

const http = require('http');

const path = require('path');

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

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

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

const express = require('express');

const app = express();

const port = 8000;

const Response = require('./responseClass');

const validationWindow = 300;

const bitcoin = require('bitcoinjs-lib');

const bitcoinMessage = require('bitcoinjs-message');

// for gets

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({ extended: true }));

//[walletAddress]:[timeStamp]:starRegistry

//let blockchain = new bc();

//let address = '142BDCeSGbXjWKaAnYXbMpZ6sbrSAo3DpZ'

//let signature = 'IJtpSFiOJrw/xYeucFxsHvIRFJ85YSGP8S1AEZxM4/obS3xr9iz7H0ffD7aM2vugrRaCi/zxaPtkflNzt5ykbc0='

//let message = '142BDCeSGbXjWKaAnYXbMpZ6sbrSAo3DpZ:1532330740:starRegistry'

//console.log(bitcoinMessage.verify(message, address, signature))

let blockchain = new Blockchain;

app.get('/', (req, res) =>

res.sendFile(path.join(\_\_dirname + '/home.html')));

app.get('/block/:id', async (req, res) => {

const blockRes = await blockchain.getBlock(req.params.id);

if (blockRes) {

res.send(blockRes) // server response

} else {

res.status(404).send("The Block was NOT Found")

}

});

app.post('/block', async (req, res) => {

console.log('----------------------------');

if (!req.body.address || !req.body.star) {

res.status(400).json({

"status": 400,

message: "Please ensure to provide the star and address."

})

} else if (encodeURI(req.body.star.story).split(/%..|./).length - 1 > 500) {

res.status(400).json({

"status": 400,

message: "The Star story size is too big. Please ensure is is under 500 bytes."

})

} else {

let starIdx = validPool.findIndex(f => f.address === req.body.address);

console.log('Validated Star Index: ' + starIdx);

if (starIdx >= 0) {

req.body.star.story = new Buffer(req.body.star.story).toString('hex');

await blockchain.addBlock(new Block(req.body));

const height = await blockchain.getBlockHeight();

const response = await blockchain.getBlock(height);

res.send(response);

} else {

res.status(400).json({

"status": 400,

message: "The address could not be verified"

})

}

}

});

internal\_db = [];

app.post('/requestValidation', async (req, res) => {

if (!req.body.address) {

res.status(400).json({

"status": 400,

message: "Address must not be empty"

})

console.log('----------------------------');

console.log('Empty address.');

}

else {

let timeME = new Date().getTime().toString().slice(0, -3);

resp = new Response;

resp.validationWindow = validationWindow;

resp.address = req.body.address;

resp.requestTimeStamp = timeME;

resp.message = resp.address + ':' + resp.requestTimeStamp + ':starRegistry';

if (internal\_db.findIndex(f => f.address === req.body.address) === -1) {

console.log('Address received: ' + (req.body.address));

console.log('Request is valid for 5 minutes.');

console.log('Please sign/verify: ' + (req.body.message));

console.log('Please validate at \*/message-signature/validate');

console.log('The array length is: ' + internal\_db.length);

console.log('');

internal\_db.push(resp);

} else if (internal\_db.findIndex(f => f.address === req.body.address) >= 0) {

console.log('----------------------------');

let reqIdx = internal\_db.findIndex(f => f.address === req.body.address);

let timeStamp = internal\_db[reqIdx].requestTimeStamp;

let timeLeft = timeME - timeStamp;

console.log('Address: ' + (req.body.address));

console.log('timestamp: ' + timeME);

console.log('retrieved timestamp: ' + TimeStamp);

console.log('Time Remaining is: ' + timeLeft);

if (timeLeft <= validationWindow) {

console.log('............................');

console.log('Request already exists...');

console.log('Please validate at \*/message-signature/validate');

console.log('');

} else if (timeLeft > validationWindow) {

console.log('............................');

console.log('Expired request, a new request will be generated.');

console.log('Address received: ' + (req.body.address));

console.log('Request is valid for 5 minutes.');

console.log('Please validate and sing \*/message-signature/validate');

console.log('');

internal\_db.splice(reqIdx);

internal\_db.push(resp);

}

}

res.send(resp);

}

});

validPool = [];

app.post('/message-signature/validate', async (req, res) => {

console.log('----------------------------');

//console.log('req body address: '+ req.body.address)

if (!req.body.address || !req.body.signature) {

res.status(400).json({

"status": 400,

message: "Address & signature data must not be empty"

})

} else if (internal\_db.findIndex(f => f.address === req.body.address) === -1) {

console.log("A request for this address does not exist... submit at \*/requestValidation");

res.status(400).json({

"status": 400,

message: "A request for this address does not exist... submit at \*/requestValidation"

})

} else if (internal\_db.findIndex(f => f.address === req.body.address) >= 0) {

let reqIdx2 = internal\_db.findIndex(f => f.address === req.body.address);

let reqTimeStamp2 = internal\_db[reqIdx2].requestTimeStamp;

let message2 = internal\_db[reqIdx2].message;

let nowTime2 = new Date().getTime().toString().slice(0, -3)

console.log('Timestamp of signature receipt: ' + nowTime2);

let timeDiff2 = nowTime2 - reqTimeStamp2;

let status = {

address: req.body.address,

requestTimeStamp: reqTimeStamp2,

message: internal\_db[reqIdx2].message,

validationWindow: timeDiff2,

messageSignature: "invalid"

}

let sigValidity = bitcoinMessage.verify(message2, req.body.address, req.body.signature);;

if (!sigValidity) {

console.log('Invalid signature');

} else if (sigValidity) {

if (timeDiff2 <= validationWindow) {

console.log("Ownership of blockchain address is verified");

console.log("Please proceed to \*/block to complete star registration");

status.messageSignature = 'valid'

validPool.push(status);

console.log('display status object: ' + JSON.stringify(validPool[validPool.length - 1]));

} else {

console.log("Time limit exceeded, request expired, please resubmit");

}

}

let resp2 = {

registerStar: true,

status: status

}

res.send(resp2);

}

});

app.get('/stars/:address', async (req, res) => {

console.log('####################################3');

console.log('Received request: ' + req.params.address);

let lookup = req.params.address.split(':');

console.log('vlookup prefix: ' + lookup[0]);

console.log('vlookup value: ' + lookup[1]);

const starz = await blockchain.star\_validation();

starz.shift();

if (lookup[0] === 'address') {

const adrFinds = starz.filter(f => f.body.address === lookup[1]);

adrFinds.forEach(function(obj) {

obj.body.star.storyDecoded = (new Buffer(obj.body.star.story, 'hex')).toString();

});

console.log('adrFinds: ' + JSON.stringify(adrFinds));

if (adrFinds.length > 0) {

res.send(adrFinds) //

} else {

res.status(400).send("Address not found")

}

} else if (lookup[0] === 'hash') {

const hashFinds = starz.filter(f => f.hash === lookup[1]);

hashFinds.forEach(function(obj) {

obj.body.star.storyDecoded = (new Buffer(obj.body.star.story, 'hex')).toString();

});

console.log('Results: ' + JSON.stringify(hashFinds));

if (hashFinds.length > 0) {

res.send(hashFinds) // server response

} else {

res.status(400).send("Address not found in blockchain")

}

} else {

res.status(400).send("Request not found in blockchain")

}

});

app.listen(port,

() => console.log(`app listening on port ${port}!`));

/\*

app.get('/stars/hash/:id', async (req, res) => {

const star = await blockchain.starValidationHash(req.params.id);

if (star) {

res.send(star)

} else {

res.status(404).send("The Block was NOT Found")

}

});

app.get('/stars/address/:id', async (req, res) => {

const star = await blockchain.starValidationAddress(req.params.id);

if (star) {

res.send(star)

} else {

res.status(404).send("The Block was NOT Found")

}

});

\*/

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

const leveldb = require('./levelFunctions');

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

module.exports = class Blockchain {

constructor() {

this.getBlockHeight().then((height) => {

if (height === -1) {

this.addBlock(new Block("Genesis block")).then(() => console.log("Genesis block created"));

}

});

}

//

async addBlock(newBlock) {

const height = parseInt(await this.getBlockHeight());

newBlock.height = height + 1;

newBlock.time = new Date().getTime().toString().slice(0, -3);

//

if (newBlock.height > 0) {

const previousBlock = await this.getBlock(height);

newBlock.previousBlockHash = previousBlock.hash;

console.log('Previous block hash: ' + newBlock.previousBlockHash);

}

newBlock.hash = SHA256(JSON.stringify(newBlock)).toString();

console.log('New block hash: ' + newBlock.hash);

//

await leveldb.addBlock(newBlock.height, JSON.stringify(newBlock));

} //addBlock

//

async getBlockHeight() {

const height = await leveldb.getBlockHeight();

return height;

} //getBlockHeight

//

async getBlock(blockHeight) {

const block = await leveldb.getBlock(blockHeight);

//console.log(block);

return block;

} //getBlock

//

async validateBlock(blockHeight) {

// get block object

let block = await this.getBlock(blockHeight);

// get block hash

let blockHash = block.hash;

block.hash = '';

// generate block hash

let validBlockHash = SHA256(JSON.stringify(block)).toString();

// Compare

if (blockHash===validBlockHash) {

//console.log('Block ' + blockHeight + ' validation confirmed');

return true;

} else {

console.log('Block ' + blockHeight + ' hash invalid:\n' + blockHash + '<>' + validBlockHash);

return false;

}

} //validateBlock

//

async validateChain() {

let errorLog = [];

let previousHash = '';

let chainArray = [];

let validFlag2 = false;

const height = await this.getBlockHeight();

//

for (let i = 0; i <= height; i++) {

await this.getBlock(i).then((block) => {

chainArray.push(block);

let validFlag = this.validateBlock(block.height);

validFlag.then((result) => {

//console.log(result);

validFlag2 = result;

})

// checking if single block hash is valid

if (!validFlag2) {

errorLog.push(i);

// checking for a break in chain

}

if (block.previousBlockHash !== previousHash) {

errorLog.push(i);

}

previousHash = block.hash;

//print('Block' + chainArray);

//return chainArray;

});

} //loop

errorLog.shift(); // excluding genesis block, no previous hash

if (errorLog.length > 0) {

console.log('Block errors =' + errorLog.length);

console.log('Block index:' + errorLog);

console.log('Errors detected');

} else {

//console.log('No errors detected');

//console.log(block);

}

console.log('chainArray Length: ################################################' + chainArray.length);

console.log('chainArray Length: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*' + JSON.stringify(chainArray));

return chainArray;

} //validateChain

async star\_validation() {

let star\_blocks = [];

const height = await this.getBlockHeight();

//

for (let i = 0; i <= height; i++) {

await this.getBlock(i).then((block) => {

star\_blocks.push(block);

console.log('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*BLOCK INFO\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

console.log(JSON.stringify(star\_blocks[i]));

});

} //for loop

return star\_blocks;

}

} //Blockchain Class