

# **VIDUSH SOMANY INSTITUTE OF TECHNOLOGY AND RESEARCH, KADI**



**KADI SARVA VISHWAVIDYALAYA,  
GANDHINAGAR**

## **CT703D-N BLOCKCHAIN TECHNOLOGY**

**LAB MANUAL  
SEMESTER – 7**

<b>ENROLLMENT NO</b>	
<b>NAME</b>	
<b>BRANCH</b>	

## CERTIFICATE



*This is to certify that Mr. / Ms. \_\_\_\_\_ Of class*

*\_\_\_\_\_ Enrollment No.: \_\_\_\_\_ has*

*Satisfactorily completed the course in \_\_\_\_\_ at*

*Vidush Somany Institute Of Technology & Research, Kadi (Kadi Sarva*

*Vishwavidyalaya) for \_\_\_\_\_ Year (B.E.) semester \_\_\_\_\_ of Computer*

*Science & Engineering / Computer Engineering / Information Technology in*

*the Academic year \_\_\_\_\_.*

*Date of Submission: \_\_\_\_/\_\_\_\_/\_\_\_\_*

**Faculty Name with Signature**

**Head of Department**

# INDEX

SR NO	TITLE	SIGNATURE
01	1.1 Understanding Block using <a href="https://tools.superdatascience.com/Blockchain/block">https://tools.superdatascience.com/Blockchain/block</a> 1.2 Understanding Blockchain using <a href="https://tools.superdatascience.com/Blockchain/Blockchain">https://tools.superdatascience.com/Blockchain/Blockchain</a> 1.3 Understanding Distributed Blockchain using <a href="https://tools.superdatascience.com/Blockchain/distributed">https://tools.superdatascience.com/Blockchain/distributed</a> 1.4 Understanding Tokens using <a href="https://tools.superdatascience.com/Blockchain/tokens">https://tools.superdatascience.com/Blockchain/tokens</a> 1.5 Understanding coin based transaction using <a href="https://tools.superdatascience.com/Blockchain/tokens">https://tools.superdatascience.com/Blockchain/tokens</a>	
02	Using JavaScript Perform following (Source: YouTube Channel: Simply Explained – Savjee) 2.1 Creating a Blockchain 2.2 Implementing Proof-of-Work 2.3 Miner rewards & transactions 2.4 Signing transactions 2.5 Angular frontend	
03	Introduction to Geth: 3.1 Introduction to geth 3.2 Creation of private Blockchain 3.2 Creation of Account 3.4 Mining using geth	
04	Introduction to Remix Ethereum: 4.1 Introduction to Metamask 4.2 Creation of account using Metamask 4.3 Introduction to Remix Ethereum 4.4 Introduction to solidity program structure, Compilation and deployment environment. 4.5 Write a smart contract in solidity to store and get Hello World 4.6 Write a smart contract in solidity to create a function setter and getter to set and get a value. 4.7 Write a smart contract in solidity to print the array of integers and its length. 4.8 Write a solidity code to print array elements and its position.	
05	Introduction to Ethereum-Ganache: 5.1 Creation of account using Ganache. 5.2 Introduction to solidity smart contract compilation and deployment environment. 5.3 Write a smart contract in solidity to store and get Hello World	

## **Practical 1**

### **1.1 Understanding Block using**

<https://tools.superdatascience.com/Blockchain/block>

### **1.2 Understanding Blockchain using**

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

### **1.3 Understanding Distributed Blockchain using**

<https://tools.superdatascience.com/Blockchain/distributed>

### **1.4 Understanding Tokens using**

<https://tools.superdatascience.com/Blockchain/tokens>

### **1.5 Understanding coin based transaction using**

<https://tools.superdatascience.com/Blockchain/tokens>

### **1.1 Understanding Block**

#### **Objective:**

To understand the structure of a block and how mining ensures security using hashing and nonce.

#### **Procedure:**

1. Open the Block Simulator: <https://tools.superdatascience.com/Blockchain/block>
2. Enter any transaction/data into the block.
3. Observe how the block's hash changes with the data.
4. Click **Mine** to calculate a valid hash (with leading zeros).

#### **Observation:**

- The hash depends on both the data and nonce.
- Editing the block changes the hash immediately.
- Mining adjusts the nonce until a valid hash is found.

#### **Conclusion:**

A block is secured using hashing and mining. Even a small change in data invalidates the block unless it is mined again.

### **1.2 Understanding Blockchain**

#### **Objective:**

To study how multiple blocks are linked to form a blockchain and how immutability is maintained.

#### **Procedure:**

1. Open the Blockchain Simulator:  
<https://tools.superdatascience.com/Blockchain/Blockchain>
2. Add transactions/data in one of the blocks.
3. Click **Mine** to validate it.
4. Edit an older block and observe how subsequent blocks turn invalid.
5. Re-mine blocks to restore validity.

#### **Observation:**

- Each block contains the hash of the previous block.
- Changing data in one block invalidates the entire chain.
- Re-mining is required to repair the chain.

**Conclusion:**

Blockchain ensures immutability because changing one block requires re-mining all following blocks, making tampering highly impractical.

**1.3 Understanding Distributed Blockchain****Objective:**

To understand how blockchain works in a distributed network and how consensus is achieved.

**Procedure:**

1. Open the Distributed Blockchain Simulator:  
<https://tools.superdatascience.com/Blockchain/distributed>
2. Edit the data in one node's blockchain.
3. Notice that this node differs from others (conflict).
4. Click **Consensus** to synchronize all nodes.

**Observation:**

- A single altered node creates conflict in the network.
- By applying consensus, the majority chain is accepted as the valid chain.
- The tampered chain is rejected.

**Conclusion:**

Distributed blockchain ensures trust among participants by maintaining a common, agreed version of the ledger through consensus mechanisms.

**1.4 Understanding Tokens****Objective:**

To explore how tokens can be created and transferred in blockchain-based systems.

**Procedure:**

1. Open the Token Simulator: <https://tools.superdatascience.com/Blockchain/tokens>
2. Create a token (e.g., "SanjayCoin").
3. Assign some tokens to a user account.
4. Transfer tokens from one account to another.

**Observation:**

- Tokens are recorded as transactions in blocks.
- Transfers are securely validated and stored on the blockchain.
- Tokens can represent currency, points, or assets.

**Conclusion:**

Tokens demonstrate blockchain's versatility, extending beyond cryptocurrency to represent any digital or physical asset.

**1.5 Understanding Coin-based Transaction****Objective:**

To study how coin-based transactions are performed and verified on blockchain.

**Procedure:**

1. Open the Coin Transaction Simulator:  
<https://tools.superdatascience.com/Blockchain/tokens>
2. Create multiple users/wallets.
3. Perform coin transfers between users.
4. Observe how transactions are recorded into a block.
5. Mine the block to validate transactions.

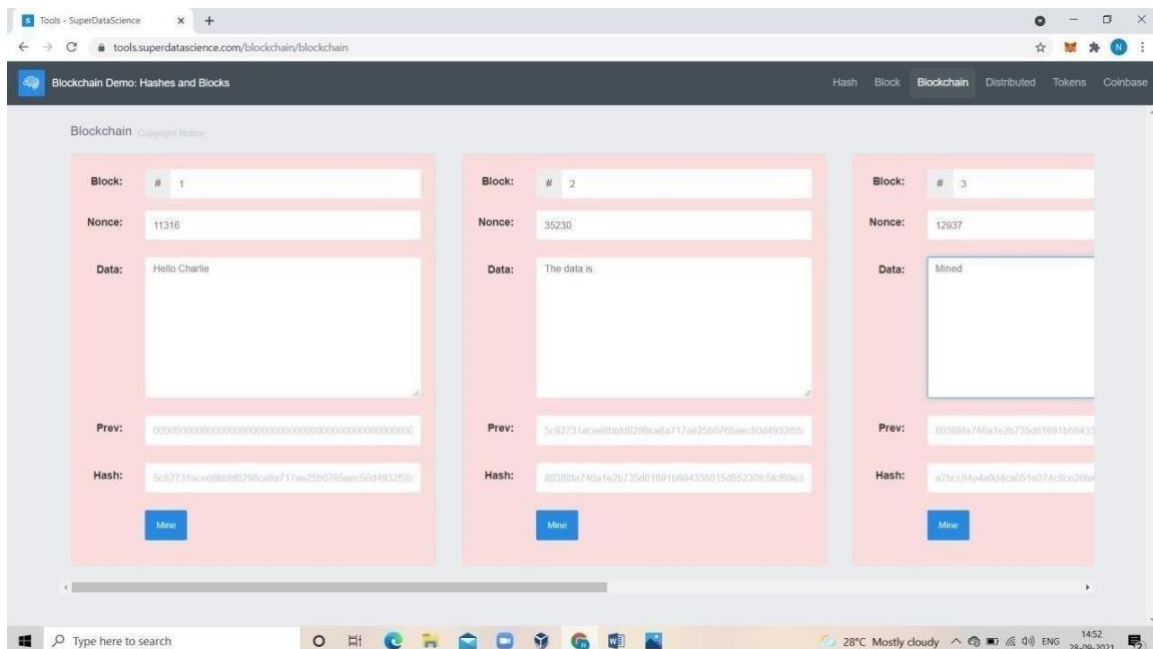
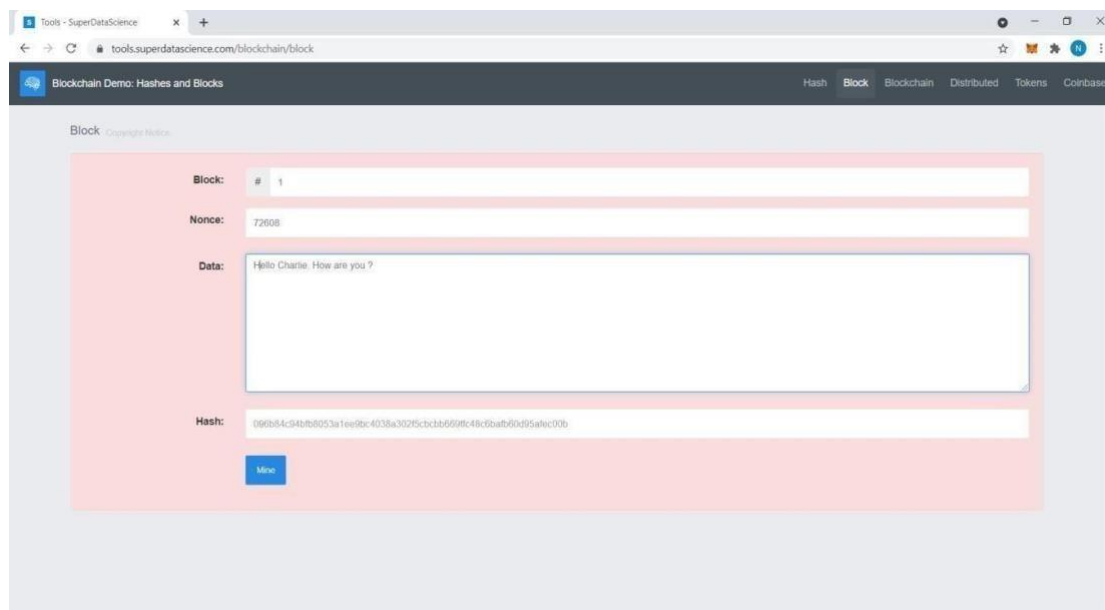
**Observation:**

- Transactions are added into the block before confirmation.

- Mining validates the transaction and secures it permanently.
- Each transfer is transparent and irreversible.

**Conclusion:**

Coin-based transactions illustrate how cryptocurrencies (like Bitcoin) enable peer-to-peer money transfer without intermediaries, secured by mining and blockchain technology.

**Screenshots:**

Tools - SuperDataScience

tools.superdatascience.com/blockchain/distributed

Blockchain Demo: Hashes and Blocks

Hash Block Blockchain Distributed Tokens Coinbase

Distributed Blockchain Copyright Notice

Peer A

Block: # 1

Nonce: 11316

Data: How are you ?

Prev: 00

Hash: a04b00b0e5c3edac8f67bd7f6c108b64f02c080817c0ca10f

Mine

Block: # 2

Nonce: 35230

Data: My name is Charlie

Prev: a04b00b0e5c3edac8f67bd7f6c108b64f02c080817c0ca10f

Hash: 10d8b94343509236816d83793be052095762d05d57e02627

Mine

Block: # 3

Nonce: 12837

Data: The hacker

Prev: 10d8b94343509236816d83793be052095762d05d57e02627

Hash: a04b00b0e5c3edac8f67bd7f6c108b64f02c080817c0ca10f

Mine

Tools - SuperDataScience

tools.superdatascience.com/blockchain/tokens

Blockchain Demo: Hashes and Blocks

Hash Block Blockchain Distributed Tokens Coinbase

Peer A

Block: # 1

Nonce: 5197

Tx: \$ 25.00 From: Darcy -> Bingley  
\$ 4.27 From: Elizabeth -> Jane  
\$ 30.22 From: Wickham -> Lydia  
\$ 106.44 From: Lady Cat -> Collins  
\$ 6.42 From: Charlotte -> Elizabeth

Prev: 00

Hash: 000048a5c44bc165c008f7db3b8647caw1250d2147a08b414

Mine

Block: # 2

Nonce: 61571

Tx: \$ 97.67 From: Rpley -> Lambert  
\$ 48.61 From: Kane -> Ash  
\$ 6.15 From: Parker -> Dallas  
\$ 10.44 From: Hicks -> Newt  
\$ 80.32 From: Bishop -> Burke  
\$ 45.00 From: Hudson -> Gorman  
\$ 92.00 From: Vasquez -> Apone

Prev: 000048a5c44bc165c008f7db3b8647caw1250d2147a08b414

Hash: 000047b02032a050a8730603f09b0cf0c0782a18d709111

Mine

Block: # 3

Nonce: 13804

Tx: \$ 10.00 From: Emily  
\$ 5.00 From: Madse  
\$ 20.00 From: Lucas

Prev: 000047b02032a050a8730603f09b0cf0c0782a18d709111

Hash: a04b00b0e5c3edac8f67bd7f6c108b64f02c080817c0ca10f

Mine

**MCQ Questions****Q1. In a blockchain block, what is the purpose of the “Nonce”?**

- a) It stores the previous block's hash
- b) It is a random number adjusted to find a valid hash
- c) It represents the number of tokens in the block
- d) It verifies the identity of the user

Answer: \_\_\_\_\_

**Q2. What happens if data inside an old block is modified in a blockchain?**

- a) The block remains valid
- b) Only the modified block becomes invalid
- c) The modified block and all following blocks become invalid
- d) Nothing changes in the Blockchain

Answer: \_\_\_\_\_

**Q3. In a distributed blockchain, how is the final version of the blockchain decided among nodes?**

- a) By the node with the highest balance
- b) By using the majority consensus
- c) By the first node to respond
- d) By random selection of a node

Answer: \_\_\_\_\_

**Q4. What do tokens on a blockchain represent?**

- a) Only cryptocurrency like Bitcoin
- b) Physical or digital assets, points, or currency
- c) Internet data packets
- d) Computer memory storage units

Answer: \_\_\_\_\_

**Q5. Which of the following best explains coin-based transactions on blockchain?**

- a) Coins are transferred between banks through blockchain
- b) Coins can be transferred between users without intermediaries, secured by mining
- c) Coins are physical money stored in blocks
- d) Coins can only be used for gaming transactions

Answer: \_\_\_\_\_

<b>Faculty Signature</b>	
<b>Date and Grade</b>	



## Practical 2

Using JavaScript Perform following

(Source: YouTube Channel: Simply Explained – Savjee)

**2.1 Creating a Blockchain**

**2.2 Implementing Proof-of-Work**

**2.3 Miner rewards & transactions**

**2.4 Signing transactions**

**2.5 Angular frontend**

### **Prerequisites**

- **Node.js** (LTS) installed
- A new folder, then:

```
npm init -y  
npm i elliptic
```

*We'll use Node's built-in crypto for SHA-256 and elliptic for ECDSA signing.*

### **2.1 Creating a Blockchain**

#### **Objective**

Build the core classes: Block, Transaction, Blockchain.

#### **Procedure**

1. Create blockchain.js.
2. Implement hashing of block contents.
3. Link blocks using previousHash.
4. Add basic chain validation.

```
// blockchain.js  
const crypto = require('crypto');  
const EC = require('elliptic').ec;  
const ec = new EC('secp256k1');  
  
// ----- UTIL -----  
const sha256 = (data) => crypto.createHash('sha256').update(data).digest('hex');  
  
// ----- MODELS -----  
class Transaction {  
  constructor(fromAddress, toAddress, amount) {  
    this.fromAddress = fromAddress || null;  
    this.toAddress = toAddress;  
    this.amount = amount;  
    this.timestamp = Date.now();  
  }  
  
  calculateHash() {  
    return sha256(this.fromAddress + this.toAddress + this.amount + this.timestamp);  
  }  
  
  signTransaction(signingKey) {
```

```
    if (signingKey.getPublic('hex') !== this.fromAddress) {
      throw new Error('You cannot sign transactions for other wallets!');
    }
    const sig = signingKey.sign(this.calculateHash(), 'base64');
    this.signature = sig.toDER('hex');
  }

  isValid() {
    // Mining reward (coinbase) has no fromAddress and is valid
    if (this.fromAddress === null) return true;
    if (!this.signature) return false;

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

class Block {
  constructor(timestamp, transactions, previousHash = '') {
    this.timestamp = timestamp;
    this.transactions = transactions; // array of Transaction
    this.previousHash = previousHash;
    this.nonce = 0;
    this.hash = this.calculateHash();
  }

  calculateHash() {
    return sha256(
      this.previousHash +
      this.timestamp +
      JSON.stringify(this.transactions) +
      this.nonce
    );
  }

  mineBlock(difficulty) {
    const target = '0'.repeat(difficulty);
    while (!this.hash.startsWith(target)) {
      this.nonce++;
      this.hash = this.calculateHash();
    }
  }

  hasValidTransactions() {
    return this.transactions.every(tx => tx.isValid());
  }
}

class Blockchain {
  constructor() {
```

```
this.chain = [this.createGenesisBlock()];
this.difficulty = 2;
this.pendingTransactions = [];
this.miningReward = 100;
}

createGenesisBlock() {
  return new Block(Date.now(), [], '0');
}

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

minePendingTransactions(rewardAddress) {
  const block = new Block(Date.now(), this.pendingTransactions,
this.getLatestBlock().hash);
  block.mineBlock(this.difficulty);
  this.chain.push(block);

  // Reset pending and add mining reward
  this.pendingTransactions = [
    new Transaction(null, rewardAddress, this.miningReward)
  ];
}

addTransaction(tx) {
  if (!tx.fromAddress || !tx.toAddress) throw new Error('Transaction must include from and
to');
  if (!tx.isValid()) throw new Error('Cannot add invalid transaction');
  this.pendingTransactions.push(tx);
}

getBalanceOfAddress(address) {
  let balance = 0;
  for (const block of this.chain) {
    for (const tx of block.transactions) {
      if (tx.fromAddress === address) balance -= tx.amount;
      if (tx.toAddress === address) balance += tx.amount;
    }
  }
  return balance;
}

isChainValid() {
  // Genesis is assumed valid; validate links, hashes, and tx signatures
  for (let i = 1; i < this.chain.length; i++) {
    const curr = this.chain[i];
    const prev = this.chain[i - 1];
```

```

    if (!curr.hasValidTransactions()) return false;
    if (curr.hash !== curr.calculateHash()) return false;
    if (curr.previousHash !== prev.hash) return false;
  }
  return true;
}
}

// ----- DEMO (run: node blockchain.js) -----
if (require.main === module) {
  const myKey = ec.genKeyPair();
  const myWalletAddress = myKey.getPublic('hex');

  const chain = new Blockchain();

  const tx1 = new Transaction(myWalletAddress, 'addrB', 10);
  tx1.signTransaction(myKey);
  chain.addTransaction(tx1);

  console.log(' Mining block...');
  chain.minePendingTransactions(myWalletAddress);

  console.log('Balance (me):', chain.getBalanceOfAddress(myWalletAddress));
  console.log('Chain valid?', chain.isChainValid());
}

module.exports = { Blockchain, Block, Transaction, ec };

```

### Observation / Expected Output

- Mining log appears, then your wallet balance becomes  $100 - 10 = 90$  after the reward comes in the *next* block.
- Chain valid? true.

## 2.2 Implementing Proof-of-Work

### Objective

Require each block's hash to start with a certain number of leading zeros (difficulty).

### Procedure

- Already implemented in Block.mineBlock(difficulty).
- Increase difficulty and observe longer mining time.

### Quick Test

```
# In blockchain.js, set: this.difficulty = 4;
node blockchain.js
```

## Observation

- Mining takes noticeably longer.
- Hashes begin with 0000....

## 2.3 Miner Rewards & Transactions

### Objective

Reward miners and process user transactions via a mempool (pendingTransactions).

### Procedure

1. Create and sign user transactions.
2. Call minePendingTransactions(rewardAddress).
3. Observe reward credited in the *next* block.

### Quick Test Snippet

```
// After initial mining in demo:
const { Blockchain, Transaction, ec } = require('./blockchain');
const chain = new Blockchain();

const minerKey = ec.genKeyPair();
const minerAddr = minerKey.getPublic('hex');

const aKey = ec.genKeyPair();
const aAddr = aKey.getPublic('hex');

const bAddr = ec.genKeyPair().getPublic('hex');

const tx = new Transaction(aAddr, bAddr, 25);
tx.signTransaction(aKey);
chain.addTransaction(tx);

console.log('⚡ Mining 1st block...');
chain.minePendingTransactions(minerAddr);

console.log('Miner balance after 1st mining (should be 0, reward queued):',
chain.getBalanceOfAddress(minerAddr));
console.log('⚡ Mining 2nd block...');
chain.minePendingTransactions(minerAddr);
console.log('Miner balance (should be 100):', chain.getBalanceOfAddress(minerAddr));
```

### Observation

- Reward is realized after the *next* mining round (classic coinbase maturity behavior in this simple model).

## 2.4 Signing Transactions

### Objective

Use ECDSA (secp256k1) to sign transactions and verify authenticity.

### Procedure

- Already implemented in `Transaction.signTransaction()` and `Transaction.isValid()`.
- Only the owner of `fromAddress` can sign.

Negative Test (Tampering)

```
const { Blockchain, Transaction, ec } = require('./blockchain');
const chain = new Blockchain();
```

```
const keyA = ec.genKeyPair();
const addrA = keyA.getPublic('hex');
const addrB = ec.genKeyPair().getPublic('hex');
```

```
const tx = new Transaction(addrA, addrB, 50);
// Forget to sign: tx.signTransaction(keyA);
try {
  chain.addTransaction(tx);
} catch (e) {
  console.log('Expected error:', e.message);
}
```

### Observation

- You should see: **“Cannot add invalid transaction”** (or signing error).
- If you sign with the wrong key, you’ll get: **“You cannot sign transactions for other wallets!”**

## 2.5 Angular Frontend (Minimal Sketch)

### Objective

Expose simple blockchain actions over HTTP and build a tiny Angular UI to interact with them.

### Procedure (Backend: Express)

1. Add Express server:

```
npm i express body-parser cors
```

2. Create `server.js`:

```
// server.js
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const { Blockchain, Transaction, ec } = require('./blockchain');

const app = express();
app.use(cors());
app.use(bodyParser.json());
```

```
const chain = new Blockchain();

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

app.post('/transaction', (req, res) => {
  const { fromPrivateKeyHex, toAddress, amount } = req.body;
  try {
    if (!fromPrivateKeyHex) {
      // Allow coinbase-like faucet by passing null fromAddress (demo only!)
      const tx = new Transaction(null, toAddress, amount);
      chain.pendingTransactions.push(tx);
      return res.json({ ok: true, message: 'Faucet tx added (unsigned, coinbase-like).' });
    }
    const key = ec.keyFromPrivate(fromPrivateKeyHex);
    const fromAddress = key.getPublic('hex');
    const tx = new Transaction(fromAddress, toAddress, amount);
    tx.signTransaction(key);
    chain.addTransaction(tx);
    res.json({ ok: true, message: 'Transaction added.' });
  } catch (e) {
    res.status(400).json({ ok: false, error: e.message });
  }
});

app.post('/mine', (req, res) => {
  const { rewardAddress } = req.body;
  chain.minePendingTransactions(rewardAddress);
  res.json({ ok: true, message: 'Block mined.' });
});

app.get('/balance/:address', (req, res) => {
  res.json({ address: req.params.address, balance:
chain.getBalanceOfAddress(req.params.address) });
});

app.listen(3000, () => console.log('API running at http://localhost:3000'));
```

Run:  
node server.js

**Procedure (Angular UI)****Create Angular app:**

```
# in a separate folder
ng new mini-chain --routing=false --style=css
cd mini-chain
ng add @angular/material
```

**Create a service src/app/chain.service.ts:**

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable({ providedIn: 'root' })
export class ChainService {
  private base = 'http://localhost:3000';
  constructor(private http: HttpClient) {}

  getChain() { return this.http.get(`${this.base}/chain`); }
  getBalance(address: string) { return this.http.get(`${this.base}/balance/${address}`); }
  addTx(fromPrivateKeyHex: string|null, toAddress: string, amount: number) {
    return this.http.post(`${this.base}/transaction`, { fromPrivateKeyHex, toAddress, amount
  });
  }
  mine(rewardAddress: string) { return this.http.post(`${this.base}/mine`, { rewardAddress
  }); }
}
```

**Simple component src/app/app.component.ts:**

```
import { Component } from '@angular/core';
import { ChainService } from '../chain.service';
import { ec as EC } from 'elliptic';
const ec = new EC('secp256k1');

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  log: string[] = [];
  myKey = ec.genKeyPair();
  myPrivHex = this.myKey.getPrivate('hex');
  myAddr = this.myKey.getPublic('hex');
  toAddr = "";

  constructor(private api: ChainService) {}

  faucet() {
```



```

    this.api.addTx(null, this.myAddr, 50).subscribe(r => this.log.push(JSON.stringify(r)));
  }
  send() {
    if (!this.toAddr) return;
    this.api.addTx(this.myPrivHex, this.toAddr, 10).subscribe(r =>
this.log.push(JSON.stringify(r)));
  }
  mine() {
    this.api.mine(this.myAddr).subscribe(r => this.log.push(JSON.stringify(r)));
  }
  showBalance() {
    this.api.getBalance(this.myAddr).subscribe(r => this.log.push(JSON.stringify(r)));
  }
  showChain() {
    this.api.getChain().subscribe(r => this.log.push(JSON.stringify(r)));
  }
}

```

Minimal template src/app/app.component.html

```

<div style="max-width:800px;margin:24px auto;font-family:system-ui;">
  <h2>MiniChain (Demo)</h2>
  <p><b>Your Address:</b> {{ myAddr | slice:0:30 }}...</p>
  <button (click)="faucet()">Faucet +50</button>
  <input placeholder="To Address" [(ngModel)]="toAddr" style="width:100%;margin:8px
0;" />
  <button (click)="send()">Send 10</button>
  <button (click)="mine()">Mine</button>
  <button (click)="showBalance()">My Balance</button>
  <button (click)="showChain()">Show Chain</button>

  <h3>Log</h3>
  <pre>{{ log.join('\n') }}</pre>
</div>

```

Run Angular:

```

ng serve
# open http://localhost:4200

```

### Observation

- Click **Faucet +50**, **Mine** → balance grows after the *next* mining round.
- **Send 10** to any address (paste another generated address if you create a second wallet in code).
- **Show Chain** prints blocks and transactions.

**MCQ Questions****Q1. What is the purpose of the previousHash field in a block?**

- a) To link the block with its parent block, maintaining chain integrity
- b) To calculate the nonce value during mining
- c) To store the miner's reward transaction
- d) To identify invalid transactions in a block

Answer: \_\_\_\_\_

**Q2. What happens if someone tampers with a transaction inside a block?**

- a) Only that single transaction becomes invalid
- b) The hash of the block changes, breaking the chain link
- c) The mining reward becomes zero for the miner
- d) The entire blockchain automatically corrects itself

Answer: \_\_\_\_\_

**Q3. Which data structure is used to hold unconfirmed transactions before mining?**

- a) Hash table
- b) Pending transaction pool (array/list)
- c) Merkle tree
- d) Digital signature log

Answer: \_\_\_\_\_

**Q4. In this blockchain, what does the signTransaction() function ensure?**

- a) Transactions are added to the mempool faster
- b) Only the owner of the wallet can authorize spending from it
- c) Mining becomes easier by reducing difficulty
- d) The miner always receives their reward first

Answer: \_\_\_\_\_

**Q5. Why is the Angular frontend calling the backend API endpoints (/transaction, /mine, /balance, /chain)?**

- a) To directly mine blocks inside Angular
- b) To act as a user interface for interacting with blockchain functions
- c) To validate hashes in the frontend without backend help
- d) To remove invalid transactions before mining

Answer: \_\_\_\_\_

<b>Faculty Signature</b>	
<b>Date and Grade</b>	

## **Practical 3**

### **Introduction to Geth:**

#### **3.1 Introduction to geth**

#### **3.2 Creation of private Blockchain**

#### **3.2 Creation of Account**

#### **3.4 Mining using geth**

### **3.1 Introduction to Geth**

#### **Objective:**

To understand the basics of Geth (Go-Ethereum) and how it provides the command-line interface to run Ethereum nodes.

#### **Procedure:**

1. Install **Geth** from the official website: <https://geth.ethereum.org/downloads> (Supports Windows, Linux, Mac).
2. Verify installation:  
geth version

Note the client version, OS, and Ethereum protocol version.

#### **Observation:**

- geth command shows version details and confirms the installation.
- Geth can run in different modes: **full node**, **light node**, **fast sync**.

#### **Conclusion:**

Geth is the official Go implementation of Ethereum that allows us to run Ethereum nodes, create private blockchains, and interact with smart contracts.

### **3.2 Creation of Private Blockchain**

#### **Objective:**

To set up a local Ethereum private blockchain for experimentation using Geth.

#### **Procedure:**

- 1 Create a working directory (e.g., mychain).  
mkdir mychain && cd mychain
- 2 Create a genesis.json file (defines chain parameters):

```
{
  "config": {
    "chainId": 2025,
    "homesteadBlock": 0,
    "eip150Block": 0,
    "eip155Block": 0,
    "eip158Block": 0
  },
  "difficulty": "1",
  "gasLimit": "2100000",
  "alloc": {}
}
```

3. Initialize blockchain with this genesis file:

```
geth --datadir ./data init genesis.json
```

Check that genesis block is created inside the data folder.

**Observation:**

- A new private chain is created with its own genesis block.
- Difficulty is set low for easy mining in private networks.

**Conclusion:**

We successfully set up a private Ethereum blockchain using a custom genesis file in Geth.

### 3.3 Creation of Account

**Objective:**

To create a new Ethereum account (wallet address) in the private blockchain.

**Procedure:**

1. Create an account using the following command:

```
geth --datadir ./data account new
```

2. Enter a password (this secures the account).

3. Copy the generated public address.

Example output: Address: {0x1234abcd...}

**Observation:**

- A keystore file is created under data/keystore.
- The public address is used to send/receive Ether.

**Conclusion:**

An Ethereum account is successfully created, which can hold Ether and interact with contracts in the private blockchain.

### 3.4 Mining using Geth

**Objective:**

To mine blocks in the private Ethereum network using Geth.

**Procedure:**

Start the Geth console with mining enabled:

```
geth --datadir ./data --networkid 2025 --http console
```

In the Geth JavaScript console, start mining:

```
miner.start(1) // (Here, 1 = number of threads).
```

Stop mining with:

```
miner.stop()
```

Check account balance after mining:

```
eth.getBalance(eth.accounts[0])
```

**Observation:**

- Mining starts and new blocks are generated quickly (since difficulty is low).
- Ether rewards are credited to the miner's account.

**Conclusion:**

Mining was successfully performed in the private Ethereum blockchain, demonstrating how Ether is generated as a reward for block creation.

**MCQ Questions**

**Q1. What is the role of the genesis.json file when creating a private blockchain with Geth?**

- a) It stores all user accounts and private keys
- b) It defines the initial block parameters and network configuration
- c) It controls the mining speed of Ethereum mainnet
- d) It automatically deploys smart contracts

Answer: \_\_\_\_\_

**Q2. When you create a new account in Geth, where is the private key stored?**

- a) In the system clipboard
- b) In a keystore file inside the data directory
- c) In the genesis block configuration
- d) On the Ethereum mainnet servers

Answer: \_\_\_\_\_

**Q3. Which Geth console command is used to start mining?**

- a) eth.start()
- b) miner.start()
- c) eth.mine()
- d) blockchain.mine()

Answer: \_\_\_\_\_

**Q4. In a private Ethereum blockchain, why is the difficulty often set very low (e.g., "1") in the genesis file?**

- a) To reduce Ether transaction fees
- b) To allow blocks to be mined quickly for testing purposes
- c) To prevent creation of multiple accounts
- d) To enable faster network synchronization with mainnet

Answer: \_\_\_\_\_

**Q5. What does the eth.getBalance(eth.accounts[0]) command return in the Geth console?**

- a) The number of pending transactions
- b) The total Ether mined in the entire chain
- c) The balance of the first account in Wei
- d) The total number of blocks in the chain

Answer: \_\_\_\_\_

<b>Faculty Signature</b>	
<b>Date and Grade</b>	

## **Practical 4**

### **Introduction to Remix Ethereum:**

#### **4.1 Introduction to Metamask**

#### **4.2 Creation of account using Metamask**

#### **4.3 Introduction to Remix Ethereum**

#### **4.4 Introduction to solidity program structure, Compilation and deployment environment.**

#### **4.5 Write a smart contract in solidity to store and get Hello World**

#### **4.6 Write a smart contract in solidity to create a function setter and getter to set and get a value.**

#### **4.7 Write a smart contract in solidity to print the array of integers and its length.**

#### **4.8 Write a solidity code to print array elements and its position.**

### **4.1 Introduction to MetaMask**

#### **Objective:**

To understand MetaMask, a browser-based Ethereum wallet and gateway to blockchain applications.

#### **Procedure:**

1. Install MetaMask extension (Chrome/Brave/Firefox).
2. Create or import a wallet.
3. Switch networks (Ethereum mainnet, testnets, or local private blockchain).

#### **Observation:**

- MetaMask shows an Ethereum address, balance, and network status.
- It connects browser DApps to blockchain securely.

#### **Conclusion:**

MetaMask is a convenient wallet and interface to manage accounts and interact with Ethereum-based DApps.

### **4.2 Creation of Account using MetaMask**

#### **Objective:**

To create a new Ethereum account in MetaMask.

#### **Procedure:**

1. Open MetaMask → Click “Create Account.”
2. Provide an account name.
3. Copy the newly generated address.

#### **Observation:**

- A new account (public address) is displayed.
- Private keys are securely stored within MetaMask.

#### **Conclusion:**

We successfully created a new Ethereum account in MetaMask for use with Remix and smart contracts.

### **4.3 Introduction to Remix Ethereum**

#### **Objective:**

To learn Remix IDE, a web-based environment for developing Solidity smart contracts.

#### **Procedure:**

1. Open Remix IDE.

2. Explore tabs: File Explorer, Solidity Compiler, Deploy & Run Transactions.

**Observation:**

- Remix provides code editor, compiler, and deployment tools in one place.

**Conclusion:**

Remix is a powerful IDE to write, compile, and deploy Solidity smart contracts quickly.

#### 4.4 Solidity Program Structure, Compilation, and Deployment

**Objective:**

To understand Solidity program structure and deploy contracts in Remix.

**Procedure:**

1. Structure of Solidity code:
  - `pragma solidity` → compiler version
  - `contract { }` → main program
  - Functions: setter/getter, logic
2. Write code in Remix editor.
3. Compile using “Solidity Compiler.”
4. Deploy via “Deploy & Run Transactions.”

**Observation:**

- Compiler generates ABI and bytecode.
- Deployment creates a contract instance on blockchain.

**Conclusion:**

We learned Solidity program structure and successfully deployed contracts using Remix.

#### 4.5 Smart Contract — Hello World

**Objective:**

To write a basic smart contract to store and retrieve a string.

**Code:**

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract HelloWorld {
    string public message = "Hello World";

    function getMessage() public view returns (string memory) {
        return message;
    }
}
```

**Observation:**

- After deployment, calling `getMessage()` returns “**Hello World.**”

**Conclusion:**

A simple Solidity contract was created and deployed successfully.

#### 4.6 Smart Contract — Setter and Getter

**Objective:**

To create functions for setting and getting a value.

**Code:**

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Storage {
    uint256 private value;

    function setValue(uint256 _value) public {
        value = _value;
    }

    function getValue() public view returns (uint256) {
        return value;
    }
}
```

**Observation:**

- Using setValue(25) stores 25.
- Calling getValue() retrieves 25.

**Conclusion:**

The setter and getter contract demonstrates how to modify and access stored values.

#### 4.7 Smart Contract — Array of Integers and Length

**Objective:**

To store an array of integers and print its length.

**Code:**

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract IntArray {
    uint[] public numbers;

    function addNumber(uint _num) public {
        numbers.push(_num);
    }

    function getNumbers() public view returns (uint[] memory) {
        return numbers;
    }

    function getLength() public view returns (uint) {
        return numbers.length;
    }
}
```



**Observation:**

- After adding numbers (10, 20, 30), getNumbers() returns [10,20,30].
- getLength() returns 3.

**Conclusion:**

We demonstrated array handling and dynamic length retrieval in Solidity.

**4.8 Smart Contract — Array Elements and Position****Objective:**

To display array elements along with their index positions.

**Code:**

```
// SPDX-License-Identifier: MIT
```

```
pragma solidity ^0.8.0;
```

```
contract ArrayWithIndex {
    uint[] public items;

    function addItem(uint _item) public {
        items.push(_item);
    }

    function getItem(uint index) public view returns (uint) {
        require(index < items.length, "Index out of range");
        return items[index];
    }

    function getAllItems() public view returns (uint[] memory) {
        return items;
    }
}
```

**Observation:**

- Adding [5, 15, 25], calling getItem(1) returns 15.
- getAllItems() returns [5,15,25].

**Conclusion:**

We created a Solidity program to fetch array elements by their index, demonstrating indexed storage in blockchain contracts.

**MCQ Questions****Q1. What is the primary purpose of MetaMask in Ethereum development?**

- a) To compile Solidity code
- b) To store and manage Ethereum accounts securely in the browser
- c) To create genesis blocks for private blockchains
- d) To mine Ether on the Ethereum mainnet

Answer: \_\_\_\_\_

**Q2. Which tab in Remix IDE is used to compile Solidity smart contracts?**

- a) File Explorer
- b) Solidity Compiler
- c) Deploy & Run Transactions
- d) Debugger

Answer: \_\_\_\_\_

**Q3. In Solidity, which keyword specifies the compiler version to be used?**

- a) import
- b) pragma
- c) contract
- d) version

Answer: \_\_\_\_\_

**Q4. What will be the output of the following Solidity function if numbers = [5, 10, 15]?**

```
function getLength() public view returns (uint) {  
    return numbers.length;  
}
```

- a) 0
- b) 2
- c) 3
- d) 15

Answer: \_\_\_\_\_

**Q5. Why is the require statement used in the getItem(uint index) function of an array contract?**

- a) To check if the value is greater than zero
- b) To prevent division by zero errors
- c) To ensure the given index is within array bounds
- d) To initialize the array with default values

Answer: \_\_\_\_\_

<b>Faculty Signature</b>	
<b>Date and Grade</b>	

## **Practical 5**

### **Introduction to Ethereum-Ganache:**

#### **5.1 Creation of account using Ganache.**

#### **5.2 Introduction to solidity smart contract compilation and deployment environment.**

#### **5.3 Write a smart contract in solidity to store and get Hello World**

### **5.1 Creation of Account using Ganache**

#### **Objective:**

To create Ethereum accounts in Ganache for testing smart contracts.

#### **Procedure:**

1. Download and install **Ganache** from <https://trufflesuite.com/ganache>.
2. Launch Ganache → choose **Quickstart (Ethereum)**.
3. Ganache automatically creates **10 accounts** with:
  - Public address
  - Private key
  - Initial Ether balance (default 100 ETH in test mode).

#### **Observation:**

- Each account has a unique public/private key pair.
- Balances are preloaded for testing transactions.

#### **Conclusion:**

Ganache successfully generated multiple test accounts with Ether for local blockchain development.

### **5.2 Introduction to Solidity Smart Contract Compilation and Deployment Environment**

#### **Objective:**

To understand how Ganache integrates with Remix IDE for compiling and deploying Solidity contracts.

#### **Procedure:**

1. Open Remix IDE.
2. Write a Solidity program in the editor.
3. Compile the contract using the **Solidity Compiler tab**.
4. In the **Deploy & Run Transactions tab**:
  - Under “Environment,” select **Web3 Provider**.
  - Provide the Ganache RPC server (default: <http://127.0.0.1:7545>).
  - Connect MetaMask to Ganache and select one of the generated accounts.
5. Deploy the contract → Ganache records the transaction.

#### **Observation:**

- Compilation generates ABI and bytecode.
- Deployment transaction appears in Ganache’s transaction history with block number, gas used, and contract address.

#### **Conclusion:**

We connected Remix to Ganache and successfully compiled and deployed a smart contract on the local Ethereum blockchain.

### 5.3 Write a Smart Contract in Solidity to Store and Get “Hello World”

**Objective:**

To implement and deploy a simple Solidity smart contract for storing and retrieving a message.

**Code (HelloWorld.sol):**

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract HelloWorld {
    string public message;

    constructor() {
        message = "Hello World";
    }

    function getMessage() public view returns (string memory) {
        return message;
    }

    function setMessage(string memory newMessage) public {
        message = newMessage;
    }
}
```

**Procedure:**

1. Paste the code into Remix.
2. Compile using the Solidity compiler.
3. Deploy using Web3 Provider → Ganache account.
4. Interact with the contract:
  - Call getMessage() → should return “**Hello World.**”
  - Call setMessage("Welcome") → updates stored string.
  - Call getMessage() again → should return “**Welcome.**”

**Observation:**

- Contract successfully deployed on Ganache.
- Storage variable updated and retrieved correctly.

**Conclusion:**

A “Hello World” smart contract was implemented, deployed, and tested on the Ganache private Ethereum blockchain.

**MCQ Questions****Q1. What is Ganache primarily used for in Ethereum development?**

- a) Mining Ether on the Ethereum mainnet
- b) Creating a local blockchain for testing smart contracts
- c) Generating ABI files for Solidity contracts
- d) Providing a cloud-based Ethereum service

Answer: \_\_\_\_\_

**Q2. By default, how many accounts with preloaded Ether does Ganache create in a new workspace?**

- a) 5
- b) 10
- c) 20
- d) 50

Answer: \_\_\_\_\_

**Q3. Which RPC URL is typically used to connect Remix or MetaMask to Ganache?**

- a) http://localhost:3030
- b) http://127.0.0.1:8545
- c) http://127.0.0.1:7545
- d) http://localhost:9000

Answer: \_\_\_\_\_

**Q4. When a smart contract is deployed using Remix connected to Ganache, where can the transaction details be viewed?**

- a) In the Solidity Compiler tab
- b) In the MetaMask extension only
- c) In Ganache's Transactions tab (GUI)
- d) On the Ethereum mainnet explorer

Answer: \_\_\_\_\_

**Q5. In the HelloWorld contract, what is the purpose of the constructor() function?**

- a) To print values of the array
- b) To set the initial value of the message variable
- c) To calculate the length of a string
- d) To compile the contract automatically

Answer: \_\_\_\_\_

<b>Faculty Signature</b>	
<b>Date and Grade</b>	

## NOTES

[illegible]