**Cryptocurrency**

**Practical No.2**

**Aim:** To implement immutable public/private blockchain.

**1.Write a program to create the chain with Genesis block and adding block into blockchain and validating the chain for any alteration. (Part-I)**

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

class Block

{

constructor(index, timestamp, data, previousHash = '')

{

this.index=index; this.timestamp=timestamp;

this.data=data;

this.previousHash=previousHash; this.hash=this.calculateHash();} calculateHash() {

return SHA256(this.index + this.previousHash + this.timestamp +

JSON.stringify(this.data) + this.nonce).toString(); }

}

class Blockchain

{ constructor()

{

this.chain=[this.createGenesisBlock()];

}

createGenesisBlock()

{

return new Block(0,"02/11/2021", "Genesis Block","0");

}

getLatestBlock()

{

return this.chain[this.chain.length -1];

}

addBlock(newBlock)

{

newBlock.previousHash=this.getLatestBlock().hash; newBlock.hash= newBlock.calculateHash();

this.chain.push(newBlock);

} isChainValid()

{ for (let i=1;i<this.chain.length;i++) { const currentBlock = this.chain[i]; const previousBlock = this.chain[i-1];

if(currentBlock.hash !== currentBlock.calculateHash())

{

return false;

} if(currentBlock.previousHash !== previousBlock.hash)

{ return false;

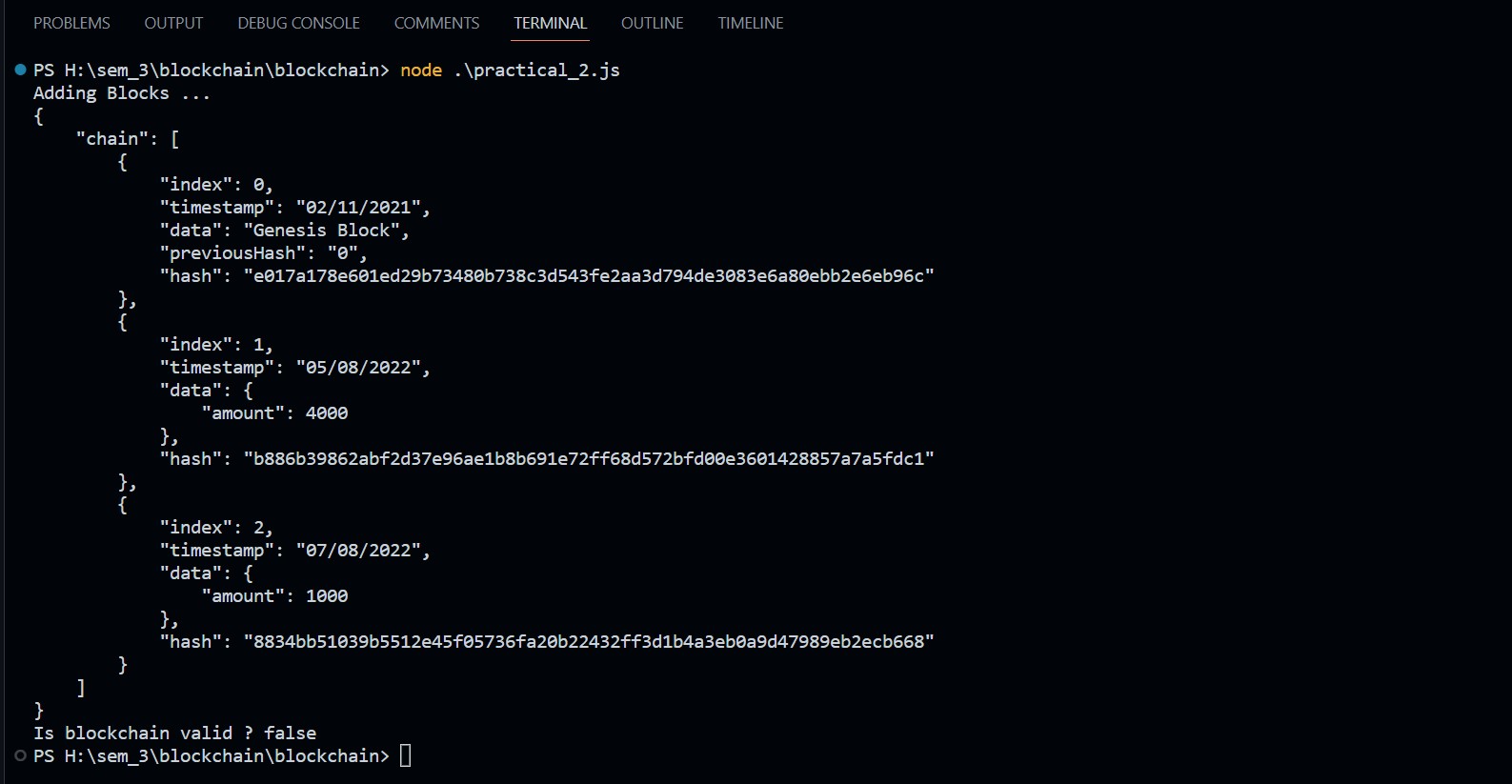
} } return true;

}}let MyCoin = new Blockchain(); console.log("Adding Blocks ...");

MyCoin.addBlock(new Block(1,"10/03/2021", { amount: 4000})); MyCoin.addBlock(new Block(2,"23/09/2021", { amount: 1000 })); console.log(JSON.stringify(MyCoin, null,4));

console.log('Is blockchain valid? ' + MyCoin.isChainValid());

**Output:**



**2. Write a program to implementing proof of work for blockchain. (Part-II)** const SHA256 = require('crypto-js/sha256'); class Block

{

constructor(index, timestamp, data, previousHash ='')

{

this.index=index; this.timestamp=timestamp;

this.data=data;

this.previousHash=previousHash; this.hash=this.calculateHash(); this.nonce=0;

}

calculateHash()

{ return

SHA256(this.index+this.previousHash+this.timestamp+JSON.stringify(this.data)+this.nonce) .toString();

}

//for difficulty

mineBlock(difficulty)

{

while(this.hash.substring(0,difficulty) !== Array(difficulty+1).join("0"))

{

this.nonce++;

this.hash=this.calculateHash();

}

console.log("Block mined: "+ this.hash);

}

}

class Blockcahin

{

constructor()

{

this.chain = [this.createGenesisBlock()]; this.difficulty = 4; //for difficulty

}

createGenesisBlock()

{

return new Block(0,"20/10/2021","Genesis Block","0");

}

getLatestBlock()

{

return this.chain[this.chain.length -1];

}

addBlock(newBlock)

{

newBlock.previousHash=this.getLatestBlock().hash; //newBlock.hash = newBlock.calculateHash(); newBlock.mineBlock(this.difficulty); //for difficulty this.chain.push(newBlock);

}

isChainValid()

{

for(let i=1; i<this.chain.length;i++)

{

const currentBlock = this.chain[i]; const previousBlock = this.chain[i-1]; if(currentBlock.hash !== currentBlock.calculateHash())

{

return false;

}

if(currentBlock.previousHash !== previousBlock.hash)

{

return false;

} }

return true;

}

}

let shCoin = new Blockcahin(); console.log('Mining the Block 1....');

shCoin.addBlock(new Block(1,"21/10/2021",{ amount: 4}));

console.log('Mining the Block 2....');

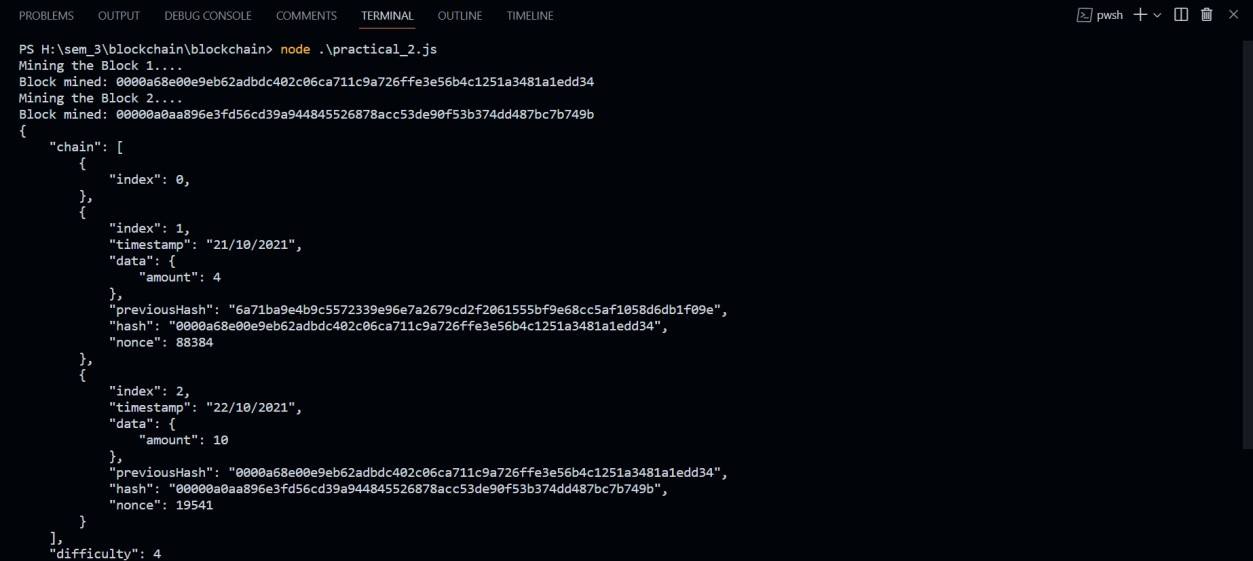
shCoin.addBlock(new Block(2,"22/10/2021",{ amount: 10}));

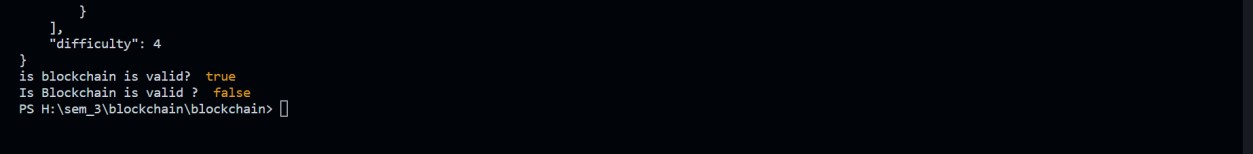
console.log(JSON.stringify(shCoin,null,4))

console.log('is blockchain is valid? ', shCoin.isChainValid());

shCoin.chain[1].data = { amount: 100}; shCoin.chain[1].hash = shCoin.chain[1].calculateHash(); console.log('Is Blockchain is valid ? ', shCoin.isChainValid());

**Output:**





**3. Write a program to add multiple transactions into block and give reward to miner for successful mining of block in blockchain. (Part-III)** const SHA256 = require('crypto-js/sha256'); class Transaction{

constructor(fromAddress, toAddress, amount)

{

this.fromAddress=fromAddress; this.toAddress=toAddress;

this.amount=amount;

}

}

class Block

{

constructor(timestamp, transactions, previousHash ='')

{

this.timestamp=timestamp; this.transactions=transactions; this.previousHash=previousHash; this.hash=this.calculateHash(); this.nonce=0;

}

calculateHash()

{

return SHA256(this.previousHash + this.timestamp + JSON.stringify(this.transactions) + this.nonce).toString();

}

mineBlock(difficulty)

{

while(this.hash.substring(0,difficulty) !== Array(difficulty+1).join("0"))

{

this.nonce++;

this.hash=this.calculateHash();

}

console.log("Block mined: "+ this.hash);

}

}

class Blockcahin

{

constructor()

{

this.chain = [this.createGenesisBlock()]; this.difficulty = 4; //for difficulty this.pendigTransactions=[];

this.miningreword=100;

}

createGenesisBlock()

{

return new Block(Date.now(),"Genesis Block","0");

}

getLatestBlock()

{

return this.chain[this.chain.length -1];

}

minePendingTransaction(miningRewordAddress)

{

let block=new Block(Date.now(),this.pendigTransactions);

block.mineBlock(this.difficulty); console.log("Block Successfully mined"); this.chain.push(block); this.pendigTransactions=[new Transaction(null,

miningRewordAddress,this.miningreword)];

}

createTransaction(transaction)

{

this.pendigTransactions.push(transaction);

}

getBalaceofAddress(address)

{ let balance=0;

for(const block of this.chain)

{

for(const trans of block.transactions)

{

if(trans.fromAddress == address)

{

balance -= trans.amount;

}

if(trans.toAddress == address)

{

balance += trans.amount;

}

}

} return balance; }

isChainValid()

{

for(let i=1; i<this.chain.length;i++)

{

const currentBlock = this.chain[i]; const previousBlock = this.chain[i-1];

if(currentBlock.hash !== currentBlock.calculateHash())

{

return false;

}

if(currentBlock.previousHash !== previousBlock.hash)

{

return false;

} }

return true;

}

}

let shCoin = new Blockcahin(); shCoin.createTransaction(new Transaction('address1', 'address2', 100)); shCoin.createTransaction(new Transaction('address2', 'address1', 50));

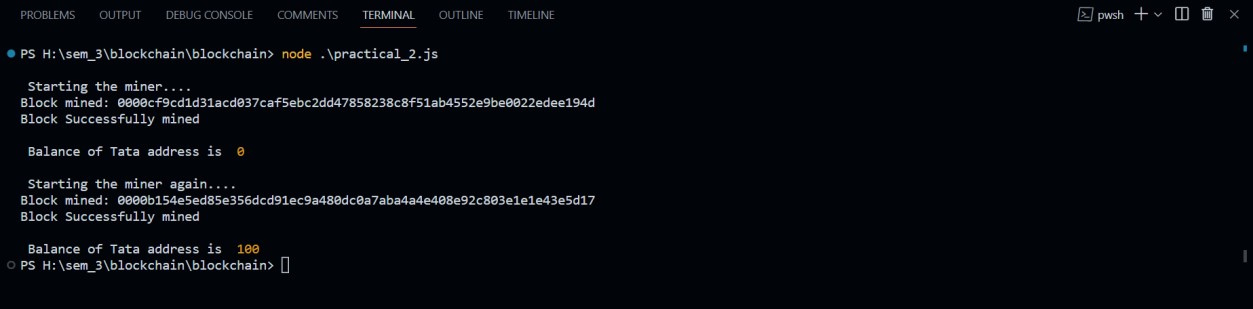
console.log("\n Starting the miner...."); shCoin.minePendingTransaction('Tata-address');

console.log('\n Balance of Tata address is ', shCoin.getBalaceofAddress('Tata-address'));

console.log("\n Starting the miner again...."); shCoin.minePendingTransaction('Tata-address');

console.log('\n Balance of Tata address is ', shCoin.getBalaceofAddress('Tata-address'));

**Output:**



**4. Write a program to sign the transaction with private key and verify the signed transactions for blockchain. (Part-IV**

# Blockchain.js

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

const EC = require('elliptic').ec;

const ec = new EC('secp256k1');

class Transaction{

constructor(fromAddress, toAddress, amount)

{

this.fromAddress = fromAddress; this.toAddress = toAddress;

this.amount = amount;

}

calculateHash()

{

return SHA256(this.fromAddress + this.toAddress + this.amount).toString();

}

signTransaction(signingKey)

{

if(signingKey.getPublic('hex') !== this.fromAddress)

{

throw new Error('You cannot sign transactions for other wallets!');

}

const hashTx = this.calculateHash(); const sig = signingKey.sign(hashTx, 'base64'); this.signature= sig.toDER('hex');

} isValid()

{

if(this.fromAddress === null) return true;

if(!this.signature || this.signature.length === 0)

{

throw new Error('No signature in this transaction');

}

const publicKey = ec.keyFromPublic(this.fromAddress, 'hex'); return publicKey.verify(this.calculateHash(), this.signature);

}

} class Block {

constructor(timestamp, transactions, previousHash = '')

{

this.timestamp=timestamp; this.transactions=transactions; this.previousHash=previousHash; this.hash=this.calculateHash(); this.nonce = 0;

}

calculateHash()

{

return SHA256(this.previousHash + this.timestamp + JSON.stringify(this.transactions) + this.nonce).toString();

}

mineBlock(difficulty)

{

while(this.hash.substring(0,difficulty) !== Array(difficulty + 1).join("0"))

{

this.nonce++;

this.hash = this.calculateHash();

}

console.log("Block mined: " + this.hash);

}

hasValidTransactions()

{ for(const tx of this.transactions)

{

if(!tx.isValid())

{ return false;

}

}

return true;

}

}

class Blockchain

{

constructor()

{

this.chain=[this.createGenesisBlock()];

this.difficulty = 2; this.pendingTransactions = [];

this.miningReward = 100;

}

createGenesisBlock()

{

return new Block("20/10/2021", "Genesis Block","0");

}

getLatestBlock()

{

return this.chain[this.chain.length -1];

}

minePendingTransactions(miningRewardAddress)

{

const rewardTx = new Transaction(null, miningRewardAddress,this.miningReward ); this.pendingTransactions.push(rewardTx);

let block = new Block(Date.now(), this.pendingTransactions); block.mineBlock(this.difficulty);

console.log("Block successfully mined!");

this.chain.push(block);

this.pendingTransactions = [];

}

addTransaction(transaction)

{

if(!transaction.fromAddress || !transaction.toAddress)

{

throw new Error('Transaction must include from and to address');

}

if(!transaction.isValid())

{

throw new Error('Cannot add invalid transaction to chain');

}

this.pendingTransactions.push(transaction);

}

getBalanceOfAddress(address)

{

let balance = 0;

for(const block of this.chain)

{

for(const trans of block.transactions)

{

if(trans.fromAddress == address)

{

balance -= trans.amount;

}

if(trans.toAddress == address)

{

balance += trans.amount;

}

}

}

return balance;

}

isChainValid()

{

for (let i=1;i<this.chain.length;i++)

{

const currentBlock = this.chain[i];

const previousBlock = this.chain[i-1];

if(!currentBlock.hasValidTransactions())

{

return false;

}

if(currentBlock.hash !== currentBlock.calculateHash())

{

return false;

}

if(currentBlock.previousHash !== previousBlock.hash)

{

return false;

}

}

return true;

}

}

module.exports.Blockchain = Blockchain;

module.exports.Transaction = Transaction

**key\_generator.js**

const EC = require('elliptic').ec;

const ec = new EC('secp256k1');

const key = ec.genKeyPair(); const publicKey = key.getPublic('hex');

const privateKey = key.getPrivate('hex');

console.log();

console.log('Private key: ', privateKey);

console.log(); console.log('Public key: ', publicKey);

# main.js

const {Blockchain, Transaction} = require('./blockchain');

const EC = require('elliptic').ec;

const ec = new EC('secp256k1');

const myKey =

ec.keyFromPrivate('7b957c401f95046875635c16d78e2754535ad51a62841b31ef0e3dcafcc97ca

4');

const myWalletAddress = myKey.getPublic('hex');

let myCoin = new Blockchain();

const tx1 = new Transaction(myWalletAddress,'public key goes here',10); tx1.signTransaction(myKey);

myCoin.addTransaction(tx1);

console.log('\n Starting the miner.');

myCoin.minePendingTransactions(myWalletAddress);

console.log('\n Balance of myWalletAddress is',myCoin.getBalanceOfAddress(myWalletAddress));

console.log('Is chain valid? ', myCoin.isChainValid());

myCoin.chain[1].transactions[0].amount = 15; console.log('Is chain valid? ', myCoin.isChainValid());

**Output:**

