LDRP INSTITUTE OF TECHNOLOGY AND RESEARCH

GANDHINAGAR

DEPARTMENT OF COMPUTER ENGINEERING

&

INFORMATION TECHNOLOGY

CERTIFICATE

Mr.Yagnik Akbari of **7th CE-b**, Enrollment No.: **20BECE30003**has satisfactorily completed his/her term work in **Blockchain Technology (CT703D-N)** for the termending in _____.

Date: 18/10/21

Prof. Occhav Patel — **Subject Coordinator**

Dr. Sandip modha — **HOD-CE**

INDEX

PRACTICAL -1

Aim: 1. Understanding Block using

- **https://tools.superdatascience.com/Blockchain/block**.
- The block in the block chain plays an important role during transaction of bitcoins , storing information related user .

OUTPUT:

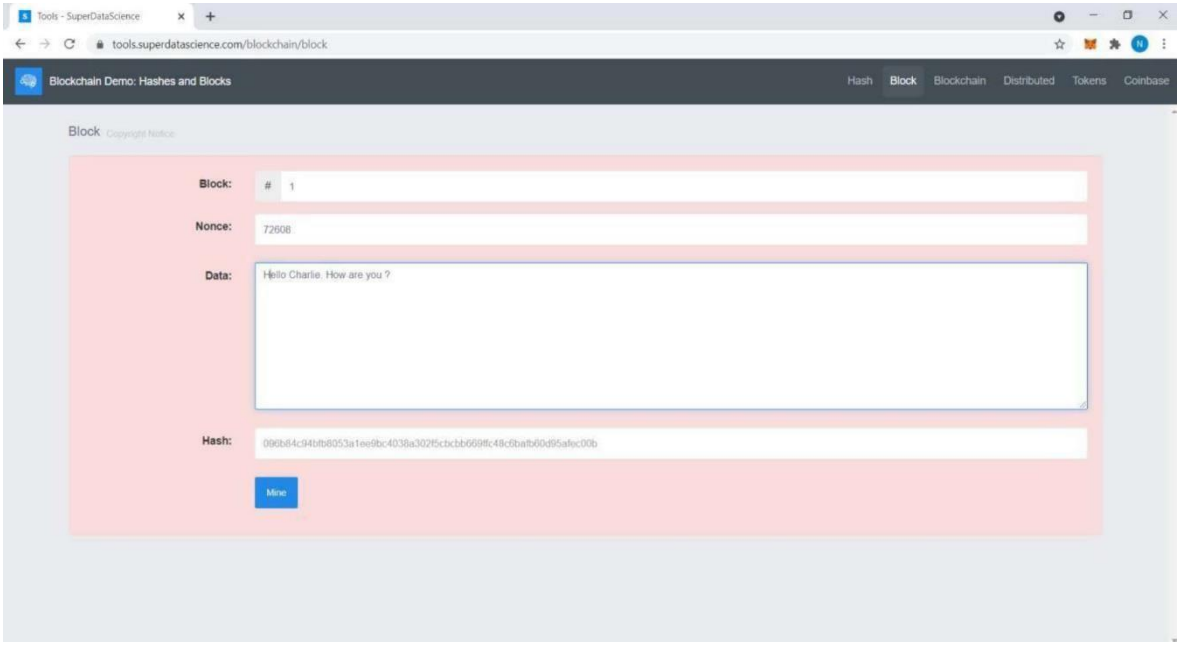

Fig. 1.1 Block

2. Understanding Block chain using

https://tools.superdatascience.com/Blockchain/Blockchain

- Block chain is a system of recording information in a way that makes it difficult or impossible to change, hack, or cheat the system.. Each block in the chain contains a number of transactions, and every time a new transaction occurs on the block chain, arecord of that transaction is added to every participant's ledger.

OUTPUT:

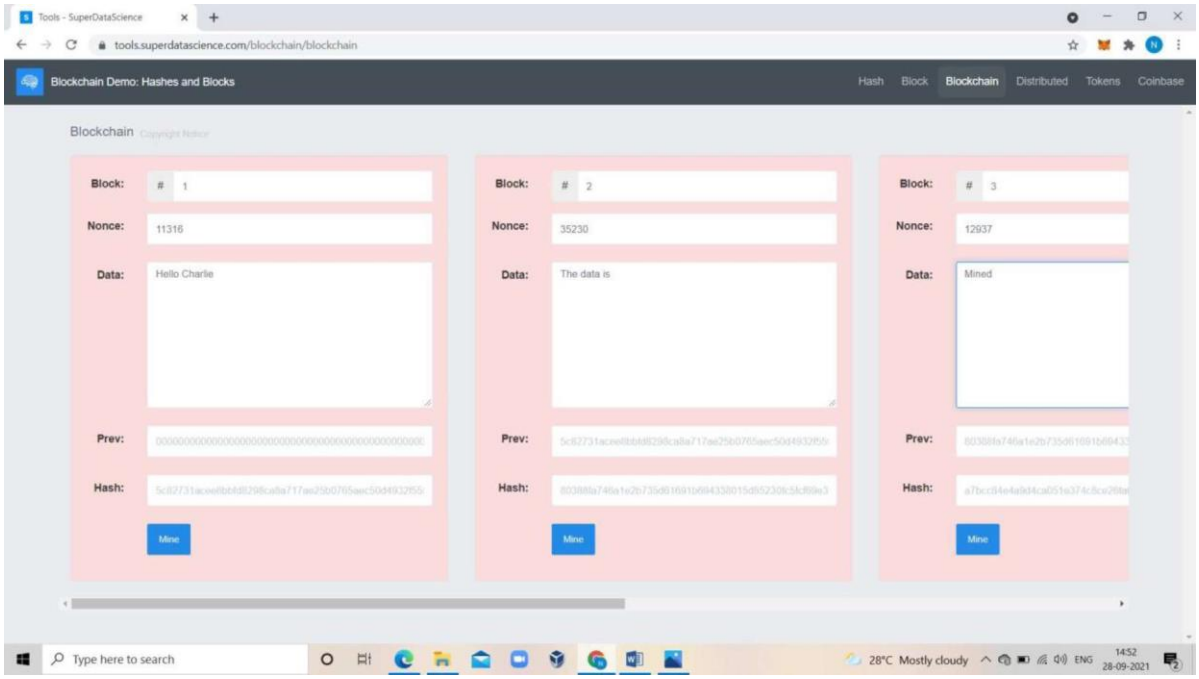

Fig. 1.2 Blockchain

3 .Understanding Distributed block chain using.

https://tools.superdatascience.com/Blockchain/Blockchain

- A distributed ledger is a database that is consensually shared and synchronized across multiple sites, institutions, or geographies , accessible by multiple people. Underlying distributed ledgers is the same technology that is used by blockchain, which is the technology that is used by bitcoin.

OUTPUT:

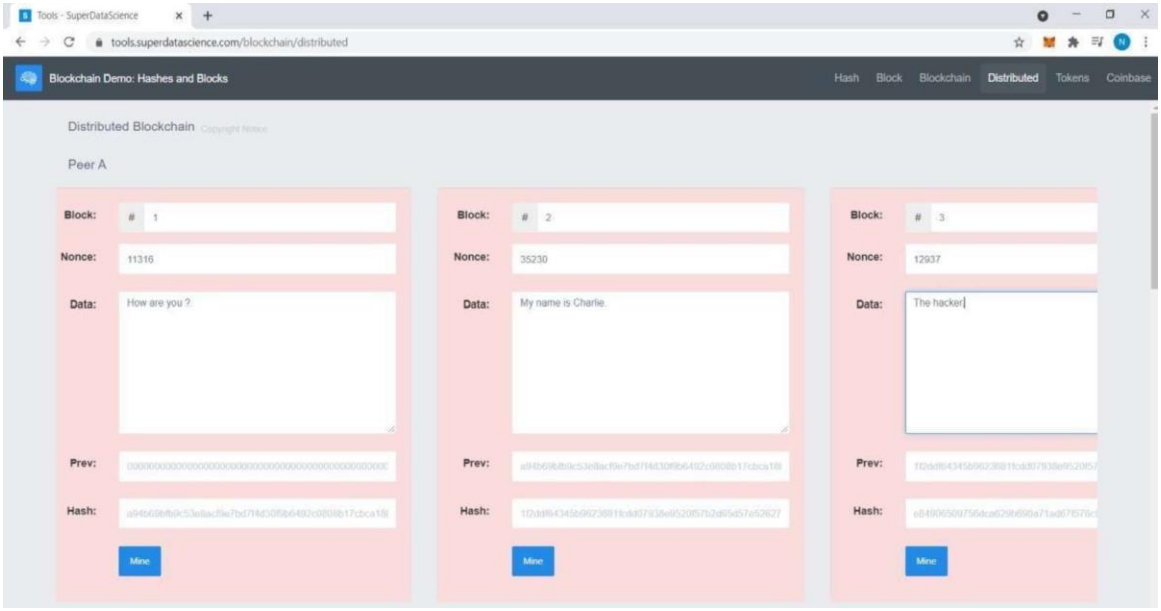

Fig. 1.3 Distributed Blockchain

4 .Understanding Tokens using.

https://tools.superdatascience.com/Blockchain/Blockchain

- TOKENS are digital assets defined by a project or smart contract and built on a specific block chain.

OUTPUT:

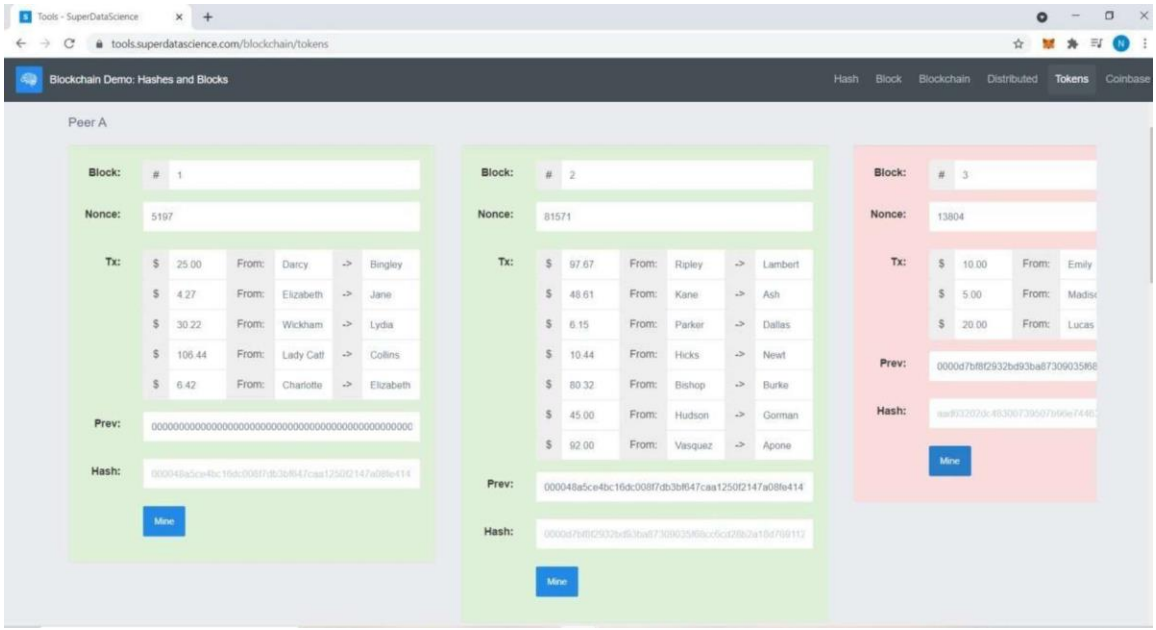

Fig. 1.4 Peer A

5 .Understanding Coin base using.

https://tools.superdatascience.com/Blockchain/Blockchain

- Coin base is an easy way for those who are new to cryptocurrency to get started. Its easy-to-use interface lets people buy and sell crypto in just a few clicks. While not every type of cryptocurrency is supported, you will find many of the most popular coins there.

OUTPUT:

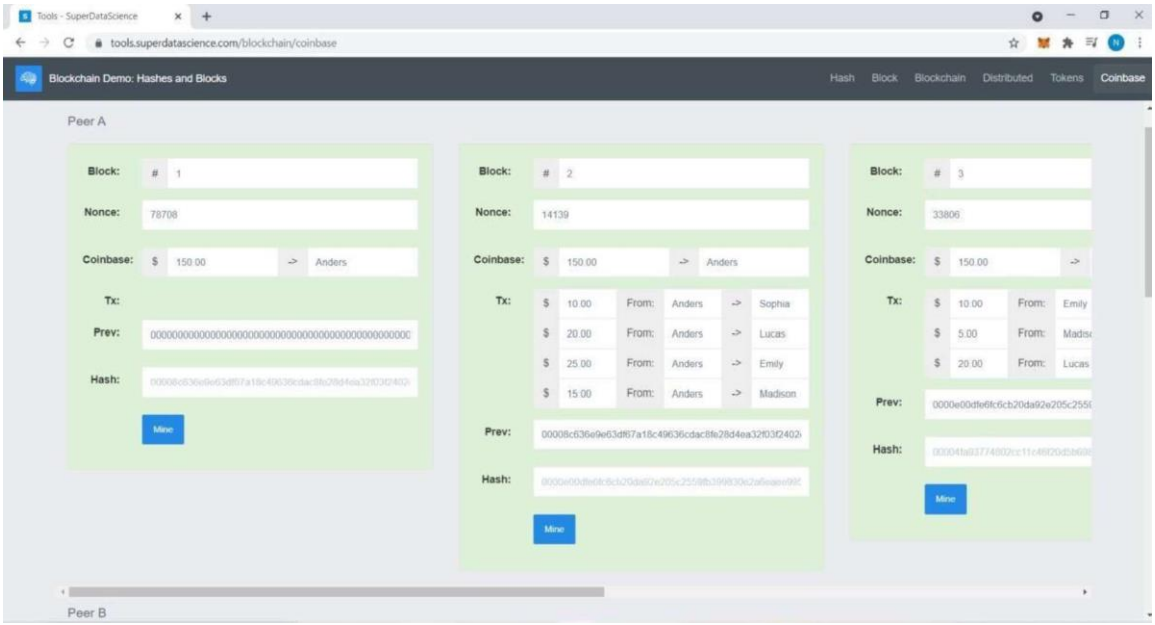

Fig. 1.5 Peer A

PRACTICAL -2

Using JavaScript Perform following

Aim: 1. Creating a block chain using Javascript.

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

class Block
{
  constructor(timestamp, data, previousHash = '')
  {
                this.previousHash = previousHash;
                this.timestamp = timestamp;
                this.data = data;
                this.hash = this.calculateHash();
  }
}

calculateHash()
{
   return SHA256(this.previousHash + this.timestamp + JSON.stringify(this.data)).toString();
}

class Blockchain
{
        constructor()
        {
                this.chain = [];
        }

        createGenesisBlock()
        {
                return new Block("01/01/2017", "Genesis block", "0");
        }

        getLatestBlock()
        {
                return this.chain[this.chain.length - 1];
        }

        addBlock(newBlock)
        {
                newBlock.previousHash = this.getLatestBlock().hash;

                // Calculate the hash of the new block
                newBlock.hash =  newBlock.calculateHash();

                // Now the block is ready and can be added to chain!
                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;
      }
        }

   if(this.chain[0] !== this.createGenesisBlock())
        {
        return false;
   }

   return true;
}

let savjeeCoin = new Blockchain();
savjeeCoin.addBlock(new Block("20/07/2017", { amount: 4 }));
savjeeCoin.addBlock(new Block("22/07/2017", { amount: 10 }));

console.log('Blockchain valid? ' + savjeeCoin.isChainValid());

savjeeCoin.chain[1].data = { amount: 100 };

savjeeCoin.chain[1].hash = savjeeCoin.chain[1].calculateHash():
console.log('Blockchain valid? ' + savjeeCoin.isChainValid());
```

2. Implementing Proof-Of-Work using Javascript.

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

class Block
{
	constructor(timestamp, data, previousHash = '')
	{
		this.previousHash = previousHash;
		this.timestamp = timestamp;
		this.data = data;
		this.hash = this.calculateHash();
		this.nonce = 0;
	}
	calculateHash()
	{
		return SHA256(this.index + this.previousHash + this.timestamp
+	JSON.stringify(this.data) + 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 Blockchain
{
	constructor()
	{
		this.chain = [this.createGenesisBlock()];
	this.difficulty = 3;
	}

	createGenesisBlock()
	{
		return new Block("01/01/2017", "Genesis block", "0");
	}

	getLatestBlock()
	{
		return this.chain[this.chain.length - 1];
```

```
        }
        addBlock(newBlock)
        {
                newBlock.previousHash = this.getLatestBlock().hash;
                newBlock.mineBlock(this.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;
      }
        }

   if(this.chain[0] !== this.createGenesisBlock())
        {
        return false;
   }

   return true;
}

let savjeeCoin = new Blockchain();
console.log('Mining block 1...');
savjeeCoin.addBlock(new Block(1, "20/07/2017", { amount: 4 }));

console.log('Mining block 2...');
savjeeCoin.addBlock(new Block(2, "20/07/2017", { amount: 8 }));
```

3. Miner Rewards and Transaction using

Javascript.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, data, previousHash = '')
        {
                this.previousHash = previousHash;
      this.timestamp  =  timestamp;
      this.transactions = transactions;
      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 Blockchain
{
        constructor()
        {
                this.chain = [this.createGenesisBlock()];
                this.difficulty = 5;
```

```
        this.pendingTransactions = [];
        this.miningReward = 100;
    }

    createGenesisBlock()
    {
        return new Block("01/01/2017", "Genesis block", "0");
    }

    getLatestBlock()
    {
        return this.chain[this.chain.length - 1];
    }

    minePendingTransactions(miningRewardAddress)
    {
        let block = new Block(Date.now(), this.pendingTransactions);
        block.mineBlock(this.difficulty);
        this.chain.push(block);
        this.pendingTransactions = [
        new Transaction(null, miningRewardAddress, this.miningReward)
        ];
    }

    createTransaction(transaction)
    {
        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;
    }

    addBlock(newBlock)
```

```
        {
                newBlock.previousHash = this.getLatestBlock().hash;
                newBlock.mineBlock(this.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;
      }
        }

   if(this.chain[0] !== this.createGenesisBlock())
        {
        return false;
   }

   return true;
}

let savjeeCoin = new Blockchain();

console.log('Creating some transactions...');
savjeeCoin.createTransaction(new Transaction('address1', 'address2', 100));
savjeeCoin.createTransaction(new Transaction('address2', 'address1', 50));

console.log('Starting the miner...');
savjeeCoin.minePendingTransactions('xaviers-address');

console.log('Balance of Xaviers address is', savjeeCoin.getBalanceOfAddress('xaviers-
address'));

console.log('Starting the miner again!');
savjeeCoin.minePendingTransactions("xaviers-address");
console.log('Balance of Xaviers address is', savjeeCoin.getBalanceOfAddress('xaviers-
address'));
```

4. Signing Transactions using Javascript.

```
const crypto = require('crypto');
const EC = require('elliptic').ec;
const ec = new EC('secp256k1');
const debug = require('debug')('savjeecoin:blockchain');

class Transaction {
  constructor(fromAddress, toAddress, amount) {
    this.fromAddress = fromAddress;
    this.toAddress = toAddress;
    this.amount = amount;
    this.timestamp = Date.now();
  }


  calculateHash() {
    return crypto.createHash('sha256').update(this.fromAddress + this.toAddress + this.amount
+ this.timestamp).digest('hex');
  }

  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.previousHash = previousHash;
    this.timestamp = timestamp;
    this.transactions = transactions;
    this.nonce = 0;
```

```
    this.hash = this.calculateHash();
  }
  calculateHash() {
    return crypto.createHash('sha256').update(this.previousHash + this.timestamp +
JSON.stringify(this.transactions) + this.nonce).digest('hex');
  }

  mineBlock(difficulty) {
    while (this.hash.substring(0, difficulty) !== Array(difficulty + 1).join('0')) {
      this.nonce++;
      this.hash = this.calculateHash();
    }

    debug(`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(Date.parse('2017-01-01'), [], '0');
  }
  getLatestBlock() {
    return this.chain[this.chain.length - 1];
  }

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

    const block = new Block(Date.now(), this.pendingTransactions,
this.getLatestBlock().hash);
    block.mineBlock(this.difficulty);

    debug('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');
  }

  // Verify the transactiion
  if (!transaction.isValid()) {
    throw new Error('Cannot add invalid transaction to chain');
  }

  if (transaction.amount <= 0) {
    throw new Error('Transaction amount should be higher than 0');
  }

  // Making sure that the amount sent is not greater than existing  balance
  if (this.getBalanceOfAddress(transaction.fromAddress) < transaction.amount) {
    throw new Error('Not enough balance');
  }

  this.pendingTransactions.push(transaction);
  debug('transaction added: %s',  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;
      }
    }
  }

  debug('getBalanceOfAdrees: %s', balance);
  return balance;
}
getAllTransactionsForWallet(address) {
  const txs = [];

  for (const block of this.chain) {
    for (const tx of block.transactions) {
      if (tx.fromAddress === address || tx.toAddress === address) {
```

```
          txs.push(tx);
        }
      }
    }

    debug('get transactions for wallet count: %s', txs.length);
    return txs;
  }


  isChainValid() {
    // Check if the Genesis block hasn't been tampered with by comparing
    // the output of createGenesisBlock with the first block on our chain
    const realGenesis = JSON.stringify(this.createGenesisBlock());

    if (realGenesis !== JSON.stringify(this.chain[0])) {
      return false;
    }

    // Check the remaining blocks on the chain to see if there hashes and
    // signatures are correct
    for (let i = 1; i < this.chain.length; i++) {
      const currentBlock = this.chain[i];
      const previousBlock = this.chain[i - 1];

      if (previousBlock.hash !== currentBlock.previousHash) {
        return false;
      }

      if (!currentBlock.hasValidTransactions()) {
        return false;
      }

      if (currentBlock.hash !== currentBlock.calculateHash()) {
        return false;
      }
    }

    return true;
  }
}

module.exports.Blockchain = Blockchain;
module.exports.Block = Block;
module.exports.Transaction = Transaction;
```

5. Angular Frontend using Javascript.

```
const crypto = require('crypto');
const EC = require('elliptic').ec;
const ec = new EC('secp256k1');
const debug = require('debug')('savjeecoin:blockchain');

class Transaction {
  constructor(fromAddress, toAddress, amount) {
    this.fromAddress = fromAddress;
    this.toAddress = toAddress;
    this.amount = amount;
    this.timestamp = Date.now();
  }


  calculateHash() {
    return crypto.createHash('sha256').update(this.fromAddress + this.toAddress + this.amount
+ this.timestamp).digest('hex');
  }

  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 = '') {
{
  "$schema": "./node_modules/@angular/cli/lib/config/schema.json",
  "version": 1,
  "newProjectRoot": "projects",
```

```
"projects": {
  "savjeecoin-frontend": {
    "root": "",
    "sourceRoot": "src",
    "projectType": "application",
    "prefix": "app",
    "schematics": {
      "@schematics/angular:component": {
        "styleext": "scss"
      }
    },
    "architect": {
      "build": {
        "builder": "@angular-devkit/build-angular:browser",
        "options": {
          "outputPath": "dist/savjeecoin-frontend",
          "index": "src/index.html",
          "main": "src/main.ts",
          "polyfills": "src/polyfills.ts",
          "tsConfig": "src/tsconfig.app.json",
          "assets": [
            "src/favicon.ico",
            "src/assets"
          ],
          "styles": [
            "src/styles.scss"
          ],
          "scripts": []
        },
        "configurations": {
          "production": {
          "fileReplacements": [
              {
                "replace": "src/environments/environment.ts",
                "with": "src/environments/environment.prod.ts"
              }
            ],
            "optimization": true,
            "outputHashing": "all",
            "sourceMap": false,
            "extractCss": true,
            "namedChunks": false,
            "aot": true,
            "extractLicenses": true,
            "vendorChunk": false,
            "buildOptimizer": true,
            "budgets": [
              {
                "type": "initial",
                "maximumWarning": "2mb",
```

```
          "maximumError": "5mb"
        }
       ]
     }
    }
  },
  "serve": {
   "builder": "@angular-devkit/build-angular:dev-server",
   "options": {
     "browserTarget": "savjeecoin-frontend:build"
   },
   "configurations": {
     "production": {
       "browserTarget": "savjeecoin-frontend:build:production"
     }
   }
  },
  "extract-i18n": {
   "builder": "@angular-devkit/build-angular:extract-i18n",
   "options": {
     "browserTarget": "savjeecoin-frontend:build"
   }
  },
  "test": {
   "builder": "@angular-devkit/build-angular:karma",
   "options": {
     "main": "src/test.ts",
     "polyfills": "src/polyfills.ts",
     "tsConfig":   "src/tsconfig.spec.json",
     "karmaConfig":   "src/karma.conf.js",
     "styles": [
       "src/styles.scss"
     ],
     "scripts": [],
     "assets": [
     "src/favicon.ico",
     "src/assets"
     ]
   }
  },
  "lint": {
   "builder": "@angular-devkit/build-angular:tslint",
   "options": {
     "tsConfig": [
       "src/tsconfig.app.json",
       "src/tsconfig.spec.json"
     ],
     "exclude": [
       "**/node_modules/**"
     ]
```

```
        }
       }
      }
    },
    "savjeecoin-frontend-e2e": {
      "root":  "e2e/",
      "projectType": "application",
      "prefix": "",
      "architect": {
        "e2e": {
          "builder": "@angular-devkit/build-angular:protractor",
          "options": {
            "protractorConfig": "e2e/protractor.conf.js",
            "devServerTarget": "savjeecoin-frontend:serve"
          },
          "configurations": {
            "production": {
              "devServerTarget": "savjeecoin-frontend:serve:production"
            }
          }
        },
        "lint": {
          "builder": "@angular-devkit/build-angular:tslint",
          "options": {
            "tsConfig": "e2e/tsconfig.e2e.json",
            "exclude": [
              "**/node_modules/**"
            ]
          }
        }
      }
    }
  },
  "defaultProject": "savjeecoin-frontend"
}
```

PRACTICAL -3

Aim: 1. Introduction to Geth.

- Geth is an implementation of an Ethereum node in the Go programming language

- In simpler terms, Geth is a program which serves as a node for the Ethereum blockchain, and via which a user can mine Ether and create software which runs on the EVM – the Ethereum Virtual Machine.

- This software can be things like crypto tokens, decentralized apps, and more.

- After startup, Geth will connect to the existing live blockchain or create its own, depending on provided settings. Left at the default values, Geth will connect to the live Ethereum blockchain (the Mainnet) which we use daily.

- Simply by running Geth, you're already participating in making the Ethereum network better and stronger. Geth also serves as a console for inputting certain commands and executing specific functions.