Estudo da implementação de plataformas de criptomoedas em ambientes compartilhados e distribuídos: Criando a GvCoin

Relatório Final

PIBIC - Programa Institucional de Bolsas de Iniciação Científica

Aluno: Yan Tadeu Kaled e Silva Barbosa
Orientador: Júlio Figueiredo

São Paulo – Julho de 2018

1. Resumo

O projeto tem como primeiro objetivo responder, a partir de uma investigação científica nos campos de Blockchain, Criptomoedas e Arquitetura Peer-to-Peer, as seguintes perguntas:

1. Como são implementadas as seguintes tecnologias de suporte de criptomoedas, smart contracts e blockchain: BitCoin, Ethereum e Hyperledger?
2. Como é possível aprimorar computacionalmente e economicamente os sistemas de criptomoedas e smart contracts com base em estudos de suas implementações?

O segundo objetivo do projeto consiste no desenvolvimento de uma plataforma de criptomoedas para utilização na FGV, a "GvCoin".
A primeira etapa do projeto "GvCoin" consistiu na pesquisa de tecnologias de ledgers distribuídos e plataformas de criptomoedas, no desenho da arquitetura da "GvCoin" e na implementação de uma versão simplificada da moeda. A partir da pesquisa, foram levantadas questões acerca da implementação da GvCoin, dadas as particularidades do ambiente de uso proposto para a criptomoeda, como a adequação das tecnologias Ethereum e Hyperledger em termos de escalabilidade, facilidade de manutenção e uso do sistema.
A segunda etapa do projeto consistiu em um aprofundamento dos estudos em Ethereum e Hyperledger, e no desenvolvimento de duas versões da GvCoin. As construções das versões da GvCoin envolveram design de software e produto, e implementação com linguagens e tecnologias disponíveis em ambientes abertos - "open-source".
Com este trabalho, espera-se contribuir com o desenvolvimento de mecanismos de engajamento e correção de externalidades em organizações e comunidades. Em particular, espera-se que o projeto desenvolvido seja implementado na FGV, e replicado em outras universidades.ve

2. Palavras-chaves

Blockchain, criptomoedas, plataformas descentralizadas, arquitetura e engenharia de software.

3. Introdução

Com o crescimento da popularidade de plataformas de criptomoedas e aplicações descentralizadas, agentes de diversos setores econômicos buscam soluções para problemas de diferentes naturezas nas tecnologias de registros distribuídos. De aplicações empresariais a comunitárias, as tecnologias de registros distribuídos alavancaram uma nova forma de

interação entre agentes econômicos, ao reduzirem a necessidade de intermediários em operações de troca de valor.
Neste cenário, três tecnologias obtiveram destaque: Bitcoin, Ethereum e Hyperledger. A Bitcoin foi a primeira criptomoeda a causar grande impacto social e conquistar ampla popularidade, além de tornar-se o modelo principal para diversas criptomoedas e aplicações em registros distribuídos. A popularização da "blockchain" - registro distribuído - trouxe plataformas com propostas alternativas à proposta monetária da Bitcoin, generalizando aplicações de tecnologias descentralizadas. A plataforma Ethereum foi idealizada por Vitalik Buterin para dar suporte ao desenvolvimento de aplicações descentralizadas - tanto por empreendimentos e fundações, quanto por indivíduos em comunidades "open source". Já o projeto Hyperledger surgiu na Linux Foundation, com o intuito de popularizar as tecnologias de registro distribuídos em corporações.
Uma categoria de aplicações dentre as mais populares baseadas na Bitcoin é formada pelos "criptotokens". Tais "tokens" são implementados utilizando protocolos e tecnologias disponíveis tanto na Bitcoin quanto em Ethereum. Um aplicação particular dos "criptotokens" é o engajamento de agentes em uma comunidade ou organização, como Buterin explicou em seu "Ethereum White Paper".
O projeto proposto tem o intuito de contribuir com o fortalecimento de comunidades universitárias, incentivando ações produtivas e inovação. Para isso, uma plataforma de "criptotokens" foi construída, a GvCoin.
Neste estudo, análises densas das tecnologias de Ethereum, Hyperledger e Bitcoin foram conduzidas. O entendimento do funcionamento dessas tecnologias e de seus mecanismos permite ao leitor explorar outras ferramentas e tecnologias, como a Litecoin e a Everledger.
O projeto foi desenvolvido conforme o cronograma abaixo:

Atividade	Mês											
	1	2	3	4	5	6	7	8	9	10	11	12
Pesquisa de Tecnologias	■	■	■									
Estudo Comparativo	■	■	■	■	■	■						
Análise de Resultados						■	■	■	■	■	■	■
Implementação da Plataforma de Criptomoeda				■	■	■	■	■	■	■	■	■
Documentação	■	■	■	■	■	■	■	■	■	■	■	■
Preparação do Relatório Final											■	■

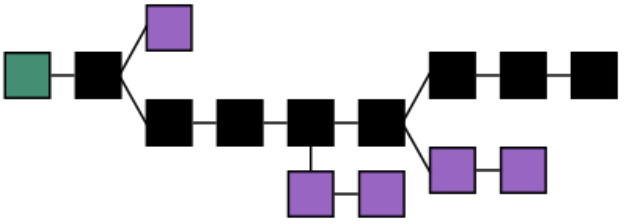
O texto está distribuído em duas partes: Tecnologias de Registro Distribuído, na qual as análises sobre as tecnologias de "blockchain" são apresentadas, e GvCoin, na qual a plataforma de "criptotoken" construída é apresentada.

4. Tecnologias de Registro Distribuído

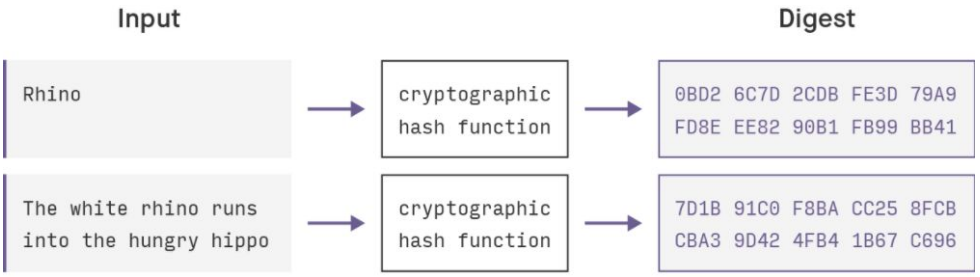
As Tecnologias de Registro Distribuído (Distributed Ledger Technologies) compõem a base das plataformas de criptomoedas. Um Registro Distribuído é uma estrutura de dados que permite que informações transacionais sejam replicadas, compartilhadas e sincronizadas entre participantes de uma rede por meio de algoritmos de consenso.
Um Registro Distribuído é composto por três componentes principais: um Modelo de Dados - que descreve o estado do registro em determinado momento – Transações – que modificam o registro - e um Protocolo – que define as regras para que o consenso seja atingido entre os membros da rede sobre quais transações serão aceitas no registro.
Abaixo, segue o estudo das tecnologias de blockchain e criptomoedas, correspondentes à primeira pergunta de pesquisa.

4.1. Bitcoin e Blockchain

A Blockchain é um caso particular da estrutura de dados de Registro Distribuído. Essa estrutura de dados se popularizou após o lançamento da Bitcoin, pois a criptomoeda é construída com base numa estrutura distribuída de Blockchain.
A Blockchain é construída com uma sequência cronológica de blocos, que são conjuntos de transações adicionadas à sequência, como na figura abaixo:

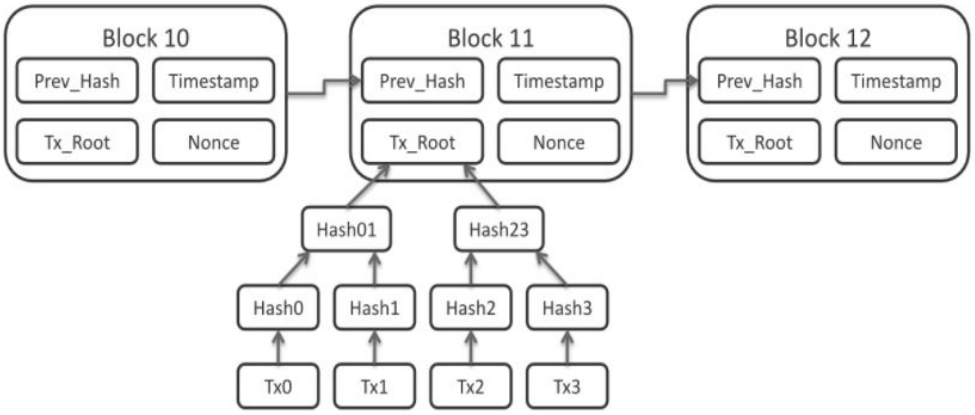


O bloco verde é chamado de Bloco Gênese, pois é o primeiro bloco gerado. Os blocos subsequentes devem ser validados por algum mecanismo de consenso para entrarem na sequência cronológica oficial – em preto, na figura - que será distribuída e sincronizada entre todos os participantes da rede. Um bloco que não foi aceito na sequência – em roxo, na figura – é chamado de Órfão. Um bloco terá, então, quatro tipos de metadados: uma referência ao bloco anterior, uma Prova de Trabalho (Proof of Work) ou outro protocolo de segurança, uma "timestamp" - que identifica quando certa transação ocorreu – e a raiz da Árvore de Merkle das transações incluídas no bloco.
As referências ao bloco anterior e à raiz da Árvore de Merkle utilizam resultados de funções de hash criptográficas. Esse tipo de função de hash – que mapeia dados de tamanho arbitrário para uma sequência de caracteres de tamanho fixo – não pode ser invertida sem métodos de tentativa e erro.



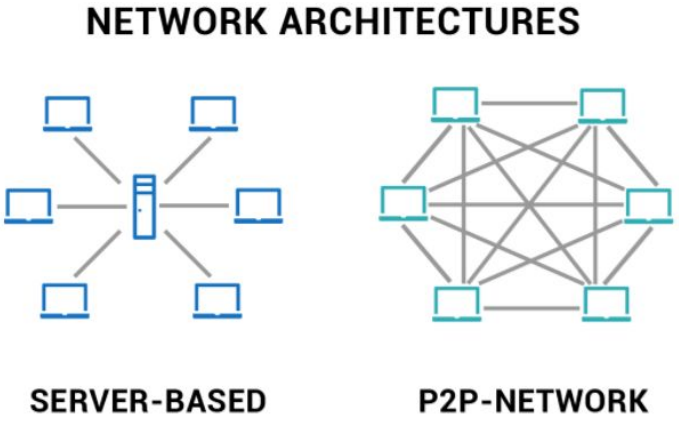
Exemplo do funcionamento de uma função de hash criptográfica.

A Árvore de Merkle é uma estrutura de dados que guarda hashes de dados individuais – representados na figura abaixo por $Txi,\ i \in \{0, 1, 2, 3\}$. Essa estrutura de dados permite que buscas sejam realizadas de modo eficiente, mesmo com grandes volumes de dados.



Exemplo de uma Árvore de Merkle em uma sequência de Blocos.

A estrutura de dados Blockchain é distribuída em redes de arquitetura Peer-to-Peer, ou seja, não tem um servidor central. Os nós que contribuem com poder computacional para processar e guardar dados pertinentes à rede são chamados de “Peers”.

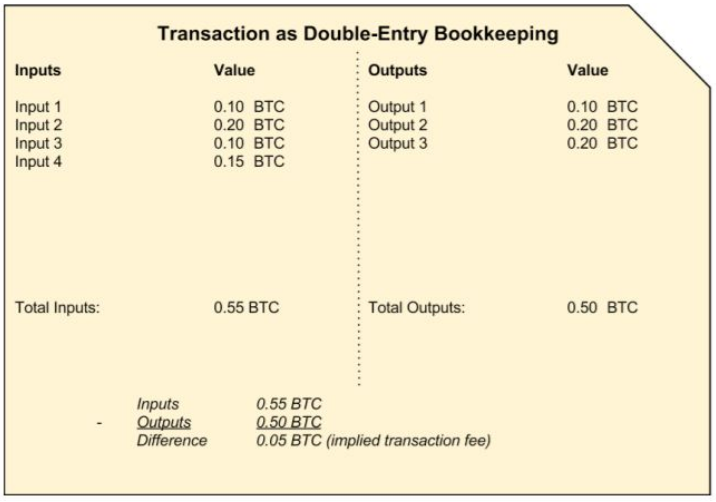


Comparação entre as arquiteturas de rede centralizada e descentralizada

A imutabilidade de dados é principal diferencial da estrutura de dados Blockchain, que segue a lógica de armazenamento append-only. Essa característica é desejável em contextos

socioeconômicos diversos, pois reforça a confiança entre agentes conhecidos ou desconhecidos – em particular, reduz a assimetria de informação. A imutabilidade é garantida pela estrutura de blocos e hashes: para alterar as informações de uma transação, é necessário alterar os hashes de todos os blocos anteriores. Em aplicações como a Bitcoin, o custo do poder computacional necessário para esse tipo de operação inviabiliza fraudes.

A Bitcoin é uma aplicação particular da Blockchain e das Tecnologias de Registro Distribuído. Essa criptomoeda foi apresentada em 2008 pelo pseudônimo Satoshi Nakamoto, como um sistema de pagamentos descentralizado baseado em uma rede Peer-to-Peer. As transações no sistema da Bitcoin seguem um modelo de input-output baseado no histórico de transações de cada usuário.



Exemplo de uma transação. Os inputs da transação são provenientes de transações anteriores.

Um usuário da rede Bitcoin opera através de uma ou múltiplas contas. Cada conta contém uma chave pública – que é apresentada para outros agentes da rede com quem o usuário transaciona – e uma chave privada. As aplicações que realizam transações são chamadas de Carteiras (Wallets).

Exemplo de uma sequência de transações. Os inputs de uma transação são usados como outputs em outra transação.

Para construir uma transação, a carteira irá retirar fundos de transações anteriores que não foram gastas. Caso a carteira não guarde esses dados, é possível realizar uma query na rede da Bitcoin, que retorna informações como os hashes das transações – tx_hash – e os valores, como no exemplo abaixo:


```
{
        "unspent_outputs":[
                {
                        "tx_hash":"186f9f998a5...2836dd734d2804fe65fa35779",
                        "tx_index":104810202,
                        "tx_output_n": 0,
                        "script":"76a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac",
                        "value": 10000000,
                        "value_hex": "00989680",
                        "confirmations":0
                }
        ]
}
```


Na figura, nota-se algumas informações relevantes, como o índice da transação no registro distribuído, o valor transacionado em Satoshis - cada um corresponde a 0.00000001 Bitcoins - e o número de confirmações já realizadas pelos *peers* da rede.
Caso a conta tenha fundos suficientes, a carteira irá construir os outputs necessários para a transação. O usuário define para qual conta irá enviar os fundos, e um script bloqueia os fundos até que a conta receptora assine a transação com sua chave privada.
Uma transação de bitcoins é adicionada ao Registro da Bitcoin por meio de um sistema de mineração. Esse processo é realizado por nós da rede que dedicam poder computacional para resolver um problema difícil, cuja solução pode ser facilmente verificada - uma boa analogia ao "problema difícil" é um desafio de Sudoku. Além de gerar novas moedas como recompensa aos mineradores pelo poder computacional empenhado, a mineração garante que uma transação será validada apenas após um certo montante de poder computacional ter sido

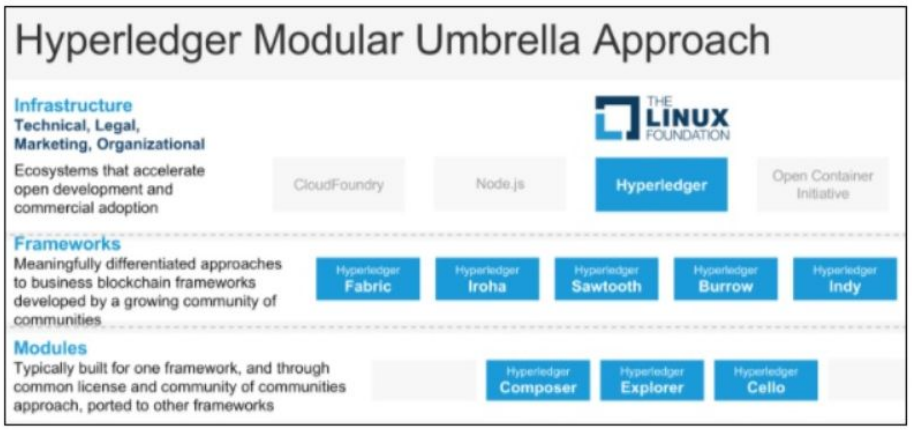
empenhado no processo. Utilizando o algoritmo SHA256, o primeiro minerador a encontrar a solução do problema irá publicar um bloco, com um conjunto de transações, na Blockchain.

4.2. Ethereum

A plataforma Ethereum foi criada em 2014 por Vitalik Buterin. O projeto visa expandir as capacidades da Blockchain e do protocolo original da Bitcoin. A plataforma Ethereum opera com a Ethereum Virtual Machine, uma máquina virtual Turing Completa, ou seja, é capaz de executar programas – chamados de "Smart Contracts" – que podem resolver qualquer problema computacional, caso uma quantidade arbitrariamente grande de memória esteja disponível. Um "Smart Contract" é um sistema que automatiza os procedimentos de uma transferência de valor a partir de regras arbitrárias preestabelecidas, e são implementados com a linguagem Solidity - proposta por Gavin Wood - em Ethereum. Como Solidity é uma linguagem Turing Completa, arquiteturas sofisticadas envolvendo chamadas recursivas de múltiplos contratos podem ser implementadas em Ethereum.
A plataforma Ethereum é baseada em estados, assim como a Bitcoin. Um estado inicial A é alterado por uma transação, que leva o sistema ao estado A'. A principal diferença entre os estados do sistema Bitcoin e do sistema Ethereum é a autonomia das contas que contém um contrato, isto é, cujo comportamento não é controlado por um agente humano, mas por uma lógica de software estabelecida pelo contrato. Um contrato pode receber e alocar Ethers - as moedas oficiais do sistema Ethereum - assim como um agente humano através de sua conta.
A plataforma Ethereum permite que programadores desenvolvam criptomoedas e criptotokens, organizações autônomas descentralizadas (Decentralized Autonomous Organizations), e apps para representação digital de ativos genéficos. Ethereum é, então, uma camada de base para a construção de aplicações descentralizadas genéricas.

4.3. Hyperledger

Hyperledger é um conjunto de projetos de código aberto mantidos pela Linux Foundation que visam acelerar a absorção das Tecnologias de Registro Distribuído em diferentes indústrias. A Hyperledger é dividida em cinco frameworks e três módulos de suporte, cada um com objetivos e funcionalidades diferentes.



4.3.1. Hyperledger Frameworks

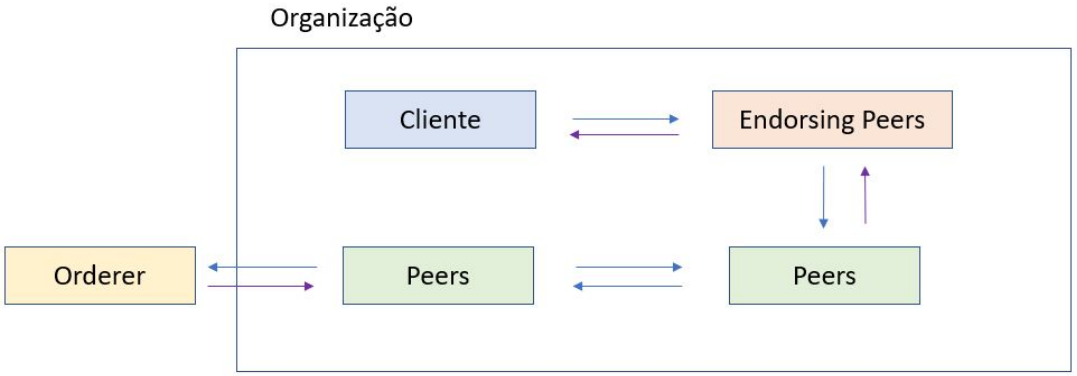
Os principais projetos apoiados pela Hyperledger consistem em frameworks para o desenvolvimento de aplicações de registros distribuídos em empresas e indústrias. Cada framework da Hyperledger contém um registro do tipo “append-only”, um algoritmo de consenso para a alteração do registro, um mecanismo de permissão para garantir a privacidade das transações, e um “smart contract”, que contém a lógica de negócios.
O framework Iroha é o único desenvolvido para aplicações mobile, com pacotes para Android e iOS. O Iroha foi desenhado para complementar os frameworks Fabric e Sawtooth.
O framework Sawtooth utiliza uma plataforma modular para para o desenvolvimento e distribuição de registros distribuídos, e permite tanto o desenvolvimento de redes permissionadas, quanto o de não permissionadas. Os algoritmos de consenso utilizados no Sawtooth variam de acordo com o tamanho da rede, garantindo a flexibilidade da implementação dos projetos que utilizam esse framework. O algoritmo padrão desse framework, Proof of Elapsed Time (PoET) tem grande escalabilidade, mas não consome grandes quantidades de energia, como o algoritmo de consenso da Bitcoin. O algoritmo PoET consiste em um tempo aleatório de espera por cada nó participante da rede. O nó com o menor tempo de espera irá gravar um novo bloco no registro.
O framework Indy tem uma arquitetura específica para o desenvolvimento de aplicações descentralizadas de identidade. Os participantes de uma rede Indy têm controle sobre suas identidades digitais, enquanto negócios e organizações que utilizam tais dados têm acesso apenas a ponteiros de dados. Uma vez processados em uma aplicação particular por uma organização, os ponteiros dos dados são descartados. O framework Burrow se destaca por utilizar implementar a especificação da Ethereum Virtual Machine. O Burrow é composto por um “gateway”, que contém interfaces para sistemas e usuários, uma engine para “smart contracts” e outra para a execução de algoritmos de consenso, e uma interface de blockchain para aplicações.
A seguir, explicaremos o framework Fabric com maior profundidade, dada sua relevância no desenvolvimento da aplicação GvCoin.

4.3.1.1. Hyperledger Fabric

Hyperledger Fabric é um framework com arquitetura modular, que permite que aplicações sejam desenvolvidas com mecanismos de consenso e permissão que podem ser modificados. Os principais componentes do Fabric são:

1. Canais - permitem que participantes da rede realizem transações com privacidade de dados
2. “Chaincode” - implementa a lógica de negócios da aplicação
3. Registro - contém o estado da rede com a sequência das transações ao longo do tempo
4. Rede - conjunto de processadores de dados chamados de “peers”, que suportam a rede computacionalmente
5. “Ordering Service” - conjunto de nós que ordenam as transações em blocos
6. Estado Global - contém os dados de todos os ativos da rede. Esses dados são guardados em bancos de dados, em particular, o LevelDB e o CouchDB
7. “Membership Service Provider”(MSP) - mecanismo que administra identidades para acesso permissionário à rede

O primeiro componente de um processo de transação no Fabric é o registro, que contém informações das transações realizadas na rede e um banco de dados do estado da rede. Apenas as operações criar (Create - C) e ler (Read - R) podem ser realizadas no registro de informações das transações, mas, no banco de dados do estado da rede, além das operações citadas, é possível atualizar (Update - U) e deletar (Delete - D) dados, completando as operações clássicas do modelo CRUD.
Uma transação é iniciada com uma requisição de uma aplicação cliente para os “peers” da rede. A requisição é propagada até chegar aos “endorsing peers”, que simulam a transação para verificar se todos os requisitos do “chaincode” são cumpridos, ou seja, se a transação é válida. Cada “endorsing peer” assina a requisição, e envia a assinatura para o “Ordering Service”, que irá ordenar as transações válidas em blocos. Por fim, os “endorsing peers” retornam ao usuário o resultado da transação - sucesso ou falha. O sucesso ou falha da transação é definido no “chaincode”, com critérios como a porcentagem ou um número mínimo de aprovações dos “endorsing peers”.

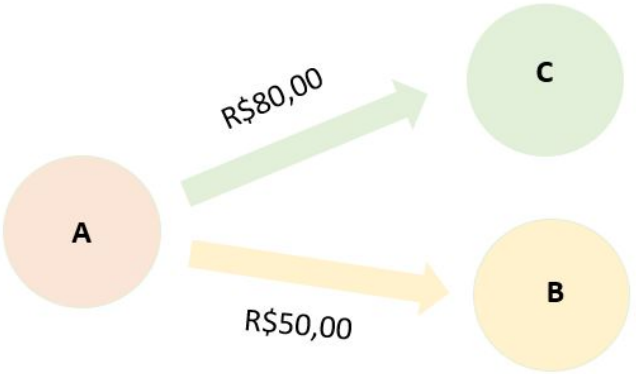


Modelo de transação

Essa arquitetura é altamente escalável, dado que apenas os "endorsing peers" precisam executar os códigos de verificação. É possível que a rede seja composta por um conjunto de "peers" dedicados a transmitir informações, e um conjunto menor de "endorsing peers", dedicados a validar informações, por exemplo.

É importante notar que a escolha das funções de cada membro da rede no sistema é definida pelos arquitetos do sistema. A flexibilidade de arquitetura oferecida pelo Hyperledger Fabric possibilita aos engenheiros de software construir aplicações genéricas, assim como em Ethereum, com o diferencial de cada blockchain em Fabric ser privada ao ambiente da empresa.

A privacidade dos dados pode ser reforçada no Fabric com o uso de canais - um sistema de comunicação entre usuários da rede. Um ativo pode ser trocado a valores diferentes em canais privados, com parte das informações restritas aos participantes daquele canal. No esquema abaixo, apenas os agentes A e C têm acesso ao valor da transação em verde, e apenas os agentes A e B têm acesso ao valor da transação em amarelo.



Canais Privados

A ordenação das transações em blocos, que finaliza o processo de consenso na rede, pode ser

implementada com o mecanismo de ordenação SOLO, em ambientes de desenvolvimento, e com o Kafka, em ambientes de produção. Como o SOLO contém apenas um nó responsável por atender todas as requisições do sistema, ele não é um mecanismo com tolerância a falhas, e portanto, é inadequado aos ambientes de produção.
O serviço de streaming de mensagens Apache Kafka, por outro lado, é ideal para produção, pois contém uma arquitetura clusterizada, e grande escalabilidade.
O componente de permissão do Fabric, MSP, contém os componentes necessários para a autenticação e autorização dos participantes da rede. Os componentes de autenticação verificam se os certificados do participante são válidos, e se ele tem permissão para acessar a rede. Já os componentes de autorização concedem permissões para que o participante realize certas operações na aplicação - por exemplo, um administrador tem permissão para modificar o "chaincode".

4.3.2. Hyperledger Modules

Os módulos de Hyperledger são softwares de suporte, que permitem aos desenvolvedores adicionar funcionalidades às aplicações construídas com os frameworks. O módulo Cello é utilizado para o deploy de aplicações com Blockchain-as-a-Service - um modelo de serviço no qual toda a infraestrutura de blockchain é oferecida em nuvem, reduzindo a complexidade do desenvolvimento de aplicações. A ferramenta permite que a Blockchain seja armazenada em nuvem, em uma arquitetura de clusters de containers.
O Explorer é uma ferramenta para visualização de operações em blockchain, que permite que usuários pesquisem informações como blocos, transações, informações de rede e estrutura de "smart contracts" em redes permissionadas, sem afetar a privacidade da rede.

4.3.2.1. Hyperledger Composer

Hyperledger Composer é um conjunto de ferramentas que otimizam o desenvolvimento de aplicações de redes de negócios em blockchain. Essas ferramentas permitem que os arquitetos de software, especialistas de negócios e desenvolvedores criem modelos de redes de blockchain adequadas ao domínio do negócio, gerem APIs REST - uma arquitetura de criação de serviços web que contribui com a escalabilidade e agilidade de sistemas - para a interação com a rede de blockchain e aplicações web com o framework para desenvolvimento de software na linguagem Javascript, Angular.
O processo de desenvolvimento com a ferramenta Composer é iniciado com a modelagem da rede de negócios pelo especialista de negócios, na Composer Modeling Language, uma linguagem orientada a objetos que permite a construção de um modelo de domínio para a aplicação. Na etapa seguinte, o desenvolvedor programa as especificações das transações da aplicação em javascript, com base no modelo de domínio escrito na Composer Modeling Language. Em seguida, o administrador da rede realiza o deploy da aplicação em uma rede de blockchain baseada no framework Fabric. A aplicação pode ser testada em um ambiente de

execução chamado Composer Playground. Esse ambiente de testes oferece uma interface adequada apenas para desenvolvedores, e é ideal para testar o “chaincode” de uma aplicação.

5. GvCoin

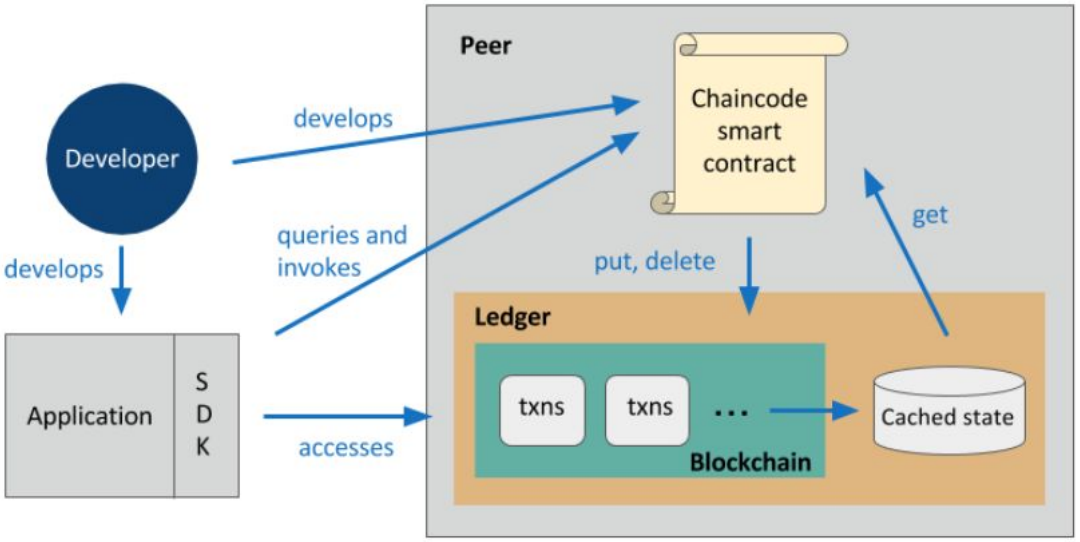
A GvCoin foi concebida como uma moeda universitária, que será utilizada em um ambiente fechado. A primeira pergunta de pesquisa direcionou a definição dos requisitos técnicos da GvCoin. É desejável que o acesso ao registro distribuído e à plataforma dessa moeda seja concedido sob permissão, e que o modelo de validação não seja fortemente dependente de poder computacional. Além disso, é possível que a criptomoeda seja mais adequada em ambiente fechado com alguma centralização. Dadas essas observações, o Hyperledger Fabric foi selecionado para o desenvolvimento da primeira versão da GvCoin, na primeira etapa da pesquisa, uma vez que este framework tem amplo suporte para aplicações com permissão e desenvolvimento com mecanismos de consenso com baixa necessidade de poder computacional, e foi construído para o desenvolvimento de aplicações para organizações, com funcionalidades para ambientes fechados. Todavia, na segunda etapa da pesquisa, as barreiras ao desenvolvimento de aplicações em Hyperledger Fabric causadas pelas extensas dependências das tecnologias utilizadas nesse framework levaram ao desenvolvimento da versão final em Ethereum, e constatou-se que tanto o problema de permissão quanto o de custo computacional não causam impedimentos relevantes ao uso do sistema implementado em Ethereum.

A segunda pergunta de pesquisa direcionou o propósito econômico da GvCoin. A otimização econômica de uma criptomoeda deve ser realizada com base nas características de seu ambiente de uso. A GvCoin poderá ter usos diversos dentro da FGV, mas, em princípio, não é concebida como uma substituta aos meios de pagamentos tradicionais. Identifica-se como maior potencial da moeda o engajamento de grupos de pesquisa e sociedades estudantis dentro do ambiente da FGV.

Como moeda de engajamento, o intuito da GvCoin é alterar os incentivos de certos agentes da FGV, direcionando esforços em atividades que contribuam positivamente com o ambiente da organização. O sistema de incentivos introduzido pela GvCoin pode ser visto como a criação de mercados internos à universidade sobre ativos ociosos. Um laboratório pouco utilizado, por exemplo, que passa a ser utilizado por grupos estudantis para a execução de simulações computacionais, ou uma equipe de membros de uma empresa júnior que, com a GvCoin, é alocada para contribuir em projetos de um diretório acadêmico.

5.1. GvCoin em Hyperledger fabric

O modelo de produção da GvCoin em Hyperledger Fabric pode ser entendido através da figura abaixo:

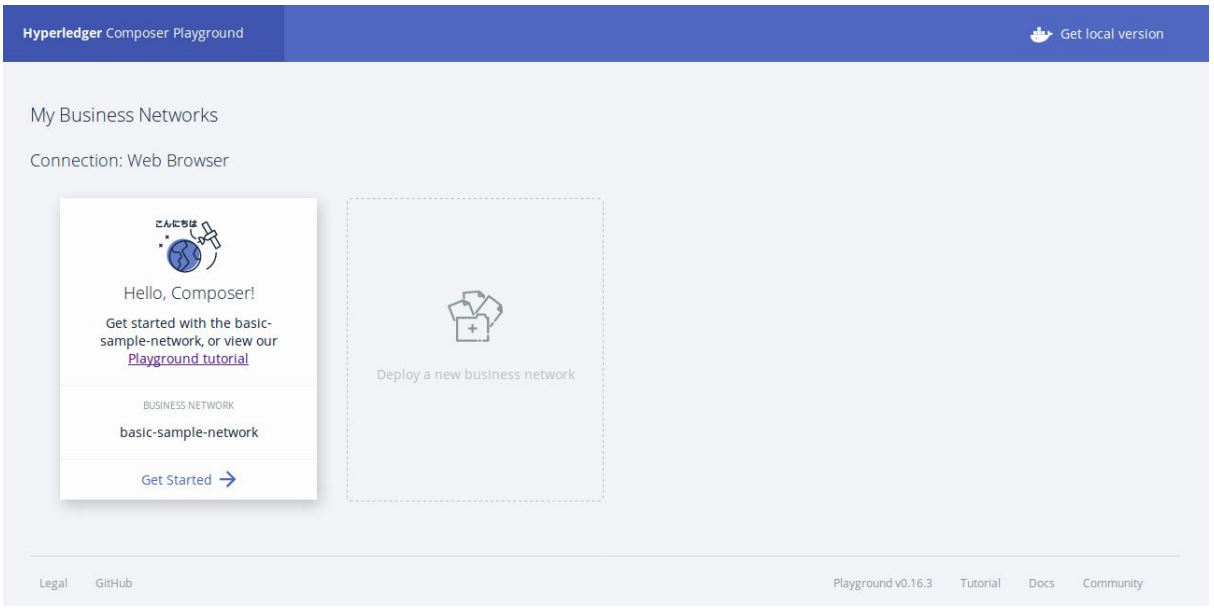


Componentes como a Blockchain, o registro e as operações genéricas entre componentes da aplicação são disponibilizados no framework. O desenvolvimento da GvCoin consiste, então, na programação da aplicação em Node.js, e do "Chaincode", em uma linguagem orientada a objetos.
Em princípio, a linguagem JavaScript foi escolhida para o desenvolvimento do "Chaincode", por ser utilizada pelo "runtime environment" Node.js. O módulo Hyperledger Composer foi escolhido para dar suporte ao desenvolvimento da GvCoin no framework Fabric.
A primeira etapa do desenvolvimento consistiu na definição de funções básicas da GvCoin, e na execução de testes na ferramenta Hyperledger Composer Playground - os dados da aplicação são armazenados pelo Playground no próprio browser, sem armazenamento remoto.
Os principais requisitos iniciais da plataforma de testes da GvCoin são:

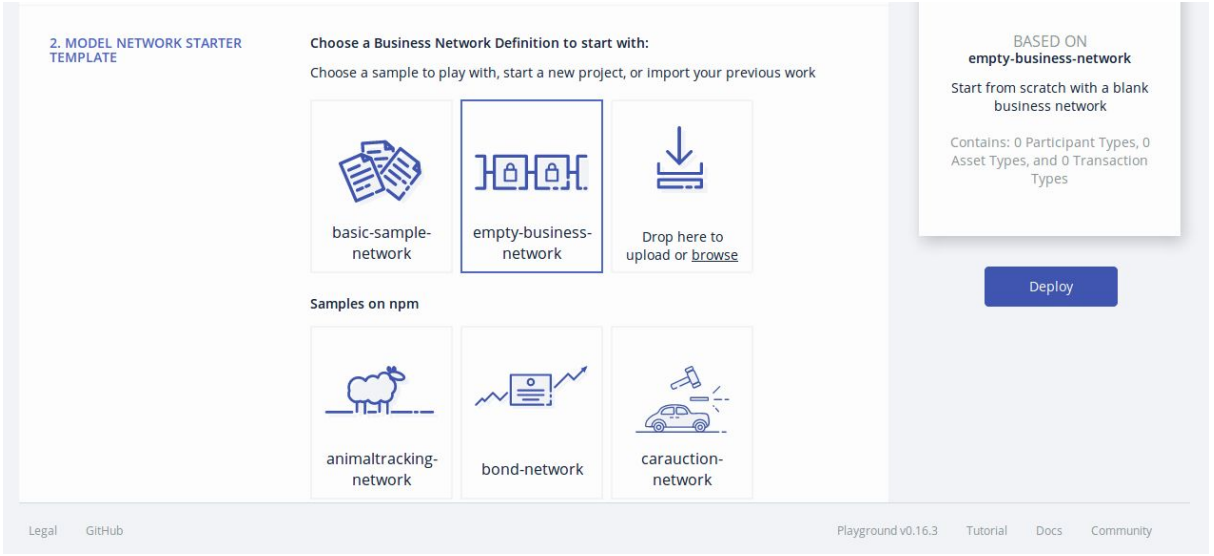
1. (Admin) Criar perfis de participantes
2. (Admin) Criar GvCoins
3. Transacionar GvCoins
4. Visualização de transações

Pelo padrão do Hyperledger Fabric, essa primeira versão da GvCoin segue o modelo Model-View-Controller. Neste modelo, a aplicação é dividida em um componente de modelagem dos objeto do sistema - a Model - em um componente intermediário, no qual as funções do sistema são definidas - o Controller - e em um terceiro, através do qual os usuários podem interagir com a aplicação - a View. Uma sessão de uso da primeira versão de testes, com simplificações dos três componentes do MVC, pode ser executada com o código disponível em:
https://github.com/yankaled/GvCoin/tree/47e1180d28ce38883a37559a755357ae795f1988/gvcoin-mvp.
Em seguida, deve ser feito o deploy da aplicação na ferramenta Hyperledger Composer

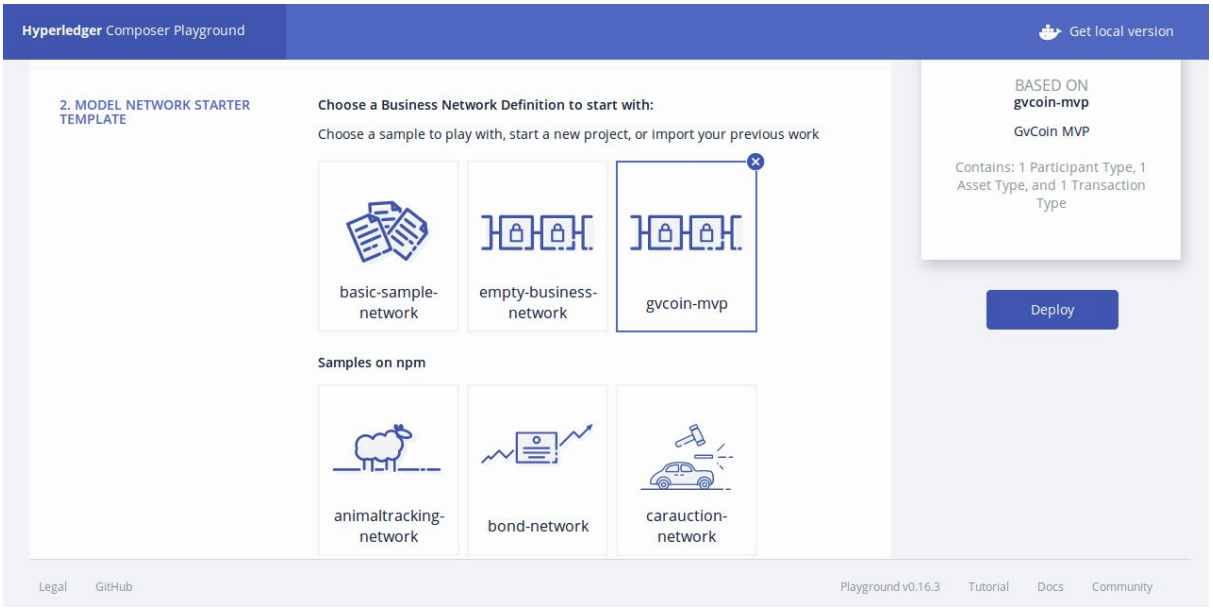
Playground, com o seguinte fluxo de trabalho:
Inicialização de uma nova “business network”:



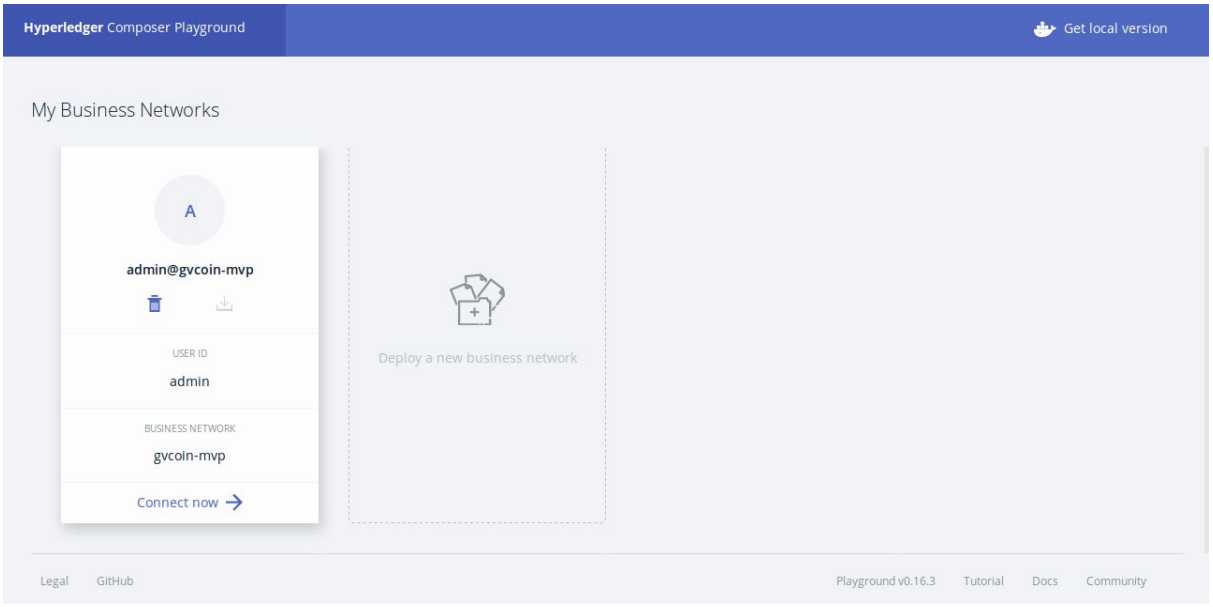
Importação do arquivo “gvcoin-mvp@0.0.1.bna”, da pasta “Dist”.



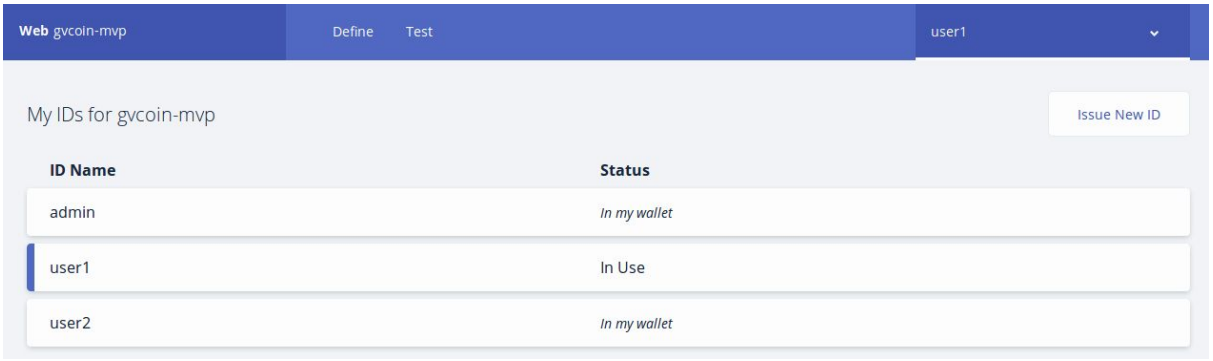
Deploy da aplicação:



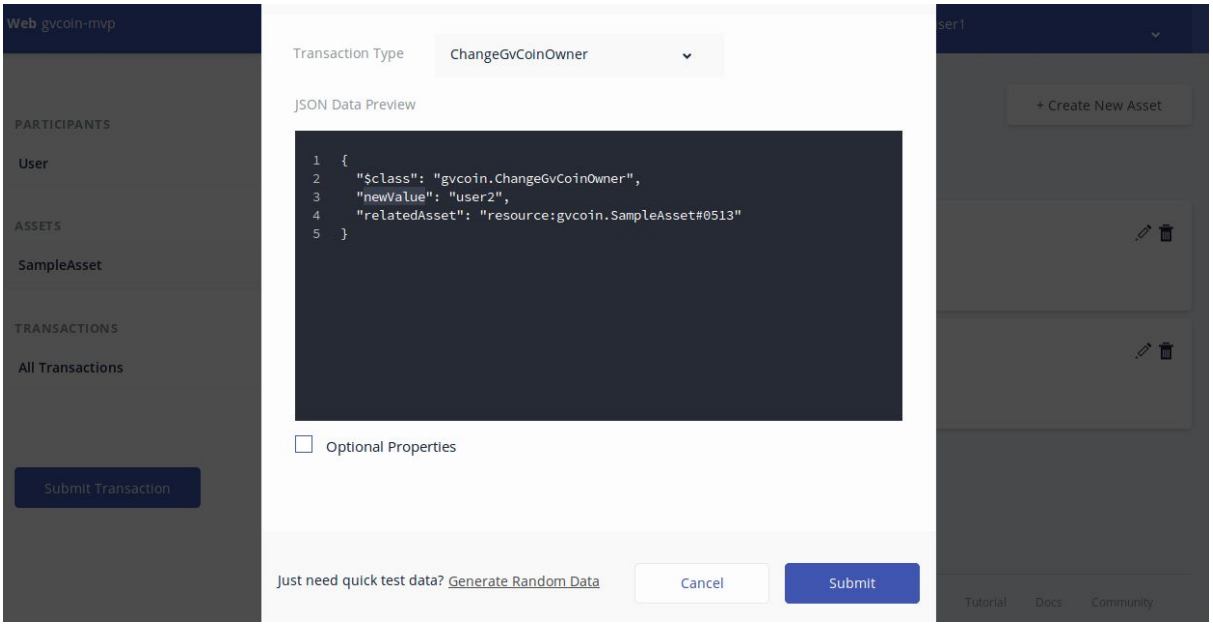
Habilitação da conexão:



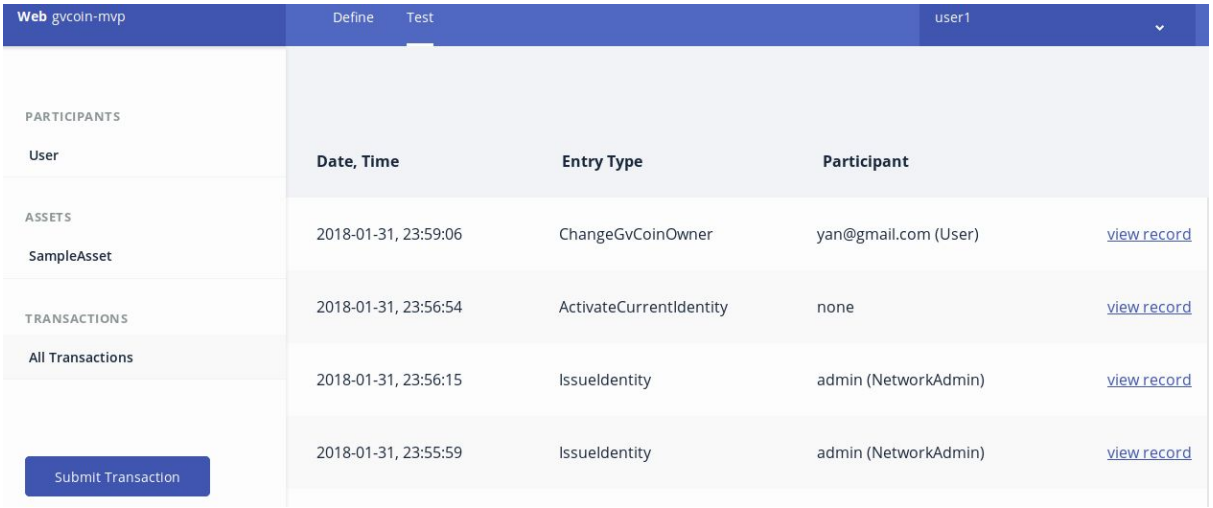
Criação e escolha de usuários:



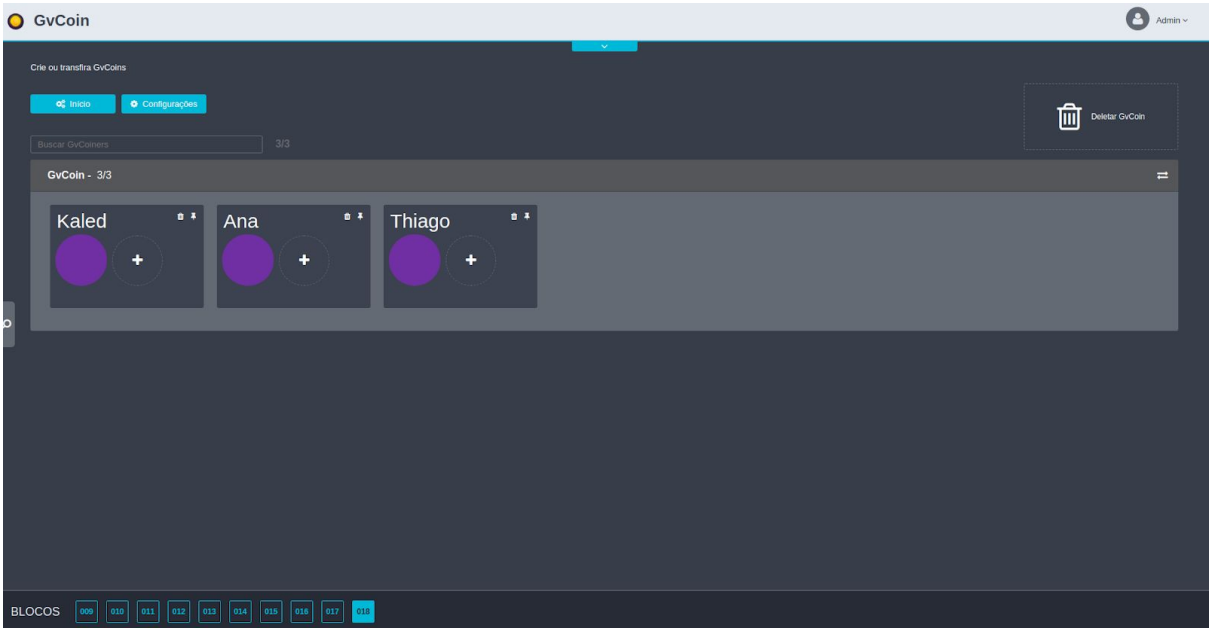
Execução de uma transação de transferência de posse com um usuário criado:



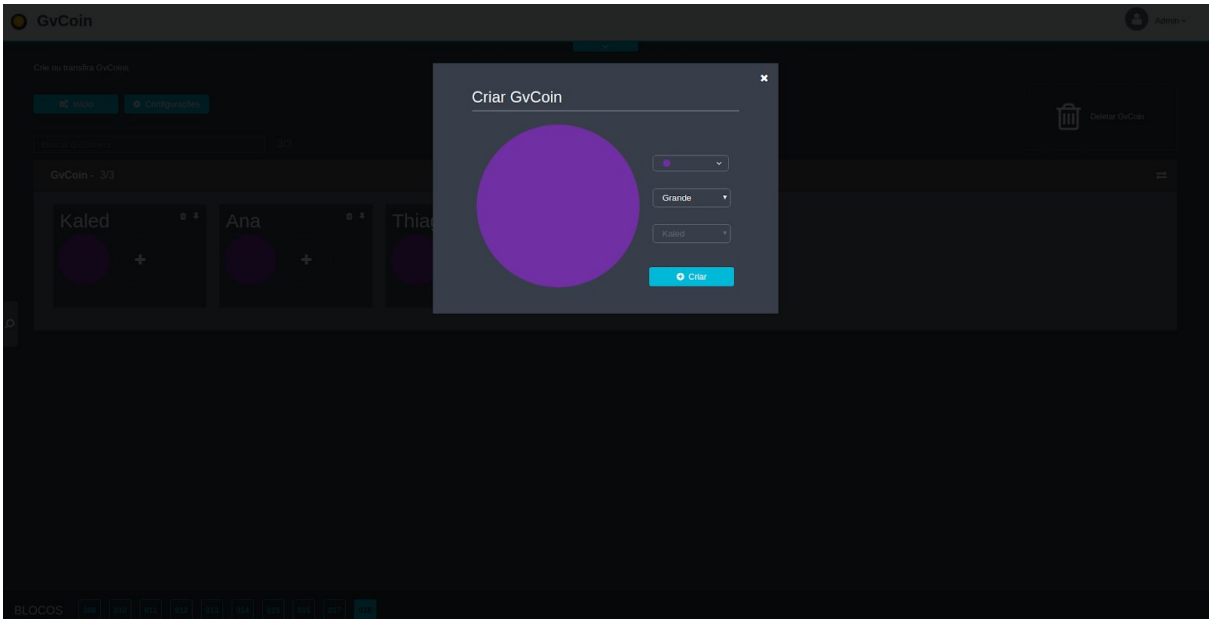
Visualização das transações:



A segunda versão da GvCoin em Hyperledger foi construída com base na aplicação de testes Marbles, da IBM. Essa versão contém apenas a tela do administrador do sistema, responsável por criar GvCoins para cada participante.



Na tela inicial, o administrador tem acesso aos usuários do sistema e às suas respectivas GvCoins. Dada a implementação baseada na aplicação Marbles, cada GvCoin é um token indivisível.

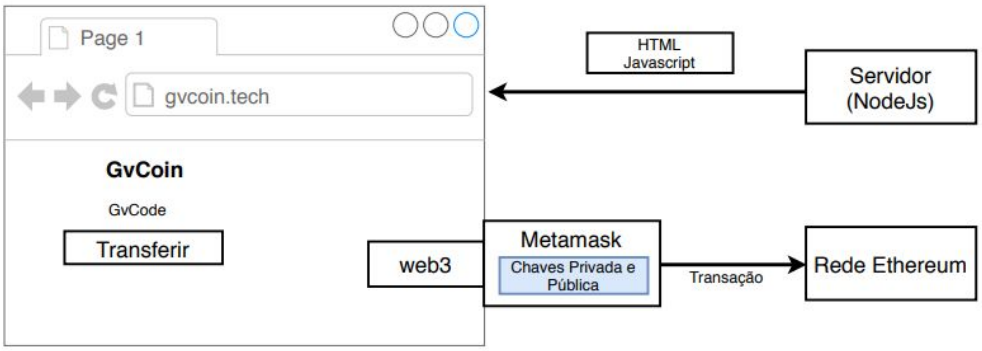


O administrador pode, então, criar tokens para cada usuário, bem como deletar e transferir tokens entre usuários.
Ao longo do desenvolvimento com Hyperledger, diversos problemas com dependências -

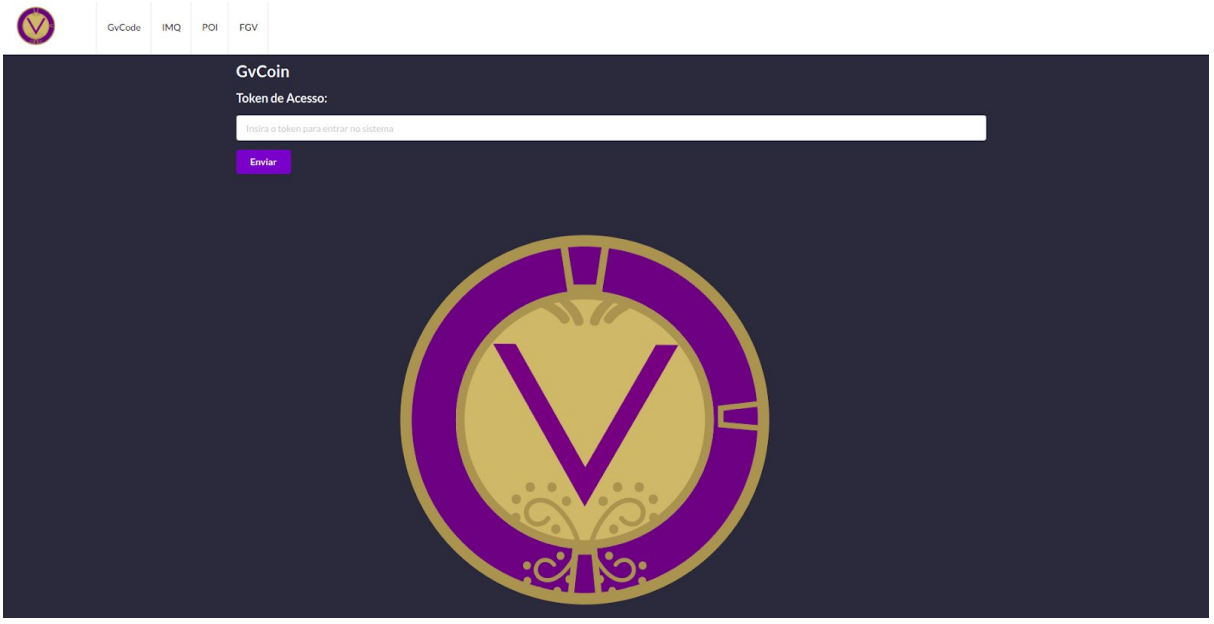
softwares utilizados pela aplicação que devem ser instalados no sistema do desenvolvedor e no servidor de produção - da tecnologia foram encontrados. Ambas as tecnologias de Hyperledger e Ethereum estão em constante mudança, com alta frequência de criação de módulos e frameworks por membros das comunidades. Todavia, os posicionamentos de ambas as tecnologias têm forte impacto nos recursos disponíveis para desenvolvedores. Enquanto a comunidade de Hyperledger é composta por desenvolvedores interessados em construir aplicações para gerar ganhos de eficiência em empresas, a comunidade de Ethereum é composta por desenvolvedores de aplicações descentralizadas. Como consequência, as soluções em Ethereum possuem amplo suporte da comunidade, e o volume de ferramentas disponíveis para o desenvolvimento de "Dapps" - aplicações descentralizadas - é consideravelmente maior do que em Hyperledger.

5.2. GvCoin em Ethereum

Dadas as limitações das tecnologias de Hyperledger, a versão final da GvCoin foi desenvolvida em Ethereum. Dois problemas principais foram encontrados no desenvolvimento com Ethereum: a ausência de um mecanismo de permissão e o custo em Ether de realizar qualquer transação - qualquer procedimento que altera o estado da "blockchain" é uma transação em Ethereum.
O diferencial oferecido por Hyperledger consiste no mecanismo de permissão que possibilita ao desenvolvedor construir redes privadas para o uso dentro de uma organização. Uma solução encontrada para construir um sistema de permissão em um aplicativo baseado em Ethereum consiste em uma arquitetura mista, unindo conceitos de arquitetura tradicional de software e arquitetura de Dapps.
Em uma arquitetura tradicional de aplicações web - acessadas por browsers, como o Chrome e o Firefox - as requisições feitas pelo usuário podem ser processadas em um servidor, e enviadas para o browsers. Todavia, o problema com esse modelo em uma aplicação em Ethereum é o custo de gravação de transações na blockchain. Toda transação na Main Network - a rede de produção - é minerada antes de ser gravada no registro distribuído, e, portanto, consome poder computacional de "peers" da rede que deve ser pago com "gas" - o custo de processamento computacional na rede Ethereum em Wei, equivalente à 18ª raiz de Ether. Tal custo impõe restrições acerca dos padrões de desenvolvimento e dos tipos de aplicações economicamente viáveis que podem ser oferecidas ao público.
Uma arquitetura mista consiste em um modelo no qual um parte dos recursos que serão apresentados ao usuário são processados em um servidor, outra no browser e ainda outra parte enviada à blockchain. A ilustração abaixo representa esse modelo.



Um sistema semelhante aos tradicionais mecanismos de cadastro em websites pode ser utilizado como ferramenta de permissionamento. O sistema funciona com um token de acesso, cujo processamento é feito no próprio browser do usuário. Usuários com o token de acesso correto têm acesso ao sistema, e podem interagir com a aplicação através de uma conta criada no "Metamask" - um plugin desenvolvido para browsers conecta o usuário às diversas redes com protocolo de Ethereum. Na figura abaixo, vemos a tela com a solicitação do token de acesso do usuário.



Tela inicial da plataforma GvCoin

O segundo problema enfrentado diz respeito ao custo de executar operações computacionais em uma transação que será gravada na blockchain. As soluções convencionais para esse problema envolvem otimizações de algoritmos e estruturas de dados - que também foram utilizadas na GvCoin - mas, no caso da aplicação desenvolvida para esta pesquisa, uma solução mais abrangente foi adotada: utilizar a rede de testes Rinkeby.
Na Rinkeby, todo Ether transacionado é "falso", sem conversão para o Ether utilizado na

Main Network ou para moedas reais, como o Dólar e o Yen. Logo, nenhuma transação realizada na pesquisa e gravada na rede Rinkeby teve custo monetário. Ainda assim, o custo das transações realizadas no sistema proposto não é alto - cada transação tem valor esperado de alguns centavos de Dólares americanos - e a rede principal poderá ser utilizada em produção por usuários da GvCoin.
Mesmo com a flexibilidade para desenvolvimento oferecida pela Rinkeby, adotou-se no desenvolvimento da GvCoin uma série de padrões para otimização dos códigos. Iniciaremos a apresentação do sistema com uma exposição e explicação do contrato da GvCoin, programado na linguagem Solidity:

```
pragma solidity ^0.4.17;

contract GvCoin {
    struct Request {
        string description;
        uint value;
        address recipient_society;
        string recipient_name;
        bool accepted;
    }

    struct Society {
        uint wealth;
        uint revenue_total;
        uint finance_total;
        string name;
        address society_address;
        uint[] revenue_series;
        uint[] finance_series;
    }
```

É relevante notar que um contrato em Solidity é equivalente ao conceito de classe em outras linguagens de programação, e, portanto, é responsável pela definição das características e métodos do software - uma diferença relevante entre classes em orientação a objetos e contratos em Solidity é que definimos os agentes do sistema como variáveis ou estruturas no contrato, enquanto cada agente é definido pela própria classe na orientação a objetos.
No início do contrato, definimos duas estruturas com a palavra especial “struct”. A estrutura Request representa uma requisição que será enviada ao administrador do sistema para a emissão de novas GvCoins, enquanto a estrutura Society representa uma sociedade estudantil ou entidade que participará do sistema.
A estrutura Request é composta por uma variável do tipo “string” - um texto, como uma frase, por exemplo - chamada descrição, que receberá a descrição do motivo da solicitação de GvCoins pela sociedade. A Request também contém uma variável do tipo “uint” - um número inteiro e positivo - que representa o valor de GvCoins solicitado, uma variável do tipo “address” que representa o endereço da conta da sociedade - composto por 42 caracteres - na rede Ethereum. O endereço utilizado pela sociedade GvCode nesta pesquisa, por exemplo, é: 0x0b9110fbFABeB517F29cBF93A7D424d3a4e641b0. A variável “recipient_name” tem a mesma função da variável “recipient_society”: retornar a sociedade em funções de busca. Cada variável será útil em partes específicas do programa. Por fim, a Request contém um campo chamado, “accepted” do tipo “bool”, que recebe apenas os valores

“true” - verdadeiro - ou “false” - falso. Esse campo será utilizado para que o sistema verifique se uma requisição já foi aprovada, e impeça requisições com valor “true” para “accepted” de serem aprovadas novamente.
A estrutura “Society” é composta por uma variável do tipo “string” chamada “name”, que é o nome da sociedade, e será utilizada no processo de transferência de GvCoins entre participantes do sistema. A estrutura também é composta por uma variável do tipo “uint” chamada “wealth”, que representa o volume de GvCoins que a sociedade possui, e por outra variável que representa o endereço da sociedade na rede. Essa estrutura contém ainda quatro variáveis que são de grande relevância para o propósito da GvCoin: “finance_total”, “revenue_total”, “finance_series” e “revenue_series”. As duas variáveis “total” são métricas que explicam a interação da sociedade com o sistema social criado pela GvCoin, e as variáveis “series” guardam “arrays” - estrutura utilizada para guardar coleções de variáveis e acessá-las com sua posição na coleção - de financiamentos e recebimentos de cada sociedade, permitindo ao administrador e aos interessados visualizar as mudanças dos estados de cada sociedade no tempo. As variáveis “finance” captam mensuram o uso de recursos universitários por cada entidade, enquanto as variáveis “revenue” mensuram o quão útil uma entidade estudantil ou grupo de pesquisa é para outros grupos universitários.

```
Request[] public requests;
Society[] public societies;
address public admin;
uint public gvconomy;
mapping(string => uint) index_society_name;
mapping(address => uint) index_society_address;
```

No trecho acima, definimos mais algumas variáveis e métodos que serão úteis aos usuários e ao sistema. Definimos dois arrays de variáveis dos tipos “Request” e “Society” que guardarão, respectivamente, requisições e sociedades.
Em seguida, duas variáveis são definidas: o endereço do administrador do sistema e o valor total de GvCoins na economia - a gvconomy. Os dados enviados pelo administrador ao sistema consistem em permissões para emissão de GvCoins e criação de sociedades, e, portanto, o usuário com funções administrativas precisa apenas que o sistema reconheça seu endereço como um com permissões administrativas. Futuramente, a função do administrador pode ser estendida para uma lógica mais sofisticada, com múltiplos administradores e um sistema de votos administrativos para emitir GvCoins.
Por fim, dois mapas são definidos. Um mapa é um tipo de estrutura de dados que facilita a busca por valores específicos utilizando um esquema de pares de chaves e valores - uma chave é informada ao algoritmo de busca, e este retorna um valor associado àquela chave. O uso de mapas é necessário em Solidity, pois o curso de executar buscas em um array pode se tornar restritivo à medida que o array cresce.

```
modifier restricted() {
    require(msg.sender == admin);
    _;
}

constructor () public {
    admin = msg.sender;
}
```

Em seguida, duas funções são definidas: um modificador e um construtor. O modificador “restricted()” será utilizado para restringir o acesso às funções administrativas apenas para o administrador. O endereço do administrador - msg.sender - é capturado na função construtora, responsável por construir uma instância do contrato GvCoin. O usuário que criar uma instância da GvCoin será o administrador do sistema.

```
function createSociety (string name, address society_address) public {
    Society memory newSociety = Society({
        name: name,
        revenue_total: 0,
        revenue_series: new uint[](0),
        finance_total: 0,
        finance_series: new uint[](0),
        wealth: 0,
        society_address: society_address
    });

    societies.push(newSociety);
    index_society_name[name] = societies.length - 1;
    index_society_address[society_address] = societies.length - 1;
}
```

A função acima, “createSociety”, é responsável por criar uma sociedade e adicioná-la ao array “societies”. Ao final da função, associamos o nome e o endereço da sociedade seu índice no array “societies” através de um mapa. Tal associação irá facilitar a busca por uma sociedade em outras funções.

```
function issueGvCoins(uint amount, uint recipient_index) private
restricted {
    gvconomy += amount;
    societies[recipient_index].wealth += amount;
    societies[recipient_index].finance_total += amount;
    societies[recipient_index].finance_series.push(amount);
}

function transferGvCoins(uint amount, uint sender_index,
uint recipient_index) public {
    require(societies[sender_index].wealth >= amount);
    societies[sender_index].wealth -= amount;
    societies[recipient_index].wealth += amount;
    societies[recipient_index].revenue_total += amount;
    societies[recipient_index].revenue_series.push(amount);
}
```

Em seguida, definimos as funções “issueGvCoins”, que será utilizada - na função

“approveRequest” - para emitir GvCoins, e “transferGvCoins”, que será utilizado pela sociedade para transferir GvCoins para outro grupo universitário. É relevante notar que, apesar de a implementação atual em Ethereum conter campos numéricos para entradas de valores de transferências de GvCoins, e não “tokens” selecionáveis como no sistema proposto em Hyperledger, tecnicamente a GvCoin é um “token” com valores discretos, dado que todos as variáveis de emissão e transferência recebem apenas valores inteiros.

```
function createRequest(string description, uint value, address recipient)
public {
    Request memory newRequest = Request({
        description: description,
        value: value,
        recipient_society: recipient,
        recipient_name: societies[getSocietyByAddress(recipient)].name,
        accepted: false
    });

    requests.push(newRequest);
}

function approveRequest(uint request_index, string society_name,
uint amount)
public restricted {
    Request storage request = requests[request_index];
    request.accepted = true;
    issueGvCoins(amount, getSocietyByName(society_name));
}
```

Em seguida, as funções “createRequest” e “approveRequest” são definidas. A primeira função será utilizada pela sociedade para criar uma requisição de emissão de GvCoins, enquanto a segunda será usada pelo administrador para aprovar requisições.

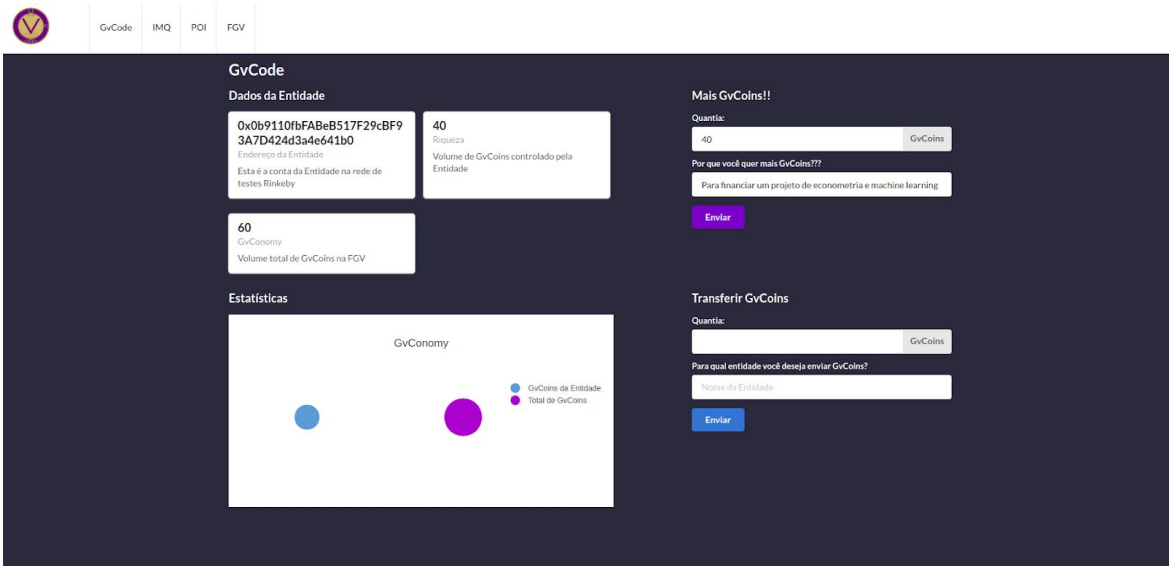
```
function destroyGvCoins(uint society_index, uint amount ) public
restricted {
    societies[society_index].wealth =
    societies[society_index].wealth - amount;
    gvconomy = gvconomy - amount;
}

function getSocietyByName(string name) public view returns(uint) {
    return index_society_name[name];
}

function getSocietyByAddress(address society_address) public view
returns(uint) {
    return index_society_address[society_address];
}
```

A função “destroyGvCoins” foi construída com o objetivo de corrigir emissões erradas em ambiente de desenvolvimento. Em uma ambiente de produção, esse tipo de função não é recomendado, pois um dos principais fatores de adoção de sistemas de blockchain é a imutabilidade do sistema. Após essa função, duas funções de busca são construídas, “getSocietyByName” e “getSocietyByAddress”. Ambas retornam o índice de uma entidade no array “societies”, a primeira com base no nome da entidade e a segunda com base no endereço da sociedade na rede Rinkeby.

```
function getRequestsLength() public view returns(uint) {
    return requests.length;
}

function getSocietiesLength() public view returns(uint) {
    return societies.length;
}

function getFin(string society_name) public view returns(uint[]) {
    Society memory society = societies[getSocietyByName(society_name)];
    return society.finance_series;
}

function getRev(string society_name) public view returns(uint[]){
    Society memory society = societies[getSocietyByName(society_name)];
    return society.revenue_series;
}
```

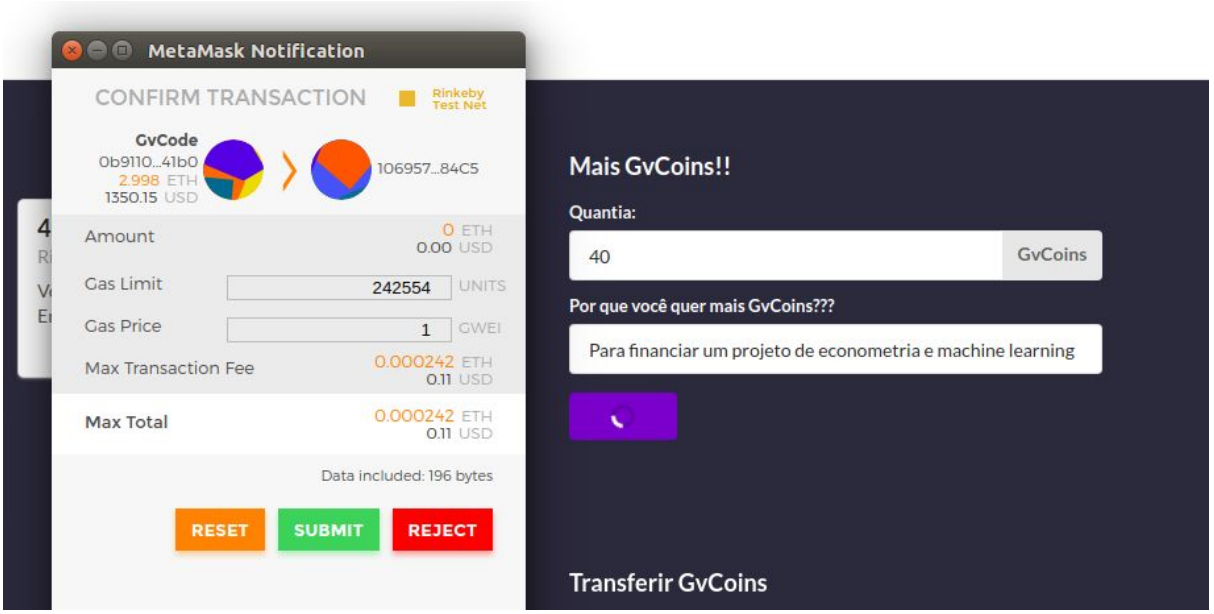
As últimas funções do contrato apenas retornam dados. As funções “getRequestsLength” e “getSocietiesLength” retornam os tamanhos dos arrays “societies” e “requests”, enquanto as funções “getFin” e “getRev” retornam as os arrays “finance_series” e “revenue_series”. Caso o sistema precise retornam uma sociedade ou requisição específica - bem como suas propriedades - é possível utilizar funções criadas automaticamente pelo contrato - os chamados “getters”. As últimas duas funções do contrato são necessários porque funções Solidity não retornam arrays dinâmicos, isto é, que não possuem predefinido.
O contrato apresentado foi enviado à rede Rinkeby, e pode ser utilizado em produção - com ressalva para a função “destroyGvCoins”. Todavia, recomenda-se que uma arquitetura mais sofisticada de “meta-contratos” seja utilizada para resolver o problema de modificações no contrato original. Cada vez que os desenvolvedores realizam um “deploy” do contrato à rede Ethereum, uma nova instância do contrato é criado em um conta completamente independente da conta relativa ao contrato anterior. Para não haver perda de dados, é possível construir uma arquitetura a partir da qual um contrato principal - que, em tese, não será alterado - realize chamadas a outros contratos contendo lógicas que podem ser alteradas ao longo do tempo.

O estudo do contrato acima revela o funcionamento adotado para o sistema: uma “Society” é criada pelo administrador para representar uma sociedade da organização, e cada sociedade envia requisições de emissão de GvCoins para financiar suas atividades dentro da organização. Como o sistema GvCoin foi construído com a finalidade de engajamento dentro da FGV, não há conversão de GvCoins para moedas tradicionais, e, portanto, uma sociedade estudantil da FGV pode comprar serviços apenas de outras sociedades e da própria FGV com GvCoins. A seguir, apresentamos um exemplo de uso do sistema:

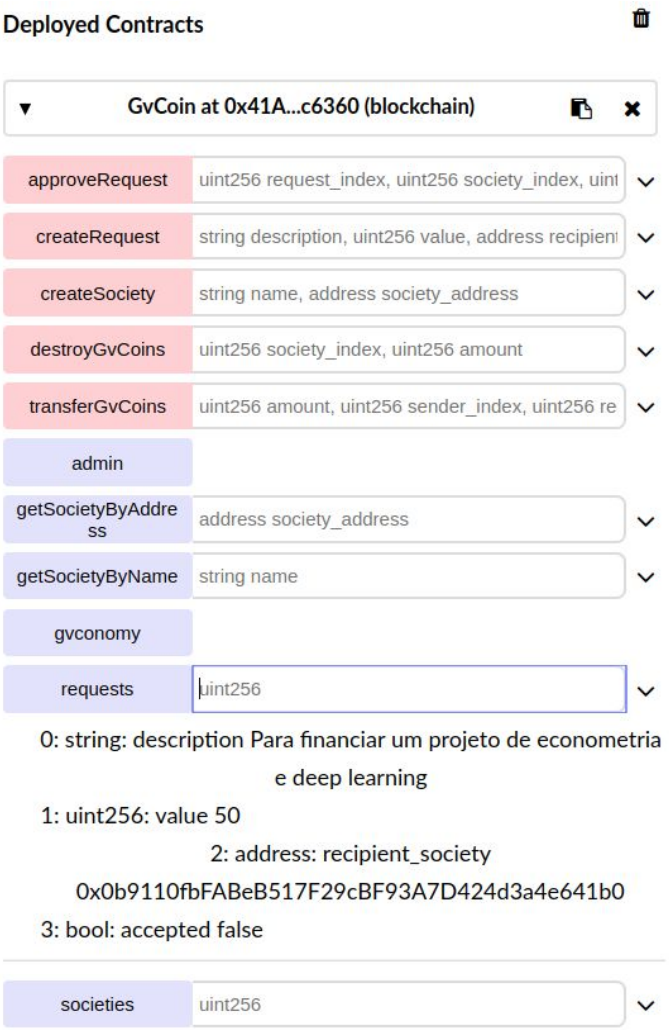


Na tela da sociedade estudantil, o usuário encontra informações sobre sua conta, estatísticas da sociedade e da “gvconomy”, além dos campos de requisição e transferência de GvCoins.

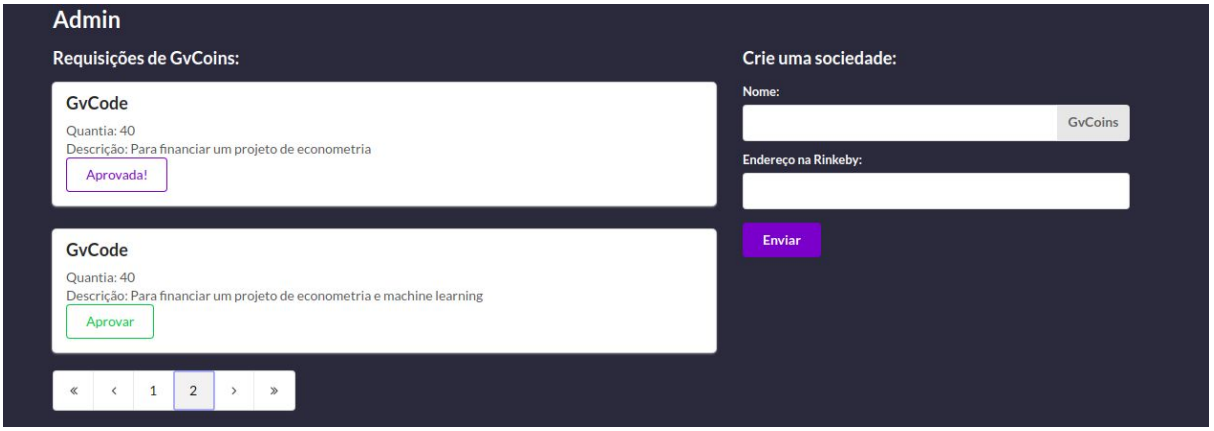
No campo “Mais GvCoins!!”, o usuário informa a quantia de GvCoins desejada e o motivo pelo qual necessita de mais GvCoins. É através desse mecanismo que o administrador do sistema pode direcionar o engajamento - por exemplo, aprovando requisições com valores maiores para projetos científicos e tecnológicos.



Ao clicar no botão “Enviar”, uma notificação do plugin “Metamask” será exibida ao usuário, informando o custo da transação. O usuário deve, então, confirmar a transação selecionando a opção “submit”, e, após o tempo de processamento da transação na blockchain da rede Rinkeby, a transação será finalizada.
O administrador pode, então, aprovar a requisição por meio do sistema ou do ambiente “Remix”, utilizado para testes de contratos e interação com contratos enviados às redes de Ethereum.

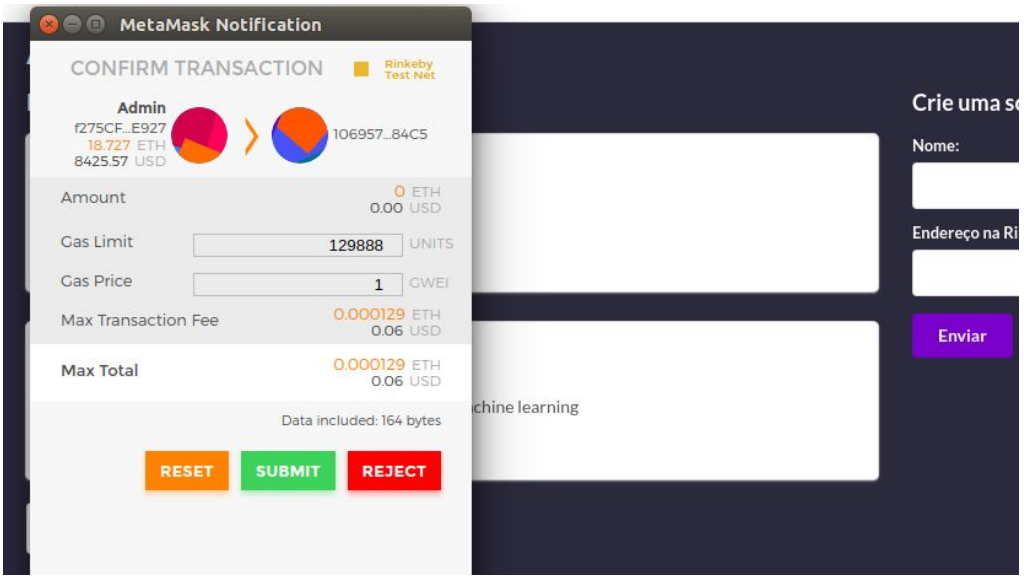


Através da ferramenta “Remix”, o administrador pode aprovar a requisição, e os valores da riqueza da sociedade - variável “wealth” - e da quantidade total de GvCoins - variável “gvconomy” - serão atualizados no sistema. O administrador também pode aprovar requisições a partir de sua tela no sistema da GvCoin, como indica a figura abaixo.

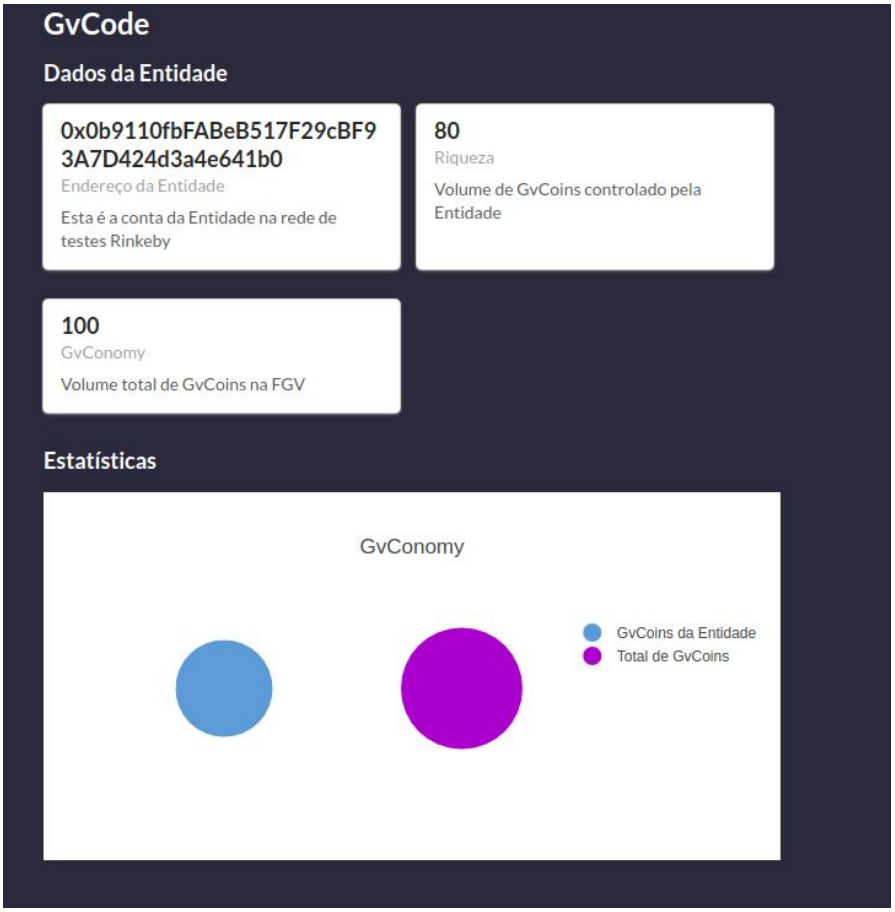


A aprovação de uma requisição também deve ser feita através de uma transação, com o

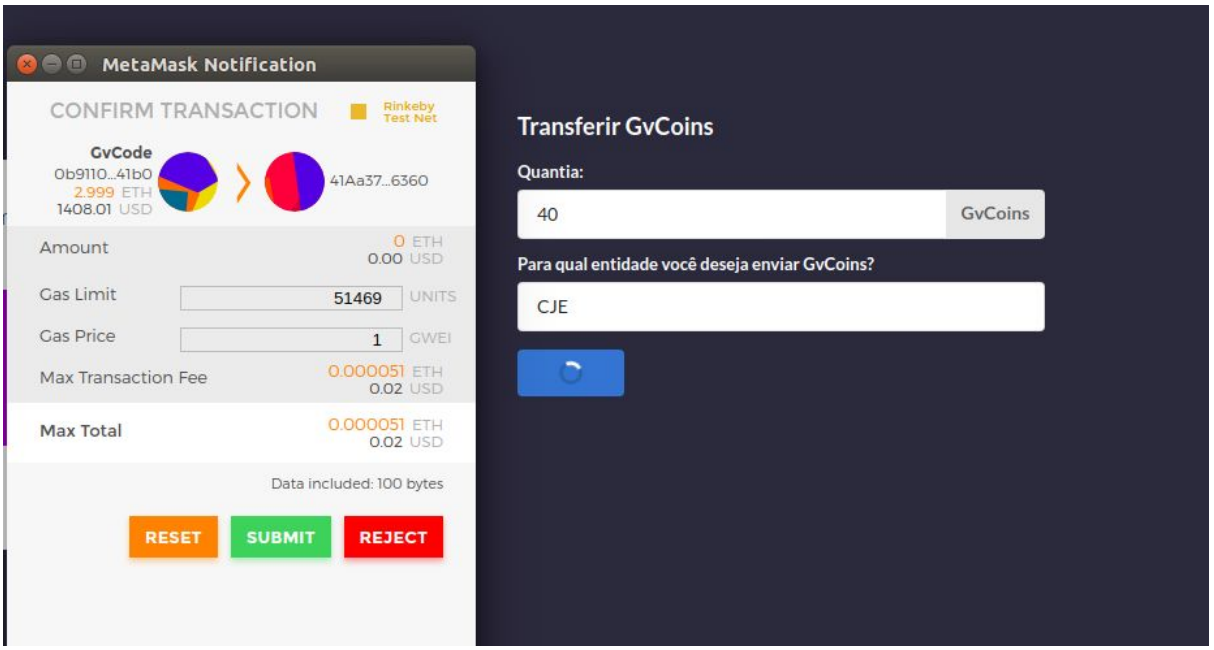
plugin “Metamask”.



Após a confirmação da transação pelos “mineradores” da rede Ethereum, os valores atualizados estarão disponíveis na tela da Sociedade.

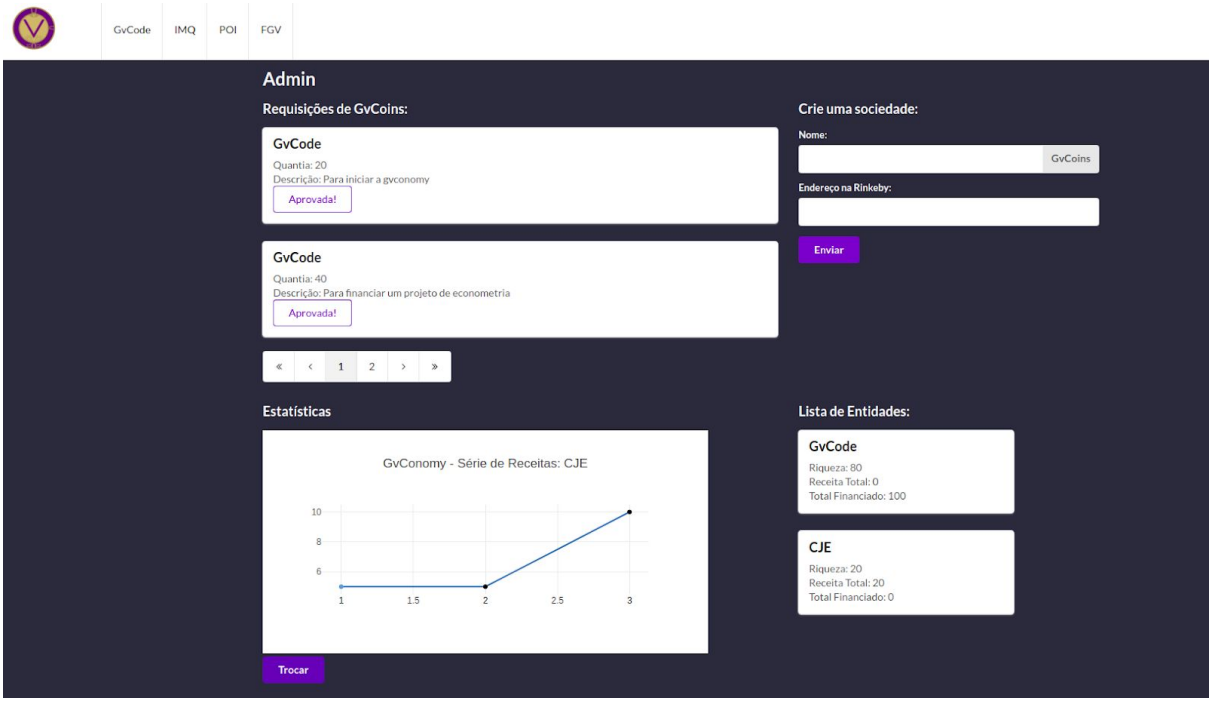


Um procedimento semelhante é realizado para a transferência de GvCoins entre as sociedades, mas sem a necessidade de aprovação pelo administrador.



O sistema permite que o administrador consulte estatísticas de cada sociedade. A figura abaixo mostra um exemplo: o administrador pode consultar a “Série de Receitas” de uma socieade.

No canto superior direito da tela do administrador, há uma opção para criar uma nova sociedade. Neste projeto não foi implementado nenhum mecanismo de envio de solicitações para a criação de uma nova sociedade, pois supõe-se que haja mecanismos suficientemente adequados para a comunicação dos diversos grupos dentro da FGV, como e-mails e grupos em redes sociais. Em universidades com diversos campi, um sistema de solicitação para uma plataforma análoga à GvCoin pode ser útil, todavia.



Tela do administrador

Como a figura abaixo indica, todas as transações serão guardadas na blockchain da rede Rinkeby.

TxHash	Block	Age	From		To	Value	[TxFee]
0x25b33bcad58263…	2712149	39 secs ago	0xf275cfbbf04efc32…	IN	0x41aa37ea9cfae66…	0 Ether	0.000039245
0x5a5843797d6951…	2712133	4 mins ago	0x0b9110fbfabeb51…	IN	0x41aa37ea9cfae66…	0 Ether	0.000134383
0xf4aa1cbbb8c6179…	2706345	1 day 11 mins ago	0x0b9110fbfabeb51…	IN	0x41aa37ea9cfae66…	0 Ether	0.000034313
0x192a07852aa3cfa…	2706023	1 day 1 hr ago	0x0b9110fbfabeb51…	IN	0x41aa37ea9cfae66…	0 Ether	0.000098626
0x1aedc9c14b11bb…	2705928	1 day 1 hr ago	0xf275cfbbf04efc32…	IN	0x41aa37ea9cfae66…	0 Ether	0.000138362
0x38658ba6d5fee57…	2705917	1 day 1 hr ago	0x0b9110fbfabeb51…	IN	0x41aa37ea9cfae66…	0 Ether	0.00021393
0x585894b78933a8…	2705886	1 day 2 hrs ago	0xf275cfbbf04efc32…	IN	0x41aa37ea9cfae66…	0 Ether	0.00023297

Podemos, então, verificar os detalhes de cada transação. Na figura abaixo, vemos um exemplo de uma transação realizada no contrato da GvCoin. Note que o campo “value” da transação indica 0 Ether. O sistema da GvCoin não abrange transferências de outros ativos que não GvCoins, logo, o único gasto de Ethers pelos usuários é indicado pelo campo “Actual Tx Cost/Fee”, que é a taxa paga em Ethers pelo processamento computacional necessário para a mineração e gravação da transação na blockchain.

[This is a Rinkeby **Testnet** Transaction Only]

TxHash:	0x13923ad1bb576161209589ace1dc9355991929a3913733db96563e4de83ec20c
TxReceipt Status:	Success
Block Height:	2701546 (10637 block confirmations)
TimeStamp:	1 day 20 hrs ago (Jul-26-2018 08:21:36 AM +UTC)
From:	0xf275cfbbf04efc3228b5b0a6851962cf3d9ce927
To:	Contract 0x67003ae3e2c33b68d6c8a5947131b569af8a7916
Value:	0 Ether ($0.00)
Gas Limit:	70353
Gas Used By Txn:	70353
Gas Price:	0.000000001 Ether (1 Gwei)
Actual Tx Cost/Fee:	0.000070353 Ether ($0.000000)

Apesar de a implementação em Ethereum da GvCoin modelar apenas sociedades estudantis, o Contrato pode ser alterado para que estudantes individuais também possam utilizar o sistema. Um estudante que deseje realizar uma pesquisa ou desenvolver um produto poderá utilizar GvCoins para comprar serviços das sociedades estudantis, por exemplo.

Além das tecnologias de Ethereum, utilizou-se as tecnologias NodeJs como servidor da aplicação, as bibliotecas NextJs, React e Semantic-UI-Reatic para o desenvolvimento visual e renderização em servidor da aplicação. A figura ao lado representa a estrutura do projeto GvCoin, relevante do ponto de vista de arquitetura e engenharia de software. Na pasta “components”, localizam-se componentes visuais que serão utilizados pelas telas principais da aplicação. Na pasta “ethereum” estão os componentes diretamente relacionados às tecnologias de Ethereum, como o arquivo contendo os “bytecodes” representando o contrato GvCoin, e que serão lidos pela EVM. O arquivo “GvCoin.sol” é o código em Solidity analisado acima, e os arquivos “compile.js”, “deploy.js”, “gvcoin.js” e “web3.js” são módulos de suporte à comunicação da aplicação com a rede Ethereum.

Na pasta “pages” estão os arquivos relativos às páginas principais da aplicação - a tela principal, e as telas do administrador e da sociedade. Na pasta “static” estão as imagens utilizadas na aplicação e, na pasta principal, arquivos de configurações da aplicação que não são diretamente relacionados a Ethereum.

O sistema da GvCoin pode ser acessado no endereço http://gvcoin.tech/, com “tokens” de acesso “GvCode”, para o sistema de sociedades estudantis, e “IMQ”, para o sistema do administrador. O código do sistema está disponível em: https://github.com/yankaled/GvCoin.

5.3. Caso de uso e conclusão

O primeiro caso de uso da GvCoin envolveu duas entidades estudantis: O GvCode, cujos membros se especializam em ciências de dados, e a CJE, cujos membros se especializam em projetos de economia e finanças. A entidade CJE desejava otimizar seu website, e, para isso, contratou serviços do GvCode, cujos integrantes são treinados em programação de software. As GvCoins adquiridas pelo GvCode servirão ao grupo tanto como uma moeda de troca para adquirir serviços de outras entidades, quanto como uma métrica de quanto a entidade contribui com o ambiente da FGV.

O sistema da GvCoin pode ser replicado em qualquer universidade, com maior possibilidade de benefícios quanto maior for a cultura científica e tecnológica da instituição. Espera-se que esta pesquisa e o sistema construído contribuam com fomento de projetos universitários inovadores, e com a cultura de compartilhamento de recursos e conhecimento nas comunidades universitárias.

6. Agradecimentos

Além da orientação da pesquisa pelo professor e pesquisador Júlio Figueiredo, a GvCoin tem participação de outros agentes.

A equipe de arquitetura da GvCoin conta com os seguintes integrantes do grupo de estudos de ciências de dados GvCode: Ana Pessoa, Thiago Chiaro, Murilo Coelho e Gabriela Aboud. Membro do GvCode, Gustavo Grivol deu suporte à equipe de arquitetura em plataformas linux.

O professor e pesquisador Eduardo Diniz, o graduando e pesquisador Filipe Oliveira e o tecnologista André Salem contribuíram com a idealização econômica da GvCoin.

7. Referências

NARAYANAN, A.; BONNEAU, J.; FELTEN, E.; MILER, A.; GOLDFEDER, S.; CLARK, J. **Bitcoin and Cryptocurrency Technologies. Draft,** 2016, p. 4-13.

ANTONOPOULOS, A. M. **Mastering Bitcoin.** 6ª edição, O'Reilly Media, In, 2014.

SANG, B. **LearnCoin: An Educational Implementation of Cryptocurrencies.**

NAKAMOTO, S. **Bitcoin: A Peer-to-Peer Electronic Cash System,** 2008**.**

EDX, THE LINUX FOUNDATION **Blockchain for Business: An Introduction to Hyperledger Technologies**

SAKHUJA, R., **Blockchain Development on Hyperledger Fabric using Composer**,

Udemy

GRIDER, S. **Ethereum and Solidity: The Complete Developers Guide**, Udemy

DEOL, R.; WIESNER, T. **Ethereum Blockchain Developer: Build Projects Using Solidity**, Udemy

THE LINUX FOUNDATION **Hyperledger Architecture, Volume 1**, 2017.

COINDESK **Understanding Ethereum**, 2016.

WILF, HERBEERT S. **Algorithms and Complexity**, 2ª edição, A K Peters, 2002

SWAN, M. **Blockchain, Blueprint For a New Economy**, 1ª edição, 2015, p. 36.

CROSBY, M.; NACHIAPPAN; PATTANAYAK, P.; VERMA, S.; KALYANARAMAN, V. **BlockChain Technology, Beyond Bitcoin, Sutardja Center for Entrepreneurship & Technology Technical Report**, 2015.

BUTERIN, V. **Ethereum White Paper**, 2013

WOOD, G. **Ethereum: A Secure Decentralised Generalised Transaction Ledger**, 2014

BUTERIN, V. **Ethereum 2.0 Mauve Paper,** 2016.

NICHOLSON, W; SNYDER, C. **Microeconomic Theory**, 10ª edição, Thomson South-Western

VARIAN, H. R. **Intermediate Microeconomics**, 8ª edição, W. W. Norton & Company