Blockchain

Uma visualização gráfica

Paulo Jerônimo

Version 1.0, 2017-07-29 02:17:24 -03

Conteúdo

Introdução

Este projeto provê uma aplicação web, interativa, para demonstrar conceitos básicos e fundamentais de uma Blockchain (ou, traduzindo ao pé da letra para o português, corrente de blocos).

O código fonte deste projeto (incluindo este documento) é aberto e está disponível.

Você pode visualizar uma versão online deste documento acessando o endereço http://blockchain4devs.github.io/blockchain-demo/docs/pt-br.

A demonstração de uso da aplicação deste projeto pode ser visualizada em duas (2) versões:

1. Blockchain 101 - Demo: Vídeo original (em inglês) (criado por Anders Brownworth).
2. Blockchain: uma visualização gráfica: Playlist com 10 vídeos, em português do Brasil, criados por Paulo Jerônimo para a organização blockchain4devs. `Plus`: Blockchain: hangout (apresentação), sobre esse assunto, com o DFJUG.

Através dos vídeos acima é feita uma apresentação, visual, que consideramos ser bastante elucidante nos conceitos elementares de uma Blockchain. A aplicação apresentada introduz o conceito de um livro-razão (distribuído). Ela está em execução, neste momento, no seguinte endereço:

- http://anders.com/blockchain/

Você também pode fazer o download deste projeto a partir do GitHub e fazer a execução dessa aplicação, localmente, em teu ambiente. Isso é explicado logo a seguir, no primeiro tópico desse documento.

1. Instalação local

1.1. Direta, utilizando o teu próprio sistema operacional (SO)

Baixe o código:

```
git clone https://github.com/anders94/blockchain-demo
```

Para rodar essa aplicação, você deverá ter instalado o Node.js. Em seguida, instale as dependências dessa aplicação através do npm.

A instalação do comando npm pode ser executada, no macOS, através de um único comando: `brew install npm`. Isso é tudo que é necessário fazer neste sistema operacional.

```
cd blockchain-demo
npm install
```

Execute o servidor:

```
./bin/www
```

Abra o teu browser e acesse a URL da página de demonstração: http://localhost:3000.

1.2. Utilizando o Docker

Baixe o código:

```
git clone https://github.com/anders94/blockchain-demo
```

Faça a construção do contêiner Docker:

```
cd blockchain-demo
docker-compose up -d
```

Abra o teu browser e acesse a URL da página de demonstração: http://localhost:3000.

Se você estiver utilizando outro SO que não o Linux (um macOS por exemplo) antes de executar o `docker-compose`, será necessário a instalação e a configuração do docker-machine. Além disso, a URL que você acessará no browser não estará em `localhost`. Nesse caso, siga os passos no subtópico a seguir. Ele demonstra como utilizar o `docker-machine` para criar uma máquina virtual utilizando o VirtualBox.

1.2.1. Numa máquina virtual (criada no VirtualBox e gerenciada pelo docker-machine)

```
docker-machine create --driver virtualbox default
eval "$(docker-machine env default)"
```

Abra o teu browser e acesse a URL da página de demonstração que devera ser `http://<ip-no-docker-machine>:3000`.

O comando `docker-machine ip` pode ser chamada para determinar o IP da máquina virtual executando o docker. Então, no macOS, o seguinte comando pode ser utilizado para abrir a browser na URL da demonstração:

```
open http://$(docker-machine ip):3000
```

2. Demonstração

2.1. O que é um Hash?

Hashes são parte de várias informações armazenadas na Blockchain e são essenciais para garantir sua segurança.

Matematicamente falando, uma função hash é um algoritmo que mapeia dados de comprimento variável para dados de comprimento fixo. Os valores retornados por uma função hash são chamados valores hash, códigos hash, somas hash, checksums ou, simplesmente, hashes.

Nessa demonstração, a aba Hash possibilita a entrada de uma informação de tamanho qualquer (no campo `Data`) e produz um `Hash` de tamanho fixo. A função SHA256 é uma das variantes do SHA-2 e utilizada para gerar esse código hash. Esse código é geralmente representado por uma string de 64 caracteres hexadecimais (de "0" a "9" e de "a" a "f"). Obviamente, além dessa função, existem várias outras que poderiam ser utilizadas para gerar uma string de tamanho fixo a partir de uma quantidade de dados variáveis. Outros exemplos de funções (dentre várias) são o MD5 e o SHA-1.

Ao se gerar um hash, espera-se que a cadeia fixa produzida a partir de um conjunto de informações tenha sempre um valor diferente para outro conjunto de entrada. Se o hash gerado num conjunto de dados for igual ao de outro, ocorre o que chamamos de colisão. Quanto menos colisões houverem ao se gerar um hash para um conjunto de informações diferentes, melhor o algoritmo de Hash.

Foram necessários 20 anos para que fosse anunciada a primeira colisão de hash para a função SHA-1. Isso quer dizer que esse algoritmo foi considerado seguro e utilizado por todo esse espaço de tempo, sem problemas. O site https://shattered.it/ demonstra que, para se proferir um ataque onde se tentaria obter um hash igual para um `input` diferente utilizando o algoritmo SHA-1, seriam necessárias 9.223.372.036.854.775.808 de execuções do algoritmo. Ou seja, um número monstruoso de compressões. Para ser realizado esse número de compressões, seriam necessários 6.500 anos de processamento de uma CPU comum (ou 110 anos de uma GPU).

Uma colisão de SHA-1 pode ser visualizada, como na execução/resultado dos comandos apresentados a seguir:

Essa execução/resultado de comandos é feita num shell (Bash) de um macOS. Mas, ela também pode ser adaptada e realizada em outros sistemas operacionais.

```
$ for n in 1 2; do curl -s -O https://shattered.it/static/shattered-$n.pdf; done
$ open shattered-*.pdf
$ diff shattered-*.pdf
Binary files shattered-1.pdf and shattered-2.pdf differ
$ shasum shattered-*.pdf
38762cf7f55934b34d179ae6a4c80cadccbb7f0a  shattered-1.pdf
38762cf7f55934b34d179ae6a4c80cadccbb7f0a  shattered-2.pdf
```

O código JavaScript da aba Hash (`views/hash.jade`) é um código escrito em Jade. Ele calcula o hash

através de um código (em JavaScript) que executa a função sha256.

public/javascripts/lib/sha256.js

```
function sha256(block, chain) {
  // calculate a SHA256 hash of the contents of the block
  return CryptoJS.SHA256(getText(block, chain));
}
```

Obviamente, a função hash implementada em JavaScript também deve produzir o mesmo resultado que uma função que pode ser chamada via linha de comando. Sendo assim, também é possível gerar o hash de uma string informada no quadro `Data` através de uma linha de comando (em Bash, por exemplo).

O Hash para a string vazia (valor inicial do quadro), pode ser calculado assim:

```
echo -n ''|shasum -a 256
```

Então, o hash de "Paulo Jerônimo" poderia ser calculado assim:

```
echo -n 'Paulo Jerônimo'|shasum -a 256
```

Para se aprofundar um pouco mais em funções hash aplicadas a moedas digitais e saber quando colisões em valores de hash tem importância nesse contexto, recomenda-se a leitura do artigo "Lessons From The History Of Attacks On Secure Hash Functions".

2.2. O que é um Bloco?

Vamos agora para a aba Block. Adicionaremos, acima do campo `Data` aprensentado na aba `Hash`, dois outros campos: `Block` e `Nonce`. Eles serão explicados agora. Também vamos criar um botão `Miner` e explicar qual será sua utilidade. Com a adição desses novos elementos, estamos criando o que chamaremos de **Bloco**.

Um bloco, então, tem um número identificador (`Block`), um `Nonce`, e pode ser "minerado" quado clicarmos no botão `Miner`. Também notemos que o campo `Hash` agora possui um valor interessante, iniciado por quatro zeros ("0000"). Esse valor de `Hash` é bastante singular. Ele é criado através de uma **regra que é utilizada para verificar se o bloco é válido**. Nesse nosso caso, a regra é bem simples: o valor do `Hash`, calculado em função do valor inserido em `Data` (e dos outros campos), precisará começar com esses quatro zeros.

Para encontar um `Hash` que safisfaça essa regra, após inserir os dados que desejamos no campo `Data`, iremos alterar o valor de `Nonce`. O número identificador do bloco (`Block`) será gerado automaticamente (apesar de podermos modificá-lo, para testes, nesse exemplo). Esse cáculo do `Hash` será repetidamente refeito, até encontarmos um `Hash` que comece com esses quatro zeros.

Você pode notar também que, qualquer mudança nos campos (`Block`, `Nonce` e `Data`) altera o valor de `Hash`.

Manualmente, poderíamos ficar tentando modificar o valor de `Nonce` até encontarmos a solução para esse problema. Mas, fazendo isso, poderemos passar vários minutos ou horas (ou mesmo dias) em tentativas. Então, o botão `Miner` servirá para realizar essa tarefa pra nós.

A atividade de "mineração", então, é nesse nosso caso uma tarefa que será executada pelo computador. Ela será realizada em função do valor de todos os campos e buscará um `Hash` que atenda a regra estabelecida: começar com quatro zeros. Essa regra também é conhecida por "desafio matemático". Você pode notar que esse desafio (no nosso caso e em várias outras formas de propô-lo) não é uma função complexa. Por outro lado, ela é onerosa pois pode cosumir muito tempo de processamento. O trabalho de "mineração" realizado dessa forma é também conhecido por "Proof of Work".

Após um bloco ser "minerado", a conferência de sua validade é algo extremamente simples e rápido. Essa conferência é apenas o cálculo do `Hash` de todos os campos (`Block` + `Nonce` + `Data`) e a obediência a regra estabelecida (em nosso caso, um `Hash` começando com quatro zeros iniciais).

2.3. O que é uma Blockchain?

2.3.1. Uma visualização gráfica

Finalmente, vamos à aba Blockchain para entender o que é esse conceito genial, numa visualização gráfica!

Nessa aba, apenas para fins de demonstração, apresentamos cinco (5) blocos encadeados. Observe que, agora, foi adicionado o campo `Prev`. Esse campo também é um `Hash`. No primeiro bloco, seu valor é nulo. Para todos os demais blocos, o campo `Prev` armazena o valor do `Hash` do bloco anterior. Assim, agora há um encademento que forma o que chamamos de "corrente de blocos". Ou, o nome bonito em inglês: Blockchain.

O objetivo desse encadeamento entre os blocos é muito simples. Se você fizer uma alteração em qualquer campo do bloco o seu `Hash` será alterado. Consequentemente, todos os blocos posteriores a essa alteração também não serão mais considerados válidos. O motivo para isso é que, a partir do momento em que é modificado o `Hash` de um bloco, o valor de `Prev`, registrado no bloco posterior, não coincidirá mais. Logo, uma Blockchain considerará todos os blocos inválidos a partir de algum que tenha sido adulterado em qualquer um de seus campos.

Para fazer uma Blockchain adulterada ser novamente considerada válida, alguns passos seriam necessários. Por exemplo, vamos supor que estamos fazendo uma alteração de dados apenas no último bloco. Nesse caso, tornar a Blockchain válida é algo simples: basta recalcular o `Hash` desse bloco executando a operação `Miner`. Ou seja, executar o processo de mineração.

O problema real surge ao ser feita uma tentativa de alterar um bloco que não é o último! Vamos ver também que a dificuldade de se alterar um bloco é gradativamente aumentada quando ele vai se aproximando do primeiro bloco (ou se distanciando do último). Suponhamos, por exemplo, que alguém tentasse fazer a adulteração de um dado no bloco 3. Para que isso fosse possível, a operação de mineração para todos deveria ser novamente realizada a partir desse bloco. Assim, o custo de tornar um bloco válido vai aumentando exponencialmente a medida em que ele se aproxima do primeiro bloco.

Podemos observar, então, que uma Blockchain é uma estrutura (ou um banco) de dados que rejeita modificações que não sejam no último bloco. Poderíamos fazer uma comparação com um livro onde só é possível se escrever na última página. E, continuando nessa linha, esse livro só poderia ser escrito a caneta. Por fim, qualquer tentativa de se modificar um dado registrado nesse livro o deixaria manchado (ou rasurado) para sempre.

Uma Blockchain é citada por muitos como sendo "o protocolo da confiança". Um dos motivos é decorrente do fato de somente ser possível se adicionar informações. Não são permitidas alterações de informação, pois, pelo que vimos, a rede rejeita mudanças através de seu protocolo. Finalmente, é importante entender que uma Blockchain é uma estrutura de dados que fica "na mão de muitos". Ou seja, é descentralizada. Vamos agora entender como uma Blockchain é considerada ainda mais segura quando ela é distribuída entre vários nós de uma rede.

2.3.2. A Blockchain é distribuída e descentralizada

Vimos que uma Blockchain é uma estrutura de dados confiável que só aceita inserções de dados. Ou seja, ela é resistente a mudanças. Mas, além disso, uma Blockchain é também uma rede de computadores onde essa estrutura de dados é replicada. Dessa forma, sua estrutura de dados existe, distribuída, em cada um dos computadores (nós) dessa rede.

O objetivo principal da distribuição de uma Blockchain entre vários nós é impedir que mesmo sendo ela válida num único nó, ela só seja totalmente válida quando houver um consenso (na rede) de que suas informações também são válidas em todos os outros nós. Numa Blockchain, os nós pertencentes a sua rede são responsáveis por validar dados e retransmití-los de forma que eles sejam replicados em todos os nós. Mais especificamente, quando dados precisam ser inseridos num bloco, todos os nós devem verificar se esses dados são válidos. Caso não sejam, esses dados serão descartados e não retransmitidos aos próximos nós.

A validação de dados, obviamente, depende do que são esses dados. A primeira Blockchain foi construída para resolver um problema complexo: a tranferência de valores. Dando crédito: **os conceitos de Blockchain que estamos aprendendo surgiram para fazer a moeda digital Bitcoin entrar em funcionamento**. Perceba que só agora estamos falando de uma das várias possibilidades de aplicação da Blockchain. Apresentaremos, ainda, vários outros casos de uso de uma Blockchain.

O que diferencia a solução da Blockchain, dada para o Bitcoin, de soluções bancárias tradicionais que resolvem esse problema, entretanto, é a descentralização. Detalharemos mais sobre transações, logo a frente. Mas, por enquanto, é importante saber que uma Blockchain, por ser descentralizada, precisa criar algoritmos que promovam a manutenção de um consenso entre os nós.

É através de consenso que uma Blockchain identifica se uma cópia de seus dados, que está num nó, é realmente válida ou não. Acessando a aba Distributed podemos identificar que a alteração de uma das cópias da Blockchain num dos nós é possível de ser realizada. E, essa alteração pode, com certeza, tornar a cópia da Blockchain válida nesse nó. Como vimos, para isso ocorrer, seria necessário refazer os cálculos de `Hash` para cada bloco a partir do que for modificado. Mas, caso isso ocorra, outros nós da rede percebem a diferença e, automaticamente, eliminam a participação do nó que possui a Blockchain adulterada. Isso é feito através de consenso. A rede que está certa é a rede que detém a maioria do consenso (51%).

2.4. A Blockchain utilizada no controle de transações $

2.4.1. Tokens

Na aba Tokens visualizamos uma das possíveis aplicações da Blockchain. Ela pode ser utilizada para um sistema financeiro. Em essência, como já citado, a origem da Blockchain foi para tornar possível a existência do Bitcoin. Em inglês, o termo token é dado para uma coisa que esteja servindo como uma representação visível ou tangível de um fato, qualidade, sentimento, etc. Dessa forma o campo `Data` numa Blockchain pode representar um `token` que, nesse contexto, significa um registro de várias transações. Cada transação, por sua vez, registra que pessoa está enviando uma quantia (em moeda **$**) para outra pessoa.

Como já vimos, o registro de dados numa Blockchain é imutável. Logo, se esses dados forem transações, elas também serão imutáveis. E, num registro dessa espécie, é essencial essa imutabilidade pois nenhuma pessoa de boa fé gostaria de ver transações sendo adulteradas. A Blockchain resolve isso. Se tentarmos fazer uma alteração nos dados de qualquer transação poderemos notar que o bloco será invalidado.

Nessa demonstração, as transações são representadas de forma muito simplificada. Basicamente, cada transação possui apenas três (3) campos: quem está enviando um valor, quem está recebendo e, por fim, o valor. Fundamentalmente, para um sistema de transações, essas são as três informações chave. É muito importante que saibamos de onde uma quantia saiu e para onde ela foi.

É claro que apenas essas três informações não são suficientes para se contruir um sistema realmente utilizável. Afinal, as pessoas querem privacidade e anonimato por razões óbvia (de segurança). Ninguém utilizaria um sistema, como esse, onde estão públicas as informações do nome da pessoa e dos valores que ela movimenta. Então, num sistema como o Bitcoin, ao invés de utilizarmos nomes, utilizamos endereços.

Da mesma forma que não queremos adulterações nos dados, devemos evitar outras possibilidades de fraudes. Há um tipo de fraude que não seria possível de ser verificada numa Blockchain implementada da forma como estamos fazendo agora. Essa fraude seria uma pessoa tentar "gastar" o mesmo diheiro duas vezes. Isso é conhecido como "gasto duplo".

Exemplificando a possibilidade de gasto duplo através de um passo a passo:

1. Darcy tem 25 dólares e quer comprar algo de Bingley.
2. Ela passa seus 25 dólares para o Bingley.
3. Utilizando essa Blockchain, ela também tenta comprar algo de Jane, gastando mais 25 dólares.
4. O problema: será que Darcy tinha, realmente, 50 dólares? Ou será que ela está tentando gastar os mesmos 25 dólares que já possuia? Como nessa Blockchain não há o registro do saldo anterior de Darcy, isso poderia ser possível. Concorda?

Então, claramente, essa Blockchain precisa ser evoluída.

2.4.2. Transações Gênese

Como vimos uma Blockchain voltada ao controle de transações precisa manter um registro imutável dessas transações. Mas, além disso, em se tratando de transações, é essencial conseguirmos verificar se elas realmente são válidas. Num sistema financeiro, para que uma pessoa possa gastar algum dinheiro, ela precisa ter saldo, óbvio. Então, esse dinheiro deve ter vindo de alguém que, por sua vez, recebeu de outro alguém e assim suscessivamente. Voltando no tempo dessa forma, chegaremos até o ponto em que o dinheiro surgiu.

Poderíamos nos imaginar na figura de mineradores reais. Encontrando uma mina de ouro, utilizaríamos parte do ouro que minerássesmos para trocá-lo por coisas. Devemos nos lembrar que toda mina de ouro é finita. A moeda (o ouro nesse caso) é simplesmente algo que utilizamos para facilitar uma troca por algo que queremos. As pessoas que recebem esse ouro, por sua vez, poderiam dividí-lo em porções ainda menores para comprar coisas de outras pessoas. O Bitcoin funciona exatamente assim, como o ouro.

Vamos fazer uma representação sistêmica desse processo. Para entender como surge o dinheiro numa Blockchain, precisamos definir o conceito de transação gênese (ou transação coinbase).

Nessa demonstração, Anders Brownworth é o "descobridor de uma mina". Sendo assim, a primeira coisa que ele faz é criar para si um bloco gênese, de identificador 1. Esse bloco não contém nenhuma transação além de uma, especial, chamada de `Coinbase`. Essa transação especial é de "ninguém" para a pessoa que criou o bloco, no caso, um minerador. Nessa Blockchain, Anders resolve se dar 100 dólares no primeiro bloco (1).

- O bloco gênese na Blockchain do Bitcoin, criada por Satoshi Nakamoto tem o identificador 0.
- No caso do Bitcoin, o valor de Coinbase foi de 50 BTC quando a rede foi criada. Em seguida, caiu para 25 e, no ano passado, foi para 12.5 BTC. Esse valor sempre decairá.
- No Bitcoin, o valor de `Coinbase` é um valor que é reivindicado por um minerador como prêmio pela mineração do bloco.

No bloco 2, existem transações. O importante nessa implementação de Blockchain é que, agora, dá pra saber se uma transação é válida. Vejamos como.

No bloco 4, por exemplo, Sophia transfere para Jackson, 8 dólares. Será que Sophia realmente tinha 8 dólares? Para saber isso, vamos ao bloco anterior (3). Sophia não teve transações nesse bloco. Vamos ao anterior novamente (2). Nesse bloco, Sophia recebeu, de Anders, 10 dólares. Então, sem dúvida, Sophia pode gastar esse dinheiro.

Portanto, com essa nova implementação de Blockchain, agora conseguimos verificar se uma transação é válida.

3. Enviando um obrigado

Se você gostou deste projeto, incluindo seu texto e vídeos, e deseja enviar um "obrigado", saiba que Bitcoin e Ethereum são, agradecidamente, aceitos por seus desenvolvedores nos seguintes endereços:

- Anders Brownworth (desenvolvedor/criador deste projeto):
  - Bitcoin (BTC): `1K3NvcuZzVTueHW1qhkG2Cm3viRkh2EXJp`
- Paulo Jerônimo (desenvolvedor/criador deste documento e de vídeos em português)
  - Bitcoin (BTC): `1LTrDUdUw2zCS7LE93hDuvYiG326VnxL6k`
  - Ether (ETH): `0xc8780E07eE3C0f058315a20D2fD7dE6d2505f7a3`

Claramente, parte do valor depositado será revertido em melhorias neste projeto.

3.1. Melhorias que você pode aguardar neste projeto

Para o fork do projeto original são esperadas as seguintes evoluções:

1. Criação de legendas (*closed captions*) em inglês para os vídeos disponibibilizados em português (pt-br).
2. Internacionalização da aplicação para português.
3. Adição de uma cópia da aplicação (em execução) no endereço http://blockchain4devs.github.io/blockchain-demo.
   a. Geração de um site estático (para publicação no endereço acima sem a necessidade de um servidor executando o Node.js).
      i. Substituição do gerenciador de pacotes (de npm p/ yarn).
      ii. Utilização do Gulp para a construção desse site estático.
      iii. Substituição da camada de visualização (de Jade p/ Pug).
4. Alteração da página principal da aplicação, para que ela também apresente links para a visualização dos vídeos em português.
5. Criação de QR Code para os endereços (de Bitcoin e/ou Ethereum) publicados neste documento.

Por fim, espera-se que as adições realizadas neste fork sejam aceitas e incorporadas ao projeto original.