Centurion University
Shaping Lives... Empowering Communities...

School: ..... Campus: .....

Academic Year: ..... Subject Name: ..... Subject Code: .....

Semester: ..... Program: ..... Branch: ..... Specialization: .....

Date: .....

Applied and Action Learning

(Learning by Doing and Discovery)

Name of the Experiement :

* Coding Phase: Pseudo Code / Flow Chart / Algorithm

Algorithm:

1. Create Block and Blockchain classes in JavaScript
2. Use WebSocket server for P2P communication between nodes
3. Each node maintains its own copy of the blockchain
4. On receiving a new block, validate and add to local chain
5. Broadcast updated chain to all peers
6. Achieve consensus using longest chain rule

* Softwares used

1.Language: JavaScript (Node.js)
2.Libraries: ws (WebSocket)
3.Editor: VS Code

*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

* Testing Phase: Compilation of Code (error detection)

1.Created a new project folder and opened it in VS Code.
2.Initialized Node project using npm init -y.
3.Installed required packages using npm install ws crypto-js.
4.Created node.js file and pasted full blockchain + P2P code.
5.Opened first terminal and ran first node:
  -node node.js ws://localhost:6002 ws://localhost:6003
6.Opened second terminal and ran second node:
  -node node.js ws://localhost:6001 ws://localhost:6003
7.Opened third terminal and ran third node:
  -node node.js ws://localhost:6001 ws://localhost:6002
8.Typed data in any terminal → block was created and auto-synced across all nodes.

```
// Run: npm install ws crypto-js
const WebSocket = require("ws");
const SHA256 = require("crypto-js/sha256");

// ---------------- BLOCK & BLOCKCHAIN ----------------
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)
        ).toString();
    }
}

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

    createGenesisBlock() {
        return new Block(0, "01/01/2025", "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);
    }
```

```
    isValidChain(chain) {
        for (let i = 1; i < chain.length; i++) {
            const current = chain[i];
            const previous = chain[i - 1];

            if (current.previousHash !== previous.hash) return false;
            if (current.hash !== current.calculateHash()) return false;
        }
        return true;
    }
}
```


*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

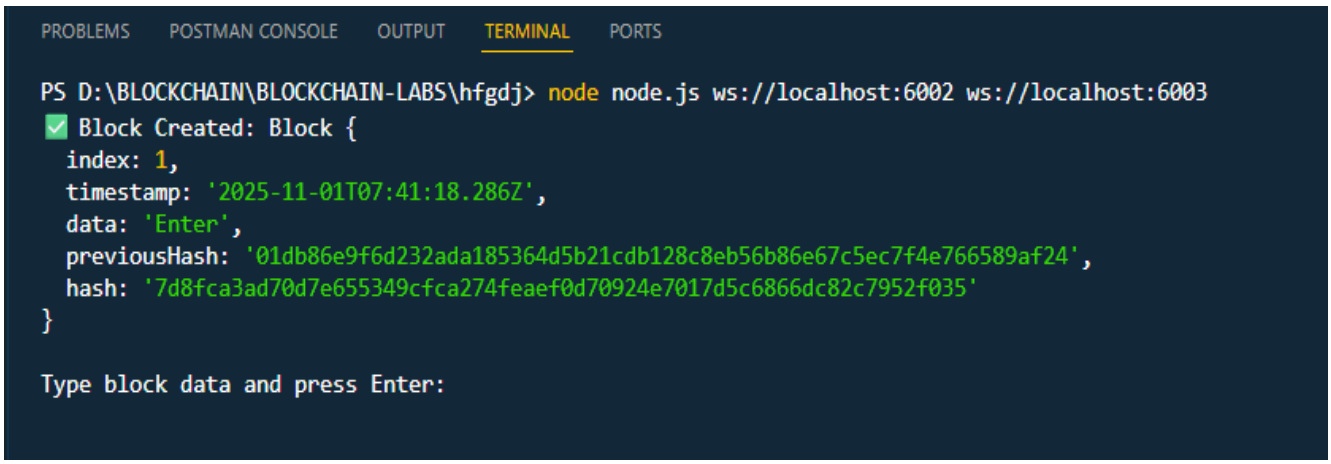
* Implementation Phase: Final Output (no error)

```
const peers = process.argv.slice(2); // command line peers
const blockchain = new Blockchain();
const sockets = [];

// WebSocket server
const PORT = 6001 + Math.floor(Math.random() * 1000);
const server = new WebSocket.Server({ port: PORT });

server.on("connection", (ws) => initConnection(ws));
console.log(`✅ Blockchain node started on port ${PORT}`);

// Connect to peers passed in arguments
function connectToPeers() {
    peers.forEach((peer) => {
        const ws = new WebSocket(peer);
        ws.on("open", () => initConnection(ws));
        ws.on("error", () => console.log("❌ Connection failed to", peer));
    });
}

function initConnection(ws) {
    sockets.push(ws);
    console.log("🔗 New peer connected");
    ws.on("message", (message) => handleMessage(JSON.parse(message)));
    send(ws, { type: "CHAIN", data: blockchain.chain });
}

function send(ws, message) {
    ws.send(JSON.stringify(message));
}

function broadcast(message) {
    sockets.forEach((socket) => send(socket, message));
}

// ---------------- MESSAGE HANDLER ----------------
function handleMessage(message) {
    switch (message.type) {
        case "CHAIN":
            if (message.data.length > blockchain.chain.length && blockchain.isValidChain(message.data)) {
                console.log("⬆️  Received new valid chain. Updating local chain...");
                blockchain.chain = message.data;
            }
            break;
```

```
function createBlockCLI() {
    readline.question("\nType block data and press Enter: ", (input) => {
        const newBlock = new Block(
            blockchain.chain.length,
            new Date().toISOString(),
            input,
            blockchain.getLatestBlock().hash
        );
        blockchain.addBlock(newBlock);
        console.log("✅ Block Created:", newBlock);
        broadcast({ type: "NEW_BLOCK", data: newBlock });
        createBlockCLI();
    });
}

// Start peer connections and input loop
connectToPeers();
createBlockCLI();
```


*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

* Implementation Phase: Final Output (no error)



* Observations

1.Each node runs independently and syncs with others.
2.Blockchain validation ensures integrity of data.
3.Real-time propagation makes the network decentralized.

ASSESSMENT

Rubrics	Full Mark	Marks Obtained	Remarks
Concept	10		
Planning and Execution/ Practical Simulation/ Programming	10		
Result and Interpretation	10		
Record of Applied and Action Learning	10		
Viva	10		
Total	**50**		

Signature of the Student:

Name :

Signature of the Faculty:

Regn. No. :

**As applicable according to the experiment.*
Two sheets per experiment (10-20) to be used.