Федеральное государственное автономное образовательное учреждение высшего образования

"Российский университет транспорта" (МИИТ)

**Институт управления и цифровых технологий**

Кафедра «Цифровые технологии управления транспортными процессами»

**КУРСОВОЙ ПРОЕКТ**

по дисциплине«Цифровые технологии»

на тему «**Реализация блокчейн (Proof-of-Work)**»

Выполнил:

студент учебной группы УВП-212

Толстиков А. А.

Принял:

ассистент кафедры «ЦТУТП»

Городников А. И.

**Москва – 2023**

СОДЕРЖАНИЕ

[**ВВЕДЕНИЕ** 3](#_Toc137544186)

[**1.** **ОСНОВНАЯ ЧАСТЬ** 4](#_Toc137544187)

[**1.1Реализация блокчейн без Proof-of-Work** 4](#_Toc137544188)

[**1.2 Запуск сервера. Цепочка блоков** 4](#_Toc137544189)

[**1.3 Майнинг** 5](#_Toc137544190)

[**2.** **РЕАЛИЗАЦИЯ БЛОКЧЕЙН С PROOF-OF-WORK** 6](#_Toc137544191)

[**2.1Изменения в исходном коде. Реализация задачи** 6](#_Toc137544192)

[**2.2 Майнинг. Проверка** 6](#_Toc137544193)

[**ЗАКЛЮЧЕНИЕ** 8](#_Toc137544194)

[**ПРИЛОЖЕНИЕ А** 9](#_Toc137544195)

[**ПРИЛОЖЕНИЕ Б** 13](#_Toc137544196)

# **ВВЕДЕНИЕ**

Такую технологию как - блокчейн в наше время применяются в большинстве сфер. Они используются для проведения финансовых операций, идентификации пользователей, а также для обеспечения кибербезопасности. Блокчейн технологии актуальны для банковских учреждений, а также государственных организаций. У блокчейна есть ряд преимуществ, делающих эту технологию очень востребованной во всех перечисленных сферах. Например, мы можем отнести такие преимущества, как: прозрачность, открытость и децентрализацию. Это означает что никто не контролирует блокчейн, и все его участники равноправны. Также блокчейн обеспечивает полную анонимность, что делает транзакцию практически полностью безопасной. Говоря простым языком, блокчейн — это непрерывная цепочка блоков, содержащих какую-либо информацию. Связь между блоками обеспечивается при помощи хеш-суммы. Каждый блок имеет собственную хеш-сумму, а также содержит в себе хеш-сумму предыдущего блока. При изменении информации в блоке меняется хеш-сумма блока. Хеш-сумма – набор символов, полученных в результате шифрования каких-либо данных путем математических вычислений. Существует множество различных хеш-функций, позволяющих сгенерировать хеш-сумму. Данная курсовая работа позволит ближе познакомиться с понятием блокчейн, а также рассмотреть использование данной технологии на примере популярного в наше время майнинга. Суть майнинга предельно проста – пользователь при помощи вычислительных мощностей компьютера должен первым подтвердить серию транзакций, тем самым добавив блок в цепочку. В данной работе были реализованы два вида цепочки блоков, а именно реализация блокчейна без Proof-of-Work, а также реализация с Proof-of-Work. Далее рассмотрим каждый из этих видов.

# **ОСНОВНАЯ ЧАСТЬ**

## **1.1Реализация блокчейн без Proof-of-Work**

Для реализации блокчейна без POW был использован исходный код, позволяющий сгенерировать хеш, создать блоки, добавить их в блокчейн, а также запустить веб-сервер, используемый для обслуживания запросов от узла блокчейн-сети, обмена сообщениями между узлами. В исходном коде был изменен способ получения хеш-суммы. В данной работе была использована хеш-функция MD6.

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рисунок 1 - хеш-фунция

### **1.2 Запуск сервера. Цепочка блоков**

При помощи команды «$env:HTTP\_PORT=3001; $env:P2P\_PORT=6001; npm start» был запущен сервер, позволяющий увидеть нашу цепочку блоков. Для этого необходимо в браузере открыть адрес <http://localhost:3001/blocks>

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рисунок 2 - Цепочка блоков. Узел 1

Далее – запуск еще одного узла. Узел был запущен при помощи команды

«**$env:HTTP\_PORT=3002;$env:P2P\_PORT=6002;$env:PEERS="ws://’ip’:6001"; npm start**»

Откроем цепочку блоков через второй узел, для этого перейдем по адресу <http://localhost:3002/blocks>

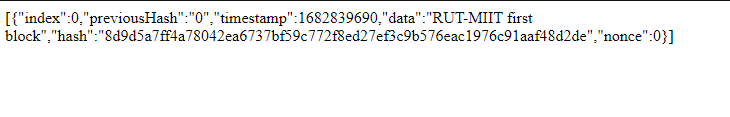


Рисунок 3 – Цепочка блоков. Узел 2

### **1.3 Майнинг**

В данной реализации майнинг простой, так как нет никакой задачи для решения, это означает, что каждый новый блок майнинга сводится к элементарному добавлению нового блока.

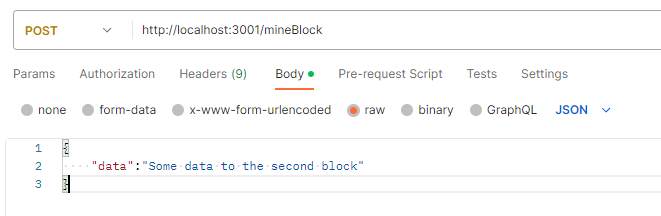
Для демонстрации майнинга воспользуемся Postman. Создадим POST запрос на первый узел

Рисунок 4 - Майнинг. Первый узел

Проверим терминал первого узла. Видим лог добавления блока

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Автоматически созданное описание

Рисунок 5 - Лог добавления блока

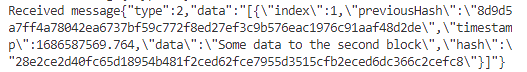
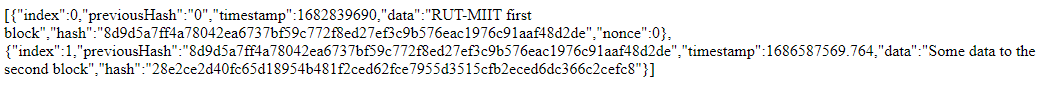
Проверим терминал второго узла. Видим синхронизацию

Рисунок 6 - Лог синхронизации

Проверим нашу цепочку блоков. Для этого обновим в браузере страницу (F5) с цепочкой

Рисунок 7 – Обновленная цепочка блоков

Блок добавился. Простой майнинг работает.

1. **РЕАЛИЗАЦИЯ БЛОКЧЕЙН С PROOF-OF-WORK**

Proof-of-Work(POW) – «доказательство работы». Это алгоритм, который требует определенное количество вычислений для выполнения заданного условия. POW обеспечивает большую безопасность и сохранность данных в блокчейне путем вычисления более сложной задачи, требующей большого объема вычислительных ресурсов. После решения задачи другие участники сети могут проверить правилен ли результат определенной задачи, что обеспечивает согласованность блокчейна между всеми ее пользователями. В данной курсовой работе в качестве POW была разработана математическая задача, которую алгоритм будет решать для подтверждения майнинга. Задача заключается в сложении всех четных цифр хеш-суммы и выяснении, является ли полученное число больше 100.

**2.1Изменения в исходном коде. Реализация задачи**

Для внедрения POW в исходном коде необходимо совершить некоторые изменения:

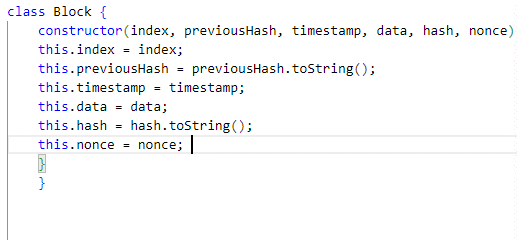
1. Отредактируем класс Block

Рисунок 8 - Изменения в классе Block

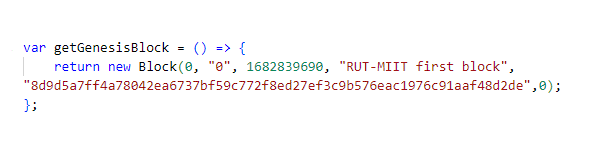
 2. Изменим создание генезис-блока.

Рисунок 9 - Изменения генезис блоке

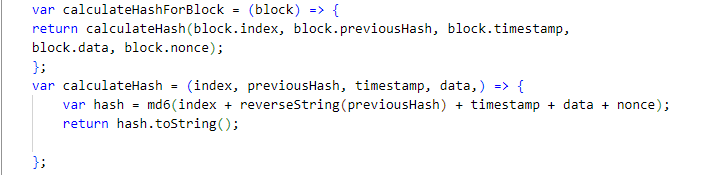
 3. Обновим функции calculateHashForBlock и calculateHash,

Рисунок 10 - Обновление функций

Рисунок 7 - Создание генезис-блока

4. Обновляем код веб-сервера для операции mineBlock

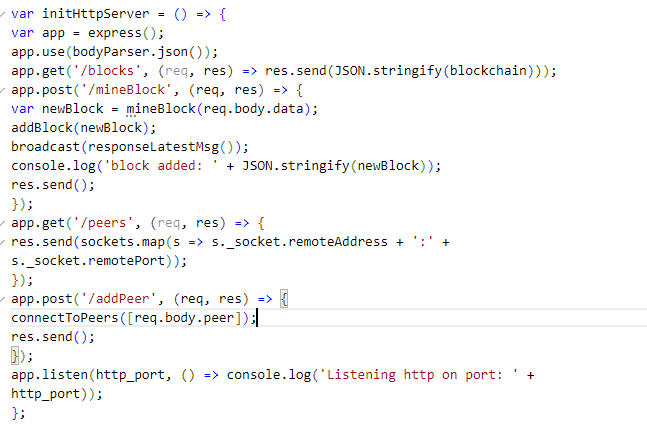


Рисунок 11 - Обновленный код веб-сервера

5. Реализация функции mineBlock

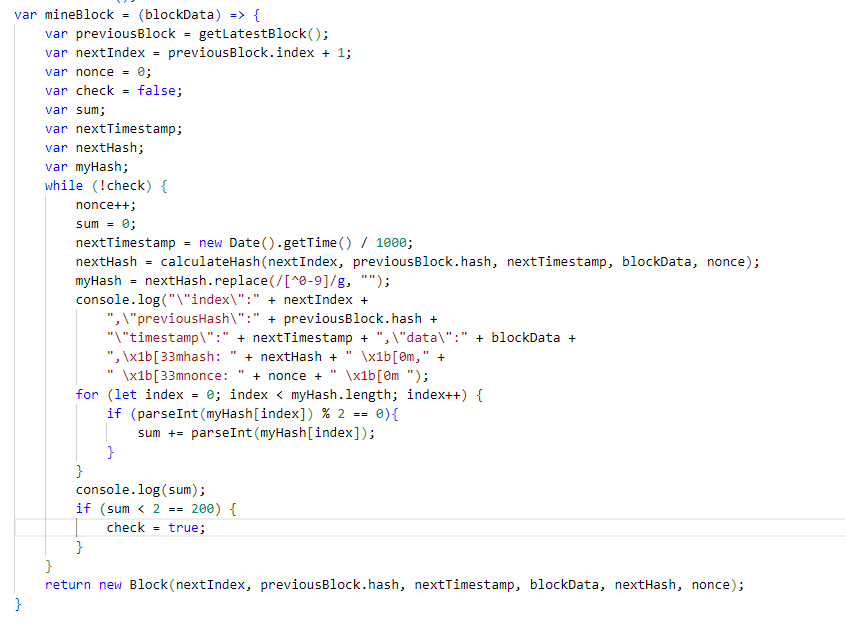


Рисунок 12 – Код функции mineBlock

**2.2 Майнинг. Проверка**

Продемонстрируем работу майнинга с POW.

Для этого создадим POST запрос на первый узел. В логе первого узла увидим процесс майнинга. С шестой попытки подходящее число найдено.

Второй узел также получил информацию

Проверим список блоков в браузере

Изображение выглядит как текст, Шрифт, линия, снимок экрана

Автоматически созданное описание

Рисунок 13 - Лог первого узла

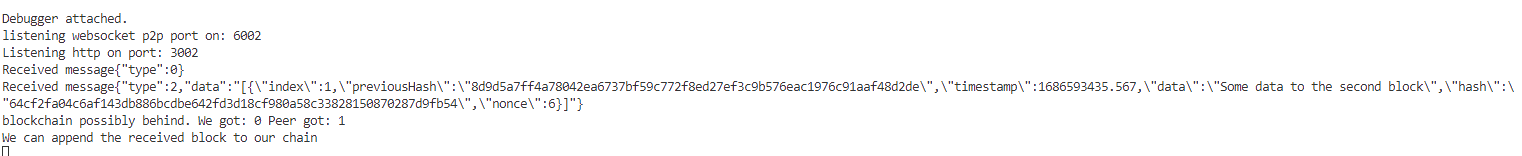


Рисунок 14 - Лог второго узла

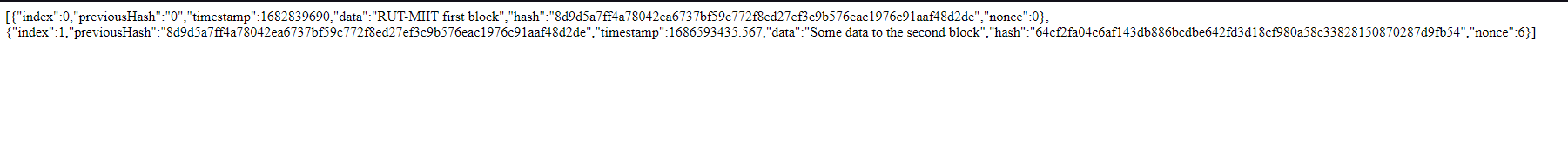


Рисунок 15 - Список блоков

# **ЗАКЛЮЧЕНИЕ**

В ходе работы над данным курсовым проектом была выполнена цель по закреплению и отработке на практике теоретических знаний о базовых принципах работы блокчейн технологий и майнинга. Знания были закреплены путем выполнения задачи по реализации блокчейна при помощи исходного кода. Основной задачей для достижения цели курсового проекта было создание двух видов блокчейна: 1. блокчейн с Proof-of-Work(со своей личной задачей) 2.блокчейн без Proof-of-Work. Для реализации данной задачи были выбраны метод хеширования MD6 и математическая задача по выяснению, является ли сумма четных цифр хеш-суммы больше числа 100. Задача была успешно выполнена, это подтверждает успешный майнинг в обоих видах блокчейна. Также была проведена проверка целостности цепочки блокчейна.

# **ПРИЛОЖЕНИЕ А**

Код реализации блокчейн без Proof-of-Work

'use strict';

var CryptoJS = require("crypto-js");

var express = require("express");

var bodyParser = require('body-parser');

var WebSocket = require("ws");

var md6 = require("md6-hash");

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(',') : [];

function reverseString(str){

    return str.split("").reverse().join("");

}

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();

    }

    }

var sockets = [];

var MessageType = {

QUERY\_LATEST: 0,

QUERY\_ALL: 1,

RESPONSE\_BLOCKCHAIN: 2

};

var getGenesisBlock = () => {

    return new Block(0, "0", 1682839690, "RUT-MIIT first block",

"8d9d5a7ff4a78042ea6737bf59c772f8ed27ef3c9b576eac1976c91aaf48d2de",0);

};

var blockchain = [getGenesisBlock()];

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));

};

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);

    };

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');

    }

    };

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);

    };

    var calculateHashForBlock = (block) => {

    return calculateHash(block.index, block.previousHash, block.timestamp,

    block.data);

    };

    var calculateHash = (index, previousHash, timestamp, data) => {

        var hash = md6(index + reverseString(previousHash) + timestamp + data);

        return hash.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;

    };

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;

};

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();

**ПРИЛОЖЕНИЕ Б**

Код реализации блокчейн с Proof-of-Work

'use strict';

var CryptoJS = require("crypto-js");

var express = require("express");

var bodyParser = require('body-parser');

var WebSocket = require("ws");

var md6 = require("md6-hash");

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(',') : [];

function reverseString(str){

    return str.split("").reverse().join("");

}

class Block {

    constructor(index, previousHash, timestamp, data, hash, nonce) {

    this.index = index;

    this.previousHash = previousHash.toString();

    this.timestamp = timestamp;

    this.data = data;

    this.hash = hash.toString();

    this.nonce = nonce;

    }

    }

var sockets = [];

var MessageType = {

QUERY\_LATEST: 0,

QUERY\_ALL: 1,

RESPONSE\_BLOCKCHAIN: 2

};

var getGenesisBlock = () => {

    return new Block(0, "0", 1682839690, "RUT-MIIT first block",

"8d9d5a7ff4a78042ea6737bf59c772f8ed27ef3c9b576eac1976c91aaf48d2de",0);

};

var blockchain = [getGenesisBlock()];

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);

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));

};

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);

    };

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');

    }

    };

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);

    };

    var calculateHashForBlock = (block) => {

    return calculateHash(block.index, block.previousHash, block.timestamp,

    block.data, block.nonce);

    };

    var calculateHash = (index, previousHash, timestamp, data,nonce) => {

        var hash = md6(index + reverseString(previousHash) + timestamp + data + nonce);

        return hash.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;

    };

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;

};

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();

var mineBlock = (blockData) => {

    var previousBlock = getLatestBlock();

    var nextIndex = previousBlock.index + 1;

    var nonce = 0;

    var check = false;

    var sum;

    var nextTimestamp;

    var nextHash;

    var myHash;

    while (!check) {

        nonce++;

        sum = 0;

        nextTimestamp = new Date().getTime() / 1000;

        nextHash = calculateHash(nextIndex, previousBlock.hash, nextTimestamp, blockData, nonce);

        myHash = nextHash.replace(/[^0-9]/g, "");

        console.log("\"index\":" + nextIndex +

            ",\"previousHash\":" + previousBlock.hash +

            "\"timestamp\":" + nextTimestamp + ",\"data\":" + blockData +

            ",\x1b[33mhash: " + nextHash + " \x1b[0m," +

            " \x1b[33mnonce: " + nonce + " \x1b[0m ");

        for (let index = 0; index < myHash.length; index++) {

            if (parseInt(myHash[index]) % 2 == 0){

                sum += parseInt(myHash[index]);

            }

        }

        console.log(sum);

        if (sum > 100) {

            check = true;

        }

    }

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

}