

# Implement Simple Proof-of-Work Mining in JS

Exercises

---

The goal of this exercise is to **understand the principles of blockchain technology** and what happens during **mining**. You will modify an existing simple JS-based blockchain network implementation to add **proof-of-work mining** inside its code.

The basic concept of blockchain is quite simple: a distributed ledger that maintains a continuously growing list of ordered records (blocks). First, you will run the simple JavaScript-based **model of blockchain** called

“Naivechain”, **add nodes** and **mine blocks**. Most of the code is based on this project: <https://github.com/lhartikk/naivechain>. Thanks to the original authors.

Later, you will touch upon the principles of **proof-of-work mining** by modifying the existing code to add a **proof-of-work calculation** (by adding **nonce** + **timestamp** in the blocks).

For this exercise you will need **Linux** or Linux-like command-line environment. We use a Debian-based distribution, but each one will work fine in most cases. Also, you will need “**node.js**”, “**npm**” and “**cURL**” installed.

## Project Dependencies

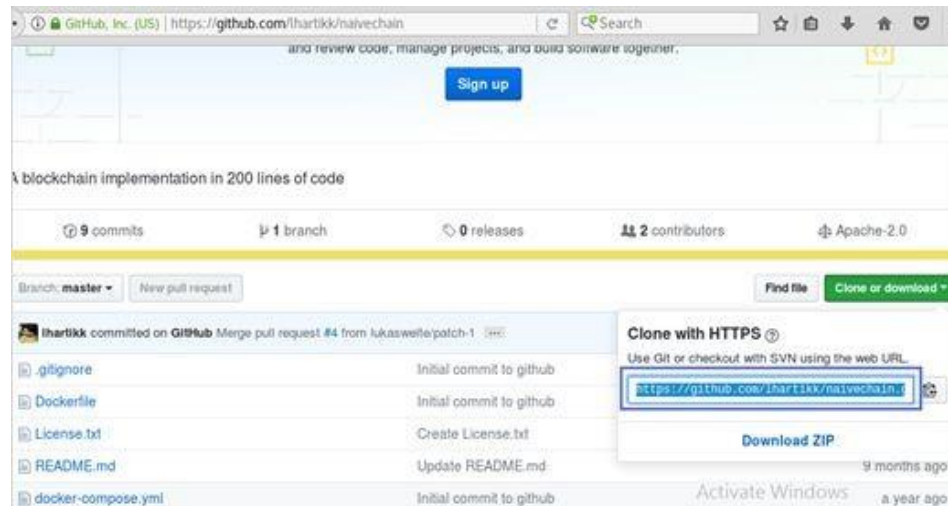
- NodeJS**v12.18** <https://nodejs.org/en/download/>
- NPM **v6.14**
- GIT **v2.28** <https://git-scm.com/downloads>

You may choose only one

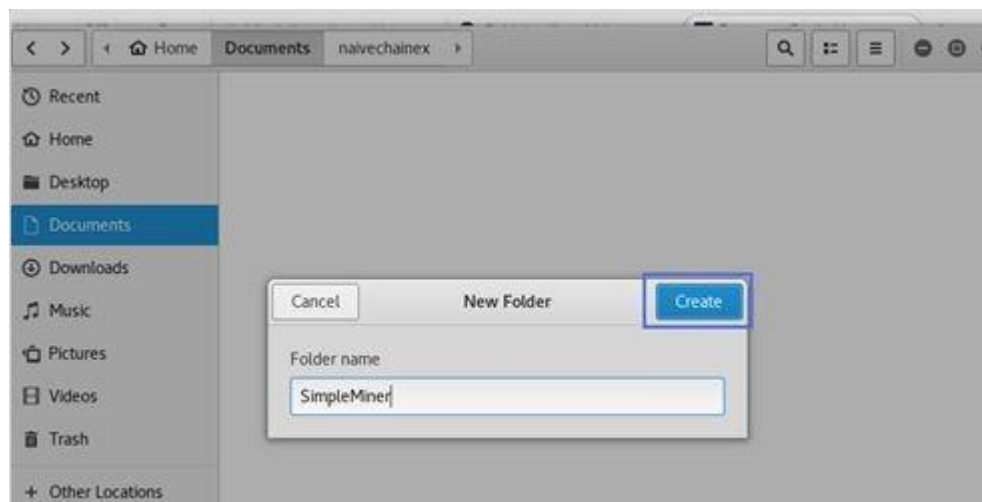
- Postman **v7.31** <https://www.postman.com/downloads/>
- cURL **v7.72** <https://curl.haxx.se/download.html>

# 1. Clone the “Naivechain” from GitHub

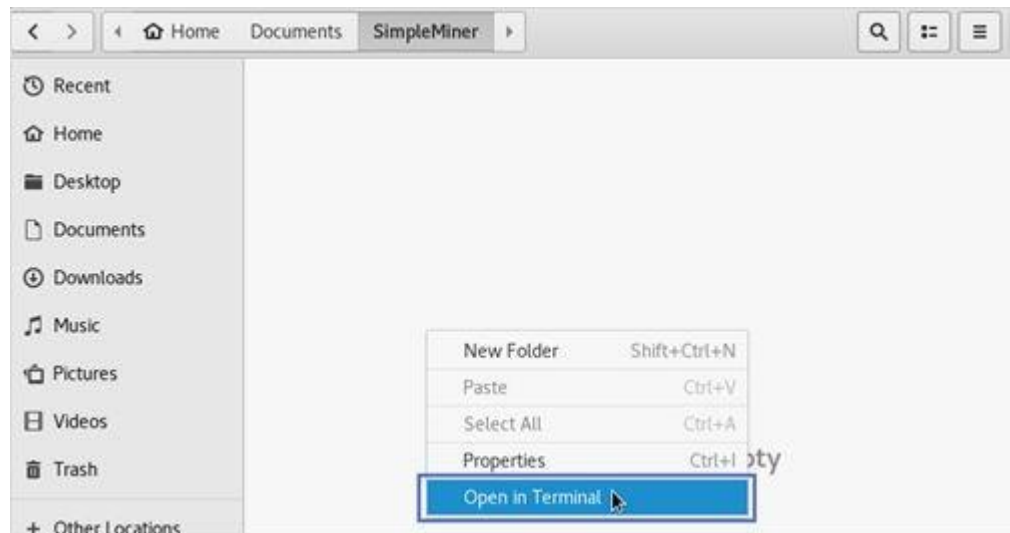
1. Go to <https://github.com/lhartikk/naivechain>. Click the green button “Clone or download” and copy the repository address.



2. Create directory for the project. Name it “SimpleMiner”.



3. Open the directory and right click to **open** it in the **terminal**.



4. First **check the versions** of “**node.js**”, “**npm**” and “**cURL**”. In terminal type: “**node -v**”, “**npm -v**” and “**curl -V**” (for cURL type capital “V” for version). If you haven’t them installed on your computer, please install them.

```
node -v  
npm -v  
curl -V
```

```
root@RIS: ~/Documents/SimpleMiner  
File Edit View Search Terminal Help  
root@RIS:~/Documents/SimpleMiner# node -v  
v9.2.1  
root@RIS:~/Documents/SimpleMiner# npm -v  
5.5.1  
root@RIS:~/Documents/SimpleMiner# curl -V  
curl 7.57.0 (i686-pc-linux-gnu) libcurl/7.57.0 OpenSSL/1.0.2l zlib/1.2.8 libpsl/  
0.18.0 (+libidn2/2.0.2) libssh2/1.8.0 nghttp2/1.25.0 librtmp/2.3  
Release-Date: 2017-11-29  
Protocols: dict file ftp ftps gopher http https imap imaps ldap ldaps pop3 pop3s  
rtmp rtsp scp sftp smb smbs smtp smtps telnet tftp  
Features: AsynchDNS IPv6 Largefile GSS-API Kerberos SPNEGO NTLM NTLM_WB SSL libz  
TLS-SRP HTTP2 UnixSockets HTTPS-proxy PSL  
root@RIS:~/Documents/SimpleMiner#
```

5. Clone the GitHub repository:

```
git clone https://github.com/lhartikk/naivechain.git
```

```
root@RIS:~/Documents/SimpleMiner# node -v
v9.2.1
root@RIS:~/Documents/SimpleMiner# npm -v
5.5.1
root@RIS:~/Documents/SimpleMiner# curl -V
curl 7.57.0 (i686-pc-linux-gnu) libcurl/7.57.0 OpenSSL/1.0.2l zlib/1.2.8 libpsl/
0.18.0 (+libidn2/2.0.2) libssh2/1.8.0 nghttp2/1.25.0 librtmp/2.3
Release-Date: 2017-11-29
Protocols: dict file ftp ftps gopher http https imap imaps ldap ldaps pop3 pop3s
rtmp rtsp scp sftp smb smbs smtp smtps telnet tftp
Features: AsynchDNS IPv6 Largefile GSS-API Kerberos SPNEGO NTLM NTLM_WB SSL libz
TLS-SRP HTTP2 UnixSockets HTTPS-proxy PSL
root@RIS:~/Documents/SimpleMiner# git clone https://github.com/lhartikk/naivecha
in.git
Cloning into 'naivechain'...
remote: Counting objects: 30, done.
remote: Total 30 (delta 0), reused 0 (delta 0), pack-reused 30
Unpacking objects: 100% (30/30), done.
root@RIS:~/Documents/SimpleMiner#
```

6. Then go to “naivechain” directory by typing “cd naivechain” in the terminal.

```
cd naivechain
```

```
root@RIS:~/Documents/SimpleMiner# cd naivechain
root@RIS:~/Documents/SimpleMiner/naivechain#
```

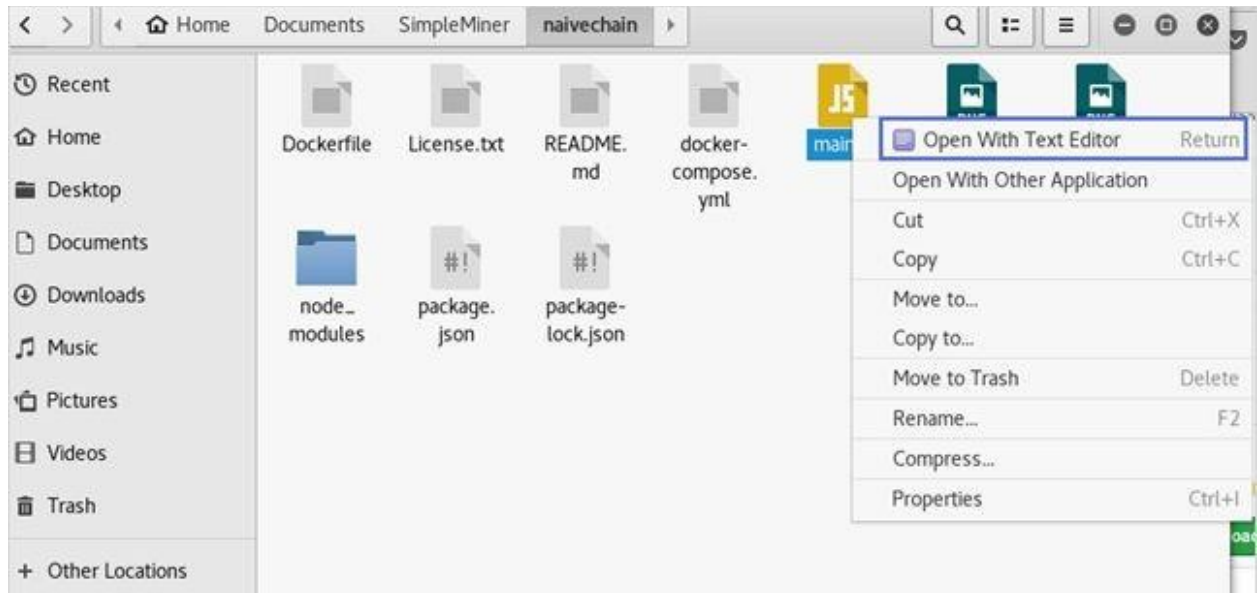
7. Now type “npm install” to get the npm packages. You should see something like this: “added 67 packages in 7.056s”.

```
npm install
```

```
root@RIS:~/Documents/SimpleMiner# cd naivechain
root@RIS:~/Documents/SimpleMiner/naivechain# npm install
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN naivechain@1.0.0 No repository field.
npm WARN naivechain@1.0.0 No license field.
added 67 packages in 7.056s
root@RIS:~/Documents/SimpleMiner/naivechain#
```

## 2. Open the File “main.js” and take a Look at the Code Inside

1. **Open** the file “**main.js**” in your preferred text editor or IDE. For example, you can open it with “**Text Editor**”.



2. Let's pay attention to the **block structure**. To keep things as simple as possible, our example contains only the most necessary elements: **index**, **timestamp**, **data**, **hash** and **previous hash**. Following the blockchain concept, the **hash** of the previous block must be **found** in the block to **preserve** the chain integrity.



3. In the code, we have **class Block** with a constructor and **five fields**.

```
class Block {
  constructor(index, previousHash, timestamp, data, hash) {
    this.index = index;
    this.previousHash = previousHash.toString();
    this.timestamp = timestamp;
    this.data = data;
    this.hash = hash.toString();
  }
}
```

4. Find the “**calculateHash**” function in the code. You will have to work on it to implement **proof-of-work mining**. This function is the critical **moment of mining** when the miner calculates the hash for the next block.

```
var calculateHash = (index, previousHash, timestamp, data) => {
  return CryptoJS.SHA256(index + previousHash + timestamp + data).toString();
};
```

The block needs to be **hashed** to keep the integrity of the data. A **SHA-256 hash function** is used to consume the content of the block. It should be noted that in our example, we want to learn where the mining happens and illustrate how the blockchain works. In our case, this hash has nothing to do with the real process of “mining”, since there is no Proof of Work problem to solve but it illustrates the **process of block generation**. In real blockchains, a Proof of Work is a piece of data which is **difficult** (costly, time-consuming) to produce but **easy for others** to verify and which **satisfies** certain requirements.

Producing a Proof Of Work can be a random process with a low probability of success, so a lot of trial and error is required on average before a valid proof of work is generated. In order for a block to be accepted by network participants, miners must complete a Proof of Work which covers all of the data in the block. Due to the very low probability of successful generation, this makes it **unpredictable** as to which node in the network will be able to **generate** the next block. In our example, this kind of competition to find the next block is not implemented and **any** generated block is considered valid.

5. When we want to **generate** a block, first we must know the **hash of the previous block** because this is the chain of the blockchain. Next, we must **create** the rest of the **required content** (index, hash, data and timestamp). The **Block data** is something that is provided by the end-user which you will see later.



```

var generateNextBlock = (blockData) => {
  var previousBlock = getLatestBlock();
  var nextIndex = previousBlock.index + 1;
  var nextTimestamp = new Date().getTime() / 1000;
  var nextHash = calculateHash(nextIndex, previousBlock.hash, nextTimestamp, blockData);
  return new Block(nextIndex, previousBlock.hash, nextTimestamp, blockData, nextHash);
};

```

6. An in-memory JavaScript **array** is used to **store the blockchain**. The **first block** of the blockchain is always a so-called “**genesis block**”, which is hard-coded. We create it with a function which returns a new Block with usual attributes: the Block index is 0, the “**previousHash**” doesn’t exist since it’s the first block so we give it a string value of “0”. The current Block **hash** is hardcoded and **data field** contains the string “**my genesis block**”.

```

var getGenesisBlock = () => {
  return new Block(0, "0", 1465154705, "my genesis block!!",
    "816534932c2b7154836da6afc367695e6337db8a921823784c14378abed4f7d7");
};

var blockchain = [getGenesisBlock()];

```



7. At any given time, we must be able to **validate** if a block or a chain of blocks are valid in terms of **integrity**. This is especially true when we receive **new blocks** from other **nodes** and must decide whether to **accept** them or not.

```
var isValidNewBlock = (newBlock, previousBlock) => {  
  if (previousBlock.index + 1 !== newBlock.index) {  
    console.log('invalid index');  
    return false;  
  } else if (previousBlock.hash !== newBlock.previousHash) {  
    console.log('invalid previoushash');  
    return false;  
  } else if (calculateHashForBlock(newBlock) !== newBlock.hash) {  
    console.log(typeof (newBlock.hash) + ' ' + typeof calculateHashForBlock(newBlock));  
    console.log('invalid hash: ' + calculateHashForBlock(newBlock) + ' ' + newBlock.hash);  
    return false;  
  }  
  return true;  
};
```

8. There should always be only **one** explicit **set of blocks** in the chain at a given time. In case of **conflicts** (e.g. two nodes both generate block number 72) we choose the chain that has the **longest number of blocks**.



9. In our model of blockchain, we compare the length of the **new blockchain** with the **length of the existing blockchain**. The **longest chain** is accepted as the **valid** blockchain.

```

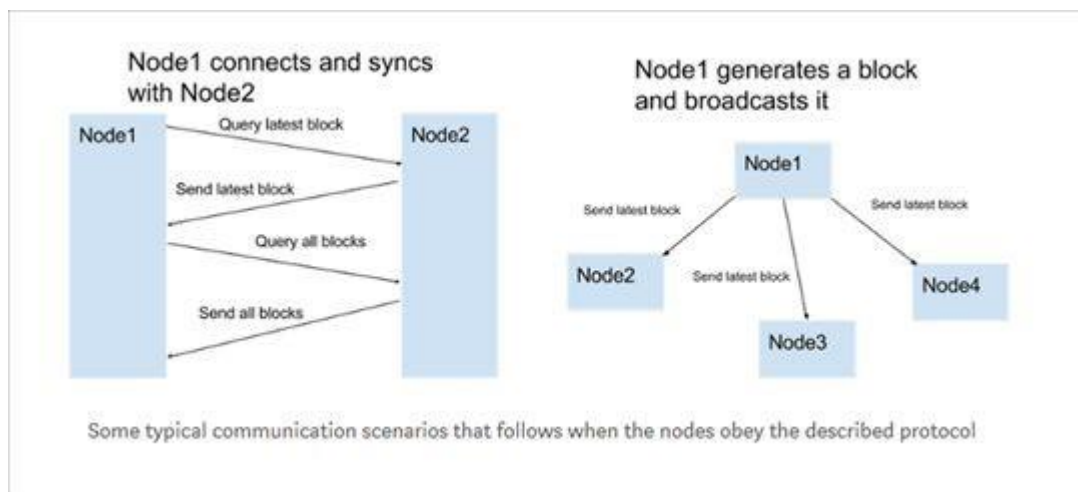
var replaceChain = (newBlocks) => {
  if (isValidChain(newBlocks) && newBlocks.length > blockchain.length) {
    console.log('Received blockchain is valid. Replacing current blockchain with received blockchain');
    blockchain = newBlocks;
    broadcast(responseLatestMsg());
  } else {
    console.log('Received blockchain invalid');
  }
};

```

10. For the purpose of modeling the blockchain and mining blocks, the program has a system of **communication** between nodes. An essential part of a node is to **share** and sync the blockchain with **other nodes**. The following **rules** are used to **keep the network in sync**.

- When a node **generates a new block**, it broadcasts it to the **network**
- When a node **connects to a new peer**, it queries for the latest block
- When a node encounters a block that has an **index larger than the current known block**, it either adds the block to its current chain or queries for the full blockchain.

No automatic peer discovery is used. The location (URLs) of peers must be **manually** added.



11. The user must be able to **control the node** in some way. This is done by setting up an **HTTP server**.

```

var initHttpServer = () => {
  var app = express();
  app.use(bodyParser.json());

  app.get('/blocks', (req, res) => res.send(JSON.stringify(blockchain)));
  app.post('/mineBlock', (req, res) => {
    var newBlock = generateNextBlock(req.body.data);
    addBlock(newBlock);
    broadcast(responseLatestMsg());
    console.log('block added: ' + JSON.stringify(newBlock));
    res.send();
  });
  app.get('/peers', (req, res) => {
    res.send(sockets.map(s => s._socket.remoteAddress + ':' + s._socket.remotePort));
  });
  app.post('/addPeer', (req, res) => {
    connectToPeers([req.body.peer]);
    res.send();
  });
  app.listen(http_port, () => console.log('Listening http on port: ' + http_port));
};

```

The API enables the user to **interact** with the node in the following ways:

- GET requests to **/blocks** will make a **get request** and **list all blocks** in the existing blockchain
- POST requests to **/mineBlock** will send a **post request** which will **create a new block** with a content given by the user, calculate the new block's hash and **add** this new block to the blockchain. Then later it will broadcast a message to other peers.
- GET requests to **/peers** will make a **get request** and take an **array with existing peers**. If there are no peers we will receive an empty array [].
- POST requests to **/addPeer** will send a **post request** and connect **peer** to the given one

The most straightforward way to control the node is by using cURL.

For example, to **get all blocks** from the node, issue this command:

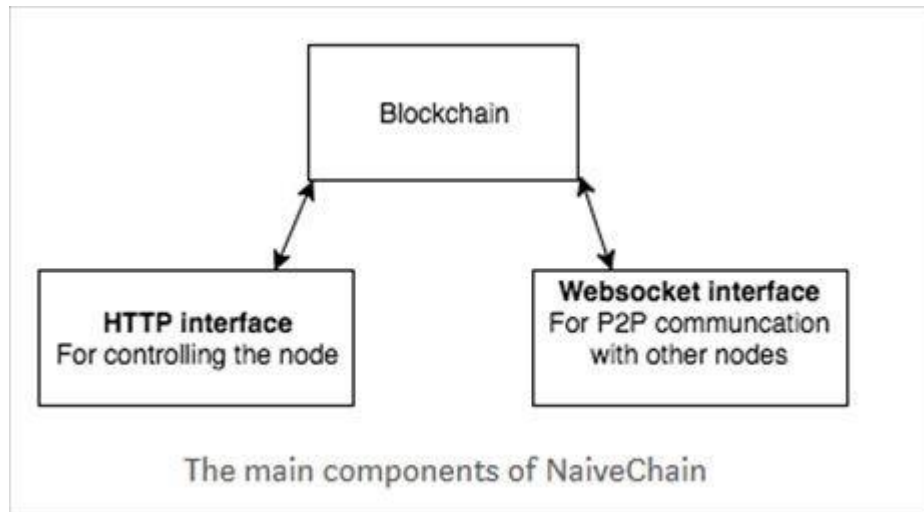
```
curl IP_OR_LOCAL_ADDRESS/blocks
```

For a host PC with address **localhost:3001** it will be:

```
curl http://localhost:3001/blocks
```

12. It should be noted that the node actually exposes **two web servers**:

- One for the **user to control the node (HTTP server)**
- and one for the **peer-to-peer communication** between the nodes (Websocket HTTP server).



### 3. Setup Connected Nodes and Mine a Block

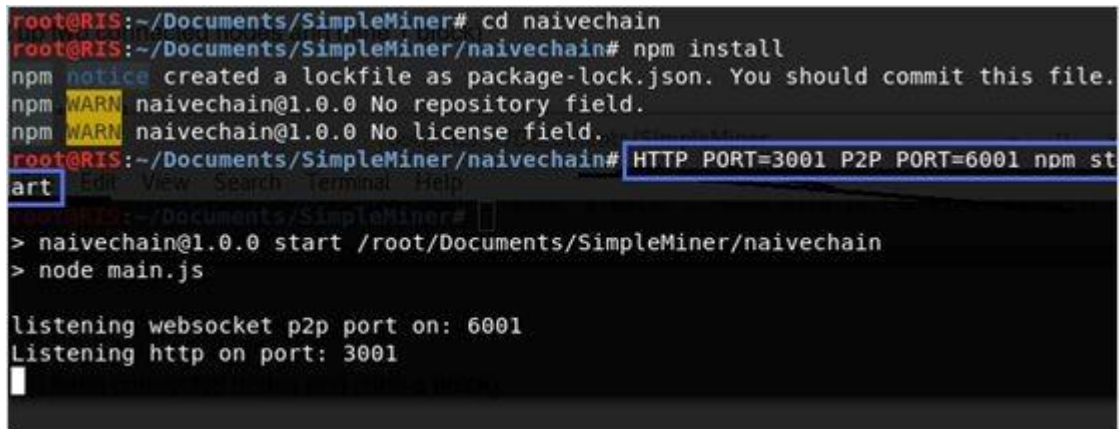
1. First let's establish the **first node**. Get back to the Linux terminal and type "**HTTP\_PORT=3001 P2P\_PORT=6001 npm start**" then press "Enter". The node will listen for **signals from another nodes** via websocket interface on port 6001 and will listen for commands via HTTP interface on port 3001. We can see the node's info on address: **localhost:3001**.

Linux/MacOS:

```
HTTP_PORT=3001 P2P_PORT=6001 npm start
```

Windows:

```
set HTTP_PORT=3001 && set P2P_PORT=6001 && npm start
```

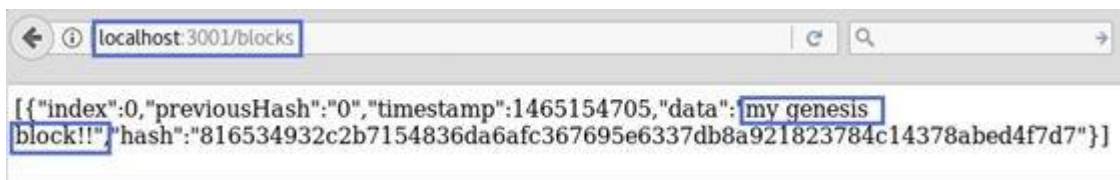


```
root@RIS:~/Documents/SimpleMiner# cd naivechain
root@RIS:~/Documents/SimpleMiner/naivechain# npm install
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN naivechain@1.0.0 No repository field.
npm WARN naivechain@1.0.0 No license field.
root@RIS:~/Documents/SimpleMiner/naivechain# HTTP_PORT=3001 P2P_PORT=6001 npm start
> naivechain@1.0.0 start /root/Documents/SimpleMiner/naivechain
> node main.js

listening websocket p2p port on: 6001
Listening http on port: 3001
```

2. Open your preferred web browser. Go to node's address and attach the endpoint "**blocks**" in attempt to receive the **list of all blocks**. In the browser url bar, write "**localhost:3001/blocks**". Here is the **first block** in the blockchain that was **hardcoded**.

```
http://localhost:3001/blocks
```



```
localhost:3001/blocks
[{"index":0,"previousHash":"0","timestamp":1465154705,"data":"my genesis block!!!"hash":"816534932c2b7154836da6afc367695e6337db8a921823784c14378abed4f7d7"}]
```

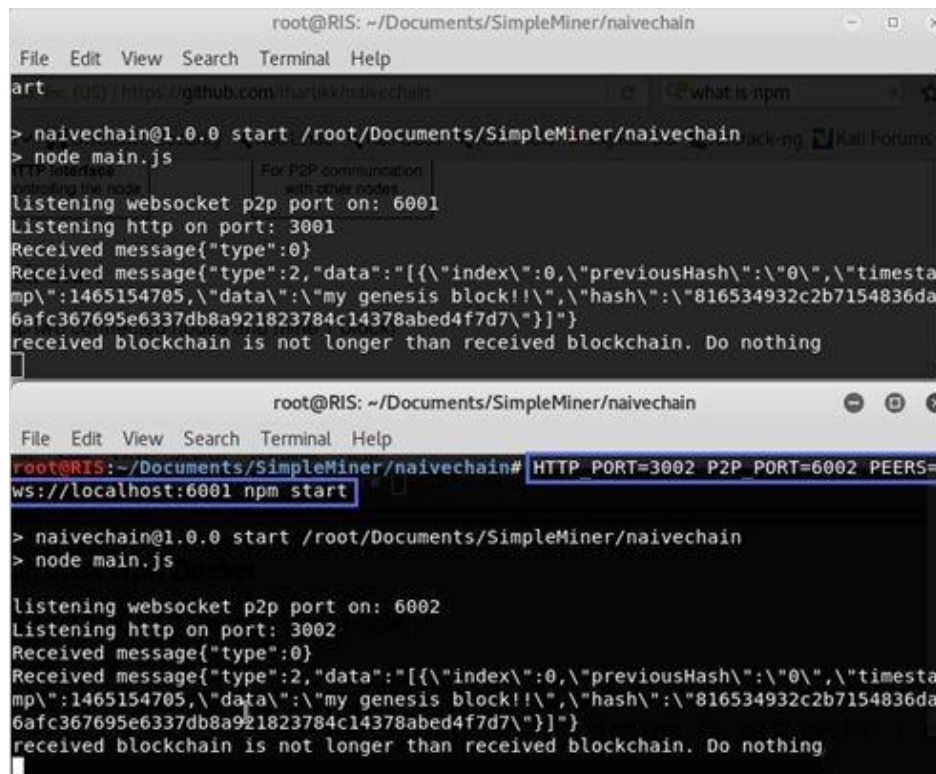
3. Now open a **second terminal** window and start the **second peer** of the network. Type command: **"HTTP\_PORT=3002 P2P\_PORT=6002 PEERS=ws://localhost:6001 npm start"**. The **second peer** will listen for **signals from another nodes** on **port 6002** and will listen for commands via HTTP interface on **port 3002**. This node will receive information from the first peer via P2P communication **via port 6001**. We can see the node's info on address: **localhost:3002**. Here in the picture are both nodes' terminals.

**Linux/MacOS:**

```
HTTP_PORT=3002 P2P_PORT=6002 PEERS=ws://localhost:6001 npm start
```

**Windows:**

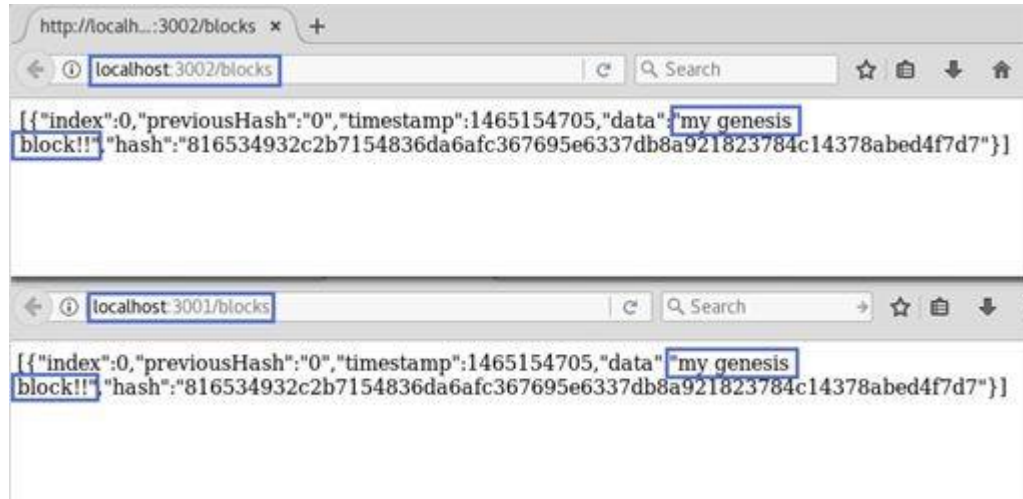
```
set HTTP_PORT=3002 && set P2P_PORT=6002 && set  
PEERS=http://localhost:6001 && npm start
```



```
root@RIS: ~/Documents/SimpleMiner/naivechain
File Edit View Search Terminal Help
naivechain@1.0.0 start /root/Documents/SimpleMiner/naivechain
> node main.js
listening websocket p2p port on: 6001
Listening http on port: 3001
Received message{"type":0}
Received message{"type":2,"data":[{"index":0,"previousHash":"","timestamp":1465154705,"data":"my genesis block!!","hash":"816534932c2b7154836da6afc367695e6337db8a921823784c14378abed4f7d7"}]}
received blockchain is not longer than received blockchain. Do nothing

root@RIS: ~/Documents/SimpleMiner/naivechain
File Edit View Search Terminal Help
root@RIS:~/Documents/SimpleMiner/naivechain# HTTP_PORT=3002 P2P_PORT=6002 PEERS=ws://localhost:6001 npm start
> naivechain@1.0.0 start /root/Documents/SimpleMiner/naivechain
> node main.js
listening websocket p2p port on: 6002
Listening http on port: 3002
Received message{"type":0}
Received message{"type":2,"data":[{"index":0,"previousHash":"","timestamp":1465154705,"data":"my genesis block!!","hash":"816534932c2b7154836da6afc367695e6337db8a921823784c14378abed4f7d7"}]}
received blockchain is not longer than received blockchain. Do nothing
```

4. Let's see **both nodes** in the browser and get the **lists of all blocks**. As we can see, they have **identical blockchains**.



5. Now comes the most interesting part. Open Postman and send a POST request to **http://localhost:3001/mineBlock**.

Alternatively, if Postman is not available for your platform, open a **new terminal** and issue the curl command:

```
curl -H "Content-type:application/json" --data '{"data": "Data to the first block"}' http://localhost:3001/mineBlock
```

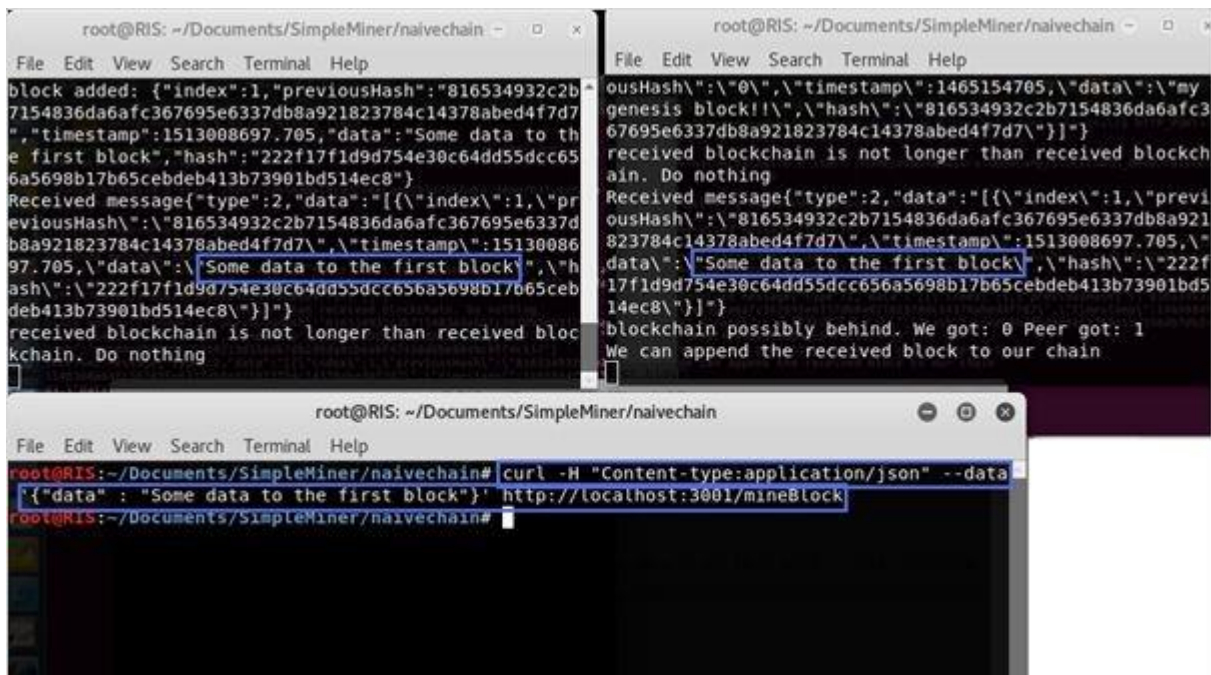
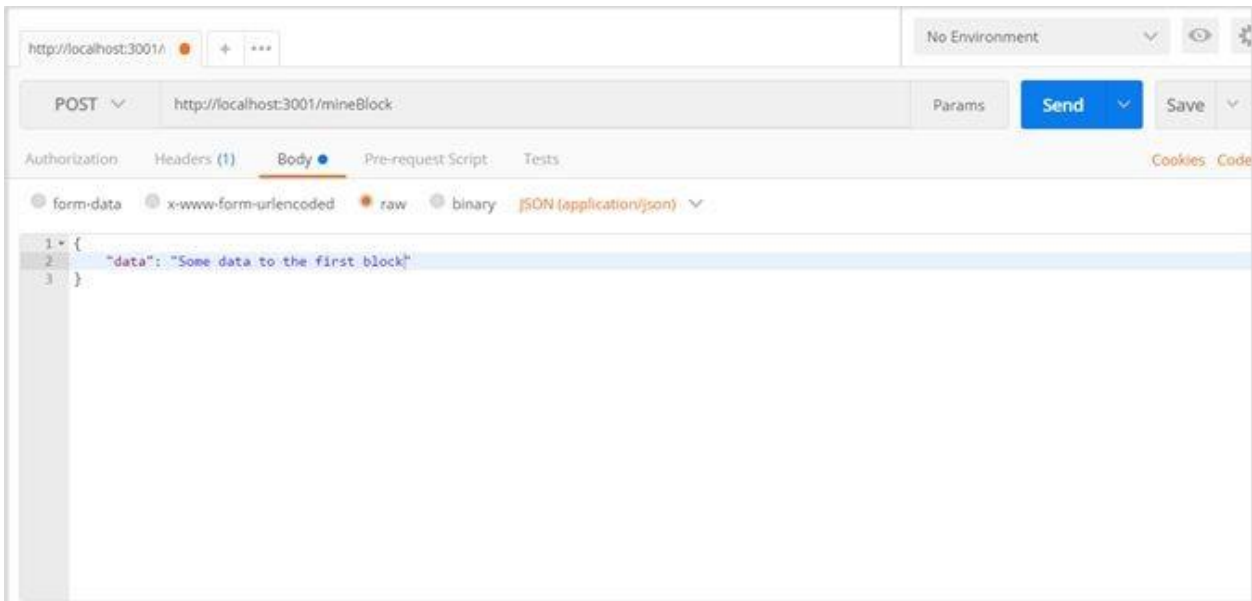
With this POST request, we command the first node to **mine a new block**. Its **index** will be the index of the previous block **+1**. It contains the following data: **Data to the first block**.



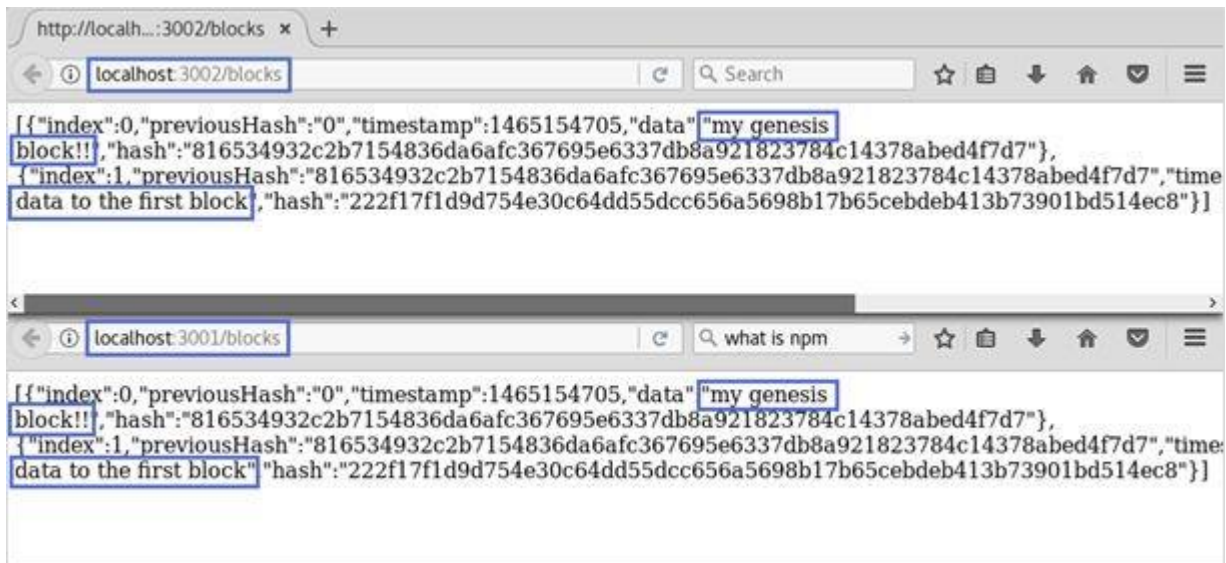
cURL:

```
curl -H "Content-type:application/json" --data '{"data": "Data to the first block"}' http://localhost:3001/mineBlock
```

Postman:



6. Let's see how blockchain was changed. Open **both nodes addresses** in the browser and get the **lists of all blocks**. The **new block** is added after the genesis block.



7. Now let's mine the **third block**. Issue this command in a terminal:

```
curl -H "Content-type: application/json" --data '{"data": "Data to the third block"}' http://localhost:3001/mineBlock
```

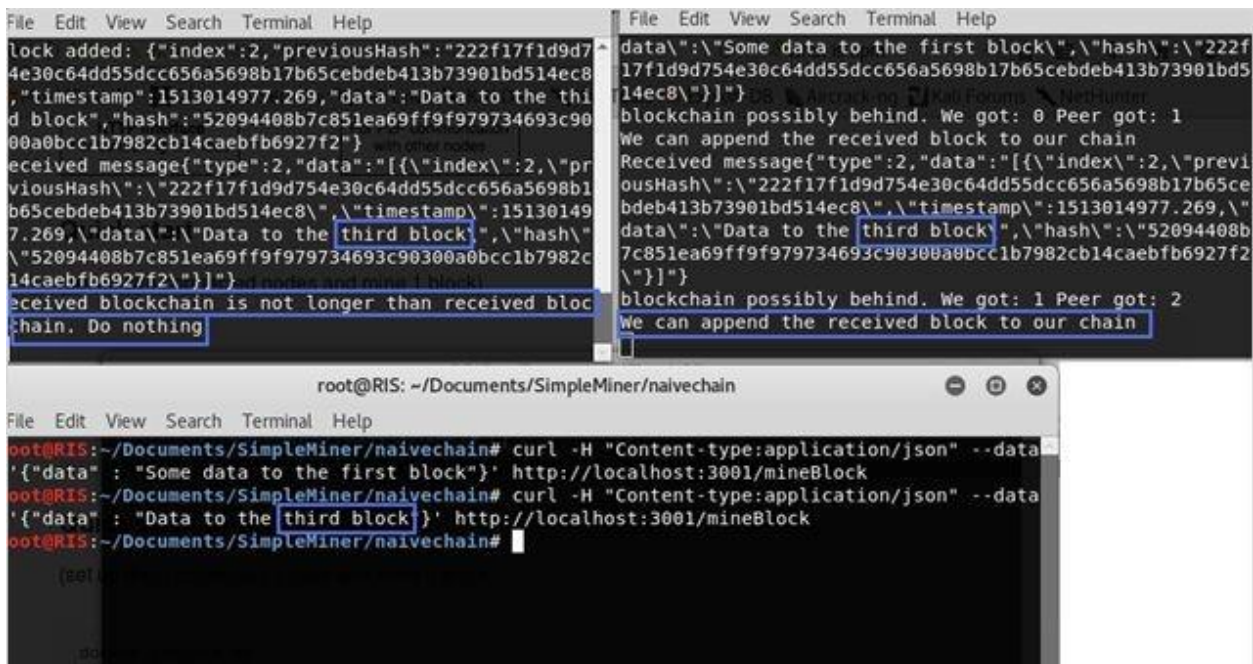
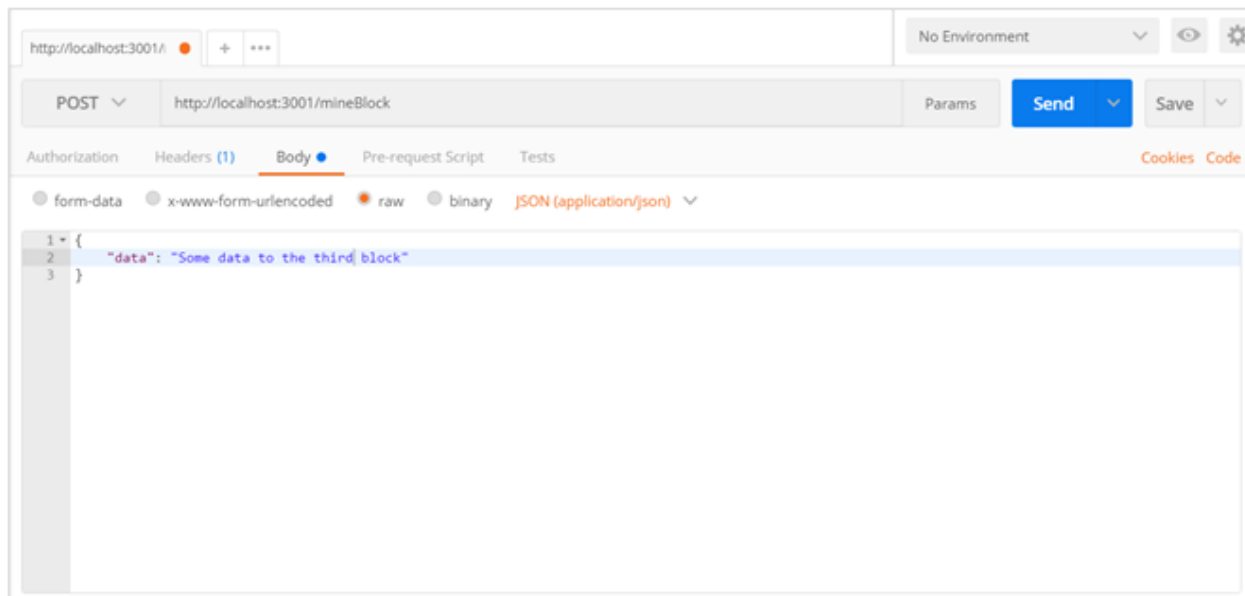
Pay attention since the **first node** is the emitter of the block so it will report that new blocks sequence is no longer than the existing blockchain with the following message: **Received blockchain is not longer than current blockchain. Do nothing.**

But the other node will report that **"We can append the received block to our chain"**.

cURL:

```
curl -H "Content-type:application/json" --data '{"data": "Data to the third block"}' http://localhost:3001/mineBlock
```

Postman:



8. The next task is to create a **third node**. In terminal type the command:  
"HTTP\_PORT=3003 P2P\_PORT=6003 PEERS=ws://localhost:6001 npm start".

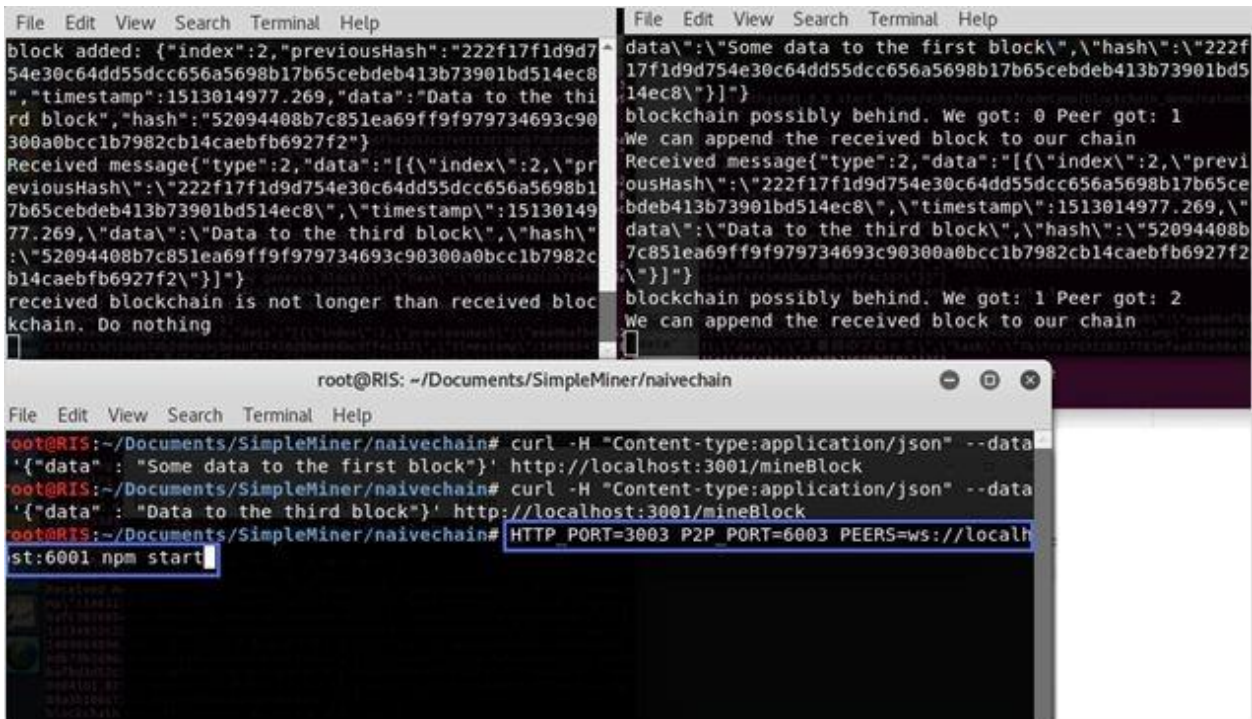
Linux:

```
HTTP_PORT=3003 P2P_PORT=6003 PEERS=ws://localhost:6001 npm start
```

Windows:

```
set HTTP_PORT=3003 && set P2P_PORT=6003 && set PEERS=http://localhost:6001  
&& npm start
```

This is the screen before executing the command.



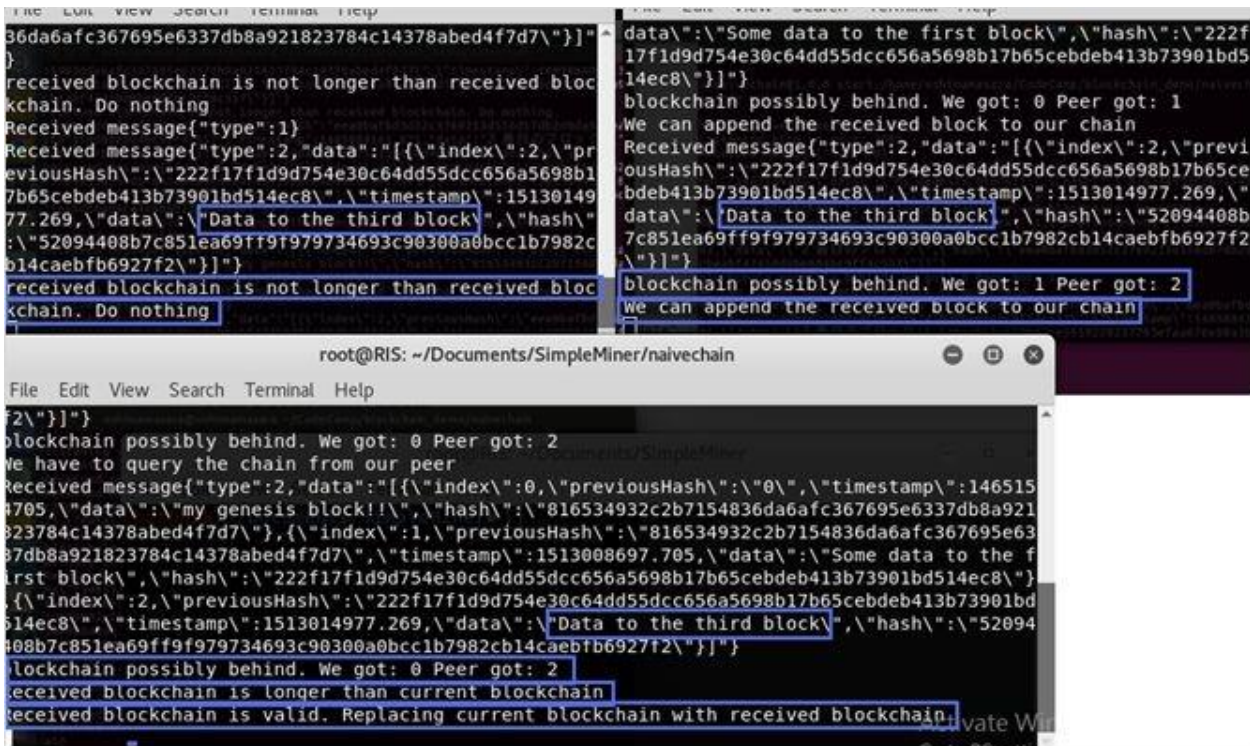
```
File Edit View Search Terminal Help
block added: {"index":2,"previousHash":"222f17f1d9d754e30c64dd55dcc656a5698b17b65cebdeb413b73901bd514ec8",
"timestamp":1513014977.269,"data":"Data to the third block","hash":"52094408b7c851ea69ff9f979734693c90300a0bcc1b7982cb14caebfb6927f2"}
Received message{"type":2,"data":[{"index":2,"previousHash":"222f17f1d9d754e30c64dd55dcc656a5698b17b65cebdeb413b73901bd514ec8",
"timestamp":1513014977.269,"data":"Data to the third block","hash":"52094408b7c851ea69ff9f979734693c90300a0bcc1b7982cb14caebfb6927f2"}]}
received blockchain is not longer than received blockchain. Do nothing

File Edit View Search Terminal Help
data\":"Some data to the first block\","hash\":"222f17f1d9d754e30c64dd55dcc656a5698b17b65cebdeb413b73901bd514ec8\"}]}
blockchain possibly behind. We got: 0 Peer got: 1
We can append the received block to our chain
Received message{"type":2,"data":[{"index":2,"previousHash":"222f17f1d9d754e30c64dd55dcc656a5698b17b65cebdeb413b73901bd514ec8",
"timestamp":1513014977.269,"data":"Data to the third block","hash":"52094408b7c851ea69ff9f979734693c90300a0bcc1b7982cb14caebfb6927f2"}]}
blockchain possibly behind. We got: 1 Peer got: 2
We can append the received block to our chain

root@RIS: ~/Documents/SimpleMiner/naivechain
File Edit View Search Terminal Help
root@RIS:~/Documents/SimpleMiner/naivechain# curl -H "Content-type:application/json" --data
'{"data": "Some data to the first block"}' http://localhost:3001/mineBlock
root@RIS:~/Documents/SimpleMiner/naivechain# curl -H "Content-type:application/json" --data
'{"data": "Data to the third block"}' http://localhost:3001/mineBlock
root@RIS:~/Documents/SimpleMiner/naivechain# HTTP_PORT=3003 P2P_PORT=6003 PEERS=ws://localhost:6001 npm start
```

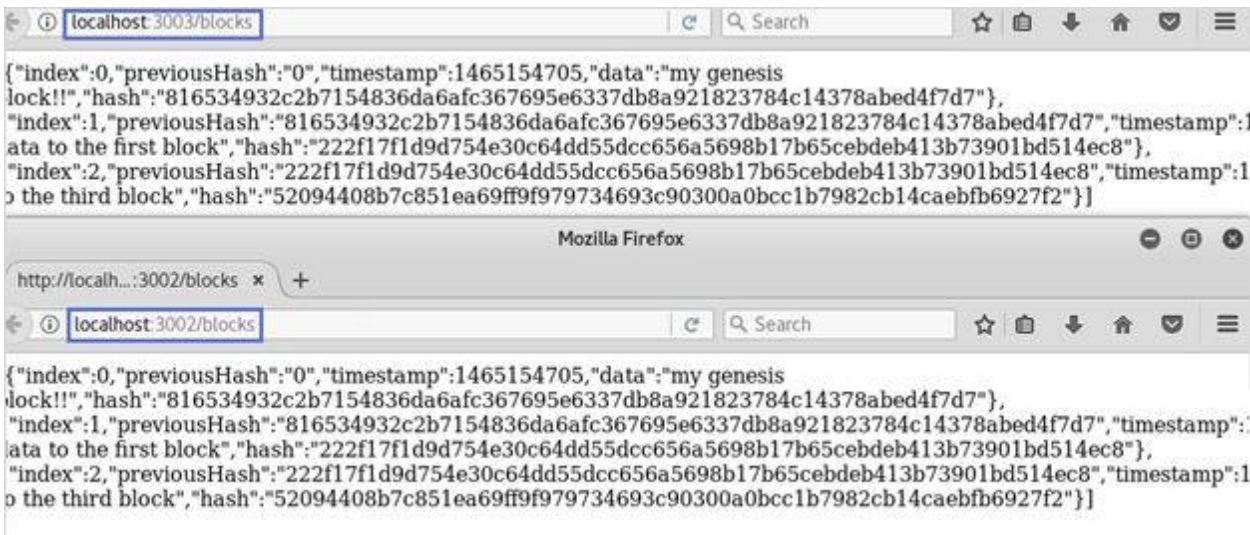


9. And this is the screen after the enter key was pressed. The **third** node was created and is participating in the blockchain.



```
36da6afc367695e6337db8a921823784c14378abed4f7d7\}}"}
received blockchain is not longer than received block
chain. Do nothing
Received message{"type":1}
Received message{"type":2,"data":[{"index":2,"previousHash":"222f17f1d9d754e30c64dd55dcc656a5698b17b65cebdeb413b73901bd514ec8","timestamp":1513014977.269,"data":"Data to the third block","hash":"52094408b7c851ea69ff9f979734693c90300a0bcc1b7982cb14caebfb6927f2"}]}
received blockchain is not longer than received block
chain. Do nothing
data":{"Some data to the first block","hash":"222f17f1d9d754e30c64dd55dcc656a5698b17b65cebdeb413b73901bd514ec8"}}
blockchain possibly behind. We got: 0 Peer got: 1
We can append the received block to our chain
Received message{"type":2,"data":[{"index":2,"previousHash":"222f17f1d9d754e30c64dd55dcc656a5698b17b65cebdeb413b73901bd514ec8","timestamp":1513014977.269,"data":"Data to the third block","hash":"52094408b7c851ea69ff9f979734693c90300a0bcc1b7982cb14caebfb6927f2"}]}
blockchain possibly behind. We got: 1 Peer got: 2
We can append the received block to our chain
root@RIS: ~/Documents/SimpleMiner/naivechain
File Edit View Search Terminal Help
{"index":2,"previousHash":"222f17f1d9d754e30c64dd55dcc656a5698b17b65cebdeb413b73901bd514ec8","timestamp":1513014977.269,"data":"Data to the third block","hash":"52094408b7c851ea69ff9f979734693c90300a0bcc1b7982cb14caebfb6927f2"}]}
blockchain possibly behind. We got: 0 Peer got: 2
We have to query the chain from our peer
Received message{"type":2,"data":[{"index":0,"previousHash":"","timestamp":1465154705,"data":"my genesis block!!","hash":"816534932c2b7154836da6afc367695e6337db8a921823784c14378abed4f7d7"},{"index":1,"previousHash":"816534932c2b7154836da6afc367695e6337db8a921823784c14378abed4f7d7","timestamp":1513008697.705,"data":"Some data to the first block","hash":"222f17f1d9d754e30c64dd55dcc656a5698b17b65cebdeb413b73901bd514ec8"},{"index":2,"previousHash":"222f17f1d9d754e30c64dd55dcc656a5698b17b65cebdeb413b73901bd514ec8","timestamp":1513014977.269,"data":"Data to the third block","hash":"52094408b7c851ea69ff9f979734693c90300a0bcc1b7982cb14caebfb6927f2"}]}
blockchain possibly behind. We got: 0 Peer got: 2
received blockchain is longer than current blockchain
received blockchain is valid. Replacing current blockchain with received blockchain.
```

10. Open the **third node's** address in the browser and compare with one of the old nodes. The third node has the **same blocks**.



```
{ "index":0,"previousHash":"0","timestamp":1465154705,"data":"my genesis block!!","hash":"816534932c2b7154836da6afc367695e6337db8a921823784c14378abed4f7d7"},
{ "index":1,"previousHash":"816534932c2b7154836da6afc367695e6337db8a921823784c14378abed4f7d7","timestamp":1513008697.705,"data":"Some data to the first block","hash":"222f17f1d9d754e30c64dd55dcc656a5698b17b65cebdeb413b73901bd514ec8"},
{ "index":2,"previousHash":"222f17f1d9d754e30c64dd55dcc656a5698b17b65cebdeb413b73901bd514ec8","timestamp":1513014977.269,"data":"Data to the third block","hash":"52094408b7c851ea69ff9f979734693c90300a0bcc1b7982cb14caebfb6927f2"}]}
Mozilla Firefox
http://localh...:3002/blocks
localhost:3002/blocks
{"index":0,"previousHash":"0","timestamp":1465154705,"data":"my genesis block!!","hash":"816534932c2b7154836da6afc367695e6337db8a921823784c14378abed4f7d7"},
{"index":1,"previousHash":"816534932c2b7154836da6afc367695e6337db8a921823784c14378abed4f7d7","timestamp":1513008697.705,"data":"Some data to the first block","hash":"222f17f1d9d754e30c64dd55dcc656a5698b17b65cebdeb413b73901bd514ec8"},
{"index":2,"previousHash":"222f17f1d9d754e30c64dd55dcc656a5698b17b65cebdeb413b73901bd514ec8","timestamp":1513014977.269,"data":"Data to the third block","hash":"52094408b7c851ea69ff9f979734693c90300a0bcc1b7982cb14caebfb6927f2"}]}
```

## 4. Implement Proof-of-Work Mining

Now, it's time to **implement proof-of-work mining** in the blockchain network. We shall modify the `/mineBlock` REST endpoint to calculate a block hash with a few leading zeros (the proof of work) instead of just the block hash.

1. **Open** the file `"main.js"` in your preferred text editor or IDE. Modify the code and **implement proof-of-work**, following the next few steps.
2. First, we should create a variable to hold the network **difficulty**. This is the required **number of leading zeros** in the next block's **hash**.

```
'use strict';
var CryptoJS = require("crypto-js");
var express = require("express");
var bodyParser = require('body-parser');
var WebSocket = require("ws");

var http_port = process.env.HTTP_PORT || 3001;
var p2p_port = process.env.P2P_PORT || 6001;
var initialPeers = process.env.PEERS ? process.env.PEERS.split(',') : [];
var difficulty = 4;
```

3. Next, we should add the **"nonce"** property and the **"difficulty"** property in the **Block** constructor. Nonce is the number which we will **increase** in the attempt to find the appropriate proof-of-work hash and **mine the next block**. The difficulty is important because without it, nobody will be able to prove whether the hash found is valid or not.

```
class Block {
  constructor(index, previousHash, timestamp, data, hash, difficulty, nonce) {
    this.index = index;
    this.previousHash = previousHash.toString();
    this.timestamp = timestamp;
    this.data = data;
    this.hash = hash.toString();
    this.difficulty = difficulty;
    this.nonce = nonce;
  }
}
```

4. We will add **nonce** and **difficulty** in every block our program creates. In this case, we can modify the **function** which **creates the hardcoded genesis block** by adding **nonce** and **difficulty** with 0 values.

```
var getGenesisBlock = () => {
  return new Block(0, "0", 1465154705, "my genesis block!!",
    "816534932c2b7154836da6afc367695e6337db8a921823784c14378abed4f7d7", 0, 0);
};

var blockchain = [getGenesisBlock()];
```

5. Then we should modify functions **calculateHashForBlock** and **calculateHash** as follows:

```
var calculateHashForBlock = (block) => {
  return calculateHash(block.index, block.previousHash, block.timestamp, block.data, block.nonce);
};

var calculateHash = (index, previousHash, timestamp, data, nonce) => {
  return CryptoJS.SHA256(index + previousHash + timestamp + data + nonce).toString();
};
```

6. Currently, when we execute the **"mineblock"** command, we can still create a new block without having a Proof of Work mechanism in place. But now we need a new **"mineBlock"** function responsible for mining with Proof of Work implemented.

```
var initHttpServer = () => {
  var app = express();
  app.use(bodyParser.json());

  app.get('/blocks', (req, res) => res.send(JSON.stringify(blockchain)));
  app.post('/mineBlock', (req, res) => {
    var newBlock = mineBlock(req.body.data);
    //var newBlock = generateNextBlock(req.body.data);
    addBlock(newBlock);
    broadcast(responseLatestMsg());
    console.log('block added: ' + JSON.stringify(newBlock));
    res.send();
  });
  app.get('/peers', (req, res) => {
    res.send(sockets.map(s => s._socket.remoteAddress + ':' + s._socket.remotePort));
  });
  app.post('/addPeer', (req, res) => {
    connectToPeers([req.body.peer]);
    res.send();
  });
  app.listen(http_port, () => console.log('Listening http on port: ' + http_port));
};
```

7. This is how the **mineBlock** function looks like. It receives the **block info** and during each iteration, it checks to see whether the generated **hash** satisfies the **difficulty** requirements. If it doesn't, we increment the **nonce**; if it does, then return the new, successfully created **block**.



```

var mineBlock = (blockData) => {
  var previousBlock = getLatestBlock();
  var nextIndex = previousBlock.index + 1;
  var nonce = 0;
  var nextTimestamp = new Date().getTime() / 1000;
  var nextHash = calculateHash(nextIndex, previousBlock.hash, nextTimestamp, blockData, nonce);
  while (nextHash.substring(0, difficulty) !== Array(difficulty + 1).join("0")){
    nonce++;
    nextTimestamp = new Date().getTime() / 1000;
    nextHash = calculateHash(nextIndex, previousBlock.hash, nextTimestamp, blockData, nonce)

    console.log("\nindex\":" + nextIndex + ",\npreviousHash\":"+previousBlock.hash+
      "\ntimestamp\":"+nextTimestamp+",\ndata\":"+blockData+
      ",\n\x1b[33mhash: " + nextHash + " \x1b[0m," + "\ndifficulty\":"+difficulty+
      " \x1b[33mnonce: " + nonce + " \x1b[0m ");
  }

  return new Block(nextIndex, previousBlock.hash, nextTimestamp, blockData, nextHash, difficulty, nonce);
}

```

## 5.Run Two “Naivechain” Nodes and Run PoW Mining

The code is ready. Now, let’s test the modified code.

1. Run two Naivechain nodes at the following ports:

- REST: 3001, P2P:6001
- REST: 3002, P2P:6002

```
root@RIS: ~/Documents/SimpleMiner_with_PoW/naivechain
File Edit View Search Terminal Help
root@RIS:~/Documents/SimpleMiner_with_PoW/naivechain# HTTP_PORT=3001 P2P_PORT=6001
01 npm start
require("crypto-js");
require("express");
> naivechain@1.0.0 start /root/Documents/SimpleMiner_with_PoW/naivechain
> node main.js require("ws");

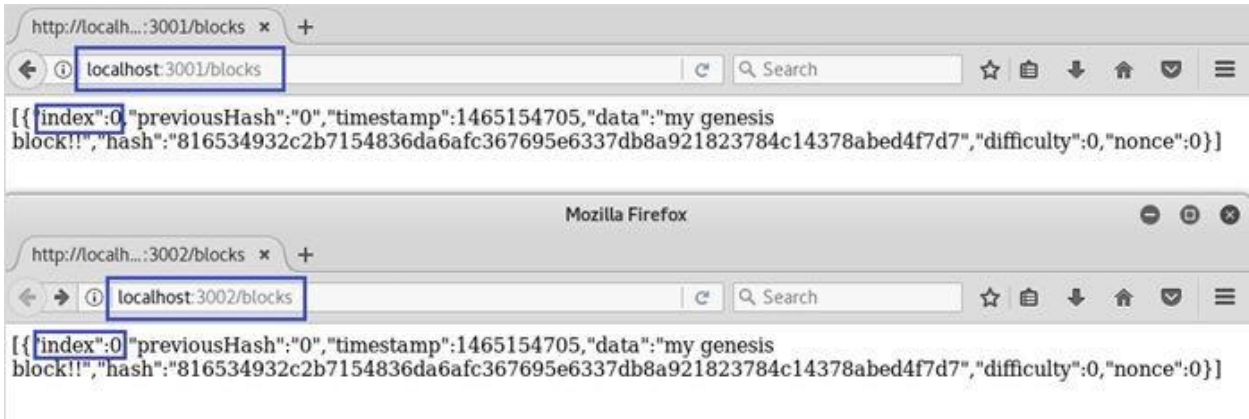
listening websocket p2p port on: 6001 | 3001;
Listening http on port: 3001 | 6001;
Received message{"type":0}env.PEERS = process.env.PEERS.split(',') : [];
Received message{"type":2,"data":[{"index":0,"previousHash":"","timestamp":1465154705,"data":"my genesis block!!","hash":"816534932c2b7154836da6afc367695e6337db8a921823784c14378abed4f7d7","difficulty":0,"nonce":0}]}
received blockchain is not longer than received blockchain. Do nothing nonce) {

root@RIS: ~/Documents/SimpleMiner_with_PoW/naivechain
File Edit View Search Terminal Help
root@RIS:~/Documents/SimpleMiner with PoW/naivechain# HTTP_PORT=3002 P2P_PORT=6002
02 PEERS=ws://localhost:6001 npm start

> naivechain@1.0.0 start /root/Documents/SimpleMiner_with_PoW/naivechain
> node main.js

listening websocket p2p port on: 6002
Listening http on port: 3002
Received message{"type":0}
Received message{"type":2,"data":[{"index":0,"previousHash":"","timestamp":1465154705,"data":"my genesis block!!","hash":"816534932c2b7154836da6afc367695e6337db8a921823784c14378abed4f7d7","difficulty":0,"nonce":0}]}
received blockchain is not longer than received blockchain. Do nothing
letGenesisBlock = () => {
```

2. Open the two nodes' blockchains in the browser. In the browser URL bar, write "**localhost:3001/blocks**" and "**localhost:3002/blocks**". Here, the only block is the hardcoded **genesis** block with **index 0**.

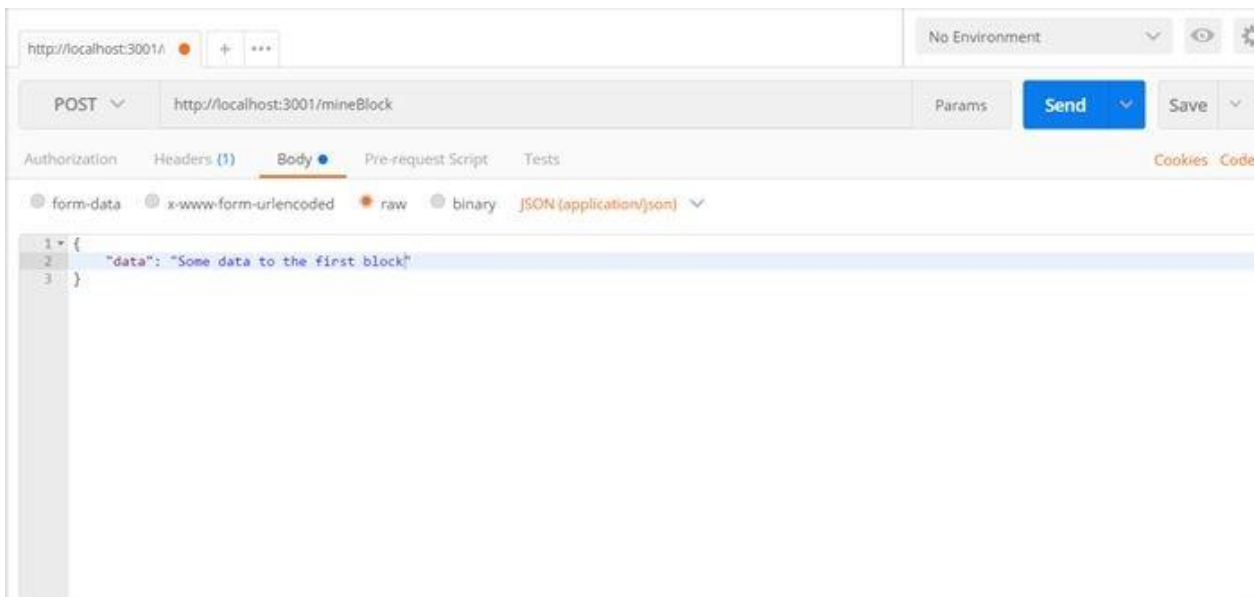


3. Now, it's time to mine. Open a new console and issue the following POST request. With this, we command the **first node** to **mine a new block**:

**cURL:**

```
curl -H "Content-type:application/json" --data '{"data": "Some data to the first block"}' http://localhost:3001/mineBlock
```

**Postman:**



4. Look at the **first node's console**. The miner starts mining and **increments the nonce** searching for the **hash that satisfies the difficulty** requirements.

```
File Edit View Search Terminal Help
13d012e1bfd8659547ee52539d4df0855ab376d27d , "difficulty":4 nonce: 20708
"index":1,"previousHash":816534932c2b7154836da6afc367695e6337db8a921823784c14378
abed4f7d7"timestamp":1517334446.12,"data":MyInfo 1,hash: 8aa7abae49a9b6d93c582ee
64b40af578d6d0455b0af10c40c04c40a68b12439 , "difficulty":4 nonce: 20709
"index":1,"previousHash":816534932c2b7154836da6afc367695e6337db8a921823784c14378
abed4f7d7"timestamp":1517334446.12,"data":MyInfo 1,hash: 5c1dc094c8e5a492c0caf2e
f29f048bc39c2f161fe31b1c3991be3b3914d9916 , "difficulty":4 nonce: 20710
"index":1,"previousHash":816534932c2b7154836da6afc367695e6337db8a921823784c14378
abed4f7d7"timestamp":1517334446.12,"data":MyInfo 1,hash: f67e949efc1d1e50b8dec64
cd8b040441d1631bbeb904339c8ddfddea3093459c , "difficulty":4 nonce: 20711
"index":1,"previousHash":816534932c2b7154836da6afc367695e6337db8a921823784c14378
abed4f7d7"timestamp":1517334446.12,"data":MyInfo 1,hash: 2721af3304f7900488450c2
fd4d30064bea9e50d38df6db9510d143836dce4fe , "difficulty":4 nonce: 20712
"index":1,"previousHash":816534932c2b7154836da6afc367695e6337db8a921823784c14378
abed4f7d7"timestamp":1517334446.12,"data":MyInfo 1,hash: 0000ad3b80f1e84f8741d05
5e248b7e3283f6a62a4425859a355a0cd22b967ca , "difficulty":4 nonce: 20713
block added: {"index":1,"previousHash":"816534932c2b7154836da6afc367695e6337db8a
921823784c14378abed4f7d7","timestamp":1517334446.12,"data":"MyInfo 1","hash":"'00
00ad3b80f1e84f8741d055e248b7e3283f6a62a4425859a355a0cd22b967ca","difficulty":4,"
nonce":20713}
Received message{"type":2,"data":[{"index":1,"previousHash":"816534932c2b7
154836da6afc367695e6337db8a921823784c14378abed4f7d7","timestamp":1517334446.1
2,"data":"MyInfo 1","hash":"'0000ad3b80f1e84f8741d055e248b7e3283f6a62a4425
859a355a0cd22b967ca","difficulty":4,"nonce":20713}]}
received blockchain is not longer than received blockchain. Do nothing
```

Note the following results:

- Block mining **takes some time** (a few seconds). This is due to the proof-of-work algorithm.
- The last block hash has **several leading zeros** depending on the **difficulty**. This is the proof-of-work result.

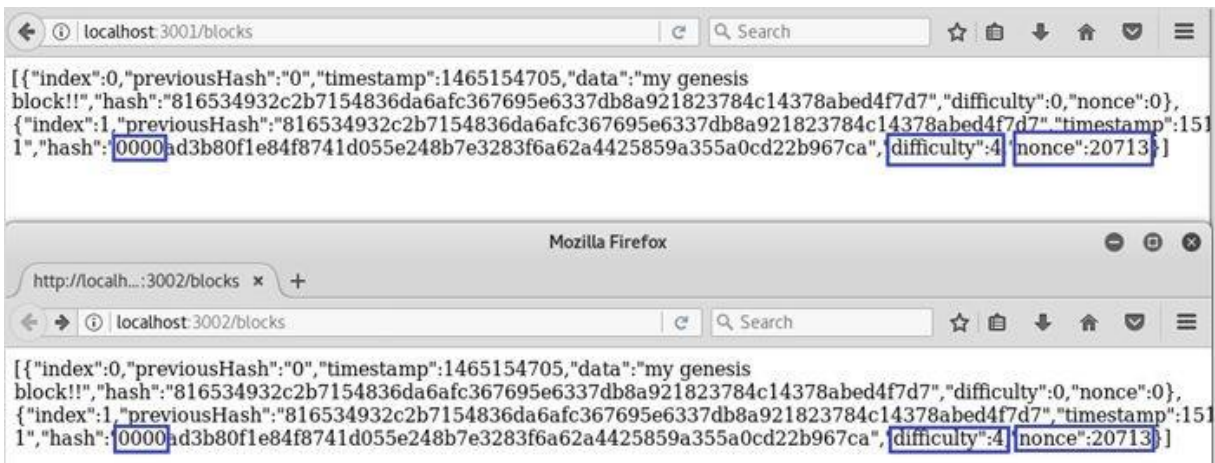


5. The second node has also accepted a new block in its blockchain.

```
File Edit View Search Terminal Help
root@RIS:~/Documents/SimpleMiner_with_PoW/naivechain# HTTP_PORT=3002 P2P_PORT=6002 PEERS=ws://localhost:6001 npm start
naivechain@1.0.0 start /root/Documents/SimpleMiner_with_PoW/naivechain
> node main.js
listening websocket p2p port on: 6002
Listening http on port: 3002
Received message{"type":0}
Received message{"type":2,"data":[{"index":0,"previousHash":"","timestamp":1465154705,"data":"my genesis block!!","hash":"816534932c2b7154836da6afc367695e6337db8a921823784c14378abed4f7d7","difficulty":0,"nonce":0}]}
received blockchain is not longer than received blockchain. Do nothing
Received message{"type":2,"data":[{"index":1,"previousHash":"816534932c2b7154836da6afc367695e6337db8a921823784c14378abed4f7d7","timestamp":1517334446.12,"data":"MyInfo 1","hash":"0000ad3b80f1e84f8741d055e248b7e3283f6a62a4425359a355a0cd22b967ca","difficulty":4,"nonce":20713}]}
blockchain possibly behind. We got: 0 Peer got: 1
We can append the received block to our chain
```

6. Look at the nodes in the browser. There is the new block with **several leading zeros, difficulty** and **nonce**.

Using this info, the hash that was found can be easily verified by other miners.



## What to Submit?

Create a **zip file** (e.g. your-name-simple-miner-exercise.zip) holding the screenshots of the following:

1. Your browser requests/responses
  - a. GET
  - b. POST
2. Your terminal
  - a. Peer syncing
  - b. Mining

Submit your **ZIP** file as **homework** at the course platform.