Experiment No.4
To implement the concept of block and blockchain using javascript
Date of Performance:24/8/23
Date of Submission:24/8/23

AIM: To implement the concept of block and blockchain using javascript

Objective: To develop a program, demonstrating the concept of block and blockchain

Theory:

Blocks are data structures within the blockchain database, where transaction data in a cryptocurrency blockchain are permanently recorded. A block records some or all of the most recent transactions not yet validated by the network. Once the data are validated, the block is closed. Then, a new block is created for new transactions to be entered into and validated.

A block is thus a permanent store of records that, once written, cannot be altered or removed.

A block stores information. There are many pieces of information included within a block, but it doesn't occupy a large amount of storage space. Blocks generally include these elements, but it might vary between different types:

- Magic number: A number containing specific values that identify that block as part of a particular cryptocurrency's network.
- Blocksize: Sets the size limit on the block so that only a specific amount of information can be written in it.
- Block header: Contains information about the block.
- Transaction counter: A number that represents how many transactions are stored in the block.
- Transactions: A list of all of the transactions within a block.

The transaction element is the largest because it contains the most information. It is followed in storage size by the block header, which includes these sub-elements:

- Version: The cryptocurrency version being used.
- Previous block hash: Contains a hash (encrypted number) of the previous block's header.
- Hash Merkle root: Hash of transactions in the Merkle tree of the current block.
- Time: A timestamp to place the block in the blockchain.
- Bits: The difficulty rating of the target hash, signifying the difficulty in solving the nonce.
- Nonce: The encrypted number that a miner must solve to verify the block and close it

Genesis Block

The genesis block is the first block of the blockchain. The genesis block is generally hardcoded in the applications that utilize its blockchain. The Genesis Block is also known as Block Zero or Block 0. It is an ancestor that every Blockchain network's block that can be traced to its origin back.

Blockchain

A blockchain in simple word is a database that stores and encrypts information in a linked fashion, so that previous information cannot be altered, and a group verifies any entries before they are finalized through a consensus—an agreement that the data is correct.

Blockchains are used in cryptocurrency, decentralized finance applications, non-fungible tokens, with more uses constantly under development.

Process:

Step 1. Open the NetBeans IDE

Step 2. Create new project of categories HTML/javascript and select Node.js application in the projects tab and click next

Step 3. Give a suitable project name in the name and location tab and click next

Step 4. Tick the Create Package.json in the Tools tab and click Finish

Step 5. In the project directory under the source directory of the project create the required .js file

[block.js, blockchain.js, crypto-hash.js, genesis.js, server.js]

Step 6. Run the server.js file, if no error then the resulting blockchain is created

Code:

block

```
// block.js
const { GENESIS_DATA } = require('./genesis.js');
const cryptoHash = require('./crypto-hash');
class Block {
  constructor({timestamp, lastHash, hash, data}) {
    this.timestamp = timestamp;
    this.lastHash = lastHash;
    this.hash = hash;
    this.data = data;
  }
  static genesis() {
    return new this(GENESIS_DATA);
  }
  static mineBlock({lastBlock, data}) {
    const timestamp = Date.now();

    const lastHash = lastBlock.hash;
    return new this({
      timestamp,
lastHash,
      data,

      hash: cryptoHash(timestamp, lastHash, data)
```

```
        });
    }
}
module.exports = Block;
```

block chain

```
// blockchain.js
const Block = require('./block');
class Blockchain {
    constructor() {
        this.chain = [Block.genesis()];
    }
    addBlock({ data }) {
        const newBlock = Block.mineBlock({
            lastBlock: this.chain[this.chain.length-1],
            data
        });
        this.chain.push(newBlock);
    }
}

module.exports = Blockchain;/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/ClientSide/javascript.js to edit this template
 */
```

crypto hash

```
// crypto-hash.js

const crypto = require('crypto');

const cryptoHash =(...inputs) => {

    const hash = crypto.createHash('sha256');

    hash.update(inputs.sort().join(' '));

    return hash.digest('hex');

}
module.exports = cryptoHash;
```

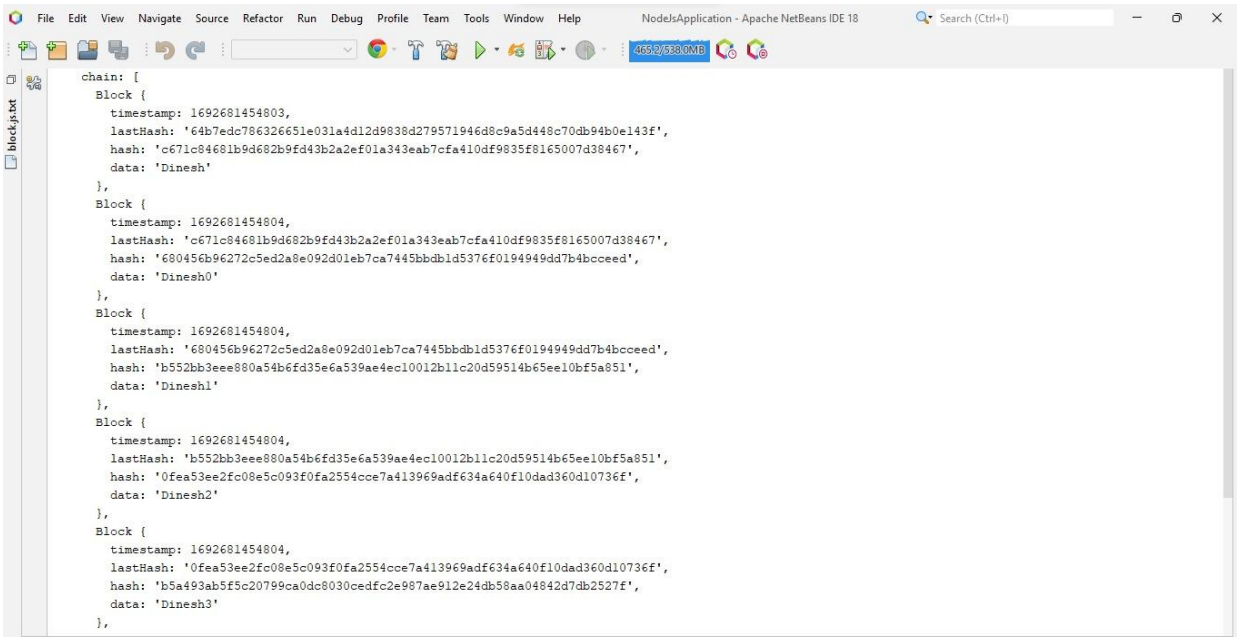
genesis

```
// genesis.js

const GENESIS_DATA = {
    timestamp: Date.now(),
    lastHash: '64b7edc786326651e031a4d12d9838d279571946d8c9a5d448c70db94b0e143f',
    hash: 'c671c84681b9d682b9fd43b2a2ef01a343eab7cfa410df9835f8165007d38467',
    data: 'Dinesh'
};

module.exports = { GENESIS_DATA };
```

server

```
// server.js
const Blockchain = require('./blockchain');
const Block = require('./block');
const blockchain = new Blockchain();
for(let i=0; i<5; i++) {
    const newData = 'Dinesh'+i;
    blockchain.addBlock({data: newData});
}
console.log(blockchain);
```

Output:



Conclusion: A programming language like JavaScript for creating blocks with their fields and forming a blockchain is justified by its versatility, widespread adoption, and developer-friendly features. JavaScript's ability to seamlessly integrate with web applications, its robust ecosystem, and accessibility make it a practical choice for efficient and scalable blockchain development across various applications and industries.