



### Реализация блокчейн (Proof-of-Work)

#### 1. Требования

Для реализации курсового проекта необходимо:

1. Ознакомиться с материалом лекции о блокчейн технологиях;
2. Установить Node.js - <https://nodejs.org/en/download>;
3. Установить редактор Visual Studio Code - <https://code.visualstudio.com>;
4. Установить Postman - <https://www.postman.com/downloads/>.

Рекомендуется перезагрузить компьютер после установки.

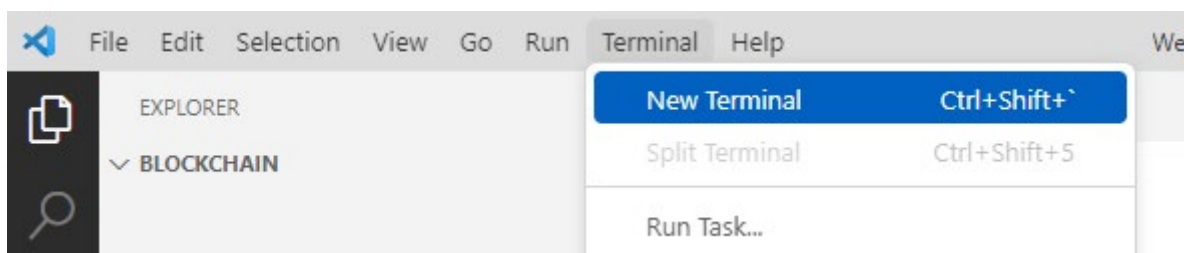
#### 2. Цель

Целью курсового проекта является закрепление теоретических знаний о базовых принципах работы блокчейн технологий и майнинга.

### Часть 1. Реализация блокчейн без Proof-of-Work

#### 3. Подготовка

Если все требования соблюдены, создайте директорию для проекта, к примеру blockchain и откройте ее в VS Code. Запустите терминал:



Введем команды для проверки установленных компонентов:

**node -v**

**npm -v**

Если в ответ вы получили версии компонентов, то все установлено корректно.

*Краткая справка:*

*Node — Node или Node.js — программная платформа, основанная на движке V8, превращающая JavaScript из узкоспециализированного языка в язык общего назначения, предоставляя возможность запуска скриптов вне браузера.*



## Цифровые технологии. Курсовой проект.

*npm* — менеджер пакетов, входящий в состав *Node.js*.

Если в ответ вы получили версии компонентов, то все установлено корректно.

Введите команду для создания проекта:

**npm init**

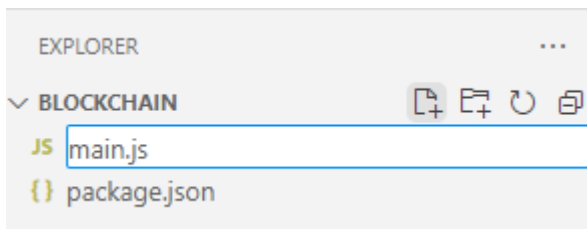
На все вопросы отвечайте нажатием клавиши «Enter».

Далее отредактируйте файл `package.json`:

```
{
  "name": "blockchain",
  "version": "1.0.0",
  "description": "",
  "scripts": {
    "start": "node main.js"
  }
}
```

Главным файлом проекта станет `main.js`, который и будет запускаться по команде `npm start`.

Создайте файл `main.js`:



Для реализации нам понадобятся внешние библиотеки, для простоты обновите `package.json`:

```
{
  "name": "blockchain",
  "version": "1.0.0",
  "description": "",
  "scripts": {
    "start": "node main.js"
  },
  "dependencies": {
    "body-parser": "^1.15.2",
    "crypto-js": "^3.1.6",
    "express": "~4.17.3",
  }
}
```



```
"ws": "^1.1.0"
},
"engines": {
  "node": ">=4.3.2"
}
}
```

Далее в терминале введите команду:

```
npm install
```

Пакеты установятся.

### 4. Реализация

В main.js начнем реализацию:

```
'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(',') : [];
```

'use strict'; говорит JavaScript-движку, что код должен быть выполнен в строгом режиме, что означает, требования при сборке проекта будут повышены, избежать рекомендаций или предупреждений при сборке не получится. К примеру, если вы определите переменную и ее не используете – это будет ошибка.

Далее мы подключаем библиотеки через переменные, которые предоставят доступ к их функционалу.

crypto-js предоставляет криптографические функции для JavaScript, express – простой фреймворк для создания веб-приложений на Node.js., bodyParser - предоставляет функциональность для обработки данных, отправленных с веб-страниц, WebSocket - предоставляет простой и гибкий API для создания WebSocket-соединений.

Далее объявляются порты как для клиент-серверного соединения, так и P2P. Если в переменных среды заранее заданных значений нет, то используем стандартные.

Далее подготавливается массив для P2P клиентов – пиров.



## Цифровые технологии. Курсовой проект.

Далее опишем блок в рамках стандартной концепции:



```
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();  
    }  
}
```

Этот код определяет класс Block, который используется для создания объектов, представляющих блоки в блокчейн-сети.

constructor(index, previousHash, timestamp, data, hash) определяет конструктор класса, который принимает следующие аргументы:

- index - число, представляющее индекс блока в цепочке блоков.
- previousHash - строка, представляющая хеш предыдущего блока в цепочке блоков.
- timestamp - число, представляющее метку времени создания блока.
- data - любые данные, которые будут сохранены в блоке.
- hash - строка, представляющая уникальный хеш текущего блока.

Далее создадим массив сокетов, через которые происходит обмен сообщениями между участниками сети. Дополнительно определим тип сообщений, отправляемых между участниками сети.

```
var sockets = [];  
var MessageType = {  
    QUERY_LATEST: 0,  
    QUERY_ALL: 1,  
    RESPONSE_BLOCKCHAIN: 2  
};
```

Далее создадим первый блок (генезис-блок):

```
var getGenesisBlock = () => {
```



## Цифровые технологии. Курсовой проект.

```
return new Block(0, "0", 1682839690, "RUT-MIIT first block",  
"8d9d5a7ff4a78042ea6737bf59c772f8ed27ef3c9b576eac1976c91aaf48d2de");  
};
```

И положим его в блокчейн:

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

Создадим и запустим веб-сервер, используемый для обслуживания запросов от узлов в блокчейн-сети.

```
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));  
};
```

Сервер отвечает по следующим endpoint'ам:

/blocks - для получения списка блоков в цепочке блоков.

/mineBlock - для майнинга (добычи) нового блока. При POST запросе сервер создает новый блок с данными, полученными от клиента, добавляет его в цепочку блоков с помощью функции addBlock() и отправляет сообщение обновления всем узлам в сети.

/peers - для получения списка узлов в сети.

/addPeer - для добавления нового узла в сеть. При POST запросе сервер сервер подключается к указанному узлу и отправляет сообщение обновления всем узлам в сети.



## Цифровые технологии. Курсовой проект.

Далее создаем и запустим веб-сервер, используемый для обмена сообщениями между узлами в блокчейн-сети.

```
var initP2PServer = () => {  
  var server = new WebSocket.Server({port: p2p_port});  
  server.on('connection', ws => initConnection(ws));  
  console.log('listening websocket p2p port on: ' + p2p_port);  
};
```

Далее проинициализируем и определим функции, используемые для обработки сообщений. Если ошибка – соединение закрывается, сокет удаляется, в противном случае сообщения обрабатываются.

```
var initConnection = (ws) => {  
  sockets.push(ws);  
  initMessageHandler(ws);  
  initErrorHandler(ws);  
  write(ws, queryChainLengthMsg());  
};  
  
var initMessageHandler = (ws) => {  
  ws.on('message', (data) => {  
    var message = JSON.parse(data);  
    console.log('Received message' + JSON.stringify(message));  
    switch (message.type) {  
      case MessageType.QUERY_LATEST:  
        write(ws, responseLatestMsg());  
        break;  
      case MessageType.QUERY_ALL:  
        write(ws, responseChainMsg());  
        break;  
      case MessageType.RESPONSE_BLOCKCHAIN:  
        handleBlockchainResponse(message);  
        break;  
    }  
  });  
};  
  
var initErrorHandler = (ws) => {  
  var closeConnection = (ws) => {  
    console.log('connection failed to peer: ' + ws.url);  
    sockets.splice(sockets.indexOf(ws), 1);  
  };  
  ws.on('close', () => closeConnection(ws));  
  ws.on('error', () => closeConnection(ws));  
};
```



```
var connectToPeers = (newPeers) => {
  newPeers.forEach((peer) => {
    var ws = new WebSocket(peer);
    ws.on('open', () => initConnection(ws));
    ws.on('error', () => {
      console.log('connection failed')
    });
  });
};

var handleBlockchainResponse = (message) => {
  var receivedBlocks = JSON.parse(message.data).sort((b1, b2) => (b1.index - b2.index));
  var latestBlockReceived = receivedBlocks[receivedBlocks.length - 1];
  var latestBlockHeld = getLatestBlock();
  if (latestBlockReceived.index > latestBlockHeld.index) {
    console.log('blockchain possibly behind. We got: ' + latestBlockHeld.index + ' Peer got: ' + latestBlockReceived.index);
    if (latestBlockHeld.hash === latestBlockReceived.previousHash) {
      console.log("We can append the received block to our chain");
      blockchain.push(latestBlockReceived);
      broadcast(responseLatestMsg());
    } else if (receivedBlocks.length === 1) {
      console.log("We have to query the chain from our peer");
      broadcast(queryAllMsg());
    } else {
      console.log("Received blockchain is longer than current blockchain");
      replaceChain(receivedBlocks);
    }
  } else {
    console.log('received blockchain is not longer than current blockchain. Do nothing');
  }
};
```

Далее проинициализируем и определим функции, используемые для генерации блока, расчета хеша, добавления блока в цепочку и проверки блока. Для генерации блока нам необходимо знать хэш предыдущего блока. Обратите внимание на проверку блока, в блокчейн сети мы всегда должны иметь возможность проверить, является ли блок допустимым.

```
var generateNextBlock = (blockData) => {
  var previousBlock = getLatestBlock();
  var nextIndex = previousBlock.index + 1;
  var nextTimestamp = new Date().getTime() / 1000;
```



## Цифровые технологии. Курсовой проект.

```
var nextHash = calculateHash(nextIndex, previousBlock.hash, nextTimestamp,
blockData);
return new Block(nextIndex, previousBlock.hash, nextTimestamp, blockData,
nextHash);
};

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

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

var addBlock = (newBlock) => {
if (isValidNewBlock(newBlock, getLatestBlock())) {
blockchain.push(newBlock);
}
};

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;
};
```

Далее определим функцию определения самой длинной цепочки и проверки цепочки на допустимость.

```
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');
    }
};

var isValidChain = (blockchainToValidate) => {
    if (JSON.stringify(blockchainToValidate[0]) !==
JSON.stringify(getGenesisBlock())) {
        return false;
    }
    var tempBlocks = [blockchainToValidate[0]];
    for (var i = 1; i < blockchainToValidate.length; i++) {
        if (isValidNewBlock(blockchainToValidate[i], tempBlocks[i - 1])) {
            tempBlocks.push(blockchainToValidate[i]);
        } else {
            return false;
        }
    }
    return true;
};
```

Далее определим реализацию вспомогательных функций и запустим веб-сервера.

```
var getLatestBlock = () => blockchain[blockchain.length - 1];
var queryChainLengthMsg = () => ({'type': MessageType.QUERY_LATEST});
var queryAllMsg = () => ({'type': MessageType.QUERY_ALL});
var responseChainMsg = () => ({
    'type': MessageType.RESPONSE_BLOCKCHAIN, 'data': JSON.stringify(blockchain)
});
var responseLatestMsg = () => ({
    'type': MessageType.RESPONSE_BLOCKCHAIN,
    'data': JSON.stringify([getLatestBlock()])
});

var write = (ws, message) => ws.send(JSON.stringify(message));
var broadcast = (message) => sockets.forEach(socket => write(socket, message));

connectToPeers(initialPeers);
initHttpServer();
initP2PServer();
```

### 5. Запуск (Windows, см. Linux ниже)

Перейдите в терминал и запустите сервер с помощью команды:

```
$env:HTTP_PORT=3001; $env:P2P_PORT=6001; npm start
```



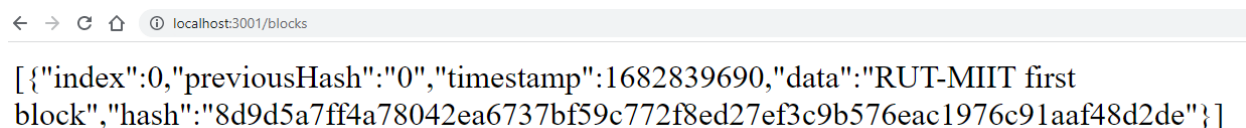
## Цифровые технологии. Курсовой проект.

```
PS G:\blockchain> $env:HTTP_PORT=3001; $env:P2P_PORT=6001; npm start
```

```
> blockchain@1.0.0 start
> node main.js
```

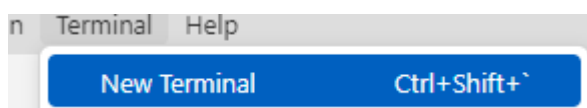
```
listening websocket p2p port on: 6001
Listening http on port: 3001
```

В браузере откройте адрес <http://localhost:3001/blocks>:



```
[{"index":0,"previousHash":"0","timestamp":1682839690,"data":"RUT-MIIT first block","hash":"8d9d5a7ff4a78042ea6737bf59c772f8ed27ef3c9b576eac1976c91aaf48d2de"}]
```

Давайте запустим еще один узел, для этого запустите дополнительный терминал:



В Windows нам понадобится локальный IP. Сперва узнаем наш локальный IP:

### ipconfig

К примеру, у нас 192.168.2.4

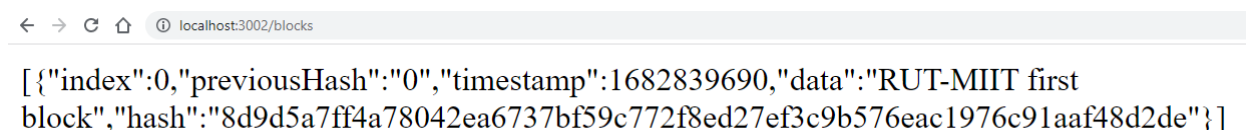
Далее введем команду

```
$env:HTTP_PORT=3002; $env:P2P_PORT=6002; $env:PEERS="ws://192.168.2.4:6001"; npm start
```

```
> blockchain@1.0.0 start
> 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":1682839690,"data":"RUT-MIIT first block","hash":"8d9d5a7ff4a78042ea6737bf59c772f8ed27ef3c9b576eac1976c91aaf48d2de"}]}
received blockchain is not longer than current blockchain. Do nothing
```

В браузере откройте адрес второго узла <http://localhost:3002/blocks>:



```
[{"index":0,"previousHash":"0","timestamp":1682839690,"data":"RUT-MIIT first block","hash":"8d9d5a7ff4a78042ea6737bf59c772f8ed27ef3c9b576eac1976c91aaf48d2de"}]
```

## 5. Запуск (Linux/Mac)

Замена для первой команды:

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

Замена для второй команды:

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

## 6. Майнинг



## Цифровые технологии. Курсовой проект.

В данной реализации майнинг очень простой – нет никакой задачи для решения майнерами, можно просто добавить новый блок.

Запустите Postman, создайте POST запрос на первый узел `http://localhost:3001/mineBlock`, в теле которого:

```
{
  "data": "Some data to the second block"
}
```

И нажмите Send



В терминале первого узла видим лог добавления блока:

```
block added: {"index":1,"previousHash":"8d9d5a7ff4a78042ea6737bf59c772f8ed27ef3c9b576eac1976c91aaf48d2de","timestamp":1682857282.344,"data":"Some data to the second block","hash":"1366718965a199c8fb769ded26fca2a6609575c25a0d52cb41fe34db7ad0e942"}
```

А в терминале второго синхронизацию:

```
Received message{"type":2,"data":[{"index":1,"previousHash":"8d9d5a7ff4a78042ea6737bf59c772f8ed27ef3c9b576eac1976c91aaf48d2de","timestamp":1682857282.344,"data":"Some data to the second block","hash":"1366718965a199c8fb769ded26fca2a6609575c25a0d52cb41fe34db7ad0e942"}]}
blockchain possibly behind. We got: 0 Peer got: 1
We can append the received block to our chain
```

Также обновите в браузере страницу с блоками любого узла:

```
localhost:3002/blocks
[{"index":0,"previousHash":"0","timestamp":1682839690,"data":"RUT-MIIT first block","hash":"8d9d5a7ff4a78042ea6737bf59c772f8ed27ef3c9b576eac1976c91aaf48d2de"}, {"index":1,"previousHash":"8d9d5a7ff4a78042ea6737bf59c772f8ed27ef3c9b576eac1976c91aaf48d2de","timestamp":1682857282.344,"data":"Some data to the second block","hash":"1366718965a199c8fb769ded26fca2a6609575c25a0d52cb41fe34db7ad0e942"}]
```

Простой майнинг работает, время внедрения Proof-of-Work.

## 7. Proof-of-Work

Для начала завершите работающие узлы (командой Ctrl+C в обоих терминалах).

В качестве демонстрации реализуем простейший Proof-of-Work алгоритм, в котором нужно найти хеш, который начинается на определенное количество нулей.

Отредактируем код `main.js`.

Добавим переменную для сложности (количества нулей). Серым отмечен код для ориентации в существующем, желтым добавочный:



## Цифровые технологии. Курсовой проект.

```
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;
```

Расширим класс Block, nonce – число, которое будет каждый раз увеличиваться с каждой попыткой поиска подходящего хеша (тем самым мы каждый раз ищем все новое число, с которым хеш будет соответствовать требованиям):

```
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;
  }
}
```

Изменим создание генезис-блока, передав 0-ые значения новых аргументов:

```
var getGenesisBlock = () => {
  return new Block(0, "0", 1682839690, "RUT-MIIT first block",
    "8d9d5a7ff4a78042ea6737bf59c772f8ed27ef3c9b576eac1976c91aaf48d2de", 0, 0);
};
```

Обновляем функции calculateHashForBlock и calculateHash:

```
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();
};
```

Обновляем код веб-сервера для операции mineBlock - вызываем метод mineBlock:

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



## Цифровые технологии. Курсовой проект.

```
app.get('/blocks', (req, res) => res.send(JSON.stringify(blockchain)));
app.post('/mineBlock', (req, res) => {
    //var newBlock = generateNextBlock(req.body.data);
    var newBlock = mineBlock(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));
};
```

Нижe добавляем реализацию функции mineBlock:

```
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("\n\"index\": " + nextIndex +
", \"previousHash\": "+previousBlock.hash+
        "\"timestamp\": "+nextTimestamp+", \"data\": "+blockData+
        ", \"\x1b[33mhash: " + nextHash + " \x1b[0m, " +
        "\"difficulty\": "+difficulty+
        " \x1b[33mnonce: " + nonce + " \x1b[0m ");
    }

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



## Цифровые технологии. Курсовой проект.

Идея майнинга простая - в цикле используется функция `calculateHash()`, чтобы генерировать хеш для нового блока и проверки, начинается ли он с заданного количества нулей, которое определяется переменной `difficulty`. Если хеш не начинается с необходимого количества нулей, `nonce` увеличивается, и генерация хеша повторяется до тех пор, пока не будет найден подходящий `nonce`.

### 8. Проверка работы

Выполните запуск двух узлов как в пункте 5.

Перейдите в Postman и также создайте POST запрос для добавление новой записи в блокчейн. Отправьте запрос, и наблюдайте в логе соответствующего узла сам процесс майнинга (поиска числа для подходящего хеша):

```
"index":1,"previousHash":"8d9d5a7ff4a78042ea6737bf59c772f8ed27ef3c9b576eac1976c91aaf48d2de","timestamp":1682859625.01,"data":"Some data to the second block","hash":"55a220b0d9a932e1c8fba8cb2d5c567a06b2be2a23b6fba566376df44eb49a6b","difficulty":4,"nonce":7661}
{"index":1,"previousHash":"8d9d5a7ff4a78042ea6737bf59c772f8ed27ef3c9b576eac1976c91aaf48d2de","timestamp":1682859625.011,"data":"Some data to the second block","hash":"497adf038daa20c015fac7f4cd5dc4ec9911c190ae7954b3f01082b194f16c4a","difficulty":4,"nonce":7662}
{"index":1,"previousHash":"8d9d5a7ff4a78042ea6737bf59c772f8ed27ef3c9b576eac1976c91aaf48d2de","timestamp":1682859625.011,"data":"Some data to the second block","hash":"47ec0e933e2af71a58fba55c630a7cf1fd68dbfc80bbb74fa30a3e6aa878f363","difficulty":4,"nonce":7663}
{"index":1,"previousHash":"8d9d5a7ff4a78042ea6737bf59c772f8ed27ef3c9b576eac1976c91aaf48d2de","timestamp":1682859625.011,"data":"Some data to the second block","hash":"0000d1ba532361c372efb2a8c16cf9535ee025cd5e5aa1fa75327990a19dddf4","difficulty":4,"nonce":132814}
```

Через некоторое время – не более минуты, такое число будет найдено - 132814 подходит:

```
"index":1,"previousHash":"8d9d5a7ff4a78042ea6737bf59c772f8ed27ef3c9b576eac1976c91aaf48d2de","timestamp":1682859685.933,"data":"Some data to the second block","hash":"0000d1ba532361c372efb2a8c16cf9535ee025cd5e5aa1fa75327990a19dddf4","difficulty":4,"nonce":132814}
block added: {"index":1,"previousHash":"8d9d5a7ff4a78042ea6737bf59c772f8ed27ef3c9b576eac1976c91aaf48d2de","timestamp":1682859685.933,"data":"Some data to the second block","hash":"0000d1ba532361c372efb2a8c16cf9535ee025cd5e5aa1fa75327990a19dddf4","difficulty":4,"nonce":132814}
Received message{"type":2,"data":[{"index":1,"previousHash":"8d9d5a7ff4a78042ea6737bf59c772f8ed27ef3c9b576eac1976c91aaf48d2de","timestamp":1682859685.933,"data":"Some data to the second block","hash":"0000d1ba532361c372efb2a8c16cf9535ee025cd5e5aa1fa75327990a19dddf4","difficulty":4,"nonce":132814}]}
```

Второй узел также получил информацию:

```
Received message{"type":2,"data":[{"index":1,"previousHash":"8d9d5a7ff4a78042ea6737bf59c772f8ed27ef3c9b576eac1976c91aaf48d2de","timestamp":1682859685.933,"data":"Some data to the second block","hash":"0000d1ba532361c372efb2a8c16cf9535ee025cd5e5aa1fa75327990a19dddf4","difficulty":4,"nonce":132814}]}
```

Проверяем в браузере список блоков:

```
localhost:3001/blocks
[{"index":0,"previousHash":"","timestamp":1682839690,"data":"RUT-MIIT first block","hash":"8d9d5a7ff4a78042ea6737bf59c772f8ed27ef3c9b576eac1976c91aaf48d2de","difficulty":0,"nonce":0}, {"index":1,"previousHash":"8d9d5a7ff4a78042ea6737bf59c772f8ed27ef3c9b576eac1976c91aaf48d2de","timestamp":1682859685.933,"data":"Some data to the second block","hash":"0000d1ba532361c372efb2a8c16cf9535ee025cd5e5aa1fa75327990a19dddf4","difficulty":4,"nonce":132814}]
```

### 9. Задание

В рамках курсового проекта необходимо реализовать собственный алгоритм майнинга, к примеру:

1. Другие требования к хешу;
2. Другая математическая задача, к примеру поиск простых чисел для хеша.