

Hash puzzle

Code:

```
//Install node.js to test the code
const { createHash } = require('crypto');
class Block{
  constructor (index, timestamp, data, previousHash=''){

    //we keep track of our properties here
    this.index = index;
    this.timestamp = timestamp;
    this.data = data;
    this.previousHash = previousHash;
    this.hash = this.calculateHash();

    //nonce property
    this.nonce = 0;
  }

  //calculating the hash value with the nonce property
  calculateHash(){
    return createHash('sha256').update(this.index + this.previousHash + this.timestamp
    +
    JSON.stringify(this.data) + this.nonce).digest('hex').toString();
  }

  //Method to mine a block
  mineBlock(difficulty){
    //while loop conditional used is a quick trick to make the substring of hash values
    exactly the length of difficulty
    while(this.hash.substring(0, difficulty) !== Array(difficulty + 1).join("0"))
    {
      //incrementing the nonce value everytime the loop runs.
      this.nonce++;

      //recalculating the hash value
      this.hash = this.calculateHash();
    }

    //logging when a block is created
    console.log("Block mined: " + this.hash);
  }
}
class Blockchain{
  constructor(){
    this.chain = [this.createGenesisBlock()];

    //adding a difficulty property to the Blockchain class
  }
}
```

CSE4080 - Blockchain Technology
Lab Experiment 6

```
this.difficulty = 4;
}

createGenesisBlock(){
  return new Block(0, "02/01/2018", "Genesis Block", "0");
}

getlatestBlock(){
  return this.chain[this.chain.length - 1];
}

addBlock(newBlock){
  newBlock.previousHash = this.getlatestBlock().hash;

  //We commented the earlier method that adds a block directly
  //newBlock.hash = newBlock.calculateHash();

  //New method to mine the block
  //Customizable difficulty value
  newBlock.mineBlock( this.difficulty );

  this.chain.push(newBlock);
}

isChainValid(){
  for(let i = 1; i < this.chain.length; i++){
    const currentBlock = this.chain[i];
    const previousBlock = this.chain[i-1];

    if(currentBlock.hash !== currentBlock.calculateHash()){
      return false;
    } //check for hash calculations

    if(currentBlock.previousHash !== previousBlock.hash){
      return false;
    } //check whether current block points to the correct previous block

  }

  return true;
}
}

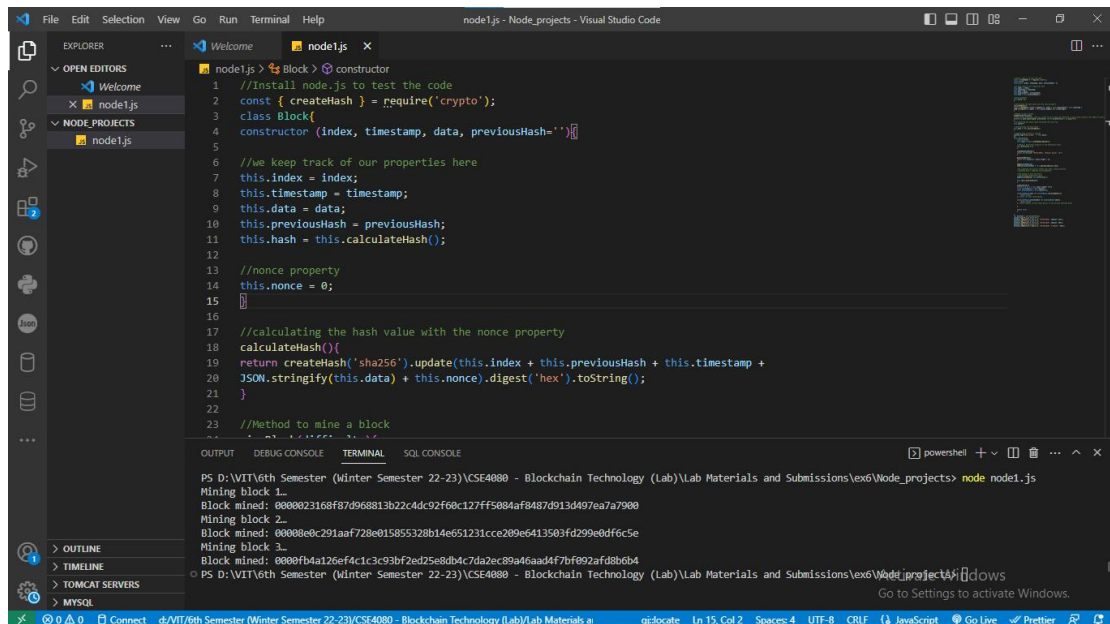
let koreCoin = new Blockchain();
console.log('Mining block 1...');
koreCoin.addBlock(new Block (1, "01/01/2018", {amount: 20}));
console.log('Mining block 2...');
koreCoin.addBlock(new Block (2, "02/01/2018", {amount: 40}));
```

CSE4080 - Blockchain Technology

Lab Experiment 6

```
console.log('Mining block 3...');
koreCoin.addBlock(new Block (3, "02/01/2018", {'amount': 40}));
```

Output:



The screenshot shows a Visual Studio Code editor with a file named 'node1.js'. The code defines a 'Block' class with a constructor that takes 'index', 'timestamp', 'data', and 'previousHash'. It includes methods for 'calculateHash' and a comment for 'Method to mine a block'. The terminal output shows the execution of 'node node1.js', resulting in three mined blocks with their respective hashes.

```
node1.js > Block > constructor
1 //Install node.js to test the code
2 const { createHash } = require('crypto');
3 class Block{
4   constructor (index, timestamp, data, previousHash=''){}
5
6   //we keep track of our properties here
7   this.index = index;
8   this.timestamp = timestamp;
9   this.data = data;
10  this.previousHash = previousHash;
11  this.hash = this.calculateHash();
12
13  //nonce property
14  this.nonce = 0;
15
16
17  //calculating the hash value with the nonce property
18  calculateHash(){
19    return createHash('sha256').update(this.index + this.previousHash + this.timestamp +
20    JSON.stringify(this.data) + this.nonce).digest('hex').toString();
21  }
22
23  //Method to mine a block
```

Terminal Output:

```
PS D:\VIT\6th Semester (Winter Semester 22-23)\CSE4080 - Blockchain Technology (Lab)\Lab Materials and Submissions\ex6\Node_projects> node node1.js
Mining block 1...
Block mined: 0000023168f87d968813b22c4dc92fe0c127ff5084af8487d913d497ea7a7900
Mining block 2...
Block mined: 0000080c291aaf728e01585328b14e651231cce209e6413503fd299e0df6c5e
Mining block 3...
Block mined: 0000f04a126efactc3c93bf2ed25e0db4c7da2ec09a46aad4f7bf092afdb0b64
```

//Observe the zeros above-5 zeros and hence met the target of 4

Code:

```
//increase difficulty level to 7
const { createHash } = require('crypto');
class Block{
  constructor (index, timestamp, data, previousHash=''){}

  //we keep track of our properties here
  this.index = index;
  this.timestamp = timestamp;
  this.data = data;
  this.previousHash = previousHash;
  this.hash = this.calculateHash();

  //nonce property
  this.nonce = 0;
}

//calculating the hash value with the nonce property
calculateHash(){
  return createHash('sha256').update(this.index + this.previousHash + this.timestamp
+
JSON.stringify(this.data) + this.nonce).digest('hex').toString();
}
```

CSE4080 - Blockchain Technology
Lab Experiment 6

```
//Method to mine a block
mineBlock(difficulty){
  //while loop conditional used is a quick trick to make the substring of hash
  values exactly the length of difficulty
  while(this.hash.substring(0, difficulty) !== Array(difficulty + 1).join("0"))
  {
    //incrementing the nonce value everytime the loop runs.
    this.nonce++;

    //recalculating the hash value
    this.hash = this.calculateHash();
  }

  //logging when a block is created
  console.log("Block mined: " + this.hash);
}
class Blockchain{
  constructor(){
    this.chain = [this.createGenesisBlock()];

    //adding a difficulty property to the Blockchain class
    this.difficulty = 7;
  }

  createGenesisBlock(){
    return new Block(0, "02/01/2018", "Genesis Block", "0");
  }

  getlatestBlock(){
    return this.chain[this.chain.length - 1];
  }

  addBlock(newBlock){
    newBlock.previousHash = this.getlatestBlock().hash;

    //We commented the earlier method that adds a block directly
    //newBlock.hash = newBlock.calculateHash();

    //New method to mine the block
    //Customizable difficulty value
    newBlock.mineBlock( this.difficulty );

    this.chain.push(newBlock);
  }

  isChainValid(){
    for(let i = 1; i < this.chain.length; i++){
```

CSE4080 - Blockchain Technology

Lab Experiment 6

```
const currentBlock = this.chain[i];
const previousBlock = this.chain[i-1];

if(currentBlock.hash !== currentBlock.calculateHash()){
  return false;
} //check for hash calculations

if(currentBlock.previousHash !== previousBlock.hash){
  return false;
} //check whether current block points to the correct previous block

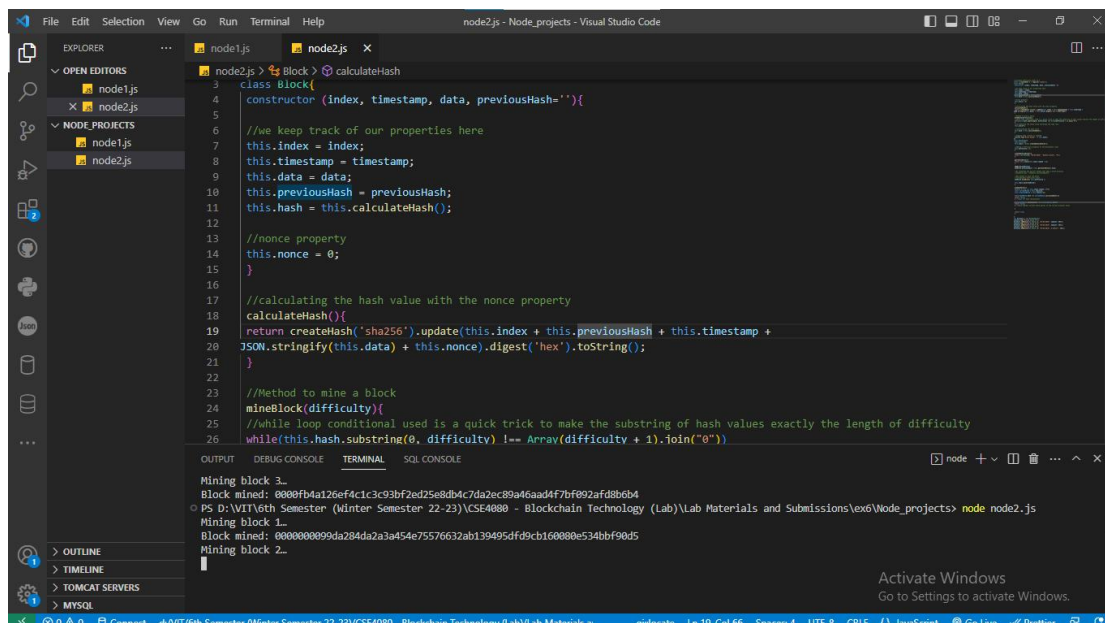
}

return true;
}

}

let koreCoin = new Blockchain();
console.log('Mining block 1...');
koreCoin.addBlock(new Block (1, "01/01/2018", {amount: 20}));
console.log('Mining block 2...');
koreCoin.addBlock(new Block (2, "02/01/2018", {amount: 40}));
console.log('Mining block 3...');
koreCoin.addBlock(new Block (3, "02/01/2018", {'amount': 40}));
```

Output:



The screenshot shows the Visual Studio Code interface with the Blockchain code in the editor and the output in the terminal. The code defines a `Block` class with properties for index, timestamp, data, previous hash, and nonce. It includes methods for calculating the hash and mining a block. The terminal output shows the execution of the code, including the creation of the `koreCoin` object and the addition of three blocks to the chain.

```
node2.js - Node_projects - Visual Studio Code
node2.js > Block > calculateHash
class Block {
  constructor (index, timestamp, data, previousHash='') {
    //we keep track of our properties here
    this.index = index;
    this.timestamp = timestamp;
    this.data = data;
    this.previousHash = previousHash;
    this.hash = this.calculateHash();
    //nonce property
    this.nonce = 0;
  }
  //calculating the hash value with the nonce property
  calculateHash(){
    return createHash('sha256').update(this.index + this.previousHash + this.timestamp +
    JSON.stringify(this.data) + this.nonce).digest('hex').toString();
  }
  //Method to mine a block
  mineBlock(difficulty){
    //while loop conditional used is a quick trick to make the substring of hash values exactly the length of difficulty
    while(this.hash.substring(0, difficulty) !== Array(difficulty + 1).join("0"))
  }
}
```

OUTPUT

```
Mining block 1...
Block mined: 0000fb4a126ef4c13c93bf2ed25e8db4c7da2ec09a46aad4f7bf092afd8bb64
Mining block 1...
Block mined: 0000000099da284da2a3a454e75576632ab139495df9cbl60080e534bbf90d5
Mining block 2...
```