Blockchain Node.js Introduction I

Blockchain is a big buzz word which is heard a lot these days.

But it's like the movie interstellar, everyone thinks it's cool but no one really gets it.

Here, we'll be looking at how the blockchain data structure works.

Introduction

A blockchain is a continuously growing list of records, called blocks, which are linked and secured using cryptography.

So basically, a blockchain is a data structure like linked lists with a growing set of nodes or blocks. Most of us would have learned about linked lists in school. Each node has several attributes including id, data, next node's address etc. So, the blockchain is a slightly more complex data structure.

In a blockchain, each node is called a "block" and each block has an index, timestamp, hash of itself, hash of the previous block and of course the data. The data can be anything.

In a cryptocurrency, the data is the details of the transaction like sender's username, receiver's username and amount.

Blockchain was initially implemented in 1991 to timestamp digital documents, to make sure they are not backdated or tampered with. But it didn't get much attention as timestamping documents is not as cool as bitcoins.

The concept

The special thing about the blockchain is that it cannot easily be tampered with.

The hash of each block gives it a unique identity like a fingerprint. And this hash is generated only once when the block is created.

It is generated using the values of all the other attributes of the block. So, if any of the values are changed, the hash will no more be valid and hence the entire block will be invalid. So, this way nobody can tamper with the values stored in the block.

Another thing is if the hash is also recalculated, then it will not be detected as tampering. To avoid this, we store the hash of the previous block. This creates a chain of the blocks and if the hash of any block changes, the chain will be broken and the entire thing will be invalid.

So, to successfully hack a blockchain, the hacker has to know the location of all blocks and simultaneously change all their hashes and relink the whole blockchain. This is not impossible but extremely tedious as there is a mandatory gap of 10 minutes between each transaction. So it beats the whole purpose of life.

Decentralized Blocks

These blockchains are not stored in a centralized server. But the blocks are distributed across the internet and they are connected using a peer-to-peer network.

When someone joins the network, they are given the full copy of the network. When they create a new block, the block is sent to all the peers and the validity of the chain is checked. If everything checks out, the block is connected to the chain.

Creating a blockchain

Let's get to the code

We will be building a simple cryptocurrency called "cilcoin".

So let's get started on building possibly your very first blockchain.

Create an empty directory anywhere on your computer. In that, create a new file called main.js.

Create a class called "Block" which will represent a block in the chain. Give it a constructor with arguments as index, data and previous hash.

Store the values of the arguments in consequent local variables. Create another attribute called timestamp and store the actual timestamp in it.

```
class Block {

   constructor(index, data, prevHash) {
```

```
        this.index = index;
        this.timestamp = Math.floor(Date.now() / 1000);
        this.data = data;

        this.prevHash = prevHash;

    }

}
```

Create a function inside the class and name it getHash.

We will be using the sha256 algorithm to generate the hash.

So import the required files.

Inside the function, calculate the hash by using the other attributes as salt. Salts are basically just arbitary string values used for generating hashes. Return the calculated hash.

Also, create an attribute in the constructor and store the calculated value by calling our function.

```
constructor (index, data, prevHash) {

    this.index = index;
    this.timestamp = Math.floor(Date.now() / 1000);
    this.data = data;

    this.prevHash = prevHash;
    this.hash = this.getHash();

}


getHash(){

    return sha(JSON.stringify(this.data) + this.prevHash + this.index + this.timestamp);

}
```

Now, let's create a new class to represent the entire block chain.

This class will have a single attribute called "chain" which will hold an array of blocks.

```
class BlockChain {

   constructor() {

      this.chain = [];

   }

}
```

Create a function in this class called addBlock.

This will be responsible for adding a new block to the chain and it will take the data as its argument.

Inside the function, calculate the index of the new block by getting the length of the existing chain.

Get the hash of the last block in the chain (if the chain is empty, use "0" by default).

Finally, create a new object of the "Block" class and plug in the calculated values in the constructor. Push this object into the "chain" array.

```
addBlock(data){

   let index = this.chain.length;
   let prevHash = this.chain.length !== 0 ? this.chain[this.chain.length - 1].hash : 0;

   let block = new Block(index, data, prevHash);

   this.chain.push(block);

}
```

Create another method in the same class and name it chainIsValid.

In this function, we will be checking if the blockchain is following all the rules.

It should have two checks as mentioned above.

One, the validity of the hash, which can be done by comparing the stored hash with a newly calculated hash. If they match, then the values haven't been tampered with.

Then two, checking if the previousHash attribute is storing the same value as the hash of the previous block. This has to be performed on all the blocks in the chain.

```
chainIsValid() {
   for(var i=0;i<this.chain.length;i++){
      if(this.chain[i].hash !== this.chain[i].getHash())
         return false;
      if(i > 0 && this.chain[i].prevHash !== this.chain[i-1].hash)
         return false;
   }
   return true;
}
```

Finished!

Add New Blocks and Test the Blockchain

We are now done with building the blockchain. Now we only have to add new blocks and test it.

Add New Blocks

First let's test if the block chain is working without any errors.

So, outside both the classes, create an object of the BlockChain class and name it CILCoin.

Then, add a few blocks to it using the addBlock function.

```
const CILCoin = new BlockChain();

CILCoin.addBlock({sender: "Thor of Asgard", receiver: "Thanos", amount: 1090});

CILCoin.addBlock({sender: "Sky Walker", receiver: "Jedi", amount: 450});

CILCoin.addBlock({sender: "The Penguin", receiver: "Dr. Steven Strange", amount: 35});

console.log(JSON.stringify(CILCoin, null, 4));
```

Go to your console and run this using the command "node main.js".

You should be getting an output like this:

Testing Blockchain Validity

Now, let's test the validity checker.

Log the validity of the blockchain without tampering any block.

Then tamper one of the blocks by changing its data.

Then log the validity again.

```
console.log("Validity: ", CILCoin.chainIsValid());
```

```
CILCoin.chain[0].data.receiver = "Joker";

console.log("Validity: ", CILCoin.chainIsValid());
```

In the first case it will be true and in the second it will be false.