada em um pacote .apk, podendo ser instalada em um dispositivo com o sistema operacional *Android* ou *iOS*. Neste processo, é necessário baixar e instalar diretamente a aplicação, pois é uma produção mais artesanal, que não conta com uma esteira de *deployment* do mesmo.

5 Conclusões

Blockchain é uma tecnologia interessante para organizações distribuídas. O objetivo primário do *Bitcoin* é ser um bem digital e escasso, que funcione como reserva de valor, e que dê maior liberdade e autonomia para seus usuários. Em nossa visão, o *Bitcoin* é uma excelente moeda, possui fundamentos fortes e é excelente reserva de valor, mas que possui falhas fundamentais de privacidade. Essas falhas permitem que a rede, desde usuários, nodos e mineradores, sejam alvos de ataques de engenharia social, como monitoramento em massa ou *spear phishing,* onde atacantes criam golpes direcionados a um indivíduo ou grupo. Durante o desenvolvimento, focamos em moedas que corrigem essas falhas, como o *Monero*. Acreditamos que ela é mais próxima da visão original que o criador do *Bitcoin* propôs para uma moeda virtual universal, distribuída e livre de censura.

O conhecimento acadêmico sobre *blockchain* é pouco desenvolvido, pois a tecnologia é relativamente nova, tendo uma validação antes econômica do que técnica. Seus fundamentos foram postos à prova na prática, através de experimentação, tentativas e erros. O *Bitcoin* completou 13 anos de operação em janeiro de 2022 e já ganhou adoção por governos, como *El Salvador* [49], e empresas como o, Itaú e BTG Pactual, provando que tem força institucional, que é uma boa reserva de valor e que pode ser melhorado ao longo do tempo.

O objetivo com a *Shadow Wallet* é permitir uma difusão e adoção dos criptoativos privados de maneira custodial, em que os usuários estarão em posse direta de suas chaves públicas e privadas, através de uma aplicação confortável e conveniente. O aplicativo não possui fins lucrativos e nem cobrará quaisquer taxas aos usuários, mas dará suporte para o pagamento de taxas pelas redes suportadas. Doações serão bem-vindas para suporte do projeto, mas não serão necessárias para a manutenção. Através de um *software* livre,

espera-se aumentar a base de conhecimento e implementação relacionada aos criptoativos com foco em privacidade.

A prototipagem levou mais tempo do que previsto, pela dificuldade de definir a identidade visual da *Shadow Wallet* e aspectos visuais que fossem simples mas que não atrapalhasse o uso da carteira. Assim que a logo e os botões primários conseguiram ser definidos, os temas conseguiram ser derivados com maior facilidade.

Durante a codificação, houveram dificuldades com a compilação dos binários de *Monero* para incluir no aplicativo, pois não tínhamos experiência com a api do *dart:ffi*. Algumas semanas foram necessárias para finalmente incluir a biblioteca de maneira adequada, e então seguir para a etapa de codificar o *front-end* da aplicação, que não levou tanto tempo.

Para os fins deste trabalho de conclusão de curso, não foi implementada uma esteira para *deployment* da mesma na *Play Store* ou *App Store*, apesar deste ser o destino de aplicações comerciais para celular. Entretanto, a aplicação é completamente funcional, e é possível instalar o .apk diretamente pelo computador e utilizá-la em produção.

De modo geral, estamos satisfeitos com o processo e os resultados obtidos. Com essa primeira versão, será possível seguir com o projeto para que se torne algo utilizável, escalável e focado no usuário final. Com o código em repositório público, esperamos conseguir adoção ampla e suporte da comunidade, para trabalhar em melhorias contínuas.

Pessoalmente, tivemos a oportunidade de revisar a bibliografia relevante a respeito de *blockchain* e criptomoedas, aprofundando nosso conhecimento técnico sobre este tema que acompanhamos desde 2016. Certamente temos maior clareza sobre o tema, e passamos a conseguir apresentá-los de forma simples e intuitiva, até mesmo para não-acadêmicos e estudantes de outros cursos de graduação.

Profissionalmente, a engenharia de software do aplicativo tem estrutura similar às engenharias usadas pelo mercado para construção de aplicativos mobile. Apesar do processo envolver apenas dois desenvolvedores, não foi uma implementação simples, já que a carteira faz comunicação direta com a *blockchain*. Foi uma excelente oportunidade para aprimorar habilidades de desenvolvimento voltado à *blockchain*, habilidades essas que estão em altíssima demanda no mercado.

Por ser um universo que muda muito rapidamente, preferimos focar na discussão dos aspectos técnicos da blockchain a focar no status de sua capitalização. Esperamos que este trabalho sirva de entrada para alunos de graduação interessados nos temas *blockchain* e criptomoedas. Trabalhos que podem ser derivados deste TCC incluem a expansão das funcionalidades da carteira, como a versão web, versão desktop com suporte para um nodo

completo, suporte para mineração, e a inclusão de moedas privadas com outras funcionalidades, como o *Haven* [37], *DERO* [28], *Equilibria* [38] e *Wownero* [39].

REFERÊNCIAS

[1] NAKAMOTO, S. **Bitcoin**: A Peer-to-Peer Electronic Cash System. Julho de 2009. Disponível em: bitcoin.org/bitcoin.pdf. Acesso em: 12 de fevereiro de 2022.

[2] YOUNG, J. Cryptocurrency market cap hits $2 trillion – Now worth as much as Apple. **Cointelegraph**. Abril de 2021. Disponível em: cointelegraph.com/news/crypto currency-market-cap-hits-2-trillion-now-worth-as-much-as-apple. Acesso em: 12 de fevereiro de 2022.

[3] DEMIR, E.; BILGIN, H.; KARABULUT G. **The relationship between cryptocurrencies and COVID-19 pandemic**. Eurasian Economic Review; 2020; 10: 349–360. DOI: doi.org/10.1007/s40822-020-00154-1.

[4] HUAWEI, H.; WEI, K.; SICONG, Z.; ZIBIN, Z.; SONG G. **A Survey of State-of-the-Art on Blockchains**: Theories, Modelings, and Tools. ACM Computing Surveys; Abril de 2021; 54 (2a): Artigo 44. DOI: doi.org/10.1145/3441692.

[5] LAKHANI, K. R.; VON HIPPEL, E. **How Open Source Software Works**: “Free” User-to-User Assistance. In: Herstatt C., Sander J.G. (eds) Produktentwicklung mit virtuellen Communities. Gabler Verlag. DOI: doi.org/10.1007/978-3-322-84540-5_13.

[6] AMERICAN NATIONAL STANDARDS INSTITUTE. **Public Key Cryptography For The Financial Services Industry**: The Elliptic Curve Digital Signature Algorithm (ECDSA). Setembro de 1998. DOI: doi.org/10.1.1.202.2977.

[7] DECKER, C.; WATTENHOFER, R. **Information propagation in the Bitcoin network**. IEEE P2P 2013 Proceedings. 2013; 1-10. DOI: doi.org/10.1109/P2P.2013.6688704.

[8] PHAM, L.; TRAN, H,.; PHAN, D.; DUONG, T.; LAM, D.; NAKASHIMA, Y. **Double SHA-256 Hardware Architecture With Compact Message Expander for Bitcoin Mining**. IEEE Access. 2020; 8; 139634-139646. DOI: doi.org/10.1109/ACCESS.2020.3012581.

[9] WOOD, G. **Ethereum**: A Secure Decentralised Generalised Transaction Ledger. 2014. Disponível em: ethereum.github.io/yellowpaper/paper.pdf. Acesso em: 12 de fevereiro de 2022.

[10] LAMPORT, L.; FISCHER, M. **Byzantine Generals and Transaction Commit Protocols**. National Science Foundation. Abril de 1982. DOI: doi.org/10.1145/3335772.3335936.

[11] MECHKAROSKA, D.; DIMITROVA, V.; POPOVSKA-MITROVIKJ, A. **Analysis of the Possibilities for Improvement of BlockChain Technology**. 26th Telecommunications Forum (TELFOR), Novembro de 2018; 1-4, DOI: doi.org/10.1109/TELFOR.2018.8612034.

[12] VISA. **Small Business Retail**. Disponível em: usa.visa.com/run-your-business/small-business-tools/retail.html. Acesso em: 12 de fevereiro de 2022.

[13] LOMBROZO, E.; LAU, J.; WUILLE, P. **BIP 141**: Segregated Witness. 2016. Disponível em: github.com/bitcoin/bips/blob/master/bip-0141.mediawiki. Acesso em: 12 de fevereiro de 2022.

[14] POON J.; DRYJA, T. **The Bitcoin Lightning Network**: Scalable Off-Chain Instant Payments. Janeiro de 2016. Disponível em: lightning.network/lightning-network-paper.pdf. Acesso em: 12 de fevereiro de 2022.

[15] Market Cap of Gold (precious metal). **Companies Market Cap**. Disponível em: companiesmarketcap.com/gold/marketcap. Acesso em: 12 de fevereiro de 2022.

[16] BURSZTYNSKY, J. Apple becomes first U.S. company to reach a $2 trillion market cap. **CNBC**. 19 de Agosto de 2020. Disponível em: cnbc.com/2020/08/19/apple-reaches-2-trillion-market-cap.html. Acesso em: 12 de fevereiro de 2022.

[17] Top Cryptocurrency Exchanges Ranked by Volume. **Coin Market Cap**. Disponível em: coinmarketcap.com/rankings/exchanges. Acesso em: 12 de fevereiro de 2021.

[18] Global Cryptocurrency Charts. **Coin Market Cap**. Disponível em: coinmarketcap.com/charts . Acesso em: 12 de fevereiro de 2022.

[19] YAKOVENKO, A. **Solana**: A new architecture for a high performance blockchain. Novembro de 2017. Disponível em: solana.com/solana-whitepaper.pdf. Acesso em: 12 de fevereiro de 2022.

[20] KIAYIAS, A.; RUSSELL, A.; DAVID, B.; OLIYNYKOV, R. **Ouroboros**: A Provably Secure Proof-of-Stake Blockchain Protocol. Springer, Cham. Advances in Cryptology – CRYPTO 2017. Lecture Notes in Computer Science, 10401. Julho de 2017. DOI: doi.org/10.1007/978-3-319-63688-7_12.

[21] ADAMS, H.; ZINSMEISTER, N.; SALEM, M.; KEEFER, R.; ROBINSON, D. **Uniswap v3 Core**. Março de 2021. Disponível em: uniswap.org/whitepaper-v3.pdf. Acesso em: 12 de fevereiro de 2022.

[22] SUSHISWAP. **Sushiswap Documentation**. Disponível em: docs.sushi.com. Acesso em 12 de fevereiro de 2022.

[23] TETHER OPERATIONS LIMITED. **Tether**: Fiat currencies on the Bitcoin blockchain. 2014. Disponível em: assets.ctfassets.net/vyse88cgwfbl/5UWgHMvz071t2Cq5yTw5vi/c9798ea8db99311bf90ebe0810938b01/TetherWhitePaper.pdf. Acesso em 12 de fevereiro de 2022.

[24] BREIDENBACH, L.; CACHIN, C.; CHAN, B.; COVENTRY, A. **Chainlink 2.0:** Next Steps in the Evolution of Decentralized Oracle Networks. Abril de 2021. Disponível em: research.chain.link/whitepaper-v2.pdf. Acesso em: 12 de fevereiro de 2022.

[25] Charles C. **Pax Gold White Paper**. Setembro de 2021. Disponível em: paxos.com/pax-gold-whitepaper. Acesso em: 12 de fevereiro de 2022.

[26] TETHER OPERATIONS LIMITED. **Tether Gold:** A Digital Token Backed by Physical Gold. Janeiro de 2022. Disponível em: gold.tether.to/Tether%20Gold%20Whitepaper.pdf . Acesso em: 12 de fevereiro de 2022.

[27] GOEKING, V. BTG lança plataforma de cripto para concorrer com Mercado Bitcoin e outras. **Valor Investe**. Setembro de 2021. Disponível em: valorinveste.globo.com/mercados/cripto/noticia/2021/09/20/btg-lanca-plataforma-de-cripto-para-concorrer-com-mercado-bitcoin-e-outras.ghtml. Acesso em: 12 de fevereiro de 2022.

[28] Dogecoin. Disponível em: dogecoin.com. Acesso em: 12 de fevereiro de 2022.

[29] CRYPTOKITTIES TEAM. **CryptoKitties WhitePapurr V2**. Disponível em: drive.google.com/file/d/1soo-eAaJHzhw_XhFGMJp3VNcQoM43byS/view. Acesso em: 12 de fevereiro de 2022

[30] BLOCKSTREAM CORPORATION INC. **Blockstream Docs**. Disponível em: docs.blockstream.com. Acesso em: 12 de fevereiro de 2022.

[31] VAN SABERHAGEN, N. **CryptoNote v 2.0**. Outubro de 2013. Disponível em: web.getmonero.org/resources/research-lab/pubs/whitepaper_annotated.pdf. Acesso em: 12 de fevereiro de 2022.

[32] DERO COMMUNITY. **Dero Project.** Outubro de 2018. Disponível em: github.com/deroproject/documentation/blob/master/Dero_Whitepaper.pdf . Acesso em: 12 de fevereiro de 2022.

[33] PEDERSEN, T. P. **Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing.** Advances in Cryptology - CRYPTO 1991. Lecture Notes in Computer Science, volume 576. Springer, Berlin, Heidelberg. DOI: doi.org/10.1007/3-540-46766-1_9.

[34] BÜNZ, B.; BOOTLE, J.; BONEH, D.; POELSTRA, A.; WUILLE, P.; MAXWELL, G. **Bulletproofs**: Short Proofs for Confidential Transactions and More. 2018 IEEE Symposium on Security and Privacy. Maio de 2018; 315-334, DOI: doi.org/10.1109/SP.2018.00020.

[35] THE TOR PROJECT. **Tor network.** Disponível em: torproject.org. Acesso em: 14 de março de 2022.

[36] WANG, Z.; WU, J.; CHEN, H.; CHAO, M.; YANG, C. **Micro-Architecture Optimization for Low-Power Bitcoin Mining ASICs**. 2019 International Symposium on VLSI Design, Automation and Test (VLSI-DAT), Abril de 2019; 1-4. DOI: doi.org/10.1109/VLSI-DAT.2019.8741726.

[37] HAVEN PROTOCOL. **Private Decentralized Finance**. Disponível em: docs.havenprotocol.org/whitepapers/english.pdf . Acesso em: 12 de fevereiro de 2022.

[38] EQUILIBRIA. **Equilibria Oracle Network**. Disponível em: equilibria.network. Acesso em: 12 de fevereiro de 2022.

[39] WOWNERO. **Wownero Whitepaper**. Disponível em: wownero.org/whitepaper.pdf. Acesso em: 12 de fevereiro de 2022.

[40] The Monero Project Repository. Disponível em: github.com/monero-project. Acesso em: 12 de fevereiro de 2022.

[41] GOOGLE LLC. **Dart Documentation**. Disponível em: dart.dev/guides. Acesso em: 12 de fevereiro de 2022.

[42] GOOGLE LLC. **Flutter Documentation**. Disponível em: docs.flutter.dev. Acesso em: 12 de fevereiro de 2022.

[43] GOOGLE LLC. **Android Keystore System**. Disponível em: developer.android.com/training/articles/keystore. Acesso em: 12 de fevereiro de 2022.

[44] DEFUSE SECURITY. **Salted Password Hashing** - Doing it Right. Disponível em: https://crackstation.net/hashing-security.htm. Acesso em: 12 de fevereiro de 2022.

[45] BARRETT, D, et al. **SSH, the Secure Shell**: the definitive guide. 1. ed. O'Reilly Media Inc; 2001.

[46] GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns**: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1994.

[47] GOOGLE LLC. **C interop using dart:ffi**. Disponível em: dart.dev/guides/libraries/c-interop. Acesso em: 12 de fevereiro de 2022.

[48] Hive Docs. Disponível em: docs.hivedb.dev. Acesso em: 12 de fevereiro de 2022.

[49] RENTERIA, N.; ESPOSITO, A. El Salvador se torna o primeiro país a adotar bitcoins como moeda oficial. **CNN Brasil**. Setembro de 2021. Disponível em: cnnbrasil.com.br/business/el-salvador-se-torna-o-primeiro-pais-a-adotar-bitcoins-como-moeda-oficial/. Acesso em: 12 de fevereiro de 2022.

[ANEXO I] Shadow Wallet - Documento de Requisitos de Software

Shadow Wallet
Documento de Requisitos de Software

Daniel Calonico
Gustavo Calonico

10 de Fevereiro de 2022

Histórico de Versões			
Versão	**Data**	**Mudanças**	**Responsável**
1.0	12/07/2021	Inicialização do Texto	Daniel Calonico
1.1	19/07/2021	Detalhamento dos Casos de uso	Daniel Calonico
1.2	26/07/2021	Detalhamento dos BPMs e Glossário	Daniel Calonico
1.3	10/02/2022	Descrição dos BPMs e edição do layout.	Daniel Calonico

Sumário

1. Introdução

Este documento descreve a *Shadow Wallet*, um aplicativo multiplataforma que funciona como gerenciador de carteiras para a criptomoeda *Monero*. O documento descreve seu modelo de projeto, além dos requisitos funcionais, requisitos não-funcionais e casos de uso e arquitetura de aplicação.

2. Visão Geral

Monero é uma criptomoeda que utiliza *blockchain*, sendo uma bifurcação do código do *Bitcoin*. A diferença de sua implementação é permitir que os usuários troquem unidades monetárias de maneira completamente anônima e ofuscada, sendo praticamente impossível para um terceiro não envolvido rastrear uma transação.

A *Shadow Wallet* é um aplicativo de celular, que funciona como gerenciador de carteiras *Monero*. Sua funcionalidade primária é permitir que usuários controlem seus ativos em *Monero*, podendo receber e enviar *Monero*, criar novas carteiras e importar suas carteiras antigas de *Monero*. O usuário também pode configurar seu oráculo [1] de *Monero*, podendo escolher sua origem de dados blockchain de preferência.

Focada em 2 aspectos primários: segurança e simplicidade, a carteira permite ao usuário configurar o bloqueio e desbloqueio da aplicação, usando um PIN (4 ou 6 dígitos) ou biometria (se seu dispositivo permitir). O bloqueio do aplicativo encripta as informações, para que elas não possam ser acessadas por outras aplicações.

Finalmente, o aplicativo permite optar por um modo claro e escuro da interface gráfica.

3. Regras de Negócio

O aplicativo possui 2 classes de regras de negócio: operações associadas à *blockchain*; e operações de aplicativo.

3.1 - Operações associadas à *blockchain*

Regras associadas à Blockchain seguem os protocolos de comunicação da rede, para que a aplicação consiga operar ativos em Monero. Estas regras farão operações dependentes de um Oráculo, uma interface que lê blocos e lança transações para a rede blockchain. Uma conexão com a internet é imprescindível para o cumprimento dessas regras.

Número	Descrição
RN1	Uma Carteira de Monero é um software que armazena localmente as informações de uma Conta, incluindo suas Chaves Privadas, transações e balanço final. Além disso, a carteira pode incluir um software de mineração de Monero, e uma lista de seus endereços e sub-endereços públicos.

RN2	Uma Carteira de Monero (Wallet) é representada por um par de Chaves Privadas: a Chave de Gasto e Chave de Visualização (Spend Key e View Key). Estas chaves podem ser representadas por uma combinação de 25 palavras mnemônicas, conhecida como Semente Mnemônica (Seed).
RN3	Uma Conta de Monero (Account) é identificada por um conjunto de 95 caracteres, começando com um “4” para endereços raíz, e 8 para endereços filhos, chamado de Endereço (Address).
RN4	Um endereço Monero é gerado a partir de uma Carteira, usando o par de Chaves Privadas, possuindo a relação de 1 para n. Uma Carteira pode ter n endereços, mas 1 endereço é sempre da mesma Carteira
RN5	Um sub-endereço Monero (Stealth Address) é um identificador associado a uma Carteira, e funciona como máscara da Conta principal, possuindo relação de 1 para n. Ou seja, uma Carteira pode ter n sub-endereços, mas os sub-endereços são sempre da mesma Carteira.
RN6	Uma carteira só pode ser adicionada/recuperada a um gerenciador de carteiras através das suas chaves privadas ou por sua Semente Mnemônica.
RN7	O usuário só pode enviar Monero para outro usuário se possuir a Chave de Gasto daquela Carteira. Um usuário que possui apenas a Chave de Visualização de uma carteira não pode fazer transferências.
RN8	Apenas transações com saldo são aceitas pela Blockchain do Monero, ou seja, um usuário não consegue transferir volumes maiores que o saldo de uma carteira.
RN9	Toda transação de Monero possui uma Taxa opcional de transação, paga aos Mineradores da rede, como incentivo para adicionarem e manterem novas informações à Blockchain. Esta taxa é configurável no momento da transação.
RN10	O usuário só pode revelar e visualizar transações de uma Conta se tiver a Chave de Visualização ou Gasto da mesma.
RN11	Toda visualização de carteira precisa estar sincronizada com a Blockchain do Monero.
RN12	Um Nodo da Blockchain é uma entidade que possui uma cópia de toda a Blockchain desta Rede, e que consegue ler e propagar informações para outros Nodos incluírem nesta Blockchain. Para leitura e escrita na Blockchain do Monero, é necessário ter um Nodo ativo.
RN13	Oráculos são serviços remotos com um Nodo ativo de Monero, que lêem e propagam informações para a internet através de uma API. Por limitações de hardware, o Gerenciador de Carteiras pode usar o serviço desses Oráculos, que pode ser configurado pelo usuário final da rede, que deseja fazer uma transação.

3.2 - Operações de aplicativo

Regras de aplicativo utilizam apenas a interface do sistema operacional, e estão restritas a operações locais que não precisam se comunicar à rede blockchain, nem com a Internet.

Número	Descrição
RN14	O acesso ao aplicativo precisa ser autenticado
RN15	O acesso a cada carteira individual precisa ser autenticado

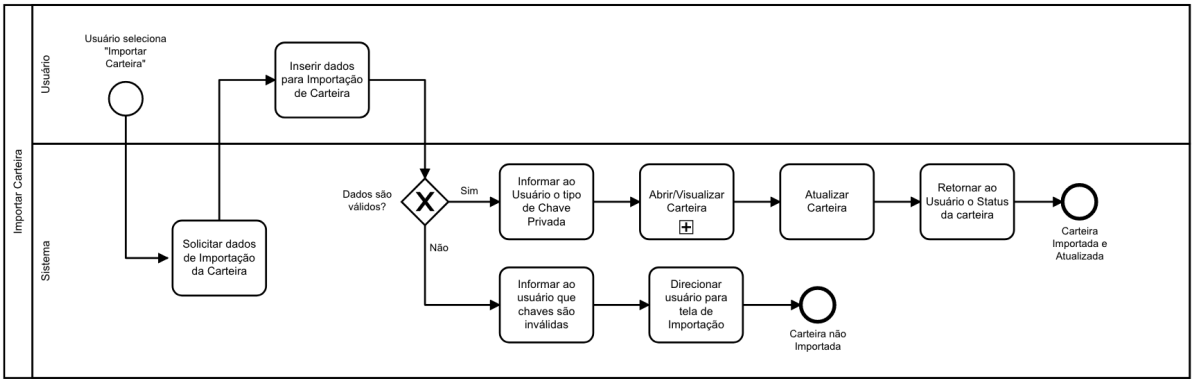
RN16	O Gerenciador de carteiras não guarda sessão de usuários, sendo permitido o uso por 1 usuário de cada vez.
RN17	Para receber pagamentos, o sistema deve disponibilizar um endereço para ser consultado sem Internet. Tal endereço pode ser consultado sem autenticação, mas deve ser um Stealth Address.

4. Modelo de Processos do Negócio

Para utilizar a carteira, o usuário deve instalar o aplicativo em seu dispositivo celular, com sistema operacional Android ou iOS.

OBS: Para melhor visualização dos diagramas, acesse:
https://cawemo.com/share/54e18863-ee2a-4397-939f-7b18294eaf02

4.1 - Importar Carteira (Restore Wallet)



Ao acessar a opção de Importar Carteira, o usuário deve escolher 1 entre 3 métodos:

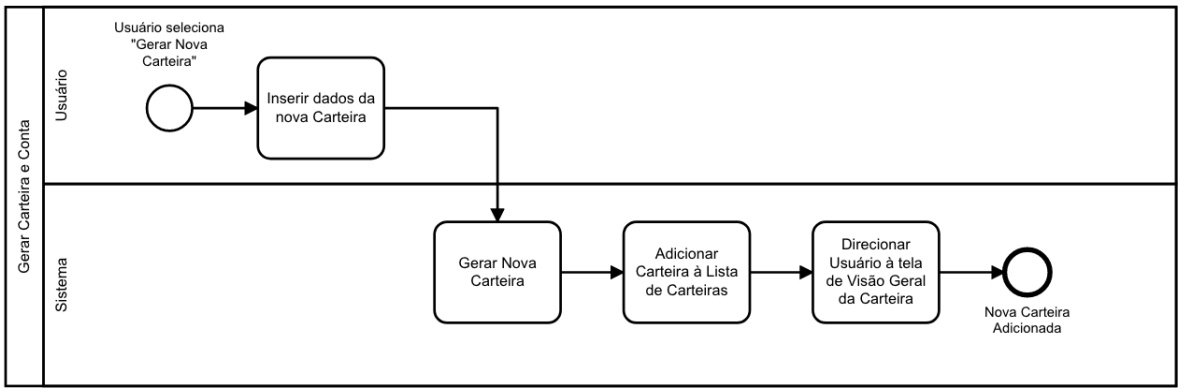
- Importar Carteira de Leitura (Restore View-Only Wallet)
- Importar Carteira Completa (Restore Wallet from Private Keys)
- Importar Carteira com 25 Palavras Seed (Mnemonic)

Caso o Usuário forneça um dado invalido, o aplicativo dispara um alerta de que o dado é inválido, sem interromper o usuário.

Assim que o usuário fornecer um dado válido, o aplicativo mostra um alerta, sem interromper o usuário. Este então nomeia a carteira e escolhe a data (ou altura) do bloco para importação. O aplicativo emite um alerta com as informações, pedindo para o usuário confirmar. O aplicativo então verifica os blocos em background, usando o Oráculo, e mostra uma notificação permanente sobre o processo.

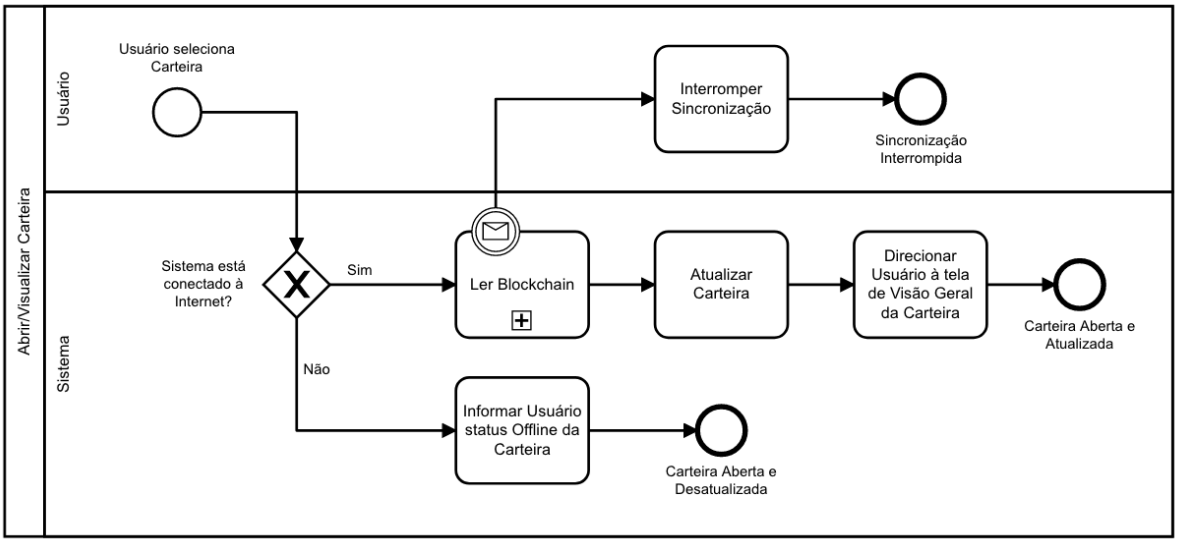
Caso a leitura de blocos seja interrompida, o aplicativo descarta os dados da carteira e lança uma notificação para que o usuário tente importar a carteira novamente. Se a leitura terminar com sucesso, o aplicativo atualiza informações da Carteira .

4.2 - Gerar Nova Carteira e Conta



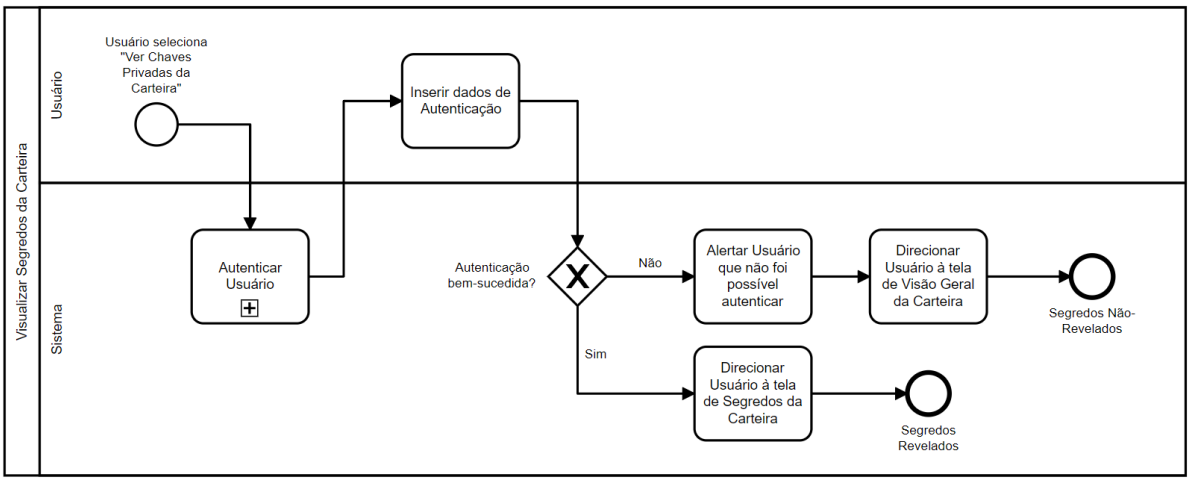
Ao acessar a opção de Gerar Nova Carteira, o usuário deve nomeá-la e dar uma senha. O aplicativo vai retornar dados sobre a nova carteira: Chave Pública, Seed Mnemônica, Altura de Restauração, View Key e Spend Key.

4.3 - Abrir/Visualizar Carteira



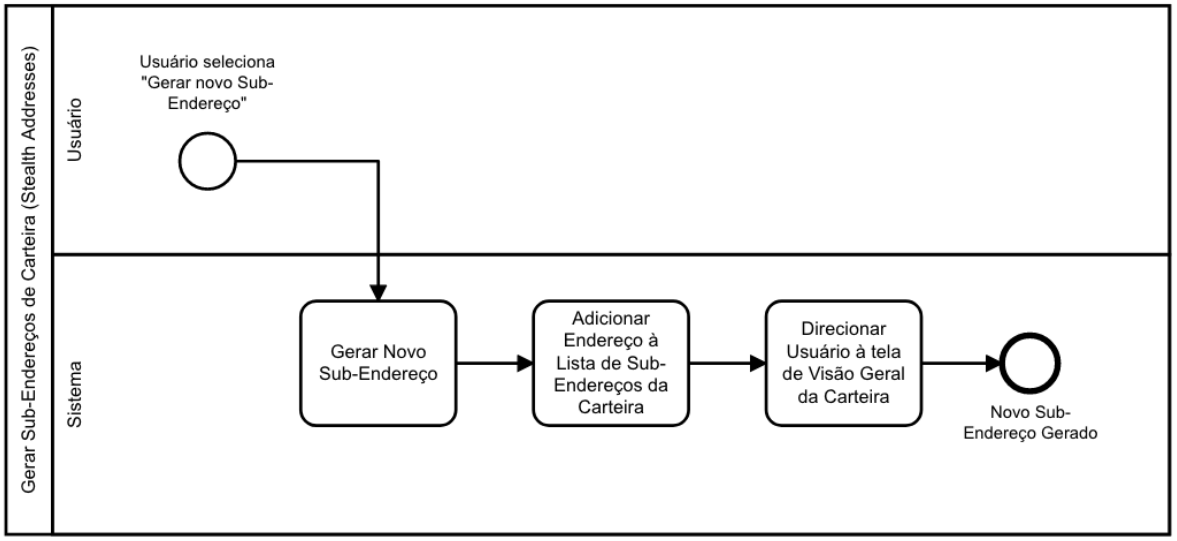
Quando um usuário seleciona uma carteira, o sistema deve verificar se está conectado com a internet. Caso não esteja conectado à internet, informa ao usuário que está offline, e abre a carteira sem atualizá-la. Caso esteja conectado à internet, o sistema lê a blockchain e atualiza a carteira, redirecionando o usuário à tela de visão geral da carteira. Caso o usuário encerre a leitura da blockchain, o sistema interrompe a sincronização.

4.4 - Visualizar Segredos da Carteira



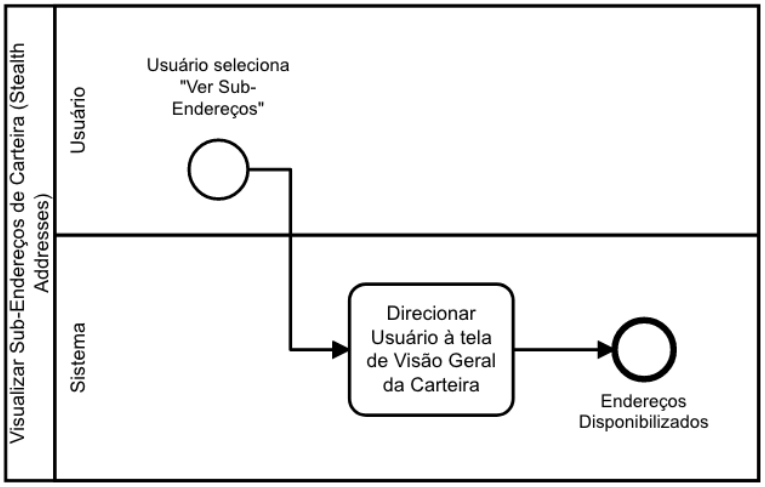
Ao acessar uma Carteira, o usuário pode escolher ver as chaves privadas da carteira. O sistema solicita os dados de autenticação do usuário, que deve os fornecer. Caso a autenticação seja bem-sucedida, o sistema revela os segredos. Caso não seja bem sucedida, o sistema direciona o usuário à tela de visão geral da carteira.

4.5 - Gerar Sub-Endereços de Carteira



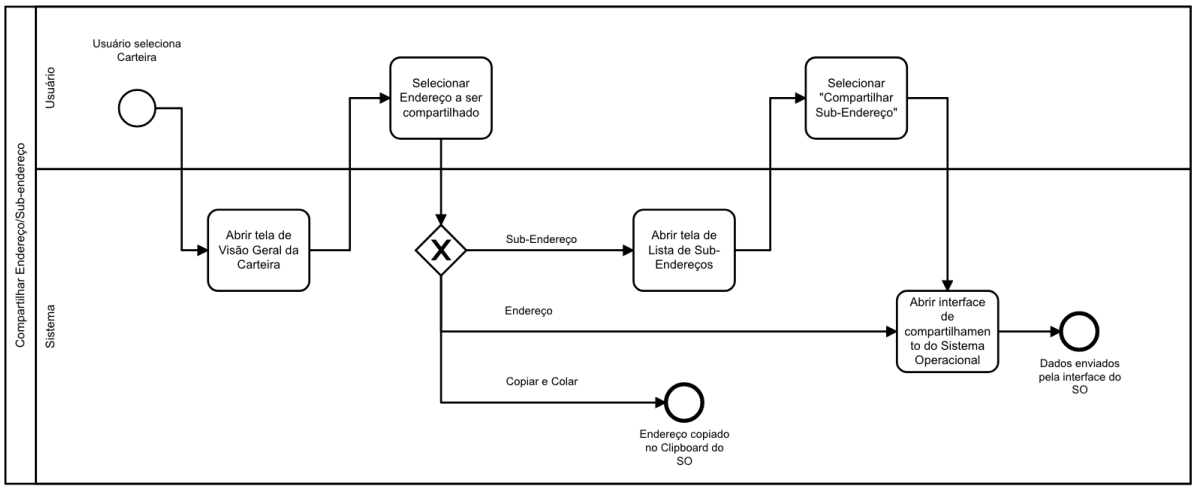
Quando o usuário seleciona “Gerar novo Sub-Endereço”, o sistema gera um novo sub-endereço, o adiciona à lista de endereços da carteira e direciona o usuário à tela de visão geral da carteira.

4.6 - Visualizar Sub-Endereços de Carteira



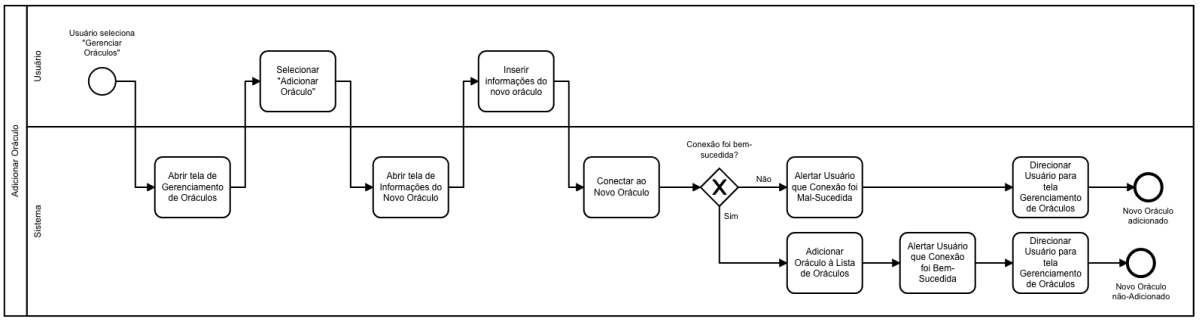
Quando o usuário deseja visualizar os sub-endereços de uma carteira, o mesmo seleciona a opção “Ver Sub-endereços”. O sistema, então, o direciona para a tela com a lista de endereços.

4.7 - Compartilhar Endereço/Sub-endereço



Ao selecionar a opção de compartilhar o endereço ou sub-endereço, o usuário deve escolher qual dos endereços quer compartilhar. Caso seja um endereço ou sub-endereço, o sistema abre a tela de compartilhamento do sistema operacional e aguarda o usuário terminar de executar suas ações. Caso o usuário escolha copiar o endereço, o sistema coloca o endereço no clipboard do sistema operacional.

4.8 - Adicionar Oráculo

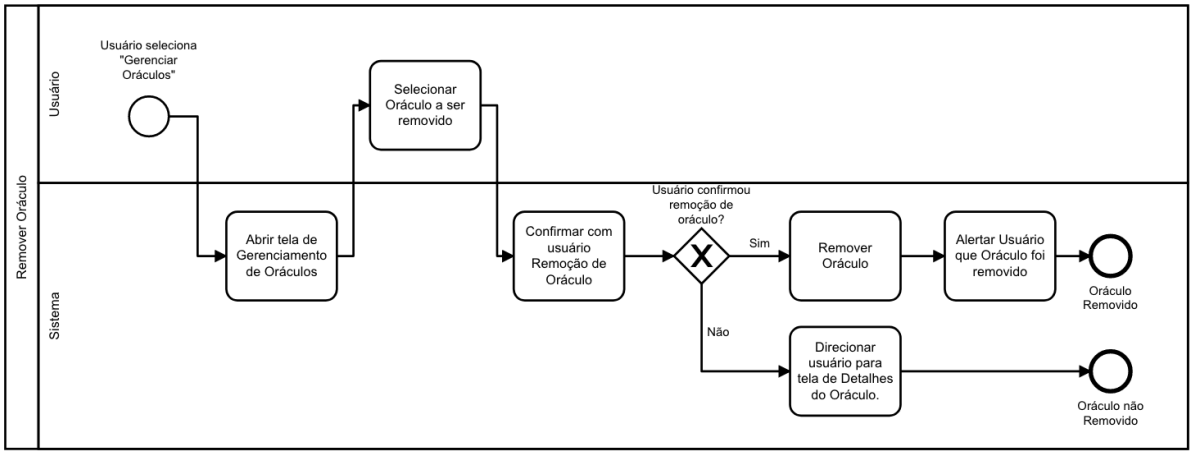


O usuário quando deseja adicionar um novo oráculo, deve selecionar “Adicionar Oráculo” na tela de gerenciamento de oráculos. O sistema deve solicitar ao usuário as informações do novo oráculo. Assim que o usuário insere as informações, o sistema deve tentar se conectar a este oráculo.

Caso seja bem-sucedido, o sistema deve adicionar este oráculo à lista de Oráculos, alertar o usuário que a adição foi bem-sucedida, e redirecionar o usuário à tela de gerenciamento de Oráculos.

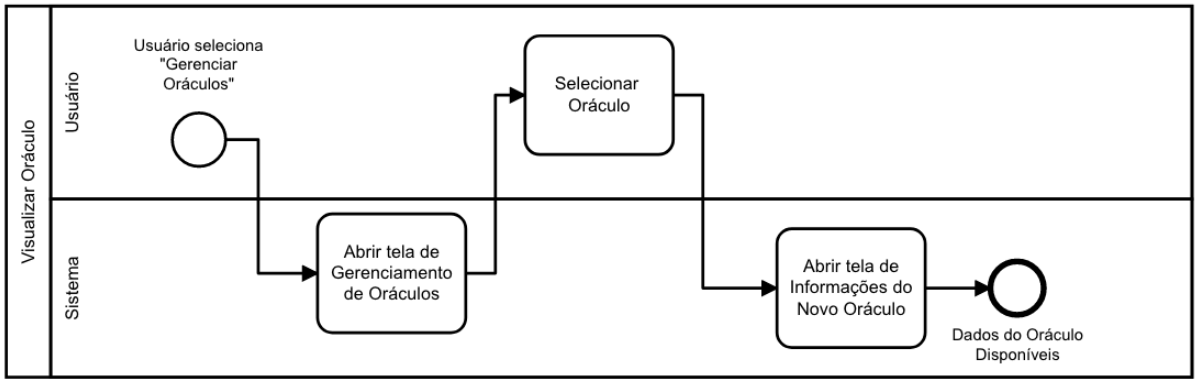
Caso seja mal-sucedido, o sistema deve alertar o usuário que a conexão foi mal-sucedida, e redirecionar o usuário à tela de gerenciamento de Oráculos.

4.9 - Remover Oráculo



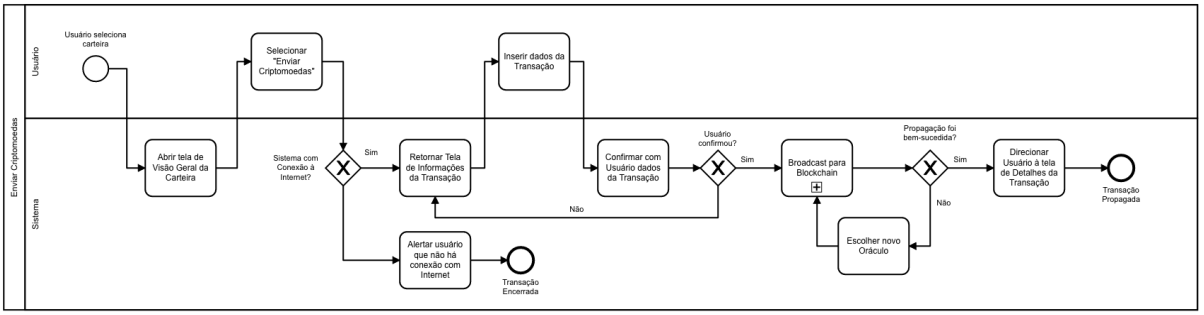
Na tela de gerenciamento de oráculos, o usuário que queira remover um oráculo deve selecionar o oráculo a ser removido, e selecionar a opção “Remover oráculo”. O sistema deve confirmar a remoção com o usuário. Caso seja confirmado, o sistema remove o oráculo e alerta o usuário que o oráculo foi removido, e encerra o processo. Caso não seja confirmado, o sistema direciona o usuário para a tela de gerenciamento de oráculos, e encerra o processo.

4.10 - Visualizar Oráculo



Para visualizar um oráculo, o usuário deve selecionar a opção “Gerenciar Oráculos”. O sistema deve abrir a tela de gerenciamento de oráculos. O usuário então seleciona o oráculo que deseja visualizar. O sistema retorna ao usuário uma tela com as informações do oráculo solicitado, e encerra o processo.

4.11 - Enviar Criptomoedas

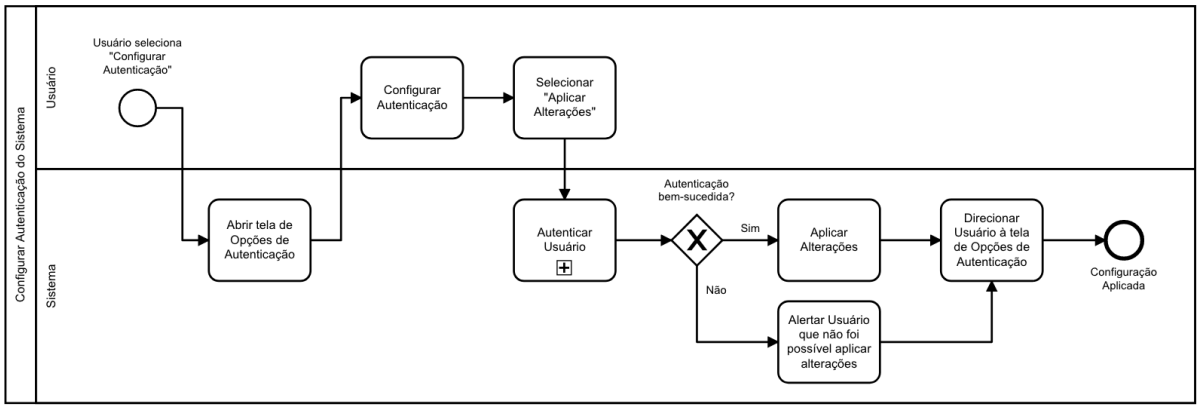


Para enviar criptomoedas, o usuário deve selecionar uma carteira a qual deseja fazer a transação. O sistema abre a tela de visão geral desta carteira, onde o usuário deve selecionar a opção “Enviar Criptomoedas”.

O sistema verifica se há conexão com a internet. Se não houver, retorna ao usuário que não tem conexão com a internet, e encerra a transação. Se houver, o sistema solicita ao usuário as informações da transação. O usuário insere os dados, e o sistema confirma a entrada destes dados. Caso o usuário não confirme, o sistema redireciona o usuário para a tela de informações da transação.

Caso o usuário confirme a transação, o sistema propaga a mensagem de transação para a blockchain, usando o oráculo configurado pelo usuário. Caso a propagação não seja bem-sucedida, o sistema escolhe um novo oráculo e tenta propagar novamente. Assim que propagar a transação para a blockchain, o sistema direciona o usuário para a tela de detalhes da transação que acabou de ser propagada, e encerra o processo.

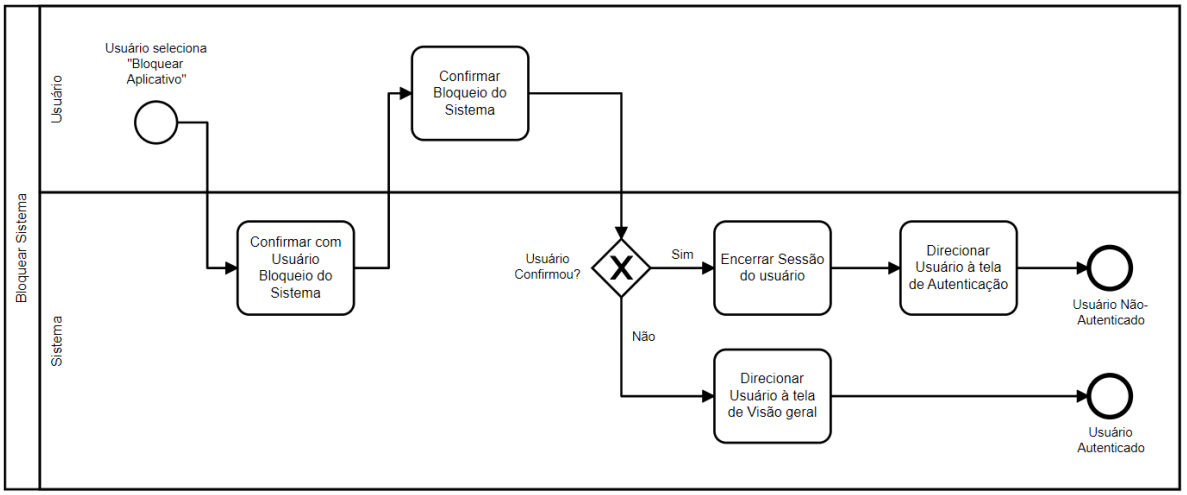
4.12 - Configurar Autenticação do Sistema



Na tela de configurações, o usuário deve escolher “Configurar Autenticação”. O sistema abre a tela de opções de autenticação. O usuário configura as opções de autenticação e seleciona “Aplicar Alterações”.

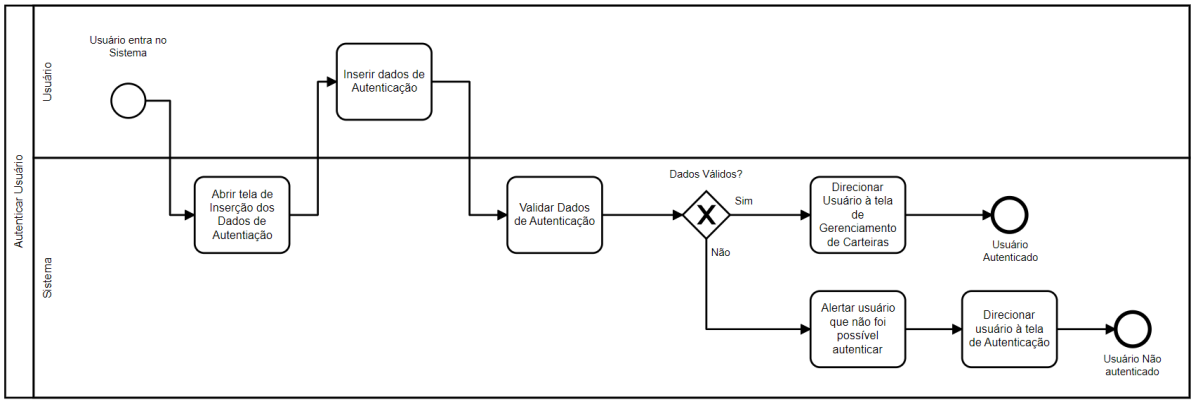
O sistema autentica o usuário. Caso seja mal-sucedido, alerta ao usuário que não foi possível autenticar, direciona o usuário para tela de opções de autenticação e encerra o processo. Caso seja bem-sucedido, o sistema aplica as alterações, direciona o usuário para tela de opções de autenticação e encerra o processo.

4.13 - Bloquear Sistema



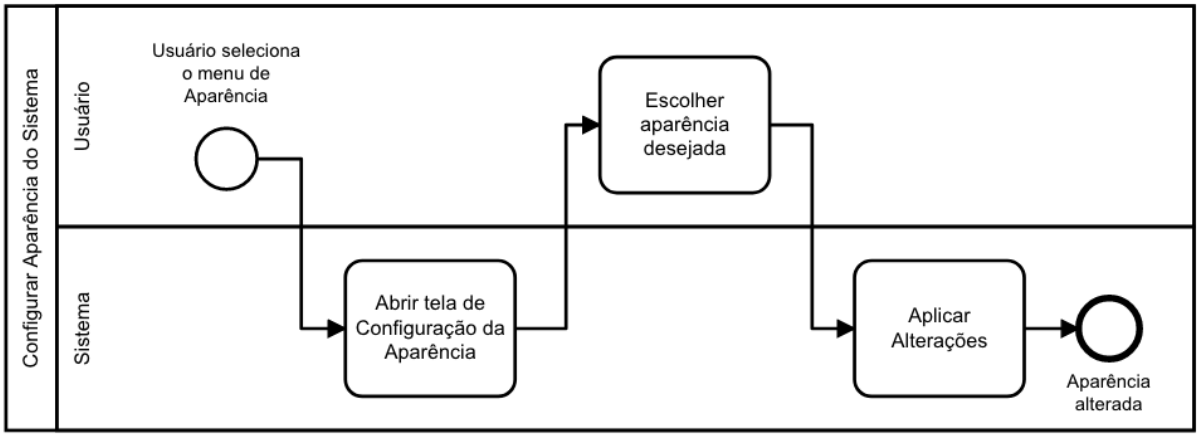
Para bloqueio do sistema, o usuário seleciona a opção “Bloquear Aplicativo”. O sistema pergunta ao usuário se deseja confirmar o bloqueio, que deve responder. Caso o usuário confirme o bloqueio, o sistema encerra a sessão do usuário e o direciona para a tela de autenticação, encerrando o processo. Caso o usuário não confirme, o sistema direciona o usuário para a tela de visão geral do sistema, encerrando o processo.

4.14 - Autenticar Usuário



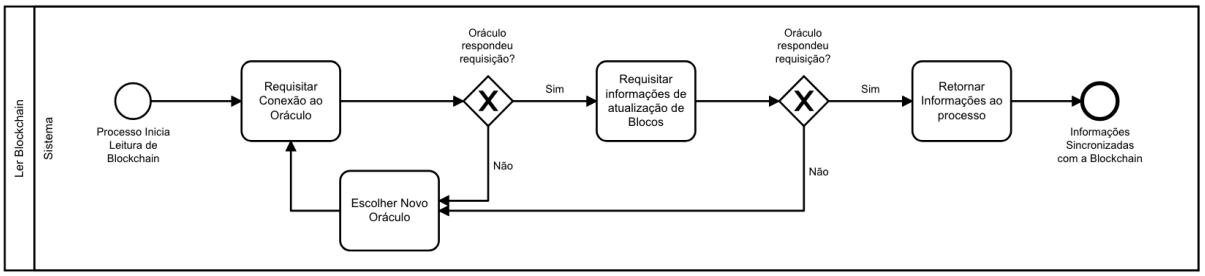
Para autenticar, o usuário não pode já estar autenticado, e deve entrar no sistema. O sistema abre a tela de inserção de dados de autenticação. O usuário insere os dados de autenticação, que pode ser senha, PIN ou biometria. O sistema valida os dados inseridos pelo usuário. Caso os dados sejam válidos, o sistema direciona o usuário para a tela de gerenciamento de carteiras, e encerra o processo. Caso os dados sejam inválidos, o sistema alerta o usuário que não foi possível autenticar, direciona o usuário à tela de inserção de dados de autenticação e finaliza o processo.

4.15 - Configurar Aparência do Sistema



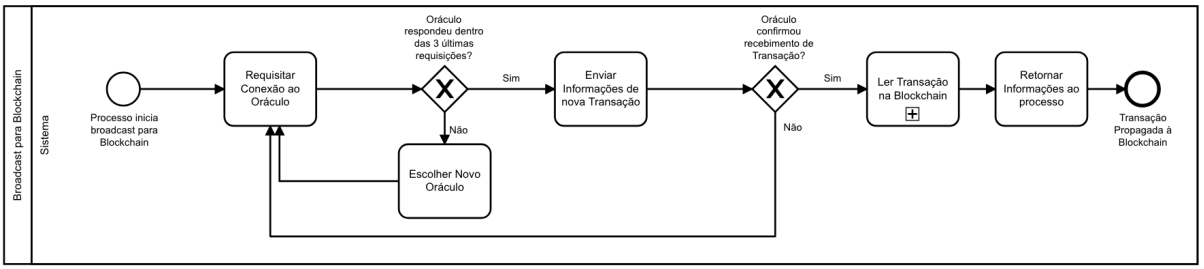
Para alterar a aparência do sistema, o usuário seleciona o menu de aparência. O sistema retorna ao usuário a tela com as opções de configuração de aparência. O usuário escolhe a aparência desejada e confirma as alterações. Ao confirmar, o sistema aplica as alterações de aparência e encerra o processo.

4.16 - Ler Blockchain



A leitura de blockchain é iniciada por outro processo. O sistema requisita uma conexão ao oráculo, e caso seja mal-sucedido, deve escolher outro oráculo para se conectar. Caso a conexão ao oráculo seja bem-sucedida, o sistema requisita as informações para atualizar seus blocos. Se o oráculo não responder, o sistema escolhe um novo oráculo e tenta novamente uma conexão. Caso o oráculo responda, o sistema retorna as informações solicitadas ao processo original, encerrando este processo.

4.17 - Broadcast para Blockchain



O broadcast para a blockchain é iniciado por outro processo. O sistema requisita uma conexão ao oráculo, e caso seja mal-sucedido 3 vezes, deve escolher outro oráculo para se conectar. Caso a conexão ao oráculo seja bem-sucedida, o sistema envia as informações para serem propagadas para a blockchain. Se o oráculo não confirmar o recebimento, o sistema escolhe um novo oráculo e tenta novamente uma conexão. Caso o oráculo responda, o sistema inicia o subprocesso “Ler Blockchain”, para atualizar as informações solicitadas e retorná-las ao processo original, encerrando este processo.

5. Requisitos

5.1 - Requisitos Funcionais

ID	Descrição
RF1	O Sistema deve permitir ao usuário gerenciar suas carteiras de Monero através de uma interface gráfica.
RF2	O Sistema deve permitir ao usuário importar/recuperar Contas de Carteiras anteriores.
RF3	O Sistema deve gerar novas Carteiras, Contas e Sub Endereços.
RF4	O Sistema deve permitir ao usuário compartilhar Endereços e Sub Endereços públicos.
RF5	O Sistema deve permitir ao usuário remover Carteiras do Gerenciador de Carteiras.
RF6	O Sistema deve permitir a visualização do saldo em Monero de uma carteira que tenha chave de Gasto e/ou Visualização.
RF7	O Sistema deve permitir visualizar as Chaves Privadas e Seed Mnemônica apenas a usuários autenticados.
RF8	O Sistema deve fazer backup das Chaves Privadas e Seed Mnemônica.
RF9	O Sistema deve permitir o envio de Monero, caso tenha a chave de Gasto.
RF10	O Sistema deve autenticar o acesso às informações no aplicativo localmente; usando um sistema de bloqueio/desbloqueio por PIN, senha ou Digital (caso o dispositivo permita).
RF11	O Sistema deve encriptar os dados da aplicação, utilizando uma senha/chave de encriptação.
RF12	O Sistema deve permitir ao usuário configurar e escolher seu Oráculo para a Blockchain do Monero.
RF13	O Sistema deve permitir ao usuário configurar aspectos do aplicativo como aparência da interface gráfica, método de bloqueio/desbloqueio do aplicativo e carteiras etc.
RF14	O Sistema deve permitir ao usuário bloquear o aplicativo.
RF15	O Sistema deve permitir ao usuário configurar rotinas automáticas de sincronismo das carteiras.

5.2 - Requisitos Não-Funcionais

ID	Descrição
RNF1	A interface gráfica do aplicativo deve ser simples, possuindo no máximo 3 subníveis de navegação a partir da tela principal. A Tela Inicial é o nível 0. Telas que são sequência de 1 operação, como transferência, contam como 1 nível.
RNF2	O Sistema deve garantir a integridade e segurança das informações sensíveis, que devem ser recuperadas e manipuladas apenas por usuários autenticados.

RNF3	O Sistema deve responder requisições em até 1 segundo, nas operações síncronas e locais, e em até 3 segundos em operações assíncronas com a Blockchain.
RNF4	O Sistema deve permitir que o usuário cancele operações assíncronas que levem mais de 2 segundos.
RNF5	Processos assíncronos longos devem ser feitos em background.

5.3 - Requisitos Inversos

ID	Descrição
RI1	O Sistema não deve manter estado, sessão ou guardar dados além das fornecidas pelo usuário. Se algum recurso do dispositivo é necessário para alguma funcionalidade, então o aplicativo deve solicitar ao usuário;
RI2	O Sistema não deve mostrar informações para usuários não-autorizados;
RI3	O Sistema não deve fazer transações pelo usuário, sem o expresso consentimento e intenção do mesmo;
RI4	O Sistema não deve minerar blocos ou rodar um Nodo local nos aplicativos mobile, e.g. Android [e iOS?];
RI5	O Sistema não deve modificar oráculos escolhidos/configurados pelo usuário sem a permissão do mesmo, redirecionando apenas quando nenhum destes oráculos estiver disponível e estiver configurado para redirecionar automaticamente.

6. Casos de Uso

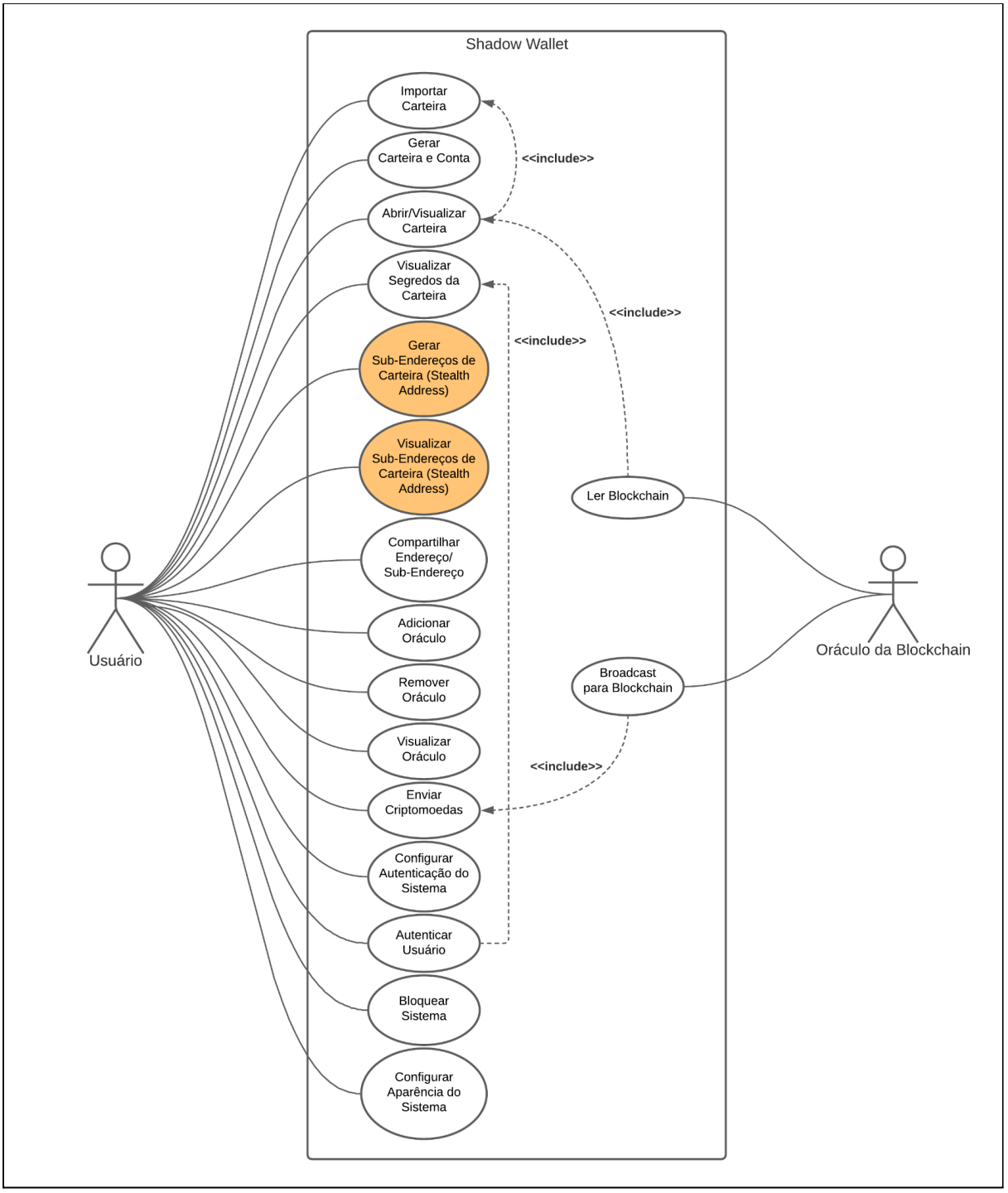


Diagrama de casos de uso

6.1 - Importar Carteira

CDU01	
Nome	Importar Carteira
Objetivo	Adicionar chaves privadas para manipulação de Criptomoedas
Atores	Usuário
Pré-condições	Usuário autenticado na tela de gerência de carteiras do Sistema; Sistema com conexão à Internet;
Trigger	Usuário seleciona “Importar Carteira”
Fluxo principal	1. Sistema abre Tela de Importação de Carteira; 2. Usuário insere dados para importação; 3. Sistema recebe chaves privadas válidas; 4. Sistema alerta ao Usuário se carteira é de Gasto ou Visualização; 5. Usuário dispensa alerta 6. Sistema direciona usuário para tela de Visão Geral da Carteira; 7. Sistema lê a carteira via Oráculo, em background.
Fluxo alternativo	Passo 4. A4. Sistema recebe chaves privadas inválidas; A5. Sistema alerta ao Usuário que chave é inválida; A6. Usuário dispensa alerta; A7. Sistema direciona usuário para Tela de Importação de Carteira.
Extensões	N/A
Pós-condições	Usuário possui uma nova Carteira no Sistema.
Regras de negócio	RN1

6.2 - Gerar Nova Carteira

CDU02	
Nome	Gerar Carteira e Conta
Objetivo	Criar nova Carteira de Monero
Atores	Usuário
Pré-condições	Usuário autenticado na tela de gerência de carteiras do Sistema.
Trigger	Usuário seleciona “Gerar Nova Carteira”
Fluxo principal	1. Sistema abre tela de Gerador de Carteira; 2. Usuário insere informações sobre a nova carteira; 3. Sistema gera nova carteira, e a adiciona à lista de carteira do usuário; 4. Sistema direciona usuário à tela de Segredos da Carteira;
Fluxo alternativo	N/A

Extensões	N/A
Pós-condições	Usuário possui uma nova Carteira no Sistema.
Regras de negócio	RN1

6.3 - Abrir/Visualizar Carteira

CDU03	
Nome	Abrir/Visualizar Carteira
Objetivo	Ver informações de chave pública e Endereços de uma Carteira de Monero
Atores	Usuário
Pré-condições	Usuário autenticado na tela de gerência de carteiras do Sistema; Sistema com pelo menos 1 carteira na Lista de carteiras;
Trigger	Usuário seleciona uma Carteira da lista de carteiras.
Fluxo principal	1. Sistema abre tela de Visão Geral da Carteira; 2. Sistema inicia Leitura da Blockchain, para sincronizar carteira e coloca tarefa em background; 3. Ao final da sincronização, Sistema atualiza tela de Visão Geral da Carteira com saldo da carteira e altura do bloco atual;
Fluxo alternativo	Passo 2, caso não haja internet: A2. Sistema abre tela de Visão Geral da Carteira; A3. Sistema alerta usuário que não há conexão com internet; A4. Usuário dispensa alerta; A5. Sistema direciona usuário à tela de Visão Geral da Carteira; Durante passo 2: A3. Usuário interrompe sincronismo com carteira; A4. Sistema encerra sincronização da carteira.
Extensões	N/A
Pós-condições	Usuário visualiza saldo, transações, altura do bloco da carteira, chaves privadas e endereços de uma carteira.
Regras de negócio	RN1

6.4 - Visualizar Segredos da Carteira

CDU04	
Nome	Visualizar Segredos da Carteira
Objetivo	Ver informações sobre Chaves Privadas de uma Carteira de Monero
Atores	Usuário

Pré-condições	Usuário autenticado na tela de gerência de carteiras do Sistema; Sistema com pelo menos 1 carteira na Lista de carteiras;
Trigger	Usuário seleciona uma carteira da lista de carteiras
Fluxo principal	1. Sistema abre tela de Visão Geral da Carteira; 2. Usuário seleciona “Ver Chaves Privadas da Carteira”; 3. Sistema solicita nova Autenticação; 4. Usuário insere dados de Autenticação; 5. Sistema consegue validar dados de Autenticação; 6. Sistema abre tela de Segredos da Carteira;
Fluxo alternativo	Passo 5: A5. Sistema não consegue validar dados de Autenticação; A6. Sistema alerta usuário que não foi possível autenticar; A7. Usuário dispensa alerta; A8. Sistema direciona usuário à tela de Visão Geral da Carteira.
Extensões	N/A
Pós-condições	Usuário visualiza Chaves Privadas e Seed Mnemônica de uma Carteira no Sistema.
Regras de negócio	RN1

6.5 - Gerar Sub-Endereços de Carteira (Stealth Address)

CDU05	
Nome	Gerar Sub-Endereços de Carteira (Stealth Address)
Objetivo	Criar novo Sub-Endereço (Stealth Address) de Monero
Atores	Usuário
Pré-condições	Usuário autenticado na tela de Visão Geral da Carteira.
Trigger	Usuário seleciona “Gerar Novo Sub-Endereço”
Fluxo principal	1. Sistema gera novo sub-endereço; 2. Sistema adiciona novo sub-endereço à lista de sub-endereços daquela carteira; 3. Sistema direciona usuário à tela de Lista de Sub-endereços da Carteira;
Fluxo alternativo	N/A
Extensões	N/A
Pós-condições	Usuário possui um novo Sub-Endereço de uma Carteira no Sistema.
Regras de negócio	RN1

6.6 - Visualizar Sub-Endereços de Carteira (Stealth Address)

CDU06	
Nome	Visualizar Sub-Endereços de Carteira (Stealth Address)
Objetivo	Ver Sub-Endereços (Stealth Address) de uma Carteira Monero
Atores	Usuário
Pré-condições	Usuário autenticado na tela de Visão Geral da Carteira.
Trigger	Usuário seleciona “Ver Sub-Endereços”
Fluxo principal	1. Sistema abre tela de Lista de Sub-endereços da Carteira;
Fluxo alternativo	N/A
Extensões	N/A
Pós-condições	Usuário visualiza os Sub-Endereços de uma Carteira no Sistema.
Regras de negócio	RN1

6.7 - Compartilhar Endereço/Sub-endereço

CDU07	
Nome	Compartilhar Endereço
Objetivo	Compartilhar chave pública de Carteira, para receber Monero de terceiros.
Atores	Usuário
Pré-condições	Usuário autenticado na tela de gerência de carteiras do Sistema; Sistema com pelo menos 1 carteira na Lista de carteiras;
Trigger	Usuário seleciona uma carteira da lista de carteiras.
Fluxo principal	1. Sistema abre tela de visão geral da carteira, com o QR code visível; 2. Usuário seleciona “Compartilhar Endereço” 3. Sistema abre a tela de compartilhamento do Sistema Operacional; 4. Usuário seleciona destinatário; 5. Sistema envia a informação pela interface.
Fluxo alternativo	Passo 2: A2. Usuário seleciona “Ver Sub-Endereços” A3. Sistema abre tela da Lista de Sub-Endereços; A4. Usuário seleciona “Compartilhar Sub-Endereço”; A5. Sistema abre interface de compartilhamento do Sistema Operacional; A6. Usuário seleciona destinatário; A7. Sistema envia a informação pela interface. Passo 2: B2. Usuário seleciona “Copiar Endereço” B3. Sistema copia Endereço para o Clipboard do Sistema Operacional.

Extensões	N/A
Pós-condições	Usuário envia Endereço da carteira desejada ao destino desejado.
Regras de negócio	RN1

6.8 - Adicionar Oráculo

CDU08	
Nome	Adicionar Oráculo
Objetivo	Adicionar novo Nodo provedor de informação da Blockchain
Atores	Usuário
Pré-condições	Usuário autenticado na tela de gerência de carteiras do Sistema; Sistema com conexão à internet.
Trigger	Usuário seleciona “Gerenciar Oráculos”
Fluxo principal	1. Sistema abre tela de Gerenciamento de Oráculos; 2. Usuário seleciona “Adicionar Oráculo”; 3. Sistema abre tela de Informações do Novo Oráculo; 4. Usuário insere informações do novo Oráculo; 5. Sistema faz requisição de conexão ao Novo Oráculo; 6. Sistema alerta usuário que conexão foi bem-sucedida; 7. Usuário dispensa alerta; 8. Sistema adiciona novo Oráculo à lista de Oráculos; 9. Sistema direciona usuário para tela de Gerenciamento de Oráculos.
Fluxo alternativo	Passo 6: A6. Sistema alerta usuário que conexão foi mal-sucedida; A7. Usuário dispensa alerta; A8. Sistema direciona usuário para tela de Gerenciamento de Oráculos.
Extensões	N/A
Pós-condições	Usuário possui novo Oráculo em sua lista de Oráculos.
Regras de negócio	RN1

6.9 - Remover Oráculo

CDU09	
Nome	Remover Oráculo
Objetivo	Remover Nodo provedor de informação da Blockchain
Atores	Usuário
Pré-condições	Usuário autenticado na tela de Gerenciamento de carteiras do Sistema. ; Sistema com pelo menos 1 Oráculo adicionado.
Trigger	Usuário seleciona “Gerenciar Oráculos”
Fluxo principal	1. Sistema abre tela de Gerenciamento de Oráculos; 2. Usuário seleciona Oráculo desejado; 3. Sistema abre tela de Detalhes do Oráculo; 4. Usuário seleciona “Remover Oráculo”; 5. Sistema confirma com Usuário remoção do Oráculo; 6. Usuário confirma remoção do Oráculo; 7. Sistema remove Oráculo da lista de Oráculos; 8. Sistema alerta usuário que remoção foi feita; 9. Usuário dispensa alerta; 10. Sistema direciona usuário para tela de Gerenciamento de Oráculos.
Fluxo alternativo	Passo 6: A6. Usuário cancela remoção do Oráculo; A7. Sistema direciona usuário para tela de Detalhes do Oráculo.
Extensões	N/A
Pós-condições	Usuário remove Oráculo de sua lista de Oráculos.
Regras de negócio	RN1

6.10 - Visualizar Oráculo

CDU10	
Nome	Visualizar Oráculo
Objetivo	Visualizar Nodo provedor de informação da Blockchain
Atores	Usuário
Pré-condições	Usuário autenticado na tela de Gerenciamento de carteiras do Sistema.
Trigger	Usuário seleciona “Gerenciar Oráculos”
Fluxo principal	1. Sistema abre tela de Gerenciamento de Oráculos; 2. Usuário seleciona Oráculo desejado; 3. Sistema abre tela de Detalhes do Oráculo;
Fluxo alternativo	N/A

Extensões	N/A
Pós-condições	Usuário visualiza informações sobre Oráculo de sua lista de Oráculos.
Regras de negócio	RN1

6.11 - Enviar Criptomoedas

CDU11	
Nome	Enviar Criptomoedas
Objetivo	Enviar criptomoedas ao endereço de destino.
Atores	Usuário
Pré-condições	Usuário autenticado na tela de gerência de carteiras do Sistema; Sistema com pelo menos 1 carteira na lista de carteiras; Carteira com saldo maior que 0 XMR/BTC.
Trigger	Usuário seleciona carteira desejada.
Fluxo principal	1. Sistema abre tela de Visão Geral da Carteira; 2. Usuário seleciona “Enviar Criptomoedas”; 3. Sistema abre tela de Informações da Transação; 4. Usuário insere Endereço de destino e valor; 5. Sistema confirma com Usuário a Transação, informando todos os dados sobre a mesma; 6. Usuário confirma Transação; 7. Sistema propaga transação via Oráculo atual; 8. Sistema alerta usuário que transação foi enviada com sucesso; 9. Usuário dispensa alerta; 10. Sistema direciona usuário para tela de Detalhes da Transação;
Fluxo alternativo	Passo 3: A3. Sistema alerta que não há conexão com Internet; A4. Usuário dispensa alerta; A5. Sistema direciona usuário para tela de Visão Geral da Carteira. Passo 8: A8. Se oráculo atual não conectar, tentar conectar a outro Oráculo; A9. Retornar ao passo 7.
Extensões	N/A
Pós-condições	Usuário possui menos criptomoedas que antes, de acordo com o valor desejado na transação; Transação é confirmada publicamente na Blockchain; Endereço destinatário possui mais criptomoedas que antes.
Regras de negócio	RN1

6.12 - Configurar Autenticação do Sistema

CDU12	
Nome	Configurar Autenticação do Sistema
Objetivo	Configurar comportamento de autenticação do usuário na carteira
Atores	Usuário
Pré-condições	Usuário autenticado na tela de Gerenciamento de Carteiras.
Trigger	Usuário seleciona “Menu > Preferências > Configurar Autenticação”
Fluxo principal	1. Sistema abre a tela de Opções de Autenticação; 2. Usuário configura comportamento de autenticação do modo desejado [expandir]; 3. Usuário seleciona “Aplicar Alterações”; 4. Sistema solicita nova Autenticação; 5. Usuário insere dados de Autenticação; 6. Sistema consegue validar dados de Autenticação; 7. Sistema aplica as alterações.
Fluxo alternativo	Passo 7: A7. Sistema não consegue validar dados de Autenticação; A8. Sistema alerta usuário que não foi possível autenticar; A9. Usuário dispensa alerta; A10. Sistema direciona usuário à tela de Opções de Autenticação.
Extensões	N/A
Pós-condições	Usuário possui uma nova Carteira no Sistema.
Regras de negócio	RN1

6.13 - Bloquear Sistema

CDU13	
Nome	Bloquear Sistema
Objetivo	Finalizar a sessão atual do usuário no Sistema, não permitindo manipular Carteiras
Atores	Usuário
Pré-condições	Usuário autenticado na tela de Visão Geral da Carteira.
Trigger	Usuário seleciona “Menu > Bloquear Aplicativo [ou Sistema]”
Fluxo principal	1. Sistema confirma com Usuário o Bloqueio do Sistema; 2. Usuário confirma Bloqueio; 3. Sistema direciona usuário à tela de Autenticação.
Fluxo alternativo	passo 6: A6. Usuário cancela Bloqueio;

	A7. Sistema direciona usuário à tela de Gerenciamento de carteiras do Sistema.
Extensões	N/A
Pós-condições	Usuário sem autenticação no Sistema.
Regras de negócio	RN1

6.14 - Autenticar Usuário

CDU14	
Nome	Autenticar Usuário
Objetivo	Iniciar a sessão atual do usuário no Sistema, permitindo manipular Carteiras.
Atores	Usuário
Pré-condições	Usuário não-autenticado na tela de Autenticação.
Trigger	Usuário seleciona “Autenticar”
Fluxo principal	1. Sistema abre tela de Inserção dos Dados de Autenticação (PIN, Digital ou Senha); 2. Usuário insere dados; 3. Sistema consegue validar os dados; 4. Sistema direciona usuário à tela de gerência de carteiras.
Fluxo alternativo	Fluxo alternativo no passo 3: A3. Sistema não consegue validar dados; A4. Sistema alerta usuário que não foi possível autenticar; A5. Usuário dispensa alerta; A6. Sistema direciona usuário à tela de Autenticação.
Extensões	N/A
Pós-condições	Usuário possui uma nova Carteira no Sistema.
Regras de negócio	RN1

6.15 - Configurar Aparência do Sistema

CDU15	
Nome	Configurar Aparência do Sistema
Objetivo	Trocar aparência visual da aplicação
Atores	Usuário
Pré-condições	Usuário autenticado na tela inicial do sistema.
Trigger	Usuário seleciona “Menu > Preferências > Aparência”
Fluxo principal	1. Sistema abre tela de configuração da aparência; 2. Usuário escolhe aparência desejada; 3. Usuário seleciona “Aplicar”; 4. Sistema aplica alterações;
Fluxo alternativo	N/A
Extensões	N/A
Pós-condições	Aparência visual alterada no sistema.
Regras de negócio	RN1

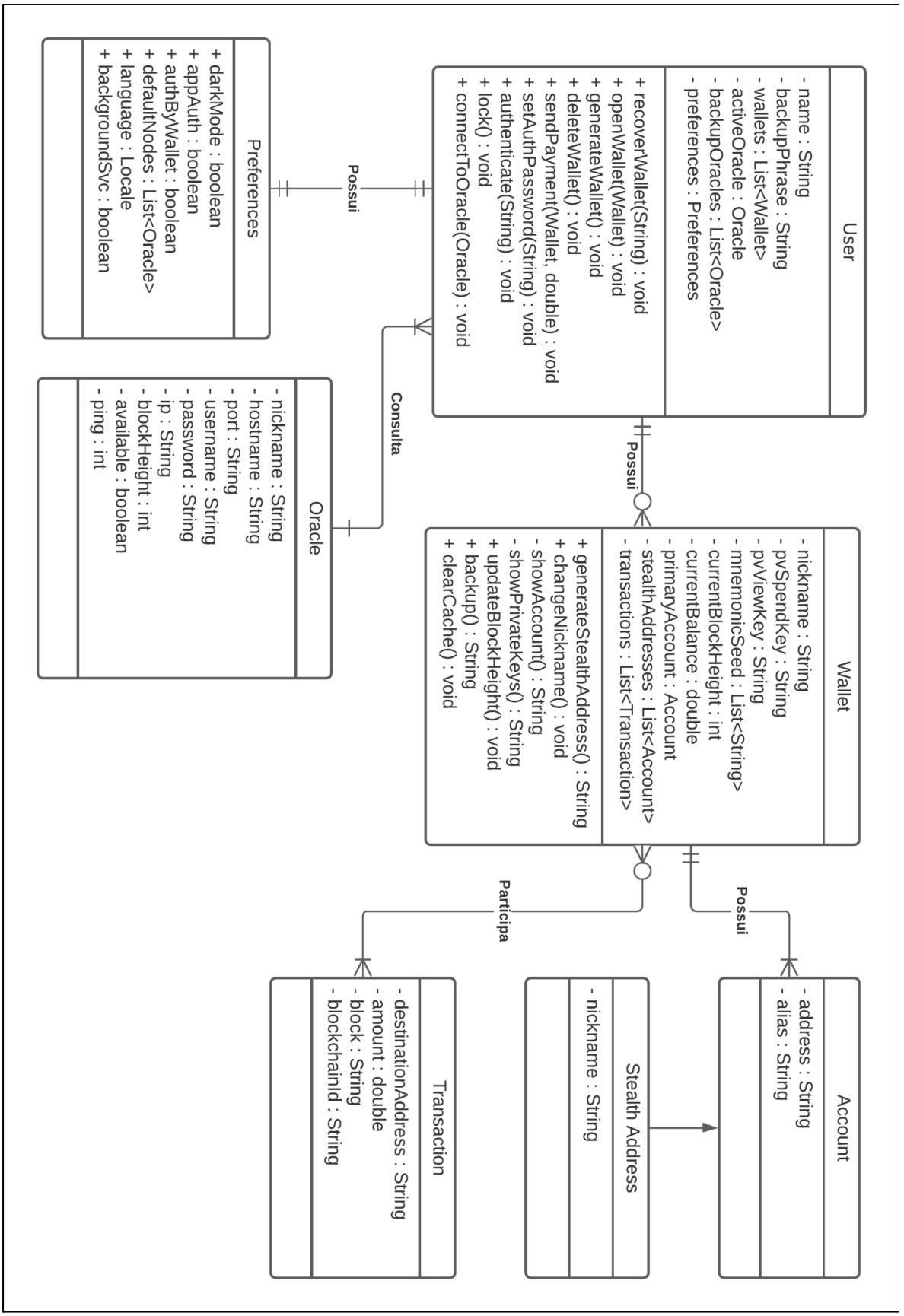
6.16 - Ler Blockchain

CDU16	
Nome	Ler Blockchain
Objetivo	Receber informações sobre a Blockchain via Oráculo
Atores	Sistema, Oráculo
Pré-condições	Sistema com conexão à Internet;
Trigger	Usuário realiza “Abre/Visualiza Carteira”
Fluxo principal	1. Sistema requisita conexão ao Oráculo; 2. Oráculo responde requisição; 3. Sistema requisita informações de atualização de Blocos ao Oráculo; 4. Sistema atualiza informações de carteira de acordo com informações recebidas.
Fluxo alternativo	Passo 2: A2. Oráculo não responde requisição; A3. Trocar Oráculo e voltar ao passo 1.
Extensões	N/A
Pós-condições	Carteira possui informações atualizadas sobre transações e saldo atual.
Regras de negócio	RN1

6.17 - Broadcast para Blockchain

CDU17	
Nome	Broadcast para Blockchain
Objetivo	Receber informações sobre a Blockchain via Oráculo
Atores	Sistema, Oráculo
Pré-condições	Sistema com conexão à Internet;
Trigger	Usuário realiza “Enviar Criptomoedas”
Fluxo principal	1. Sistema solicita conexão ao Oráculo; 2. Oráculo responde requisição; 3. Sistema propaga transação ao Oráculo; 4. Sistema solicita informações de carteira, para atualizar status da transação; 5. Sistema direciona Usuário para tela de Detalhes da Transação;
Fluxo alternativo	Passo 2: A2. Oráculo não responde requisição; A3. Trocar Oráculo e voltar ao passo 1.
Extensões	N/A
Pós-condições	Carteira possui informações atualizadas sobre transações e saldo atual.
Regras de negócio	RN1

7. Diagrama de Classes



8. Glossário

Gerenciador de Carteiras

Um Gerenciador de Carteiras nada mais é do que um software que permite a manipulação de Carteiras de Monero. O software definido neste documento de requisitos é do tipo de Gerenciador de Carteiras.

Carteira de Monero (Monero Wallet)

Uma Carteira de Monero é um software que armazena localmente as informações de uma Conta, incluindo suas Chaves Privadas, transações e balanço final. Além disso, a carteira pode incluir um software de mineração de Monero, e uma lista de seus endereços e sub-endereços públicos.

Uma Carteira de Monero (Wallet) é representada por um par de Chaves Privadas: a Chave de Gasto e Chave de Visualização (Spend Key e View Key). Estas chaves podem ser representadas por uma combinação de 25 palavras mnemônicas, conhecida como Semente Mnemônica (Seed).

Uma carteira só pode ser adicionada/recuperada a um gerenciador de carteiras através das suas chaves privadas ou por sua Semente Mnemônica.

Conta de Monero (Monero Account)

Uma Conta de Monero (Account) é identificada por um conjunto de 95 caracteres, começando com um “4”, chamado de Endereço (Address).

Chave Privada de Gasto (Private Spend Key)

O usuário só pode enviar Monero para outro usuário se possuir a Chave de Gasto daquela Carteira. Um usuário que possui apenas a Chave de Visualização de uma carteira não pode fazer transferências.

Chave Privada de Visualização (Private View Key)

Uma chave privada de visualização é necessária para que o usuário possa visualizar seu saldo. Entretanto, não consegue fazer transações com a mesma.

Chave Públicas ou Endereços (Address or Public Key)

Um endereço Monero é gerado a partir de uma Carteira, usando o par de Chaves Privadas, possuindo a relação de 1 para n. Uma Carteira pode ter n endereços, mas 1 endereço é sempre da mesma Carteira

Sub Endereços (Sub Addresses, Steath Address)

Um sub-endereço Monero (Stealth Address) é um identificador associado a uma Carteira, e funciona como máscara da Conta principal, possuindo relação de 1 para n. Ou seja, uma Carteira pode ter n sub-endereços, mas os sub-endereços são sempre da mesma Carteira.

Node e Oráculo

Um Nodo da Blockchain é uma entidade que possui uma cópia de toda a Blockchain desta Rede, e que consegue ler e propagar informações para outros Nodos incluírem nesta Blockchain. Para leitura e escrita na Blockchain do Monero, é necessário ter um Nodo ativo.

Oráculos são serviços remotos com um Nodo ativo de Monero, que lêem e propagam informações para a internet através de uma API. Por limitações de hardware, o Gerenciador de Carteiras pode usar o serviço desses Oráculos, que pode ser configurado pelo usuário final da rede, que deseja fazer uma transação.

Taxa de Mineração

Toda transação de Monero possui uma Taxa opcional de transação, paga aos Mineradores da rede, como incentivo para adicionarem e manterem novas informações à Blockchain. Esta taxa é configurável no momento da transação.