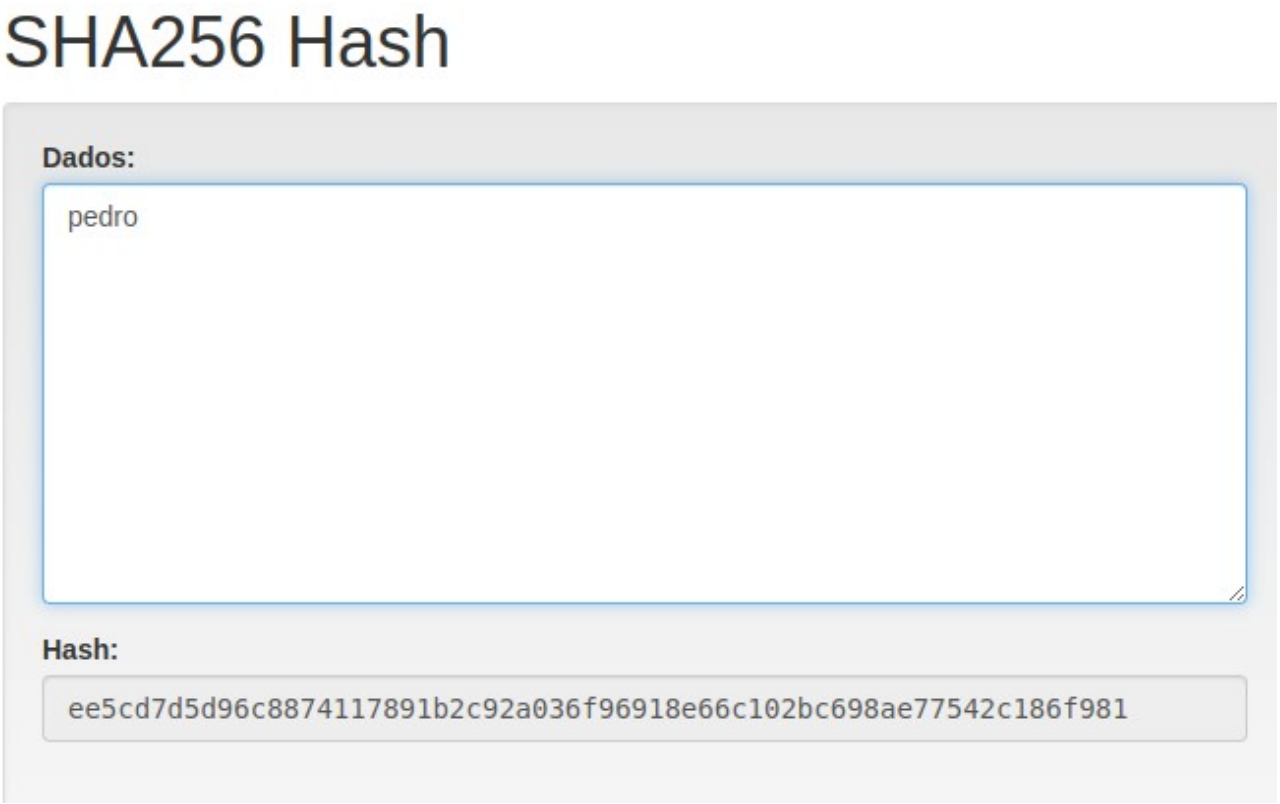
Relatório implementação Blockchain

Aluno: Pedro Arthur Cunha Anício

Relatório sobre vídeo Blockchain:

SHA256 Hash:

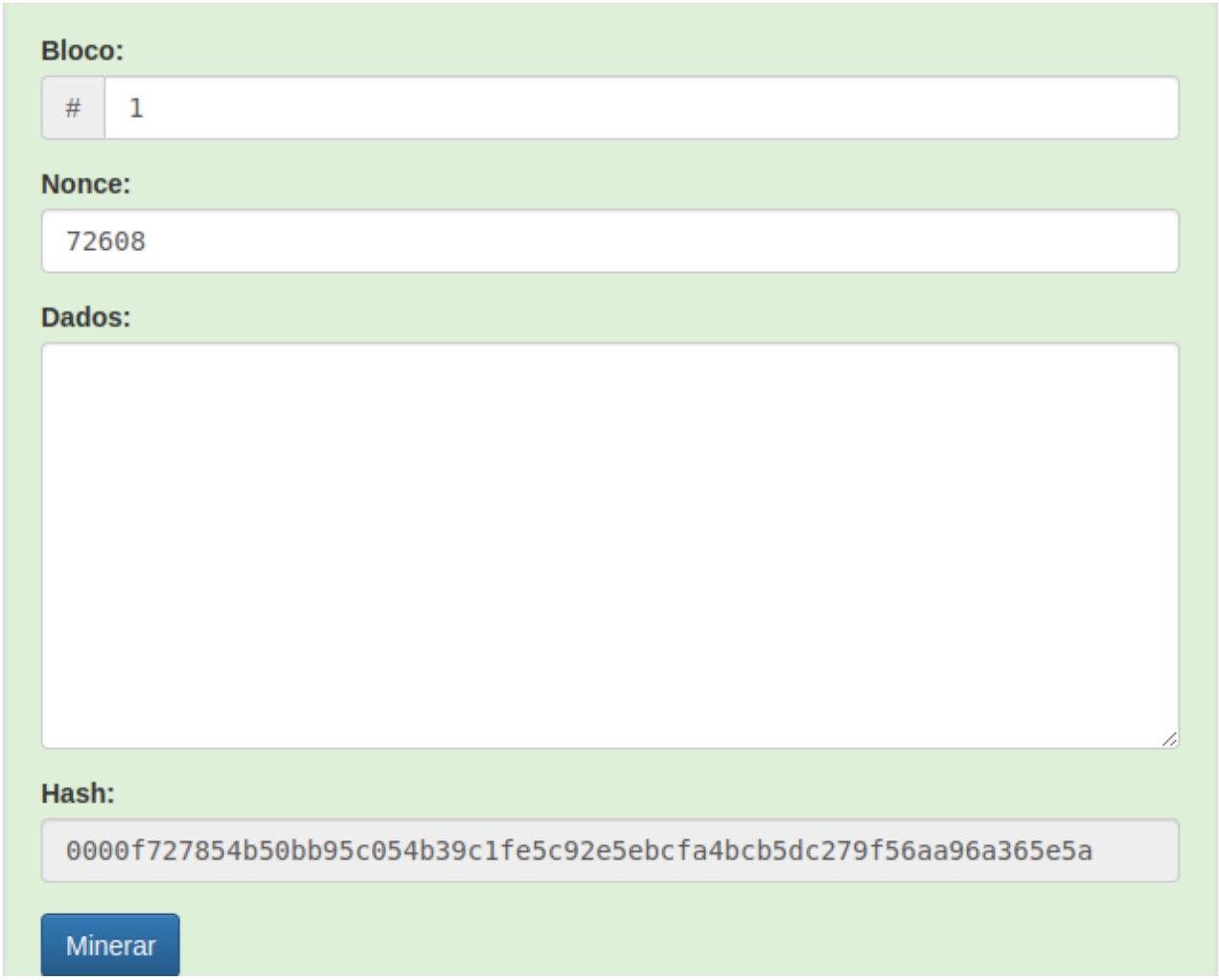
Um hash nada mais é do que uma sequência de números, que funciona como uma impressão digital de algum dado digital.



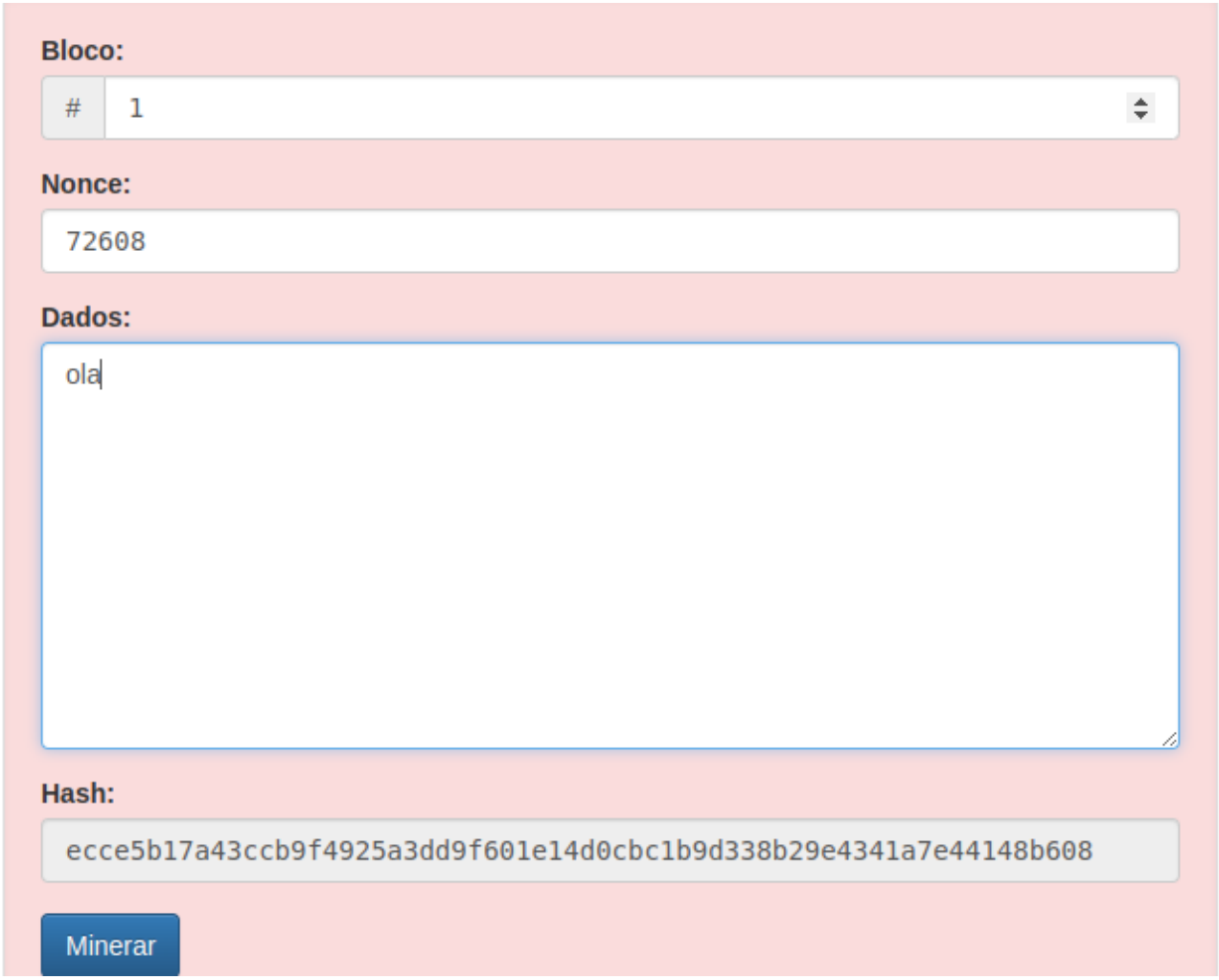
Meu nome (Pedro), tem como hash essa sequência de caracteres mostrada acima. Cada dado apresenta seu próprio hash, ou seja, sempre que for digitado a palava “pedro”, essa mesma sequência será apresentada. Essa é uma ótima forma de proteger um dado sensível.

Bloco:

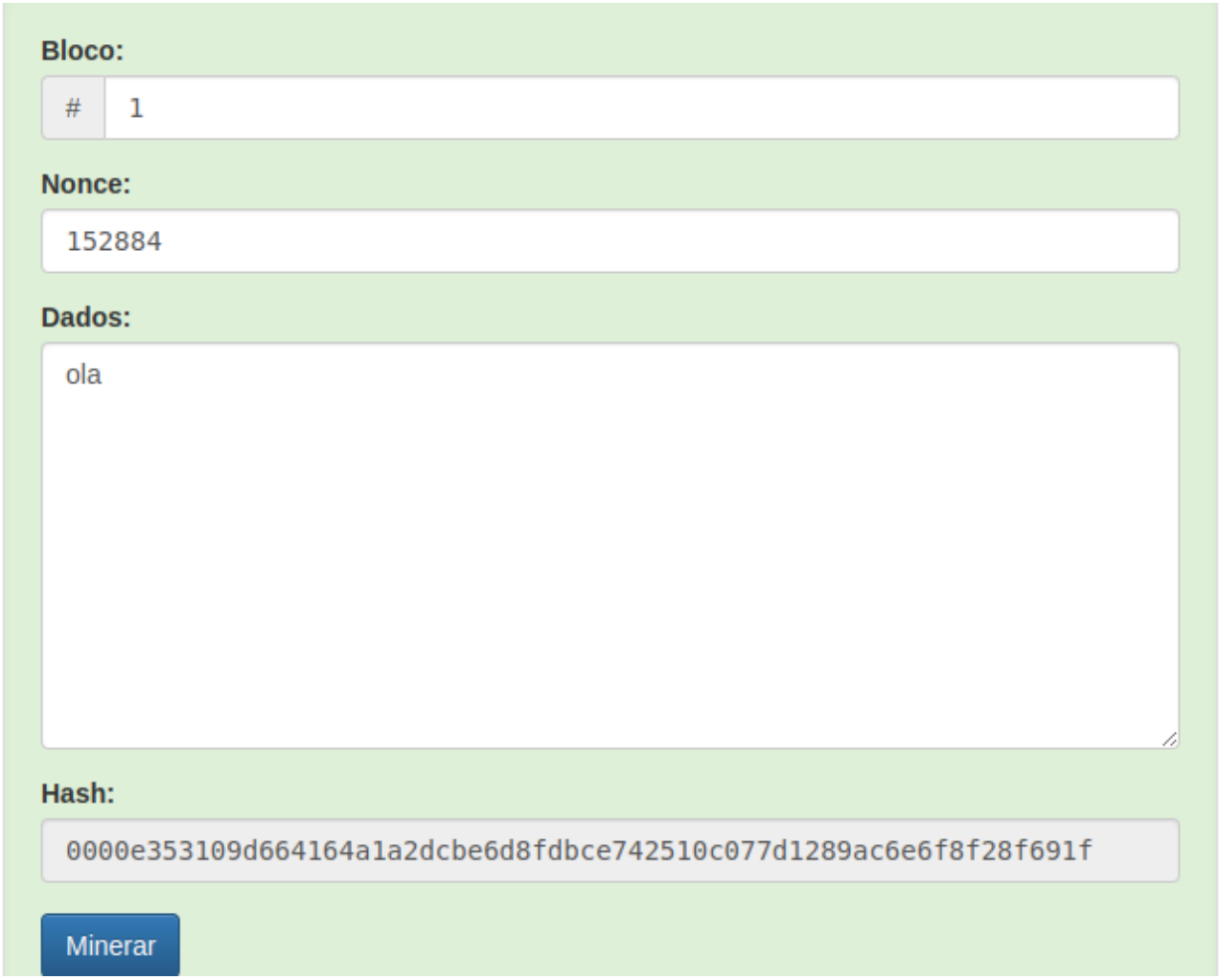
Um bloco apresenta quatro principais campos. O número do bloco, que funciona como um identificador, um Nonce, que vou explicar mais tarde sua função, os dados presentes naquele bloco, e por fim, o hash daquele bloco.



Analisando esse bloco, percebemos que seu hash começa com uma sequência de quatro números 0, o que podemos afirmar que esse bloco está assinado. Mas o que aconteceria se algum dado mudasse?

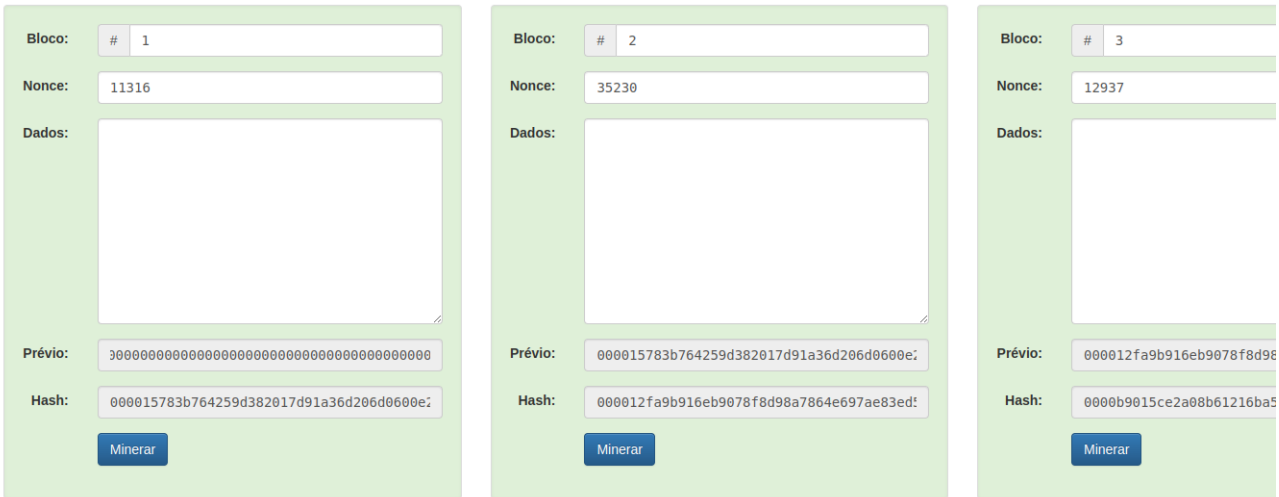


Ao alterar algum dado, o hash que antes começava com quatro zeros, não começa mais, perdendo assim a característica de estar “assinado”. Para resolver isso, o campo de Nonce tem como objetivo buscar algum valor que somado com os outros dados do bloco, façam com que os primeiros quatro dígitos do hash passem a ser números zero, e é exatamente isso que ocorre ao clicar no botão de minerar.



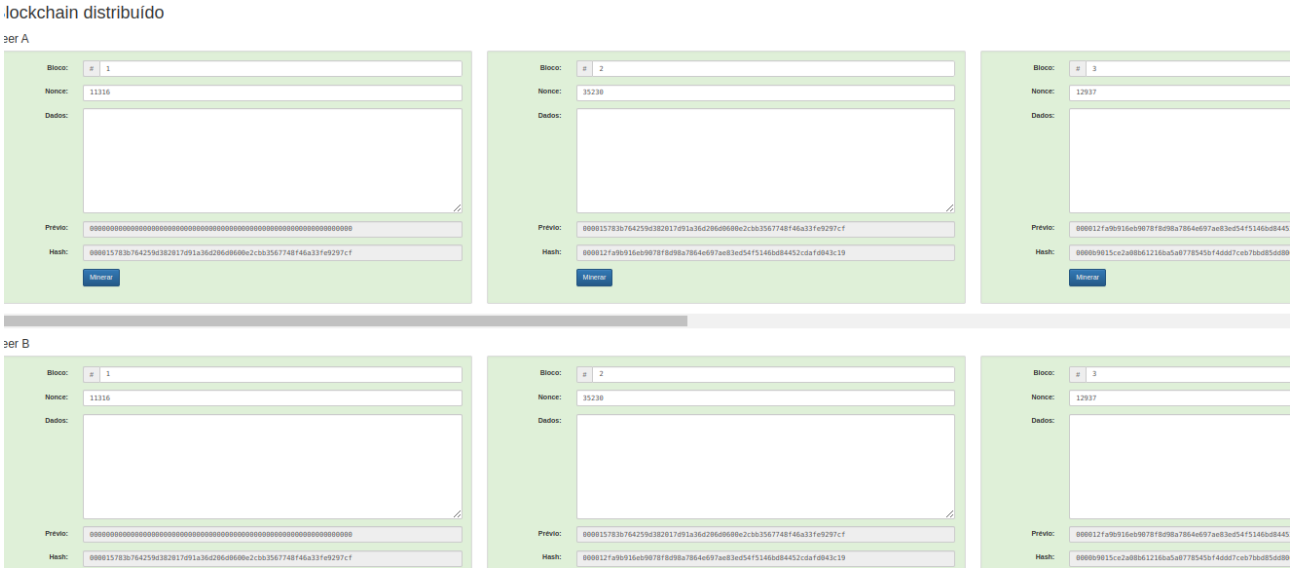
Blockchain:

Uma blockchain é uma cadeia de blocos, como aquele mostrado anteriormente. A Única diferença é que cada bloco possui um campo “prev”, que referencia o bloco anterior, por meio do hash daquele último bloco.



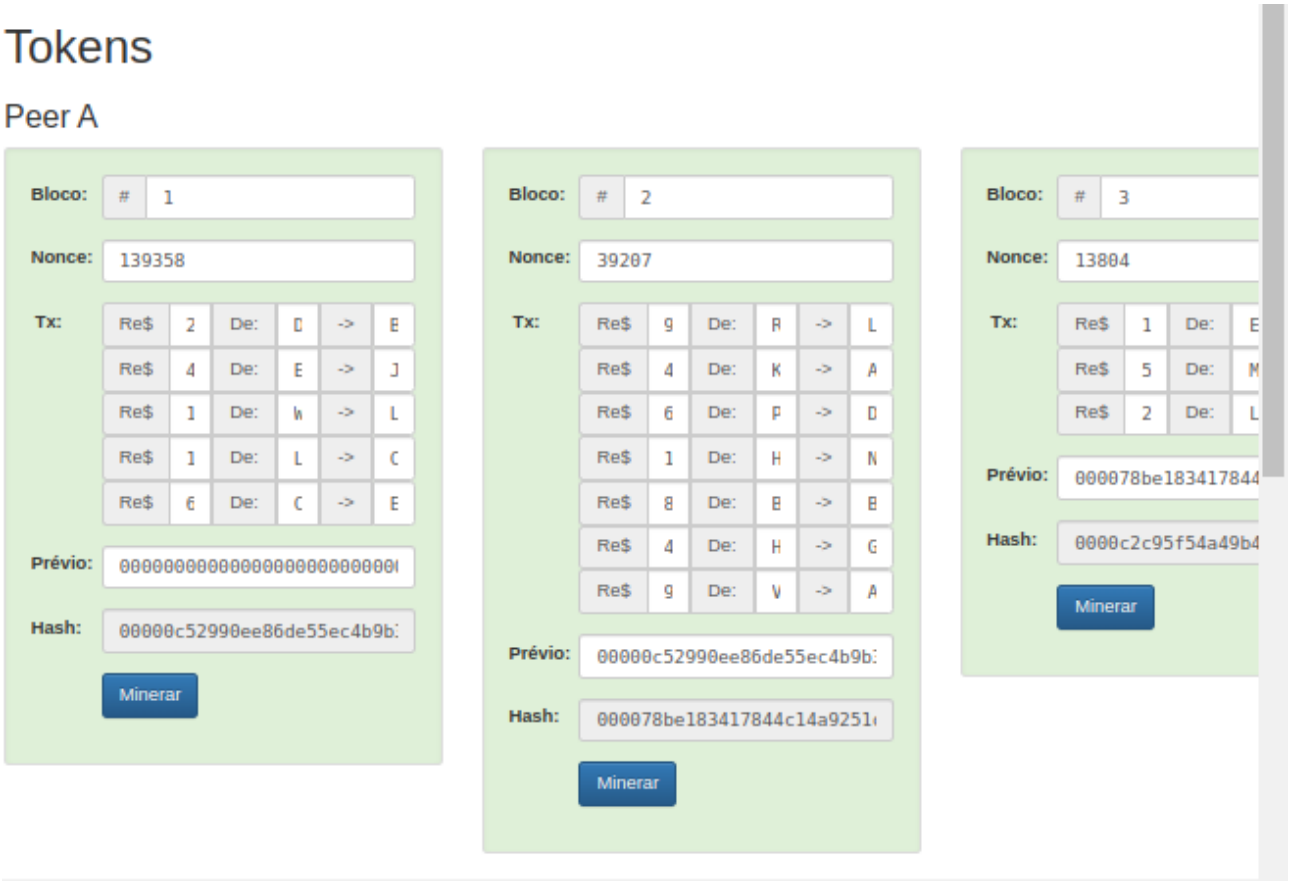
Todas as vezes que algum bloco tiver um dado alterado, será necessário fazer a mineração naquele bloco em específico, e também em toda a cadeia conseguinte a aquele bloco, de forma sequencial, para fazer com que todos tenham um hash válido.

Blockchain Distribuída:



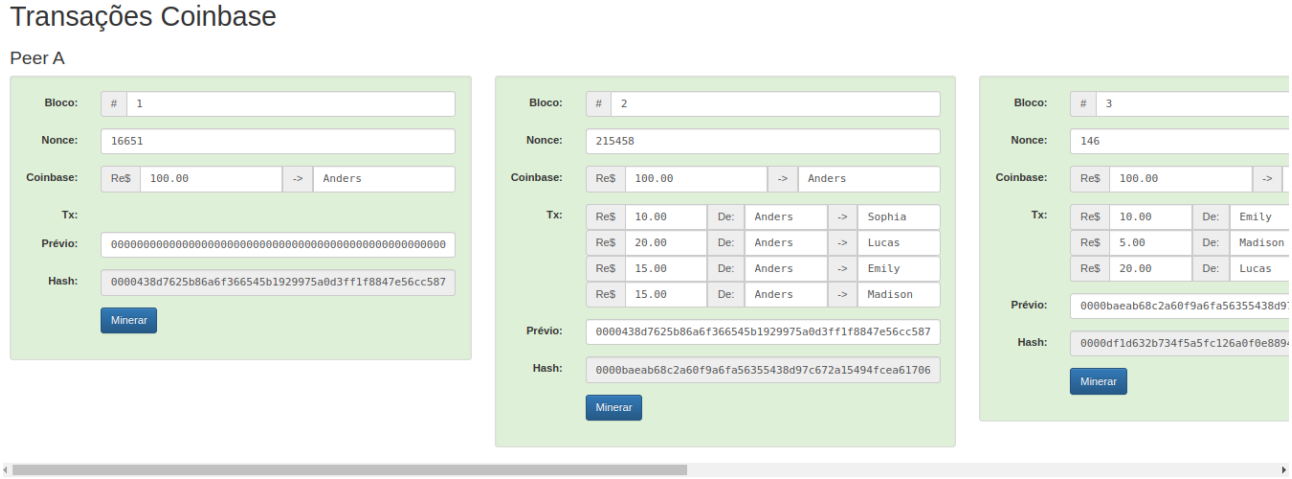
Agora a Blockchain se encontra dividida em vários computadores, com uma cópia exata de toda a Blockchain em cada um deles. Ou seja, se alguém invadir uma blockchain e conseguir alterar algum dado, isso poderá ser identificado visto que aquele dado adulterado resultará numa nova hash para todos os blocos sequenciais à aquele que recebeu a alteração e não baterá com os hashes das outras milhares de cópias daquela blockchain.

Tokens:

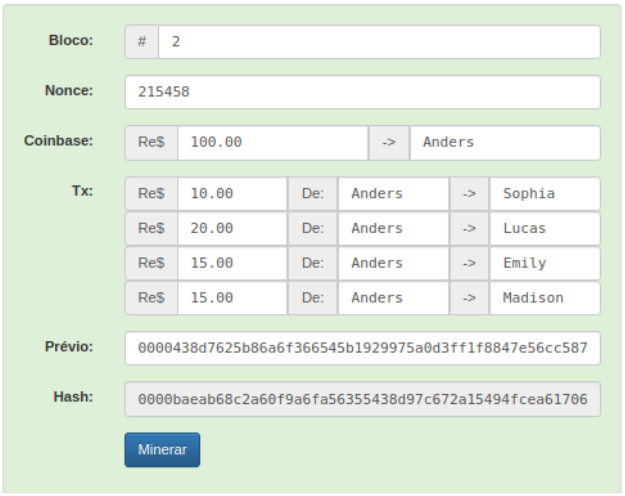


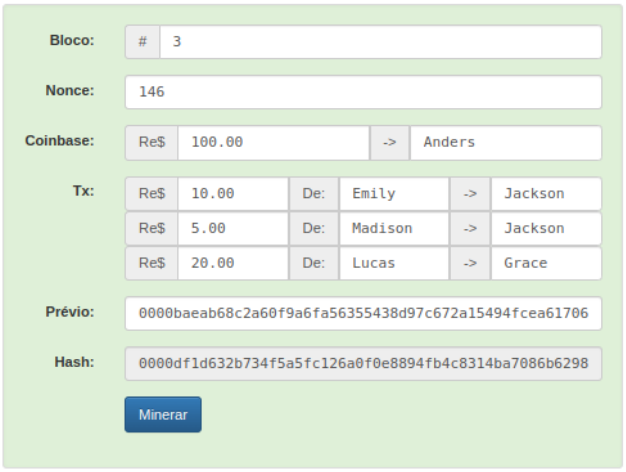
Os tokens, nesse caso, representam as transações bancárias. Por exemplo, Pedro transfere 25 Reais para Bianca. Mas como saber se o Pedro tem esses 25 Reais para serem transferidos? Aí que entra a coinbase.

Transações Coinbase:



O coinbase define uma base para o valor disponível para o usuário. Por exemplo, no primeiro bloco, estão sendo “distribuidos” 100 dólares para o Anders. Com esse dinheiro, ele faz as transações para as outras pessoas.





No segundo bloco, a Emily recebe 10 dólares do Anders, e no bloco seguinte, como ela tem dinheiro para gastar, ela faz uma transação para o Jackson.

Segurança:

Para garantir que outra pessoa não faça alguma transação com o dinheiro de outra, existem as private keys e public keys. Ao realizar uma transação, a private key é validada, e em caso positivo, a transferência pode ser realizada. A public key é uma chave pública associada a chave privada, que tem como objetivo identificar o usuário que ira receber a transação. A chave privada, como o nome diz, não deve ser compartilhada, a pública, por sua vez, deve ser compartilhada.

Construa sua própria Blockchain em Python: Guia prático

Este relatório busca descrever o funcionamento detalhado do código de uma aplicação baseada em blockchain. O código, implementado em Python, simula uma blockchain simples, com funcionalidades de mineração de blocos, adição de transações e resolução de conflitos entre nós.

Parte 1: Blockchain

A classe “Blockchain” é a base da aplicação, responsável por gereciar a cadeia de blocos, transações e a lógica de mineração.

Construtor __init__:

- Inicializa uma nova instância da blockchain;
- Ações realizadas:
  1. Cria a lista “chain” que armazena os blocos;
  2. Cria a lista “current_trasactions” para manter as transações pendentes;
  3. Cria o bloco gênesis chamando o método “new_block” com prova inicial (proof=199O) e hash anterior como “1”.

Método new_block:

- Cria e adiciona um novo bloco na cadeia.
- Parâmetros:
  1. proof: Prova de trabalho para o novo bloco (campo nonce);
  2. previous_hash: Hash do bloco anterior.
- Ações realizadas:
  1. Cria um bloco:
     - index: Posição do bloco na cadeia;
     - timestamp: tempo atual em segundos;
     - transactions: lista de transações pendentes;
     - proof: prova de trabalho fornecida;
     - previous_hash: hash do bloco anterior.
  2. Adiciona o novo bloco à lista chain.

Método new_transaction:

- Responsabilidade: Adiciona uma nova transação pendente.
- Parâmetros:
  1. sender: Identificador do remetente;
  2. recipient: Identificador do destinatário;
  3. amount: Valor transferido.
- Ações Realizadas:
  1. Adiciona um dicionário com os detalhes da transação à lista current_transactions.
  2. Retorna o índice do próximo bloco que incluirá essa transação.

Método estático hash:

- Responsabilidade: Gera um hash SHA-256 de um bloco.
- Ações Realizadas:
  1. Serializa o bloco em formato JSON ordenado.
  2. Gera e retorna o hash utilizando SHA-256.

Propriedade last_block:

- Responsabilidade: Retorna o último bloco da cadeia.

Método proof_of_work:

- Responsabilidade: Implementa o algoritmo de prova de trabalho (PoW).
- Parâmetros:
  1. last_block: O último bloco da cadeia.
- Ações Realizadas:
  1. Incrementa o valor da prova (proof) até encontrar um valor válido, ou seja, até encontrar um valor em que o hash comece com “0000”.
  2. Retorna a prova válida.

Método estático valid_proof:

- **Responsabilidade: Valida uma prova.**
- **Parâmetros:**
  1. last_proof: Prova do bloco anterior.
  2. proof: Prova atual.
  3. last_hash: Hash do bloco anterior.
- **Ações Realizadas:**
  1. Concatena os valores e gera um hash.
  2. Verifica se o hash começa com quatro zeros.
  3. Retorna True se a condição for satisfeita; caso contrário, retorna False.

Parte 2: API Rest

A aplicação utiliza o Flask para fornecer uma interface HTTP para interagir com a blockchain.

Configuração inicial:

- **UUID para o nó:** Um identificador único é gerado para o nó usando uuid4.
- **Instância da Blockchain:** Um objeto da classe Blockchain é instanciado.

Rota /mine:

- **Método:** GET
- **Responsabilidade:** Minera um novo bloco.
- **Fluxo:**
    1. Recupera o último bloco.
    2. Executa o algoritmo de PoW.
    3. Cria uma transação de recompensa para o minerador.
    4. Adiciona um novo bloco à cadeia.
    5. Retorna os detalhes do novo bloco como resposta JSON.

Rota /transactions/new:

- **Método:** POST
- **Responsabilidade:** Adiciona uma nova transação.
- **Fluxo:**
    1. Valida os campos necessários no corpo da requisição.
    2. Adiciona a transação à lista de pendentes.
    3. Retorna o índice do bloco que incluirá a transação.

Rota /chain:

- **Método:** GET
- **Responsabilidade:** Retorna toda a blockchain.
- **Fluxo:**
    1. Serializa a cadeia.

2. Retorna a cadeia e seu comprimento.

Rota /nodes/register:

- **Método:** POST
- **Responsabilidade:** Registra novos nós na rede.
- **Fluxo:**
    1. Recebe uma lista de URLs de nós no corpo da requisição.
    2. Adiciona os nós à lista local.
    3. Retorna a lista atualizada de nós.

Rota /nodes/resolve:

- **Método:** GET
- **Responsabilidade:** Implementa o algoritmo de consenso.
- **Fluxo:**
    1. Verifica a blockchain em todos os nós conectados.
    2. Substitui a cadeia local pela mais longa e válida.
    3. Retorna uma mensagem indicando se a cadeia foi substituída ou não.