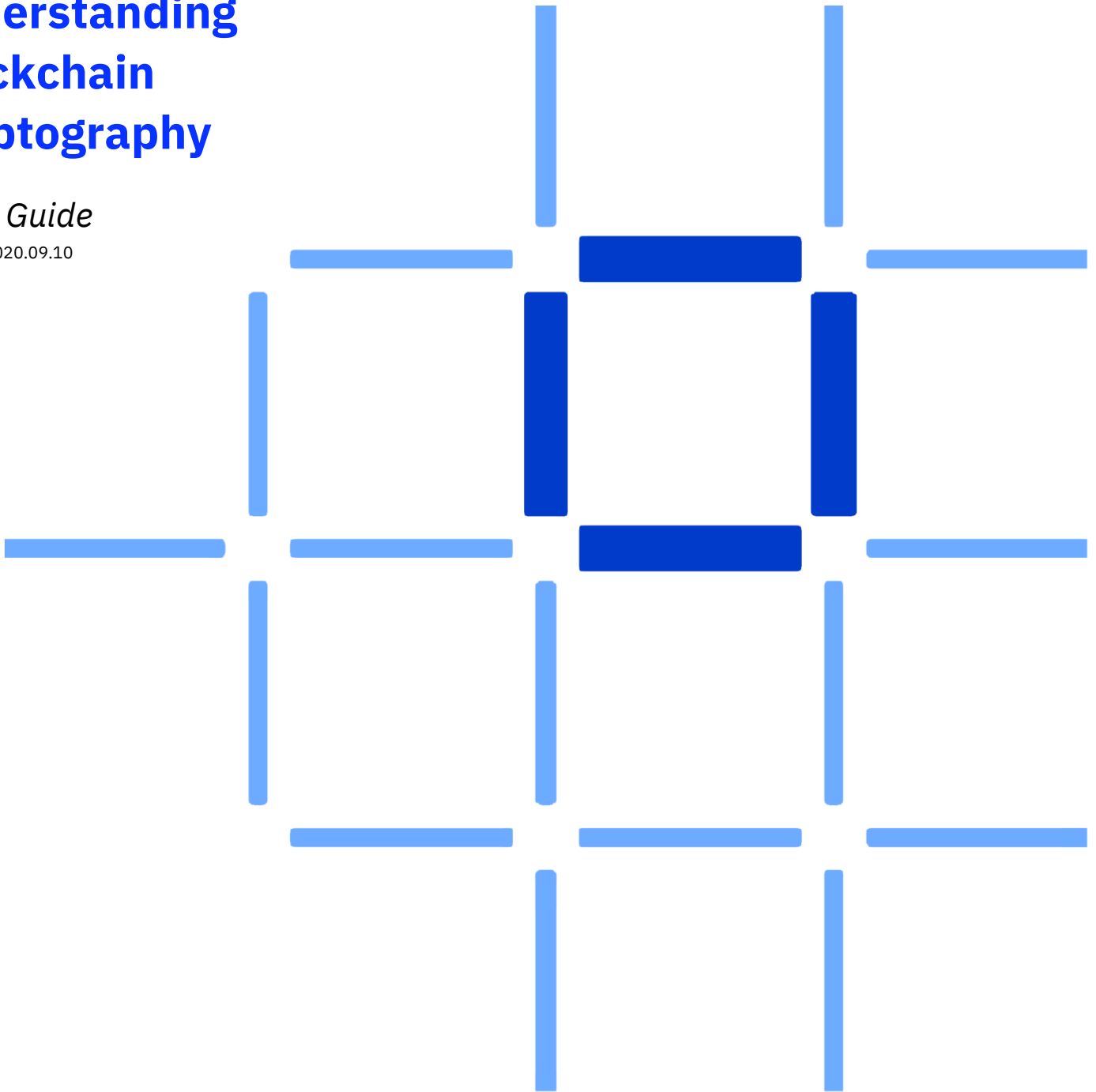


Understanding Blockchain Cryptography

Lab 1 Guide

Version: 2020.09.10



The information contained in this document has not been submitted to any formal IBM test and is distributed on an “as is” basis without any warranty either express or implied. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer’s ability to evaluate and integrate them into the customer’s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will result elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

© Copyright International Business Machines Corporation 2020.

This document may not be reproduced in whole or in part without the prior written permission of IBM.
US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP
Schedule Contract with IBM Corp.

Contents

PREFACE **4**

 OVERVIEW4

 OBJECTIVES.....4

 FLOW4

 TOOLS.....5

 PREREQUISITES5

MILESTONE 1: THE HASH CHAIN **6**

 INSTALL PREREQUISITES6

 CREATE THE DEVELOPMENT ENVIRONMENT9

 CREATE A BASIC HASH CHAIN 11

 HELP PREVENT MODIFICATION OF THE HASH CHAIN 16

Preface

Overview

Underpinning blockchain is a set of well-known computer science concepts: linked lists, hash chains and the like. These form the basis of every blockchain in existence, and in order to fully grasp blockchain it is useful to be able to see these implemented from first principles.

Objectives

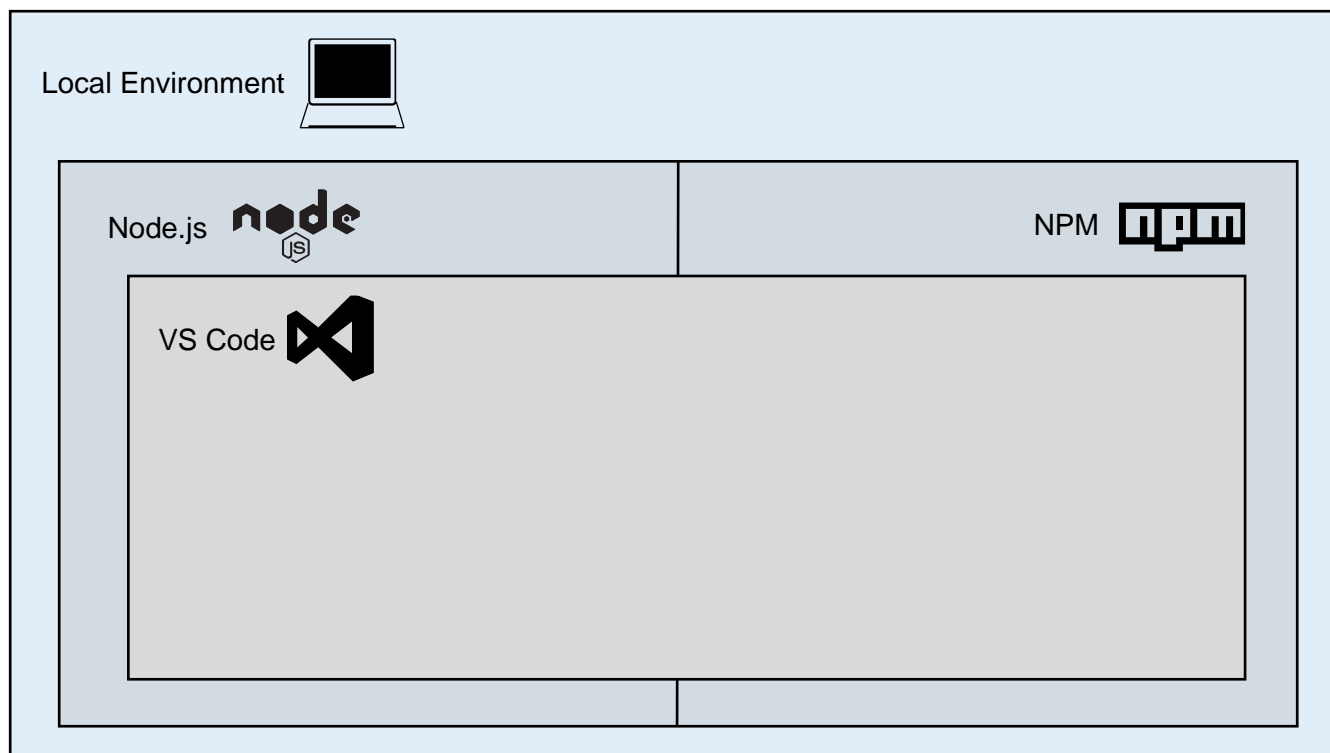
This short lab builds a complete JavaScript application to implement a very basic Hashchain to demonstrate some of the concepts of blockchain.

It does not introduce any concepts or technology unique to a real-world blockchain implementation. Nor does it attempt to implement any of the advanced features of a blockchain, such as distributed peers, smart contracts or consensus.

Flow

We will start by implementing the basic *block* data structure and linking instances of them together to form a block *chain*. We will look at how hash functions are used to provide continuity of the chain and will test this out by simulating the tampering of data within the chain.

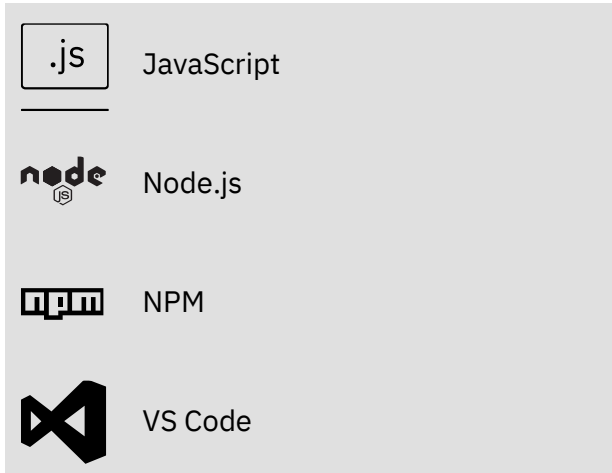
For this lab we will need to install Node.js and NPM onto our local environment. We will then install VS Code.



Tools

We will use JavaScript as the language for implementing our hash chain, with Node as the run time environment and NPM as the package manager.

We will use an editor that can edit these JavaScript files - VSCode is recommended, although if you wish to use an alternative editor, feel free to do so.



Prerequisites

Skill requirements:

- Basic knowledge of JavaScript is desirable.

Technical requirements (installed in the first section):

- Node and NPM
- An editor for JavaScript files (e.g. VSCode)

Milestone 1: The Hash Chain

Milestone Overview

This lab requires the completion of one milestone:

1. The Hash Chain

In this milestone, we will build a complete JavaScript application to implement a very basic Hashchain to demonstrate some of the concepts of blockchain.

Install prerequisites

1. Click the terminal window icon to bring up a new terminal window.



Figure 1-1 **Terminal Icon**

2. Enter:

```
node -v
```

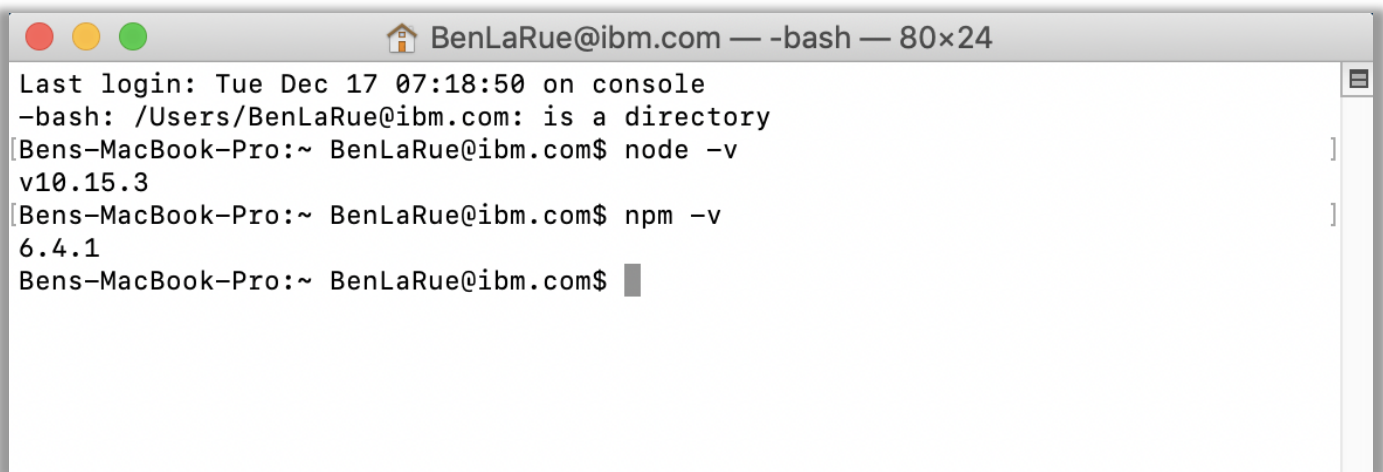
If you get a 'command not found' or similar error, you need to install Node from <https://nodejs.org/en/download/>. Simply download the version for your operating system and proceed through the installer. This lab was created using Node v8.16; other versions should work fine.

3. Enter:

```
npm -v
```

NPM should have been installed as part of Node; however if you do get a 'command not found' or similar error, you need to install NPM from <https://www.npmjs.com/get-npm>.

This lab was created using NPM 6.4.1; other versions should work fine.

A screenshot of a macOS terminal window. The title bar shows a home icon, the username 'BenLaRue@ibm.com', and the shell '-bash' with a window size of '80x24'. The terminal text shows the last login time as 'Tue Dec 17 07:18:50 on console'. The first command entered is '-bash: /Users/BenLaRue@ibm.com: is a directory'. The second command is '[Bens-MacBook-Pro:~ BenLaRue@ibm.com\$ node -v]', which outputs 'v10.15.3'. The third command is '[Bens-MacBook-Pro:~ BenLaRue@ibm.com\$ npm -v]', which outputs '6.4.1'. The prompt then returns to '[Bens-MacBook-Pro:~ BenLaRue@ibm.com\$]' with a cursor.

```
BenLaRue@ibm.com — -bash — 80x24
Last login: Tue Dec 17 07:18:50 on console
-bash: /Users/BenLaRue@ibm.com: is a directory
[Bens-MacBook-Pro:~ BenLaRue@ibm.com$ node -v
v10.15.3
[Bens-MacBook-Pro:~ BenLaRue@ibm.com$ npm -v
6.4.1
Bens-MacBook-Pro:~ BenLaRue@ibm.com$
```

Next we will install a code editor capable of editing JavaScript. For this lab we will use VSCode.

4. Navigate to <https://code.visualstudio.com/> and find the download for your operating system.

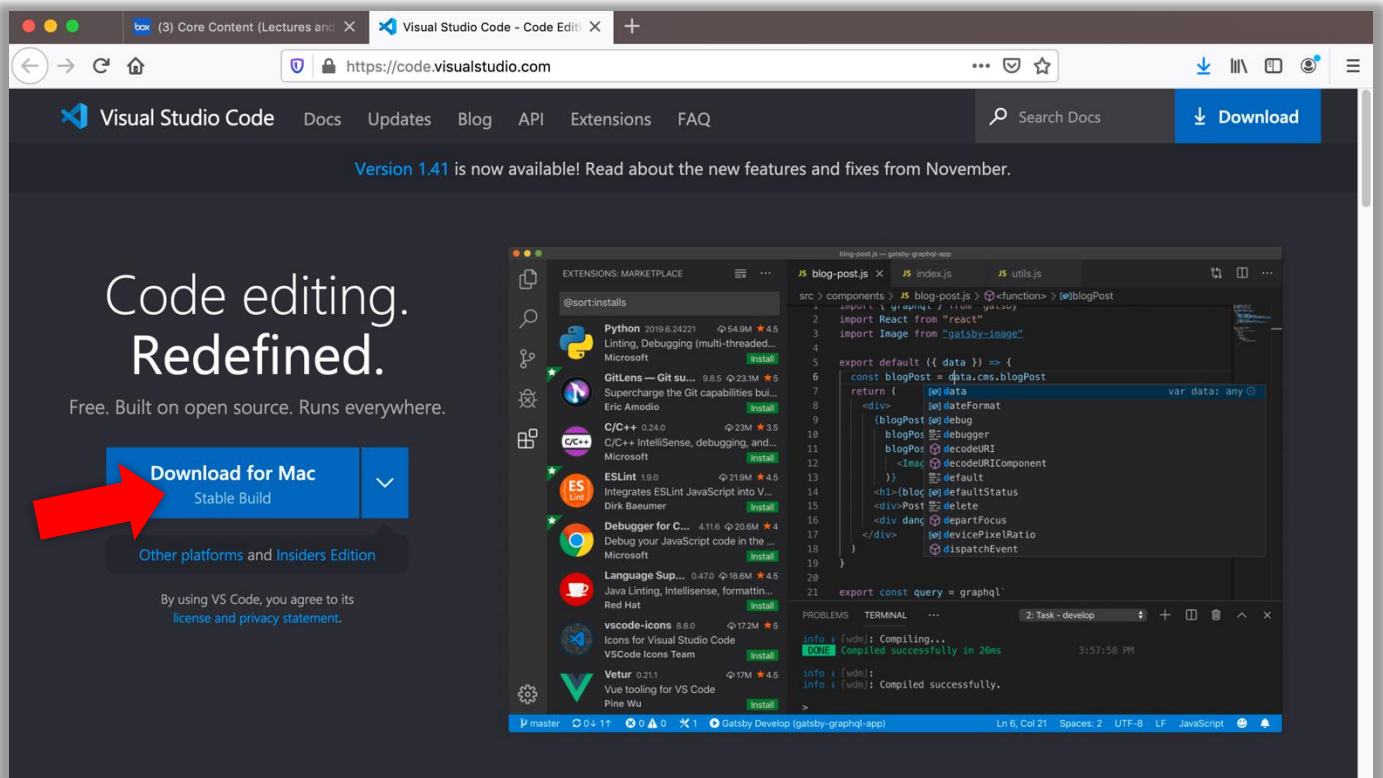


Figure 1-2 Visual Studio Code page

5. Proceed through the steps of the installer and you are now ready to continue completing Lab 2.

Create the development environment

1. Create and navigate to a folder in which you can work; in the terminal window enter:

```
mkdir ~/MyChain  
cd ~/MyChain
```

```
blockchain@ubuntu:~$ mkdir ~/MyChain  
blockchain@ubuntu:~$ cd ~/MyChain  
blockchain@ubuntu:~/MyChain$
```

In order to create the hashes for our blockchain, we will make use of a SHA256 library which is part of the Node crypto-js module. We must download this module and make it available to our application.

2. Enter:

```
npm install crypto-js
```

A node_modules folder will be created and the crypto-js library installed within it; warnings here can usually be ignored, as our application doesn't yet exist.

```
blockchain@ubuntu:~/MyChain$ npm install crypto-js  
npm WARN saveError ENOENT: no such file or directory, open '/home/blockchain/MyChain/package.json'  
npm notice created a lockfile as package-lock.json. You should commit this file.  
npm WARN endent ENOENT: no such file or directory, open '/home/blockchain/MyChain/package.json'  
npm WARN MyChain No description  
npm WARN MyChain No repository field.  
npm WARN MyChain No README data  
npm WARN MyChain No license field.  
  
+ crypto-js@3.1.9-1  
added 1 package from 1 contributor and audited 1 package in 1.452s  
found 0 vulnerabilities
```

3. Start your editor and begin creating the blockchain application. If you installed VS Code for example, enter:

```
code MyChain.js
```

This will load VS Code and create a new Javascript file for you to edit.

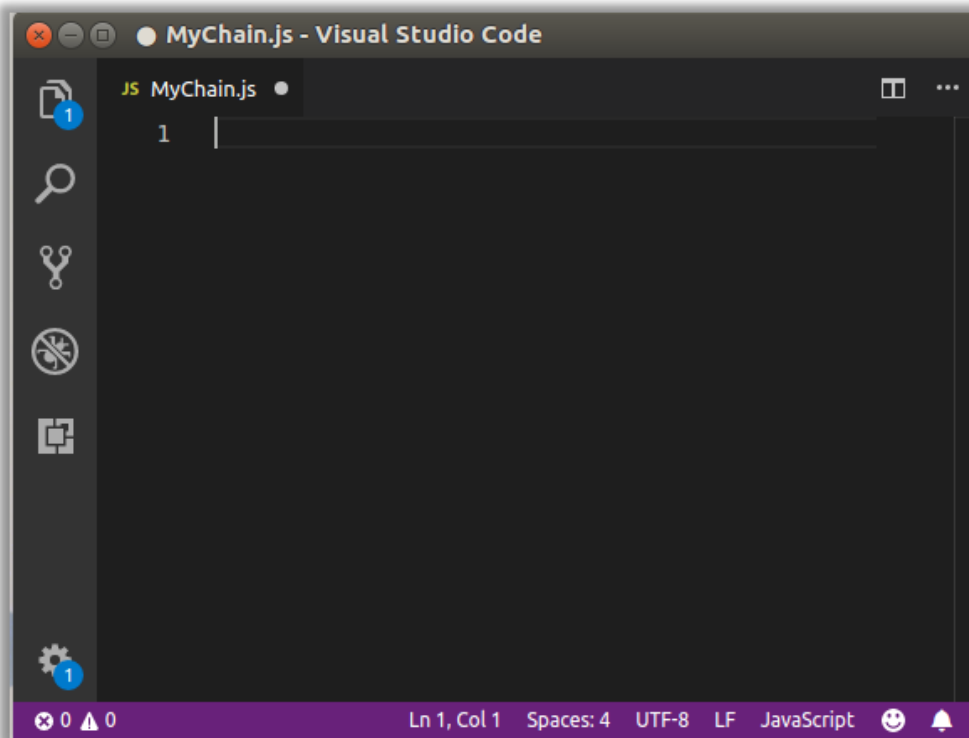


Figure 1-3 Visual Studio Code window

Note

Starting the VSCode editor for the first time might cause a web browser window to start. You can close this down.

Create a basic hash chain

1. In the JS file you created, provide the basic implementation of a block:

```
const SHA256 = require("crypto-js/sha256");

class Block {
  constructor(data, previousHash) {
    this.data = data;
    this.timestamp = Date.now();
    this.previousHash = previousHash;
    this.hash = this.getHash();
  }

  getHash() {
    return SHA256(this.previousHash + this.timestamp +
      JSON.stringify(this.data)).toString();
  }
}
```

In this implementation, when a Block is instantiated you need to provide it some data; in a real blockchain this is usually a representation of a set of transactions. You also need to supply the hash of the previous block in the chain. It uses this to generate a hash of the block, together with the data and current timestamp. This helps ensure immutability of the chain.

2. Implement in the same file a Blockchain class definition:

```
class Blockchain {  
  constructor() {  
    this.blockchain = [new Block("Genesis Block", "")];  
  }  
  getLastBlock() {  
    return this.blockchain[this.blockchain.length-1];  
  }  
  createBlock(data) {  
    this.blockchain.push(new Block(data, this.getLastBlock().hash));  
  }  
}
```

The class represents the blockchain as an array of blocks, with a mandatory block zero which is called our genesis block. There is also a method to return the most recent block in the chain, and a factory for creating new blocks based on the supplied input data.

3. At the end of the file, implement code to create and test an instance of the blockchain.

```
myChain = new Blockchain();  
console.log('\n-----\nNew blockchain created');  
console.log(myChain);  
  
myChain.createBlock("first set of transaction data");  
console.log('\n-----\nAdded a block');  
console.log(myChain);  
myChain.createBlock("another set of transaction data");  
console.log('\n-----\nAdded another block');  
console.log(myChain);
```

This instantiates a Blockchain and adds two blocks to it. At each stage, the contents of the blockchain are displayed on the console, to show how it grows.

4. Review your code and save it.

```
JS MyChain.js x
1  const SHA256 = require("crypto-js/sha256");
2
3  class Block {
4      constructor(data, previousHash) {
5          this.data = data;
6          this.timestamp = Date.now();
7          this.previousHash = previousHash;
8          this.hash = this.getHash();
9      }
10
11     getHash() {
12         return SHA256(this.previousHash + this.timestamp + JSON.stringify(this.data)).toString();
13     }
14 }
15
16 class Blockchain {
17     constructor() {
18         this.blockchain = [new Block("Genesis Block", '')];
19     }
20     getLastBlock() {
21         return this.blockchain[this.blockchain.length-1];
22     }
23     createBlock(data) {
24         this.blockchain.push(new Block(data, this.getLastBlock().hash));
25     }
26 }
27
28
29 myChain = new Blockchain();
30 console.log('\n-----\nNew blockchain created');
31 console.log(myChain);
32
33 myChain.createBlock("first set of transaction data");
34 console.log('\n-----\nAdded a block');
35 console.log(myChain);
36
37 myChain.createBlock("another set of transaction data");
38 console.log('\n-----\nAdded another block');
39 console.log(myChain);
```

5. Return to your terminal window and run your Javascript application, enter:

```
node MyChain.js
```

```
blockchain@ubuntu:~/MyChain$ node MyChain.js
```

```
-----  
New blockchain created  
Blockchain {  
  blockchain:  
    [ Block {  
      data: 'Genesis Block',  
      timestamp: 1541436759916,  
      previousHash: '',  
      hash: 'cd96a78fe54df359d69544cbdd43d19153316272f1035ff4b08d4b1d3ec400f' } ] ]  
-----  
Added a block  
Blockchain {  
  blockchain:  
    [ Block {  
      data: 'Genesis Block',  
      timestamp: 1541436759916,
```

6. Review the output from the program.

New blockchain created

Blockchain {

blockchain:

[Block {

data: 'Genesis Block',

timestamp: 1520596059987,

previousHash: '',

hash: 'df266d26ed337142cd00d128e090cdaf10a220a8e65ad6eca894577276c8e7fc' }] }

Added a block

Blockchain {

blockchain:

[Block {

data: 'Genesis Block',

timestamp: 1520596059987,

previousHash: '',

hash: 'df266d26ed337142cd00d128e090cdaf10a220a8e65ad6eca894577276c8e7fc' },

Block {

data: 'first set of transaction data',

timestamp: 1520596059993,

previousHash: 'df266d26ed337142cd00d128e090cdaf10a220a8e65ad6eca894577276c8e7fc',

hash: 'c02694901a64df111b5648da5181abfb65702ec193074bda4cd99af716596254' }] }

Added another block

Blockchain {

blockchain:

[Block {

data: 'Genesis Block',

timestamp: 1520596059987,

previousHash: '',

hash: 'df266d26ed337142cd00d128e090cdaf10a220a8e65ad6eca894577276c8e7fc' },

Block {

data: 'first set of transaction data',

timestamp: 1520596059993,

previousHash: 'df266d26ed337142cd00d128e090cdaf10a220a8e65ad6eca894577276c8e7fc',

hash: 'c02694901a64df111b5648da5181abfb65702ec193074bda4cd99af716596254' },

Block {

data: 'another set of transaction data',

timestamp: 1520596059994,

previousHash: 'c02694901a64df111b5648da5181abfb65702ec193074bda4cd99af716596254',

hash: '2be785a4020990626de8493ffd2c8dea7e346b06aec38665429e7430efa04791' }] }

Help prevent modification of the hash chain

We will now implement code in the hash chain to check that the data stored within it has not been tampered. We will then test it by modifying the chain illegally.

1. Insert a new method into the Blockchain class to check the validity of the chain.

```
isBlockchainValid() {  
  for (let i=1; i<this.blockchain.length; i++) {  
    let currentBlock = this.blockchain[i];  
    let previousBlock = this.blockchain[i-1];  
    if ((currentBlock.previousHash !== previousBlock.hash) ||  
        (currentBlock.hash !== currentBlock.getHash())) {  
      return false;  
    }  
  }  
  return true;  
}
```

This method ensures that the hash of each block is correctly refers to the hash recorded in the previous block, and that the contents of the block have been hashed correctly. This helps prevent the data in a recorded block from being modified without rendering the validity check invalid.

2. At the end of the entire program, add code to call the new method, modify a block 'illegally' and then call the method again.

```
console.log('Is the chain valid: ' + myChain.isBlockchainValid());  
myChain.blockchain[1].data = "changed set of transactions";  
console.log('Modified transaction data in a block');  
console.log('Is the chain valid: ' + myChain.isBlockchainValid());
```


3. Review your code and save it.

```

JS MyChain.js x
1  const SHA256 = require("crypto-js/sha256");
2
3  class Block {
4      constructor(data, previousHash) {
5          this.data = data;
6          this.timestamp = Date.now();
7          this.previousHash = previousHash;
8          this.hash = this.getHash();
9      }
10
11     getHash() {
12         return SHA256(this.previousHash + this.timestamp + JSON.stringify(this.data)).toString();
13     }
14 }
15
16 class Blockchain {
17     constructor() {
18         this.blockchain = [new Block("Genesis Block", '')];
19     }
20     getLastBlock() {
21         return this.blockchain[this.blockchain.length-1];
22     }
23     createBlock(data) {
24         this.blockchain.push(new Block(data, this.getLastBlock().hash));
25     }
26     isBlockchainValid() {
27         for (let i=1; i<this.blockchain.length; i++) {
28             let currentBlock = this.blockchain[i];
29             let previousBlock = this.blockchain[i-1];
30             if ((currentBlock.previousHash !== previousBlock.hash) ||
31                 (currentBlock.hash !== currentBlock.getHash())) {
32                 return false;
33             }
34         }
35         return true;
36     }
37 }
38
39
40 myChain = new Blockchain();
41 console.log('\n-----\nNew blockchain created');
42 console.log(myChain);
43
44 myChain.createBlock("first set of transaction data");
45 console.log('\n-----\nAdded a block');
46 console.log(myChain);
47
48 myChain.createBlock("another set of transaction data");
49 console.log('\n-----\nAdded another block');
50 console.log(myChain);
51
52 console.log('Is the chain valid: ' + myChain.isBlockchainValid());
53 myChain.blockchain[1].data = "changed set of transactions";
54 console.log('Modified transaction data in a block');
55 console.log('Is the chain valid: ' + myChain.isBlockchainValid());

```

4. Back in the terminal window, run the modified program.

```
node MyChain.js
```

5. Review the output.

```
hash: 102b574...e5611430dd226400  
Is the chain valid: true  
Modified transaction data in a block  
Is the chain valid: false  
blockchain@ubuntu:~/MyChain$
```

Notice that once the chain has been illegally modified, it is no longer valid.

This is only a very basic implementation of a single-process centralised hashchain. Real-world blockchains are significantly more complicated than this: they are distributed processing systems where different network participants use a process of consensus to agree on the contents of a block. The underlying data structure is based on the fundamental principles shown here however, which help prove immutability of the data stored within the chain.

Congratulations! You have completed this lab.

NOTES

NOTES

[illegible]



© Copyright IBM Corporation 2020.

The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. This information is based on current IBM product plans and strategy, which are subject to change by IBM without notice. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way.

IBM, the IBM logo and ibm.com are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.



Please Recycle
