6. Write a Node.js Program to create a HTTP server using HTTP Module

***Program:***

const http = require('http');

const server = http.createServer((req, res) => {

if (req.url === '/') {

res.write('<h1>Hello, Node.js!</h1>');

}

res.end();

});

server.listen(5000);

console.log(`The HTTP Server is running on port 5000`);

**The following starts the HTTP server:**

**node server.js**

**Output:**

**The HTTP Server is running on port 5000**

**Explanation:**

First, create a new file called server.js and include the http module by using the require() function:

**const http = require('http');**

Second, create an HTTP server using the createServer() method of the http object.

**const server = http.createServer((req, res) => {**

**if (req.url === '/') {**

**res.write('<h1>Hello, Node.js!</h1>');**

**}**

**res.end();**

**});**

The createServer() accepts a callback that has two parameters: HTTP request (req) and response (res). Inside the callback, we send an HTML string to the browser if the URL is / and end the request.

Third, listen to the incoming HTTP request on the port 5000:

**server.listen(5000);**

**console.log(`The HTTP Server is running on port 5000`);**

7. Write a Node.js Program to Perform operations on files a) Read files b) Create files c) Update files d)Delete files e)Rename files

**Create a js file named main.js having the following code to open a file input.txt for reading and writing.**

var fs = require("fs");

// Asynchronous - Opening File

console.log("Going to open file!");

fs.open('input.txt', 'r+', function(err, fd) {

if (err) {

return console.error(err);

}

console.log("File opened successfully!");

});

Now run the main.js to see the result −

$ node main.js

**Verify the Output.**

Going to open file!

File opened successfully!

**Write data into file**

var fs = require("fs");

console.log("Going to write into existing file");

fs.writeFile('input.txt', 'Simply Easy Learning!', function(err) {

if (err) {

return console.error(err);

}

console.log("Data written successfully!");

console.log("Let's read newly written data");

fs.readFile('input.txt', function (err, data) {

if (err) {

return console.error(err);

}

console.log("Asynchronous read: " + data.toString());

});

});

**Verify the Output.**

Going to write into existing file

Data written successfully!

Let's read newly written data

Asynchronous read: Simply Easy Learning!

**Read data from a file**

var fs = require("fs");

var buf = new Buffer(1024);

console.log("Going to open an existing file");

fs.open('input.txt', 'r+', function(err, fd) {

if (err) {

return console.error(err);

}

console.log("File opened successfully!");

console.log("Going to read the file");

fs.read(fd, buf, 0, buf.length, 0, function(err, bytes){

if (err){

console.log(err);

}

console.log(bytes + " bytes read");

// Print only read bytes to avoid junk.

if(bytes > 0){

console.log(buf.slice(0, bytes).toString());

}

});

});

**Verify the Output.**

Going to open an existing file

File opened successfully!

Going to read the file

97 bytes read

Tutorials Point is giving self learning content

to teach the world in simple and easy way!!!!!

**To delete a File**

var fs = require("fs");

console.log("Going to delete an existing file");

fs.unlink('input.txt', function(err) {

if (err) {

return console.error(err);

}

console.log("File deleted successfully!");

});

Output:

Going to delete an existing file

File deleted successfully!

## 8. Write a blockchain application in javascript for the simple transaction

## How To Create Blockchain In JavaScript

Now if we need to create a simple Blockchain in Node.js, then we need to have one **Genesis Block**

You can refer to Genesis Block as the first block of Blockchain because Blockchain is an array of blocks and Genesis is the first block of the **Blockchain**.

**Genesis Block**

The **genesis block** is the first **block** of the blockchain. The **genesis block** is generally hardcoded in the applications that utilize its blockchain. The Genesis Block is also known as Block Zero or Block 0. It is an ancestor that every Blockchain network’s block that can be traced to its origin back. Remember how every block in the Blockchain is linked back to the previous block using the hash in the block header? You keep going back, and you realize every block is hence connected to the genesis block.

Our Genesis Block contains the following fields or properties.

1. timestamp
2. lastHash
3. hash
4. data

So, if we combine all these properties into one object, then it will become a Genesis Block. So based on this block, we will mine the second block. Then from the second block, we will extract the third block and so on. That is why we need a Genesis Block to start a Blockchain.

Now, let us start a project by creating a project folder and then start building awesome blockchain using Javascript.

**Step 1: Create a project and create a Genesis Data**

The first step is to create a project. Type the following command.

**mkdir** cryptochain

Go inside that folder.

**cd** cryptochain

Now, generate the **package.json** file in the case in future, we need to install any node modules.

npm init -y

Remember, I am using Node v11.3.0.

Next step is to create a new file inside the root called **genesis.js**and add the following code.

*// genesis.js*

const GENESIS\_DATA = {

timestamp: Date.now(),

lastHash: '64b7edc786326651e031a4d12d9838d279571946d8c9a5d448c70db94b0e143f',

hash: 'c671c84681b9d682b9fd43b2a2ef01a343eab7cfa410df9835f8165007d38467',

data: 'krunal'

};

module.exports = { GENESIS\_DATA };

Here, we have taken an object with its initial values.

As I have defined earlier, our block contains the four properties. This is a genesis data, so we need to hardcode this values.

The properties **timestamp**and **data**are you can understand. But I have used **SHA256**algorithm to convert the simple text into hash text.

Actual text for lastHash is a**krunal,**and actual text for the hash is a **krunalHash.**

You can verify it by going to this URL: <https://passwordsgenerator.net/sha256-hash-generator/>

Here, you can enter my name as a text, and it will give the **SHA256** hash of that text. Now, you need to convert that text to lowercase by using Javascript’s toLowercase(). That is it. You will find the exact hash like me. Cool!! So now, you have Genesis Data is ready. The next thing is to create a Block from this data.

**Step 2: Create a Block**

Inside your project root, create a file called **block.js**and add the following code.

*// block.js*

const { GENESIS\_DATA } = require('./genesis.js');

**class** **Block** {

**constructor**({timestamp, lastHash, hash, **data**}) {

**this**.timestamp = timestamp;

**this**.lastHash = lastHash;

**this**.hash = hash;

**this**.**data** = **data**;

}

static genesis() {

**return** new **this**(GENESIS\_DATA);

}

}

module.exports = Block;

So, here we have imported the GENESIS\_DATA for our block.

Then we have defined the **Block**class and passed the parameters to the constructor when we create an object of the Block.

We have also defined the **static**method called **genesis()**which is responsible for returning the **Genesis Block** for our blockchain.

Remember, we have previously defined the GENESIS\_DATA not block. After creating an object of this class, it will become a Genesis Block.

**Step 3: Create a hash based on a previous block.**

We need to define a function that can create a hash based on the previous block’s hash.

So, first let us create a new file inside the root called **crypto-hash.js**and add the following code inside it.

*// crypto-hash.js*

**const** crypto = require('crypto');

**const** cryptoHash =(...inputs) => {

**const** hash = crypto.createHash('sha256');

hash.update(inputs.sort().join(' '));

**return** hash.digest('hex');

}

module.exports = cryptoHash;

Now, to create a Hash, we need the three properties.

1. timestamp
2. lastHash
3. data

So, we have first required the **crypto module**provided by the Node.js.

Then we have defined the function called **cryptoHash()**which will accept the inputs. Here we have used spread operator which is the syntax of ES6.

Inside that function, we have called the **createHash()**method and pass the **sha256**as a parameter. That means we need to create a hash based on the **sha256** algorithm.

Then we have sorted and join that three parameters and return its hex values.

So, finally, we can get the current block’s hash based on the previous block’s three properties.

**Step 4: Mine a new Block based on a previous Block**

So, we got the Genesis Block and the hash of that block. Now, we need to write the function that can generate a new Block based on the previous block. Right now, in our case, it is a Genesis Block.

Now, import the **crypto-hash**module inside the **block.js**file and create a new function called **mineBlock()**and pass the two parameters.

1. lastBlock
2. data

So, our final **block.js**file looks like this.

*// block.js*

const { GENESIS\_DATA } = require('./genesis.js');

const cryptoHash = require('./crypto-hash');

**class** **Block** {

**constructor**({timestamp, lastHash, hash, **data**}) {

**this**.timestamp = timestamp;

**this**.lastHash = lastHash;

**this**.hash = hash;

**this**.**data** = **data**;

}

static genesis() {

**return** new **this**(GENESIS\_DATA);

}

static mineBlock({lastBlock, **data**}) {

const timestamp = Date.now();

const lastHash = lastBlock.hash;

**return** new **this**({

timestamp,

lastHash,

**data**,

hash: cryptoHash(timestamp, lastHash, **data**)

});

}

}

module.exports = Block;

So, here we have defined the method called **mineBlock**and pass the two parameters. The **mineBlock()**will return a complete new Block based on the previous block.

Now, it is the time to create a Blockchain from these blocks.

**Step 5: Create a Blockchain**

We have completed the step of creating a Genesis Block and mining a new Block. Now it is the time to develop a blockchain. That is why create a new file inside the root called **blockchain.js**and add the following code inside it.

*// blockchain.js*

const Block = require('./block');

**class** **Blockchain** {

**constructor**() {

**this**.chain = [Block.genesis()];

}

addBlock({ **data** }) {

const newBlock = Block.mineBlock({

lastBlock: **this**.chain[**this**.chain.length-1],

**data**

});

**this**.chain.push(newBlock);

}

}

module.exports = Blockchain;

So, first, we have imported the **Block.js**file and then create a class called Blockchain.

The Blockchain class is responsible for adding a new block inside the blockchain.

The blockchain is an array of the blocks starts with Genesis block.

So in the constructor, we have not defined an empty array. Instead, we have filled the chain array with the Genesis block.

Then we have defined one function called **addBlock()**which accepts the data.

Now, before adding a new block to the chain array, we need to mine it. That is why we have called the Block class’s **mineBlock()**method to extract a new block.

For mining a new block, we need a previous block and data. That is why we have used this code **this.chain[this.chain.length-1]**because it will return the last block in the blockchain and data is already we are passing to that function. So we get the new block based on the previous block and data.

Next step is to add that newly mined block inside our blockchain, and that is it.

The last step is to run this project and get the blockchain.

**Step 6: Run the project and get the blockchain.**

Now, the last step is to create a file inside the root of the project called **server.js**and add the following code.

*// server.js*

**const** Blockchain = require('./blockchain');

**const** Block = require('./block');

**const** blockchain = **new** Blockchain();

**for**(**let** i=0; i<5; i++) {

**const** newData = 'krunal'+i;

blockchain.addBlock({data: newData});

}

console.log(blockchain);

So, here we have imported both the **block.js**and **blockchain.js**file and created an object of the Blockchain.

Then we will loop through that blockchain and add the data to that blockchain and created the six blocks in the blockchain because one is Genesis and five based on previous blocks.

Go to the terminal and start the node server.

**node** **server**

You will see an output like this.

[](https://appdividend.com/wp-content/uploads/2018/12/How-To-Create-Blockchain-In-JavaScript.png)