UDACITY

< Back to Blockchain Developer

Private Blockchain

REVIEW

CODE REVIEW 7

HISTORY

▼ blockchainClass.js 5

```
1  const Block = require('./blockClass');
2  const leveldb = require('./levelFunctions');
3  const SHA256 = require('crypto-js/sha256');
4
5  class Blockchain {
6      constructor() {
7          this.getBlockHeight().then((height) => {
8              if (height === -1) {
```

AWESOME

Good check!

```
9                  this.addBlock(new Block("Genesis block")).then(() => cc
10             }
11         });
       }
12     //
14     async addBlock(newBlock) {
```

```
15          const height = parseInt(await this.getBlockHeight());
16          newBlock.height = height + 1;
17          newBlock.time = new Date().getTime().toString().slice(0, -3);
18          //
19          if (newBlock.height > 0) {
20              const previousBlock = await this.getBlock(height);
21              newBlock.previousBlockHash = previousBlock.hash;
22              console.log('Previous block hash: ' + newBlock.previousBloc
23          }
24          // note block hash is performed when hash property is null
25          newBlock.hash = SHA256(JSON.stringify(newBlock)).toString();
26          console.log('New block hash: ' + newBlock.hash);
27          //
28          await leveldb.addBlock(newBlock.height, JSON.stringify(newBlock
29      } //addBlock
30      //
31      async getBlockHeight() {
32          const height = await leveldb.getBlockHeight();
33          return height;
34      } //getBlockHeight
35      //
36      async getBlock(blockHeight) {
37          const block = await leveldb.getBlock(blockHeight);
38          //console.log(block);
39          return block;
40      } //getBlock
41      //
42      async validateBlock(blockHeight) {
43          // get block object
44          let block = await this.getBlock(blockHeight);
45          // get block hash
46          let blockHash = block.hash;
47          // remove block hash to test block integrity
48          block.hash = '';
49          // generate block hash
50          let validBlockHash = SHA256(JSON.stringify(block)).toString();
51          // Compare
52          if (blockHash===validBlockHash) {
53              console.log('Block ' + blockHeight + ' validation confirmed
54              return true;
55          } else {
56              console.log('Block ' + blockHeight + ' hash invalid:\n' + b
```

AWESOME

Yes this shows when block is invalid!

```
57              return false;
58          }
59      } //validateBlock
60      //
```

```
61      async validateChain() {
62          let errorLog = [];
63          let previousHash = '';
64          let validFlag = false;
65          const height = await this.getBlockHeight();
66          //
67          for (let i = 0; i <= height; i++) {
68              this.getBlock(i).then((block) => {
69                  validFlag = this.validateBlock(block.height);
```

REQUIRED

Because this is asynchronous, this will get pushed into the background and your code wi
before this is done. You must find a way to make this resolve first before the errorLog ch

```
70                  //
71                  if (!validFlag) {
72                      errorLog.push(i);
73                  }
74                  //
75                  if (block.previousBlockHash !== previousHash) {
76                      errorLog.push(i);
77                  }
78                  //
79                  previousHash = block.hash;
80                  //
81                  if (errorLog.length > 0) {
```

REQUIRED

This should be out of the loop else you'll call this multiple times. We need to only look th

```
82                      console.log('Block errors =' + errorLog.length);
83                      console.log('Blocks:' + errorLog);
84                      console.log('Errors detected');
85                  } else {
86                      console.log('No errors detected');
87                  }
88              });
89          } //loop
90      } //validateChain
```

SUGGESTION

There are many ways to implement this correctly. The simplest syntactically would be to
like:

like:

```
await height
for every block in the chain:
     if (! await validateBlock), update errorLog
     if (i < height):
          await current block
          await  the next block
          if currentblock.hash equal to nextblock.previousblockhash, then
check errorLog contents
```

This is not the only way and you're more than welcome to try different solutions. You car channel, or Knowledge portal if you need addtional help. When done correctly, you'll get

```
> Block 0 validation confirmed    // promises are resolved first!
Block 1 validation confirmed
Block 2 hash invalid:
effd81d74364c4bc481b0ef84c9c754f40bbe949316626eaaa108e12170a82ef<>0824bd1d
05fd4376eb16b6d69a573
Block 3 validation confirmed
Block 4 hash invalid:
61aee58f7157c431ef083eba7e907c878b2081c3d9e5cedaa9948aa76215672c<>6c5c1f3c
01c5a45aeb40f5ddbad0f
Block errors =2           // errorLog is checked last!
Blocks:2,4
Errors detected
```

```
91 } //Blockchain Class
92
93
94 module.exports = Blockchain;
```

▸ privateBlockchain.js 1

▸ levelFunctions.js 1

▸ blockClass.js

RETURN TO PATH

Rate this review