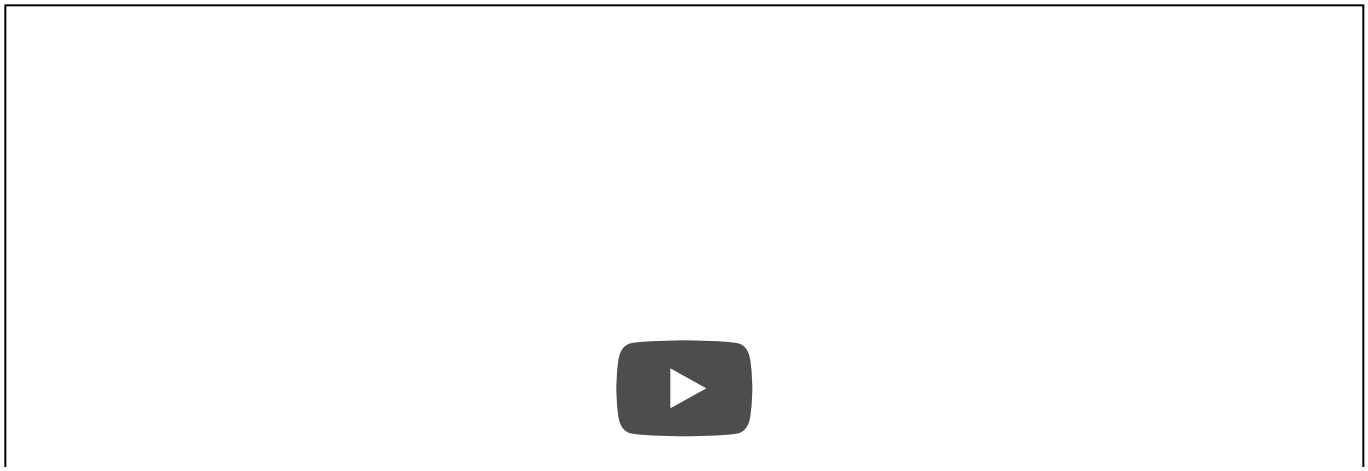
Transactions & Mining Rewards (Javascript blockchain, part 3)

This is part 3 of my blog posts series in which we create a blockchain in Javascript. In the previous blog posts we created a simple blockchain and implemented proof-of-work to protect it from spammers and attackers. However we made some shortcuts along the way: our blockchain can only store 1 transaction in a block and there are no rewards for miners. Let's fix that!

Missed the other parts? Read them here:

- Part 1: Implementing a basic blockchain.
- Part 2: Implementing proof-of-work.
- **Part 3: Transactions & mining rewards**



Restructuring the Block class

Right now a block has the index, previousHash, timestamp, data, hash and nonce properties. The `index` property isn't really useful, in fact I don't even know why I added it to begin with. So let's remove that and lets rename `data` to `transactions` which makes more sense.

```
class Block{
    constructor(timestamp, transactions, previousHash = '') {
        this.previousHash = previousHash;
        this.timestamp = timestamp;
        this.transactions = transactions;
        this.hash = this.calculateHash();
        this.nonce = 0;
    }
}
```

When we change our Block class, we also have to change its `calculateHash()` function. Right now it still uses the old `index` and `data` properties, which we just removed.

```
calculateHash() {
  return SHA256(this.previousHash + this.timestamp + JSON.stringify(this.transactio
}
```

Transaction class

Inside a block we will be able to store multiple transactions. So let's also define a `Transaction` class so we can lock down what properties a transaction should have:

```
class Transaction{
    constructor(fromAddress, toAddress, amount){
        this.fromAddress = fromAddress;
        this.toAddress = toAddress;
        this.amount = amount;
    }
}
```

In this example a transaction will be very simple, just containing a sender (`fromAddress`) a receiver (`toAddress`) and an amount. If required, you can add more fields to a transaction, but these are the absolute minimum.

Adapting our Blockchain

Now comes the biggest task of them all: making our `Blockchain` work with all these new changes. The first thing that we need is a place to store pending transactions.

As you know blockchains create blocks on a steady interval thanks to the proof-of-work algorithm. In Bitcoin's case, the difficulty is adjusted so that new blocks are created roughly every 10 minutes. However it should be possible to submit new transactions in between the creation of two blocks.

So to do that, let's start by changing our Blockchain's constructor so it has a place to store pending transactions. We'll also create a property that defines how much coins a miner gets as a reward:

```
class Blockchain{
    constructor() {
        this.chain = [this.createGenesisBlock()];
        this.difficulty = 5;
```

```
        // Place to store transactions in between block creation
        this.pendingTransactions = [];

        // How many coins a miner will get as a reward for his/her efforts
        this.miningReward = 100;
    }
}
```

Next, we need to adapt our `addBlock()` method and by adapting I mean completely remove it to rewrite it! We won't allow people to directly add blocks to our chain anymore. Instead they have to add transactions who will be included in the next block. So we'll replace the `addBlock()` method with `createTransaction()`, that makes more sense:

```
createTransaction(transaction) {
    // There should be some validation here!

    // Push into onto the "pendingTransactions" array
    this.pendingTransactions.push(transaction);
}
```

Mining blocks

People can now add new transactions to the list of pending ones. But one way or another, we need to clear those out and put them inside actual blocks. So to do that, let's create a `minePendingTransactions()` method. This method will not only mine a new block with all the pending transactions, it will also send a mining reward to the miner.

```
minePendingTransactions(miningRewardAddress) {
    // Create new block with all pending transactions and mine it..
    let block = new Block(Date.now(), this.pendingTransactions);
    block.mineBlock(this.difficulty);

    // Add the newly mined block to the chain
    this.chain.push(block);
```

```
    // Reset the pending transactions and send the mining reward
    this.pendingTransactions = [
        new Transaction(null, miningRewardAddress, this.miningReward)
    ];
}
```

Note that the method takes an argument `miningRewardAddress`. If you start mining, you can pass along your wallet address to this method. Once you successfully mined a block, the system will create a new transaction to give you your mining reward (in this case 100 coins).

One thing to note is that in this implementation we take all the pending transactions and add them to a block. In reality however that won't work because the size of a block is limited. In Bitcoin's case there is a block size limit of 2mb. If there are more transactions that can fit in a block, the miner gets to choose which transaction he includes and which he doesn't (usually the ones with the highest fee wins).

Balance of an address

Before we can test our code let's do one more thing! It would be nice to be able to check the balances of the addresses on our blockchain.

```
getBalanceOfAddress(address){
    let balance = 0; // you start at zero!

    // Loop over each block and each transaction inside the block
    for(const block of this.chain){
        for(const trans of block.transactions){

            // If the given address is the sender -> reduce the balance
            if(trans.fromAddress === address){
                balance -= trans.amount;
            }

            // If the given address is the receiver -> increase the balance
            if(trans.toAddress === address){
                balance += trans.amount;
```

```
            }
        }
    }

    return balance;
}
```

Testing it

Alright we're done and can finally test if everything is working! To do that, let's create some transactions:

```
let savjeeCoin = new Blockchain();

console.log('Creating some transactions...');
savjeeCoin.createTransaction(new Transaction('address1', 'address2', 100));
savjeeCoin.createTransaction(new Transaction('address2', 'address1', 50));
```

These transactions are now pending and in order for them to get confirmed, we have to start the miner:

```
console.log('Starting the miner...');
savjeeCoin.minePendingTransactions('xaviers-address');
```

When we start the miner, we also pass along an address on which we want to receive the mining reward. In this case, my address is `xaviers-address` (pretty complicated!)

After that, let's check the balance of `xaviers-address`:

```
console.log('Balance of Xaviers address is', savjeeCoin.getBalanceOfAddress('xavier
// Output: 0
```

It outputs that my balance is zero. Wait, what? Shouldn't I have gotten my mining reward? Well if you look closely at the code, you'll see that the system

creates a new block and then adds your mining rewards as a new pending transaction. That transaction will be included in the next block. So if we start the miner again, we will receive our 100 coin reward!

```
console.log('Starting the miner again!');
savjeeCoin.minePendingTransactions("xaviers-address");

console.log('Balance of Xaviers address is', savjeeCoin.getBalanceOfAddress('xaviers
// Output: 100
```

Limitations & conclusion

Right now our little blockchain is capable of storing multiple transactions in a block and of giving rewards to miners.

There are however still some things missing: when sending money we don't check to see if the sender has enough balance to actually make that transaction. However this is an easy thing to address. We also don't have a way to create a new wallet and to sign transactions (traditionally that would be done with public/private key encryption)

Disclaimer & Source code

I want to point out that is this by no means a complete blockchain implementation! It still lacks many features. This is just a proof-of-concept designed to help you understand how blockchains work internally.

The source code of this project is available on GitHub.

Posted on 12 Feb 2018

Louis Kemp • 24 days ago

great explanation of blockchain software implementation. Thank you.

• Reply • Share ›

RSS feed - Github - Twitter - Facebook - YouTube