Thadomal Shahani Engineering College

Bandra (W.), Mumbai - 400 050.

CERTIFICATE

Certify that Mr./Miss Parth Sandeep Dabholkar of Computer Department, Semester VII with Roll No 2103032 has completed a course of the necessary experiments in the subject Blockchain under my supervision in the **Thadomal Shahani Engineering College** Laboratory in the year 2024 - 2025

10/10/24

Teacher In- Charge

Head of the Department

Date 09/10/24

Principal

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

CONTENTS

10/10/24

Group - 38
Parth Dabholkar
C31
2103032

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

EXPERIMENT NO: 1

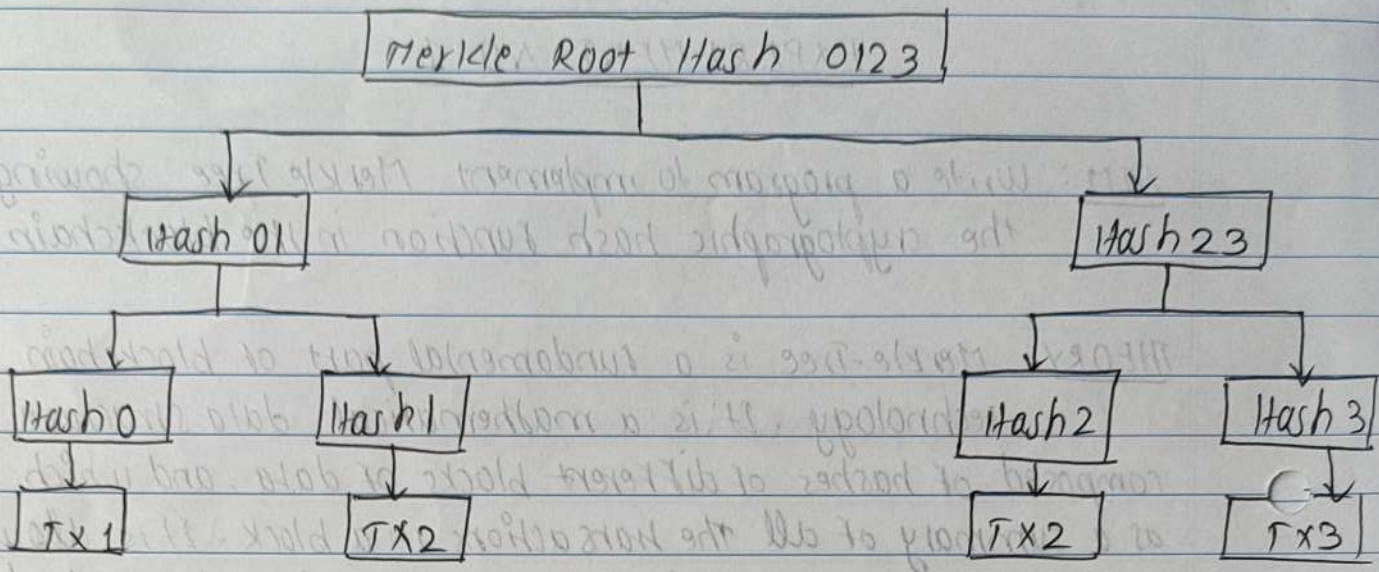
AIM: Write a program to implement Merkle Tree showing the cryptographic hash function in the blockchain.

THEORY: Merkle-Tree is a fundamental part of blockchain technology. It is a mathematical data structure composed of hashes of different blocks of data, and which serves as a summary of all the transactions in a block. It is allows for efficient and secure verification of content in a large body of data. It is also helps to verify the consistency and content of the data. Both Bitcoin and Ethereum use Merkle Tree structure. Merkle Tree is also known as Hash Tree.

Working of Merkle Tree: A Merkle Tree stores all the transactions in a block by providing a digital fingerprint of the entire set of transactions. It allows the user to verify whether a transaction can be included in a block or not.

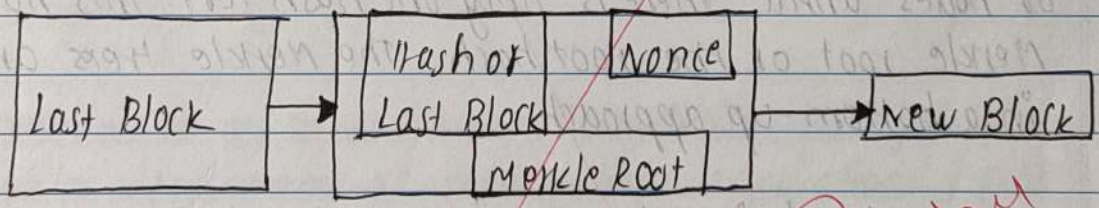
Merkle Trees are created by repeatedly calculating hashing pairs of nodes until there is only one hash left. This hash is called Merkle root or the root hash. The Merkle trees are constructed in a bottom-up approach.

Every leaf-node is a hash of transactional data and the non-leaf node is a hash of its previous hashes. Merkle Tree are in a binary tree so it requires an even number of leaf nodes. If there is an odd node number of transactions, the last hash will be duplicated once, to create an even number of leaf nodes.



Basic Structure of Merkle Tree

Merkle Root is stored in the block header. The block header is a part of Bitcoin Block which gets hash in the process of mining. It contains the hash of the last block, a Nonce and the root hash of all the transactions in the current block in a Merkle Tree. So having the Merkle root in a block header makes the transaction tamper-proof. As this root hash includes the hashes of all the transactions within the block, these transaction may result in saving the disk space.



11/10/24

Advantages of Merkle Roots:

- It provides a means to maintain the integrity and validity of data.
- It helps in saving the memory and disk space as the proofs.
- Computationally easy and fast.

```
import hashlib

class MerkleTree:
    def __init__(self, data_blocks):
        self.leaves = [(data, self.hash_data(data)) for data in
data_blocks]
        self.intermediate_hashes = []  # To store intermediate hashes
        self.root = self.build_tree([leaf[1] for leaf in self.leaves])

    def hash_data(self, data):
        # Hashes data using SHA-256 and returns the hexadecimal digest
        return hashlib.sha256(data.encode('utf-8')).hexdigest()

    def build_tree(self, leaves):
        if len(leaves) == 1:
            return leaves[0]

        # If odd number of leaves, duplicate the last leaf
        if len(leaves) % 2 != 0:
            leaves.append(leaves[-1])

        # Compute the parent level
        parent_level = []
        for i in range(0, len(leaves), 2):
            combined_hash = leaves[i] + leaves[i + 1]
            parent_hash = self.hash_data(combined_hash)
            parent_level.append(parent_hash)
            self.intermediate_hashes.append((leaves[i], leaves[i + 1],
parent_hash))  # Store intermediate hashes

        # Recursively build the tree until we get the root hash
        return self.build_tree(parent_level)

    def get_root(self):
        return self.root

    def get_leaves(self):
        return self.leaves

    def get_intermediate_hashes(self):
        return self.intermediate_hashes

if __name__ == "__main__":
    data_blocks = []

    print("Enter data blocks for the Merkle Tree (type 'done' to
finish):")
    while True:
        data = input("Enter data block: ")
        if data.lower() == 'done':
```

```
            break
        data_blocks.append(data)

    if data_blocks:
        merkle_tree = MerkleTree(data_blocks)

        print("Hashes of each block:")
        for data, hash_value in merkle_tree.get_leaves():
            print(f"Data: {data}, Hash: {hash_value}")

        print("\nIntermediate hashes:")
        for leaf1, leaf2, parent_hash in
merkle_tree.get_intermediate_hashes():
            print(f"Combined: {leaf1} + {leaf2} -> Hash:
{parent_hash}")

        print(f"\nMerkle Root: {merkle_tree.get_root()}")
    else:
        print("No data blocks entered.")
```

```
Enter data blocks for the Merkle Tree (type 'done' to finish):
Enter data block: Kanav
Enter data block: Parth
Enter data block: Vedant
Enter data block: Shirish
Enter data block: done
Hashes of each block:
Data: Kanav, Hash:
418d75f3639d15a99ecbf59516675f4959ea8e29ddd7d2c3b0543d616f7c7dfb
Data: Parth, Hash:
4ec1a76282e4f89a809b90cea83ef509a5eda8d8d128819905a7459830a2ebe5
Data: Vedant, Hash:
2f429777f4a4909d6744825d404137c5b07b2d3b7ac7e524f121318cf5664d27
Data: Shirish, Hash:
6ec5597981e8251d63f918f85d2fe5505766494beb04b24ec0e1cb0c567ca61e

Intermediate hashes:
Combined:
418d75f3639d15a99ecbf59516675f4959ea8e29ddd7d2c3b0543d616f7c7dfb +
4ec1a76282e4f89a809b90cea83ef509a5eda8d8d128819905a7459830a2ebe5 ->
Hash: 861f34c47a929057c94775ba6581e4654472ff6f4252e63ae18b50f9a8f84859
Combined:
2f429777f4a4909d6744825d404137c5b07b2d3b7ac7e524f121318cf5664d27 +
6ec5597981e8251d63f918f85d2fe5505766494beb04b24ec0e1cb0c567ca61e ->
Hash: d93f7d25cec6c1fb6cfbd8e915d1b02b99aedf6ca3a7fce5ecffcab2a7bccdc0
Combined:
861f34c47a929057c94775ba6581e4654472ff6f4252e63ae18b50f9a8f84859 +
d93f7d25cec6c1fb6cfbd8e915d1b02b99aedf6ca3a7fce5ecffcab2a7bccdc0 ->
Hash: a4967bfcd294efc4d0cbf4dc84b5eb25ed88ff11a525c5066732fcc32a0a5ea6
```

```
Merkle Root:
a4967bfcd294efc4d0cbf4dc84b5eb25ed88ff11a525c5066732fcc32a0a5ea6
```

Group - 38
Parth Dabholkar
C31
2103032

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

EXPERIMENT NO: 2

AIM: Creating Smart Contract using solidity and remix IDE.

THEORY:

- Remix IDE is a popular integrated development environment (IDE) specifically designed for Ethereum smart contract development.
- It provides a user friendly interface for creating, testing, debugging and deploying smart contract on the Ethereum blockchain.
- Solidity support is built in which is commonly used for developing Ethereum Smart Contract.
- You can write, compile and deploy smart contracts within Remix IDE. Also, you can connect to various Ethereum networks including local development networks, public testnets and the ethereum main net for contract deployment, development and testing.
- Ethereum is an open source Blockchain-base platform that enables development of decentralized applications (DAPP) and smart contracts. These applications can be used for a wide range of purposes from financial services to gaming and more.

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

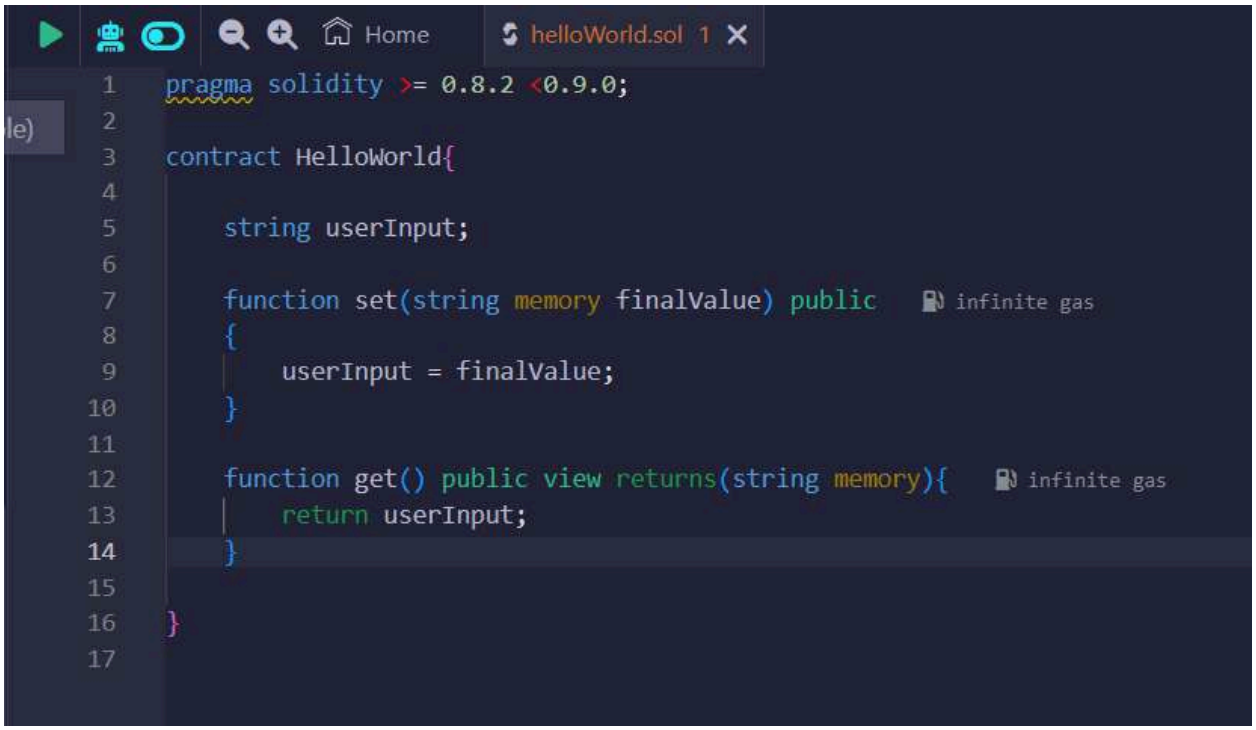
- Ethereum has its native cryptocurrency called Ether (ETH) that can be used for various purpose within the ethereum network.

conclusion:

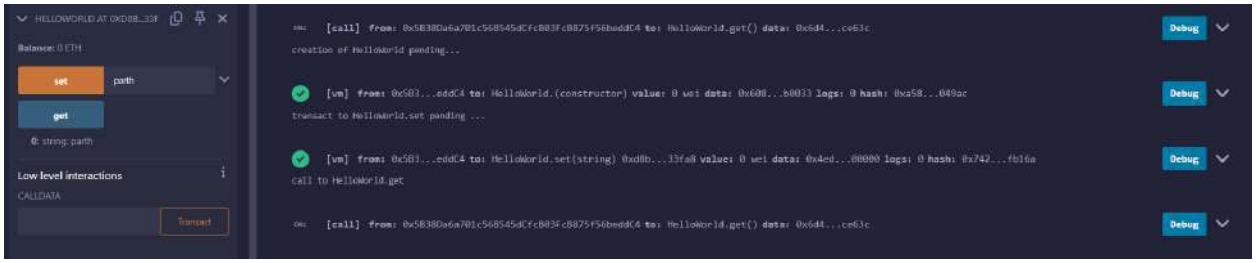
Using solidity and Remix IDE we successfully implemented various contracts and deployed them to get the desidered output.

11/10/24

Code:


Home
helloWorld.sol 1
pragma solidity >= 0.8.2 <0.9.0;
contract HelloWorld{
string userInput;
function set(string memory finalValue) public
infinite gas
{
userInput = finalValue;
}
function get() public view returns(string memory){
infinite gas
return userInput;
}
}

Output:


set
path
get
Low level interactions
CALLDATA
Transact
Debug

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

EXPERIMENT NO: 3

AIM: write a program in solidity programming to create transactions using the Remix IDE.

THEORY:

In blockchain, a transaction is a transfer of data or assets between participants. It involves recording information on the blockchain ledger ensuring a secure transparent and immutable. Once a transaction is validated by the network trough consensus mechanisms like proof of work or proof of stack.

It is grouped into a block and added into the blockchain, making it permanent part of the network. Transactions are designed to be tamper proof and transparent. Each transaction typically includes details such as the sender, recipient, amount and a timestamp.

After a transaction is initiated, it is broadcasted to the network while nodes verify its authenticity using cryptographic algorithms. Once validated, the transaction is included in a block, and block is appended to the existing chain. This processes ensures that all participants in the network have a consistent and accurate record of transactions making fraud and double speeding difficult.

Components of transaction are:

1) Sender Address: The address of the sender initiating the transaction. It is derived from the users public key.

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

2. Recipient Address: Address of the user or smart contract will reciere the funds or data.
3. Amount: The quantity of cryptography being transferred.
4. Transaction Fee: A fee paid by the sender to the validators for processing the transaction.
5. Timestamp: The exact time when the transaction was created.
6. Digital Signature: A cryptographic signature generated using the sender's private key, ensuring authenticity.
7. Transaction ID: A unique identifier for the transaction usually derived from hashing the transaction data.

11/10/24

Code:

```
// SPDX-License-Identifier: MIT
pragma solidity >= 0.8.2 <0.9.0;
contract TransactionContract {
event TransactionMade(address indexed from, address indexed to, uint amount);
struct Transaction {
address from;
address to;
uint amount;
uint timestamp;
}
Transaction[] public transactions;
mapping(address => uint) public balances;
function createTransaction(address payable _to) public payable {
require(msg.sender.balance >= msg.value, "Insufficient balance");
transactions.push(Transaction(msg.sender, _to, msg.value, block.timestamp));
balances[msg.sender] -= msg.value;
balances[_to] += msg.value;
emit TransactionMade(msg.sender, _to, msg.value);
_to.transfer(msg.value);
}
function getAllTransactions() public view returns (Transaction[] memory) {
return transactions;
}
```

```
function getBalance(address _addr) public view returns (uint) {
return balances[_addr];
}
}
```

Output:

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

EXPERIMENT NO: 4

AIM: Write a program to implement transaction by embedding of a wallet.

THEORY:

A blockchain wallet is a software application that stores a user's private and public keys and interacts with various blockchain networks to enable users to send and receive digital currency and monitor their balance. Unlike traditional wallets, the blockchain do not store the currency itself, instead they store the keys needed to access and manage the crypto-currency on the blockchain.

Types of Blockchain wallet:

1. Hot wallets: Connected to the internet, they are more convinient for daily transactions but are also more vulnerable to hacker.

2. Cold wallets: Not connected to the internet, offering higher security for storing large amounts of crypto currency

Wallet Embedding in Applications:

Embedding a wallet within an application or website integrates blockchain functionality directly into the user experience. This process involves incorporating wallets functionalities into the applications UI to allow users to perform blockchain transaction without needing to leave the platform.

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Key features of Embedded wallets:

1. Key Management
2. Transaction Initiation
3. Balance monitoring
4. Security.

Conclusion:
In this experiment, I learnt program in solidity to implement transaction by embedding of wallet.

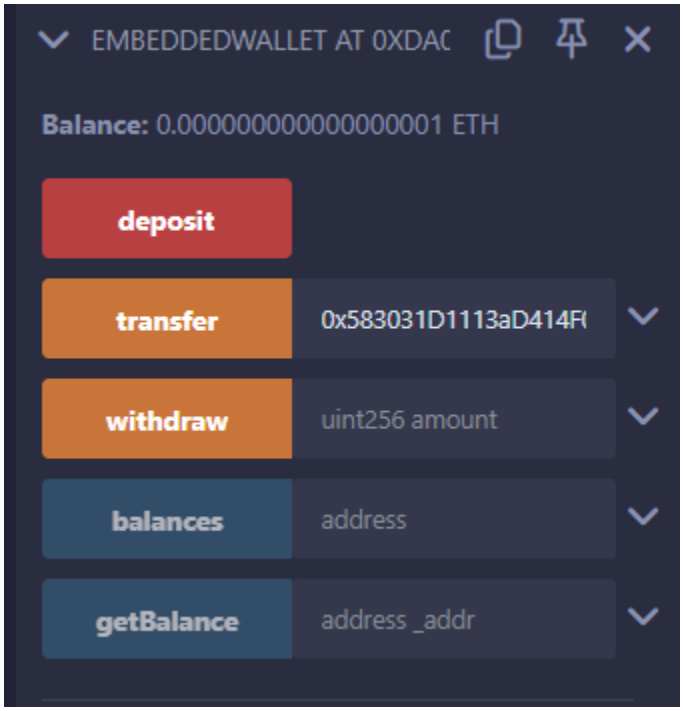
11/10/24

Code:

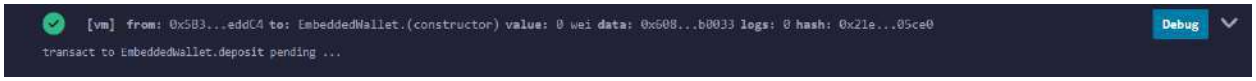
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract EmbeddedWallet {
event Deposit(address indexed user, uint amount);
event Withdrawal(address indexed user, uint amount);
event Transfer(address indexed from, address indexed to, uint amount);
mapping(address => uint) public balances;
modifier hasEnoughFunds(uint amount) {
require(balances[msg.sender] >= amount, "Insufficient funds");
_;
}
function deposit() public payable {
require(msg.value > 0, "Deposit must be greater than 0");
balances[msg.sender] += msg.value;
emit Deposit(msg.sender, msg.value);
}
function withdraw(uint amount) public hasEnoughFunds(amount) {
balances[msg.sender] -= amount;
payable(msg.sender).transfer(amount);
emit Withdrawal(msg.sender, amount);
}
function transfer(address to, uint amount) public hasEnoughFunds(amount) {
require(to != address(0), "Invalid recipient address");
```

```
require(to != msg.sender, "Cannot transfer to yourself");
balances[msg.sender] -= amount;
balances[to] += amount;
emit Transfer(msg.sender, to, amount);
}
function getBalance(address _addr) public view returns (uint) {
return balances[_addr];
}
}
```

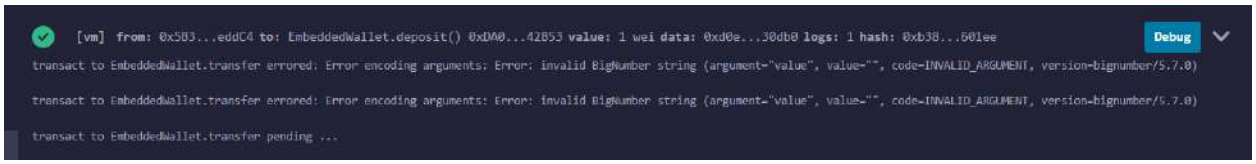
Output:



Deposit function -



Transfer function -



EXPERIMENT NO: 5

AIM: Show implementation of the blockchain platform Ethereum using Geth.

THEORY:

⇒ **ETHEREUM:**

Decentralised blockchain platform that allows developers to create and deploy smart contracts and decentralized applications (dApps). Launched in 2015, it introduced a programmable blockchain powered by its native cryptocurrency Ether (ETH), enabling automation and innovation across various industries. With its transition to Proof-of-Stake (PoS) Ethereum aims to improve scalability, security and sustainability, solidifying its position as a leading platform in blockchain space.

⇒ **GETH:**

Geth is a command line interface (CLI) tool that allows users to run an Ethereum node. By running a node, users can participated in Ethereum network by verifying its transactions, deploying smart contracts, mining ether, and managing accounts. Geth serves as the backbone for developers and blockchain enthusiasts to interact with ethereum blockchain

- **KEY FEATURES OF GETH:**

1. Node Operation
2. Account Management.

3. Transaction Handling.
4. Smart Contract Interaction
5. Mining.

⇒ GETH ROLE IN ETHEREUM DEVELOPMENT:

1. Local Development and Testing
2. Mining and Proof-of-Work.
3. DApp Interaction.

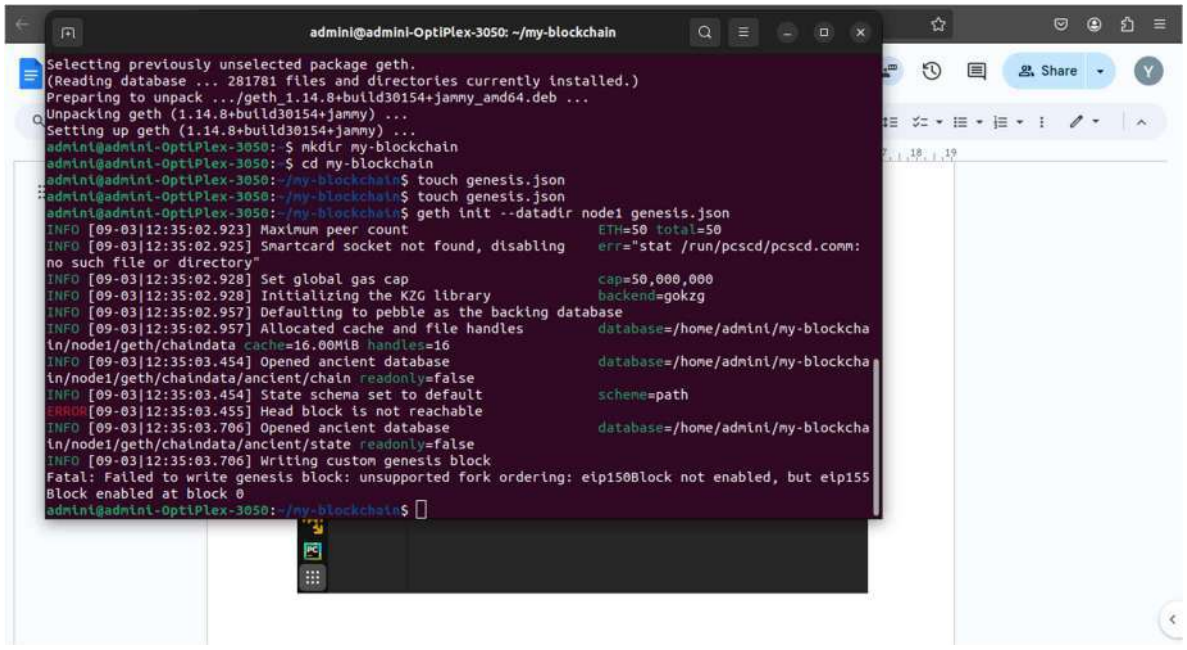
CONCLUSION:

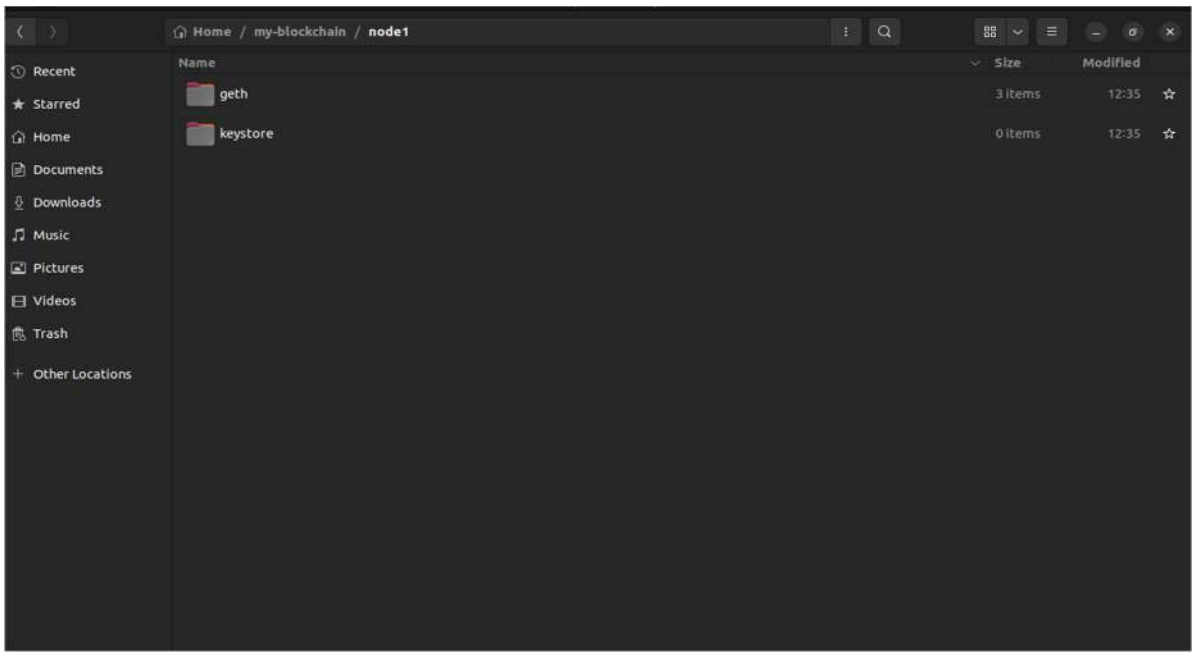
In this experiment, I have learnt the implementation of the blockchain platform Ethereum using GETH.

11/10/24

OUTPUT:


Open
genesis.json
~/my-blockchain
Save
{
"config": {
"chainId": 1234,
"homesteadBlock": 0,
"eip155Block": 0,
"eip158Block": 0
},
"difficulty": "20000",
"gasLimit": "2100000",
"alloc": {}
}
JSON
Tab Width: 8
Ln 1, Col 2
INS


admini@admini-OptiPlex-3050: ~/my-blockchain
Selecting previously unselected package geth.
(Reading database ... 281781 files and directories currently installed.)
Preparing to unpack .../geth_1.14.8+build30154+jammy_amd64.deb ...
Unpacking geth (1.14.8+build30154+jammy) ...
Setting up geth (1.14.8+build30154+jammy) ...
admini@admini-OptiPlex-3050: $ mkdir my-blockchain
admini@admini-OptiPlex-3050: $ cd my-blockchain
admini@admini-OptiPlex-3050:~/my-blockchain$ touch genesis.json
admini@admini-OptiPlex-3050:~/my-blockchain$ touch genesis.json
admini@admini-OptiPlex-3050:~/my-blockchain$ geth init --datadir node1 genesis.json
INFO [09-03|12:35:02.923] Maximum peer count ETH=50 total=50
INFO [09-03|12:35:02.925] Smartcard socket not found, disabling err="stat /run/pcscd/pcscd.comm:
no such file or directory"
INFO [09-03|12:35:02.928] Set global gas cap cap=50,000,000
INFO [09-03|12:35:02.928] Initializing the KZG library backend=gokzg
INFO [09-03|12:35:02.957] Defaulting to pebble as the backing database
INFO [09-03|12:35:02.957] Allocated cache and file handles database=/home/admini/my-blockcha
in/node1/geth/chaindata cache=16.00MiB handles=16
INFO [09-03|12:35:03.454] Opened ancient database database=/home/admini/my-blockcha
in/node1/geth/chaindata/ancient/chain readonly=false
INFO [09-03|12:35:03.454] State schema set to default scheme=path
ERROR[09-03|12:35:03.455] Head block is not reachable
INFO [09-03|12:35:03.706] Opened ancient database database=/home/admini/my-blockcha
in/node1/geth/chaindata/ancient/state readonly=false
INFO [09-03|12:35:03.706] Writing custom genesis block
Fatal: Failed to write genesis block: unsupported fork ordering: eip150Block not enabled, but eip155
Block enabled at block 0
admini@admini-OptiPlex-3050:~/my-blockchain$
Share


Home / my-blockchain / node1
Recent
Starred
Home
Documents
Downloads
Music
Pictures
Videos
Trash
Other Locations
Name
Size
Modified
geth
3 items
12:35
keystore
0 items
12:35

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

EXPERIMENT NO: 6

AIM: Show implementation of Blockchain platform Ganache.

THEORY:

Ganache is a popular blockchain development tool used primarily in Ethereum based decentralized applications (DApp). It serves as a local blockchain emulator, providing developers with a personal Ethereum blockchain to deploy and test smart contracts in a safe and controlled environment. Here are key features and functionalities of Ganache:

1. LOCAL BLOCKCHAIN ENVIRONMENT:
Ganache runs a personal ethereum blockchain locally, enabling safe and controlled smart contract testing.

2. INSTANT MINING:
Blocks are mined instantly, speeding up development by providing immediate feedback on transaction.

3. DETERMINISTIC BEHAVIOUR:
Ganache offers consistent, repeatable conditions by providing the same account and address on each restart.

4. PRE-FUNDED ACCOUNTS:
It includes several accounts with pre-loaded ether, eliminating the need to mine to external faucets for testing.

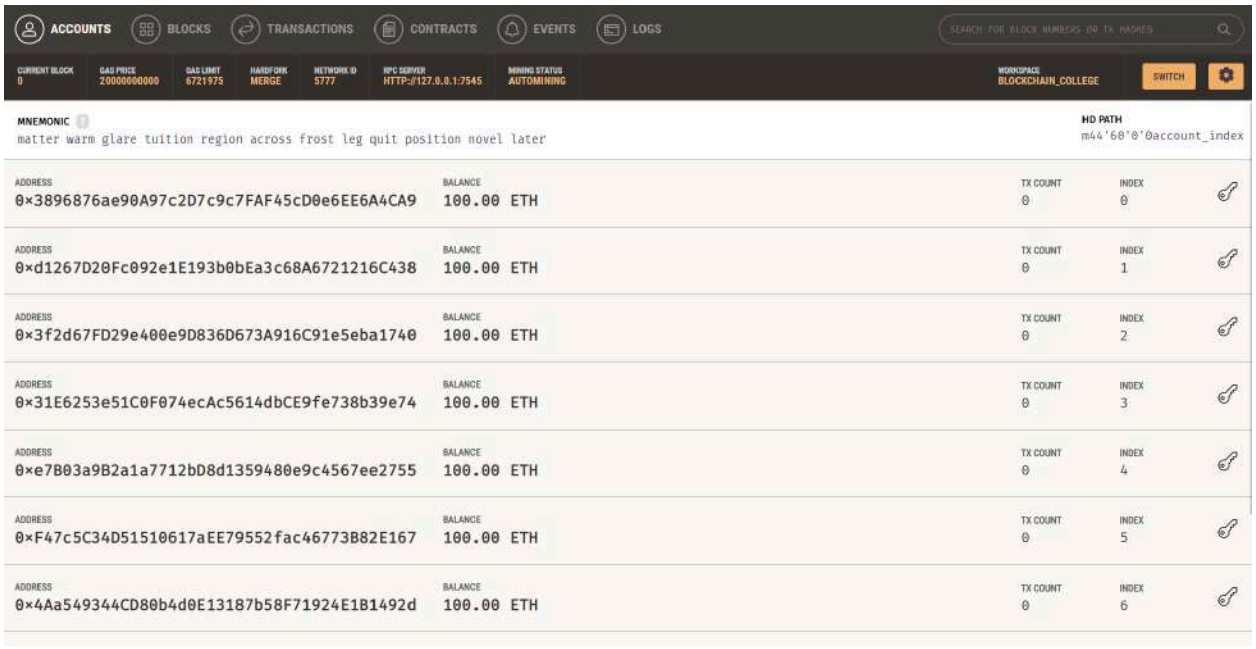
5. TRANSACTION INSIGHTS:
Detailed transaction data including gas usage and event logs, helps with debugging and optimizing smart contracts.

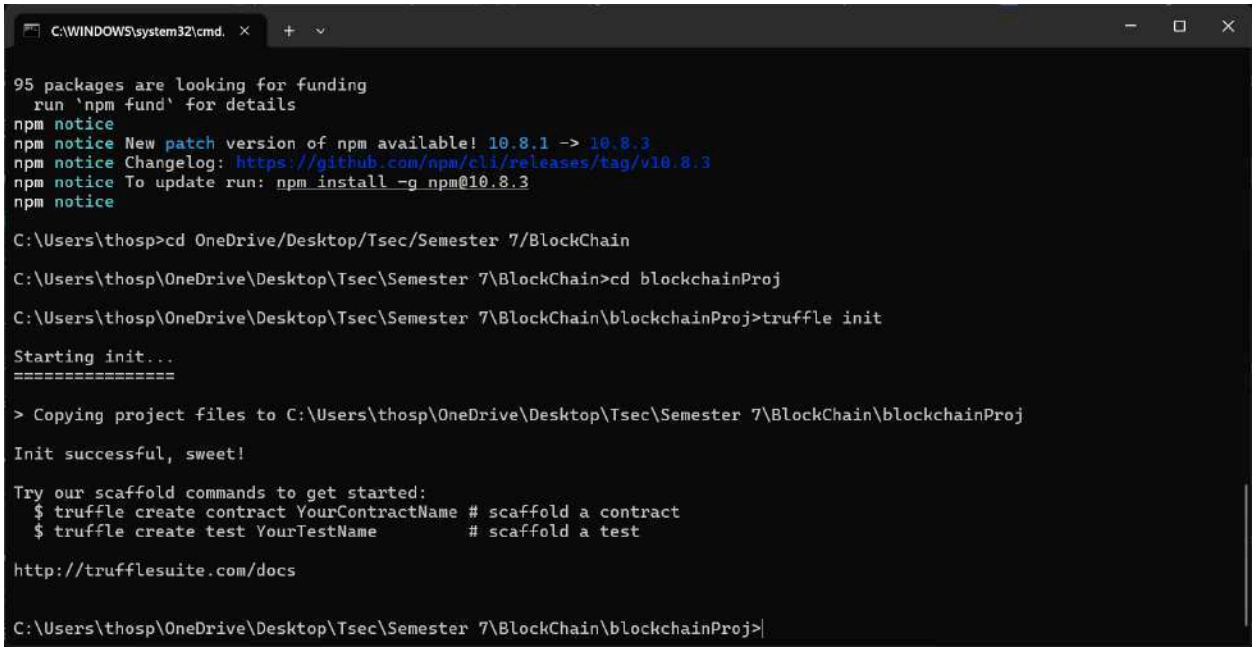
6. CLI and GUI versions:
Available in both CLI and GUI, with the GUI offering a user-friendly interface for real-time blockchain activity.

7. FRAMEWORK COMPATIBILITY:
Works seamlessly with Ethereum Development tools like ~~Hardhat~~ Truffle and Drizzle, making it integral to the development ecosystem.

11/10/24

Output:


ACCOUNTS
BLOCKS
TRANSACTIONS
CONTRACTS
EVENTS
LOGS
SEARCH FOR BLOCK NUMBERS OR TX HASHES
CURRENT BLOCK 0
GAS PRICE 20000000000
GAS LIMIT 6721975
HARDFORK MERGE
NETWORK ID 5777
RPC SERVER HTTP://127.0.0.1:7545
MINING STATUS AUTOMINING
WORKSPACE BLOCKCHAIN_COLLEGE
SWITCH
MNEMONIC
matter warm glare tuition region across frost leg quit position novel later
HD PATH
m44'60'0'0account_index
ADDRESS 0x3896876ae90A97c2D7c9c7FAF45cD0e6EE6A4CA9 BALANCE 100.00 ETH TX COUNT 0 INDEX 0
ADDRESS 0xd1267D20Fc092e1E193b0bEa3c68A6721216C438 BALANCE 100.00 ETH TX COUNT 0 INDEX 1
ADDRESS 0x3f2d67FD29e400e9D836D673A916C91e5eba1740 BALANCE 100.00 ETH TX COUNT 0 INDEX 2
ADDRESS 0x31E6253e51C0F074ecAc5614dbCE9fe738b39e74 BALANCE 100.00 ETH TX COUNT 0 INDEX 3
ADDRESS 0xe7B03a9B2a1a7712bD8d1359480e9c4567ee2755 BALANCE 100.00 ETH TX COUNT 0 INDEX 4
ADDRESS 0xF47c5C34D51510617aEE79552fac46773B82E167 BALANCE 100.00 ETH TX COUNT 0 INDEX 5
ADDRESS 0x4Aa549344CD80b4d0E13187b58F71924E1B1492d BALANCE 100.00 ETH TX COUNT 0 INDEX 6


C:\WINDOWS\system32\cmd.
95 packages are looking for funding
run `npm fund` for details
npm notice
npm notice New patch version of npm available! 10.8.1 -> 10.8.3
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.8.3
npm notice To update run: npm install -g npm@10.8.3
npm notice
C:\Users\thosp>cd OneDrive/Desktop/Tsec/Semester 7/BlockChain
C:\Users\thosp\OneDrive\Desktop\Tsec\Semester 7\BlockChain>cd blockchainProj
C:\Users\thosp\OneDrive\Desktop\Tsec\Semester 7\BlockChain\blockchainProj>truffle init
Starting init...
> Copying project files to C:\Users\thosp\OneDrive\Desktop\Tsec\Semester 7\BlockChain\blockchainProj
Init successful, sweet!
Try our scaffold commands to get started:
$ truffle create contract YourContractName # scaffold a contract
$ truffle create test YourTestName # scaffold a test
http://trufflesuite.com/docs
C:\Users\thosp\OneDrive\Desktop\Tsec\Semester 7\BlockChain\blockchainProj>

Welcome myContract.sol

contracts > myContract.sol

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.0;
3
4  contract MyContract {
5      string public greeting;
6
7      constructor(string memory _greeting) {
8          greeting = _greeting;
9      }
10
11     function setGreeting(string memory _greeting) public {
12         greeting = _greeting;
13     }
14 }
15
```

```
Compiling your contracts...
===========================
✓ Fetching solc version list from solc-bin. Attempt #1
✓ Downloading compiler. Attempt #1.
> Compiling .\contracts\myContract.sol
> Artifacts written to C:\Users\thosp\OneDrive\Desktop\Tsec\Semester 7\BlockChain\blockchainProj\build\contracts
> Compiled successfully using:
   - solc: 0.8.21+commit.d9974bed.Emscripten.clang
```

```
module.exports = {

  networks: {

    development: {
     host: "127.0.0.1",
     port: 7545,
     network_id: "*",
    },
  },

  mocha: {
  },

  compilers: {
    solc: {
      version: "0.8.21",

    }
  },
};
```

myContract.sol × | 2_deploy_contracts.js | truffle-config.js

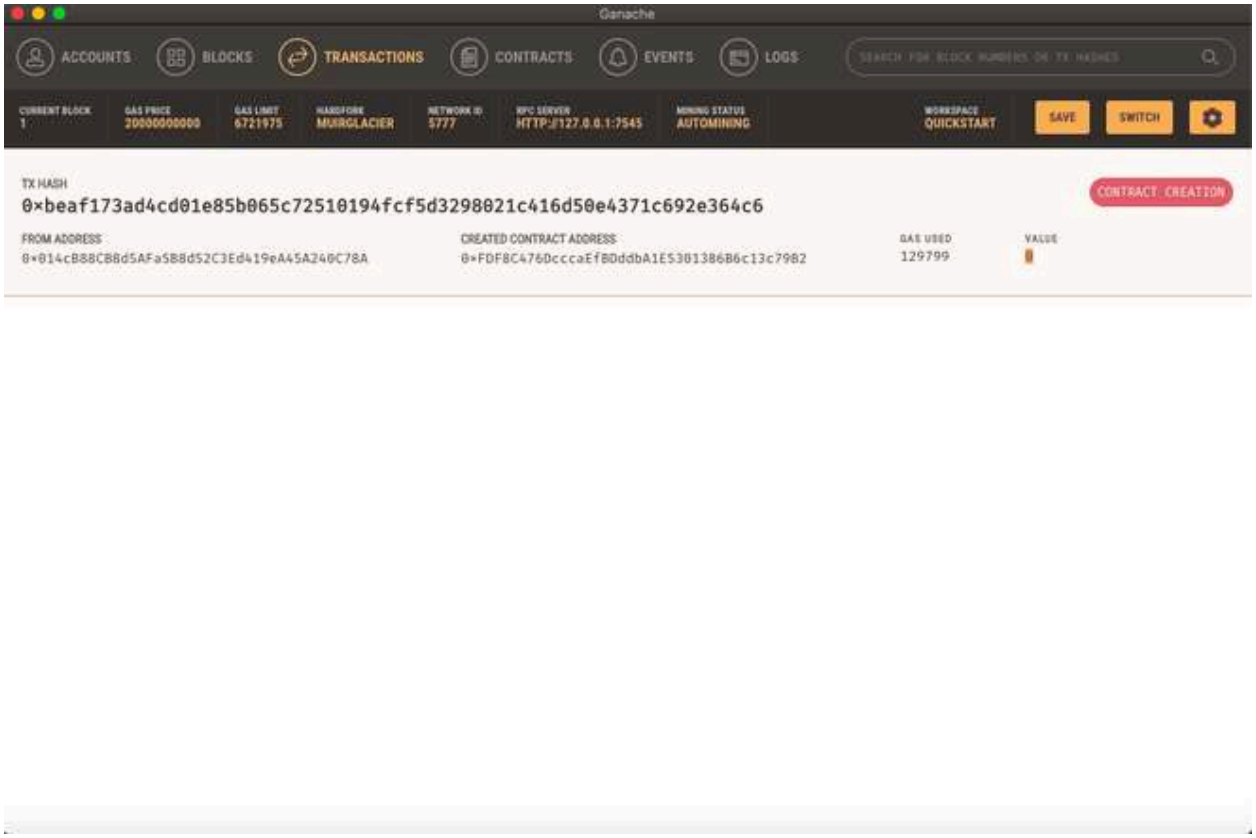
contracts > myContract.sol

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

contract MyContract {
    string public greeting;

    constructor(string memory _greeting) {
        greeting = _greeting;
    }

    function setGreeting(string memory _greeting) public {
        greeting = _greeting;
    }
}
```


Ganache
ACCOUNTS
BLOCKS
TRANSACTIONS
CONTRACTS
EVENTS
LOGS
CURRENT BLOCK
1
GAS PRICE
20000000000
GAS LIMIT
6721975
HARDFORK
MUIRGLACIER
NETWORK ID
5777
RPC SERVER
HTTP://127.0.0.1:7545
MINING STATUS
AUTOMINING
WORKSPACE
QUICKSTART
SAVE
SWITCH
TX HASH
0×beaf173ad4cd01e85b065c72510194fcf5d3298021c416d50e4371c692e364c6
CONTRACT CREATION
FROM ADDRESS
0×814cB88CBBd5AFa5B8d52C3Ed419eA45A240C78A
CREATED CONTRACT ADDRESS
0×FDF8C476DcccaEfBDddbA1E5301386B6c13c79B2
GAS USED
129799
VALUE

C31 Group No: 38

Blockchain Experiment No: 7

Case Study on Hyperledger

Introduction

Hyperledger is an open-source collaborative effort hosted by The Linux Foundation, launched in 2015. It is designed to advance cross-industry blockchain technologies by developing enterprise-grade, distributed ledger frameworks and tools. Unlike public blockchains like Bitcoin and Ethereum, Hyperledger focuses on permissioned blockchains, which are more suitable for business-to-business use cases requiring privacy, scalability, and control over data.

Hyperledger was created to address the challenges enterprises face with traditional blockchain networks. While public blockchains provide transparency and decentralization, they often lack scalability, privacy, and governance controls, which are crucial for organizations in industries such as finance, supply chain, and healthcare. By offering a modular and flexible architecture, Hyperledger allows businesses to build customized blockchain solutions suited to their specific needs.

Features of Hyperledger:

1. **Permissioned Network**
   Hyperledger operates in a permissioned environment, meaning only authorized participants can join the network, ensuring better control over data access and privacy.
2. **Modular Architecture**
   Hyperledger is designed with a modular architecture, allowing developers to plug in various components such as consensus mechanisms, identity services, and smart contracts, depending on business needs.
3. **Smart Contracts**
   Hyperledger supports smart contracts known as "Chain code." These are self-executing contracts where the terms are written directly into the code, allowing automated transactions based on predefined conditions.
4. **Privacy and Confidentiality**
   With Hyperledger, businesses can create private channels between specific network participants, ensuring that only selected participants have access to certain data or transactions.
5. **Performance and Scalability**
   Hyperledger is designed to handle high throughput and scale easily for large enterprise needs. It optimizes performance by allowing selective consensus among participants, rather than requiring the entire network to validate every transaction.

Advantages of Hyperledger

1. **Privacy and Security**:
   Its permissioned nature ensures that sensitive data is kept private and accessible only to authorized users, making it suitable for industries like finance, healthcare, and supply chain management.

2. **High Efficiency**:
   Hyperledger does not require mining like public blockchains, leading to faster transaction processing times and lower energy consumption.

3. **Customizable Consensus**:
   Businesses can select a consensus algorithm that best fits their needs, allowing for higher flexibility and optimization based on the specific use case.

4. **Governance and Compliance**:
   Hyperledger's permissioned structure ensures that all participants are known and accountable, making it easier to comply with regulatory and legal requirements.

5. **Interoperability**:
   Hyperledger supports interoperability, enabling different blockchain networks and traditional systems to integrate and work together seamlessly.

Use Cases of Hyperledger

1. **Supply Chain Management:**
   Companies like Walmart and IBM have used Hyperledger to enhance traceability and transparency in their supply chains, tracking goods from production to final delivery.

2. **Financial Services:**
   Hyperledger is used in areas like cross-border payments, fraud detection, and KYC (Know Your Customer) compliance to enable secure, transparent, and efficient financial transactions.

3. **Healthcare:**
   Hyperledger is applied in healthcare for managing medical records, ensuring that patient data is securely shared between authorized parties without compromising privacy.

4. **Government Services:**
   Hyperledger can be used for managing digital identities, land registries, and voting systems to improve transparency and reduce fraud in government processes.

5. **Digital Identity Management:**
   Hyperledger frameworks like Indy help create decentralized identity solutions, where users control their digital identities and how they are shared.

Disadvantages of Hyperledger

1. **Not Fully Decentralized**:
   While Hyperledger offers more privacy and control, it is not fully decentralized like public blockchains (e.g., Ethereum or Bitcoin), which might be seen as a drawback for users who prefer decentralization.

2. **Complex Setup and Maintenance**:
   Due to its modular architecture and highly customizable nature, setting up and maintaining a Hyperledger network can be complex, requiring significant technical expertise.

3. **Limited Use Cases in Public Domains**:
   Hyperledger is primarily designed for private, permissioned use cases and is less suited for public blockchain applications, limiting its scope compared to public blockchain platforms.

Hyperledger Fabric in Fraud Detection

Problem Statement: Fraud detection in the financial industry is a significant challenge due to the complex nature of transactions, large volumes of data, and the need for real-time monitoring. Traditional systems struggle to handle these demands, often leading to delays in fraud identification and response. The current systems are also vulnerable to data tampering and inefficiency in tracking fraudulent activities across multiple organizations.

Issues faced in Fraud Detection:

- **Data Silos:**
  Financial data is often stored in isolated systems across different organizations, making it difficult to get a holistic view of transactions to detect fraud in real time.
- **Lack of Transparency:**
  Current systems lack transparency, and audits require manual processes, making fraud harder to detect early and quickly.
- **Slow Data Exchange:**
  Fraud detection requires quick access to information, but financial institutions face delays in exchanging data with other parties due to privacy concerns and a lack of trust.
- **Data Tampering Risks:**
  Fraudsters often exploit vulnerabilities in centralized systems by tampering with transaction data, making it difficult to track the source of fraudulent activity.
- **Scalability Issues:**
  Traditional fraud detection systems are not equipped to handle the massive amount of data generated by financial transactions, limiting their scalability.

Solution: Hyperledger, particularly Hyperledger Fabric, offers a Blockchain-based solution that addresses the key challenges of Fraud Detection in financial industry, providing enhanced security, transparency and traceability.

1. Decentralized Ledger for Real-Time Monitoring

- **Description**: Hyperledger Fabric operates on a decentralized, distributed ledger technology (DLT) framework, enabling multiple organizations to maintain a shared, up-to-date record of transactions.
- **Benefit**: This real-time visibility allows financial institutions to monitor transactions across their networks and identify anomalies or patterns that suggest fraudulent activity. For instance, if a transaction occurs outside of normal patterns—such as an unusually high amount or a sudden spike in frequency—alerts can be triggered for immediate investigation.

2. **Privacy Through Private Channels**

- **Description**: Hyperledger Fabric supports the creation of private channels, where only authorized participants can view specific transactions or data. This allows institutions to share sensitive information securely while maintaining confidentiality.
- **Benefit**: For fraud detection, this feature enables organizations to collaborate on identifying suspicious transactions without exposing all their data to every participant in the network. For example, two banks can create a private channel to discuss a potential fraud case without revealing customer details to other members of the network.

3. **Immutable Record of Transactions**

- **Description**: Every transaction recorded on the Hyperledger Fabric blockchain is immutable, meaning once a transaction is logged, it cannot be altered or deleted.
- **Benefit**: This immutability ensures that all transaction histories are preserved, making it easier to audit transactions and trace fraudulent activities back to their origin. In case of fraud detection, investigators can rely on an accurate history of all interactions related to a transaction, which is vital for understanding the fraud's nature and scope.

4. **Smart Contracts for Automated Fraud Detection**

- **Description**: Smart contracts (or Chaincode in Hyperledger Fabric) are self-executing contracts with the terms directly written into the code. They can automatically enforce rules and execute actions based on predefined conditions.
- **Benefit**: Financial institutions can program smart contracts to automatically flag transactions that meet certain risk criteria, such as transactions over a specified limit, transfers to high-risk jurisdictions, or rapid multiple transactions from the same account. This automation speeds up the detection process significantly and reduces the reliance on manual monitoring.

5. **Consensus Mechanism for Trust**

- **Description**: Hyperledger Fabric utilizes a pluggable consensus mechanism, allowing organizations to choose the consensus algorithm that best suits their needs. This mechanism ensures that all transactions are verified by multiple entities before being added to the blockchain.
- **Benefit**: By requiring consensus among participating institutions, the risk of a single entity manipulating transaction data is significantly reduced. For instance, if a bank suspects fraudulent activity, they can consult with others in the network to validate the transaction's legitimacy before proceeding with further action.

Implementation Steps

1. **Identify Stakeholders**:
   Financial institutions, regulatory bodies, and insurance companies form a consortium and participate in the Hyperledger Fabric network.
2. **Design the Blockchain Network**:
   Set up the permissioned blockchain network using **Hyperledger Fabric**. Define the roles and permissions of each stakeholder to control data access and participation in fraud detection.

3. **Develop Smart Contracts (Chaincode)**:
   Create smart contracts that will automatically detect unusual patterns, such as high-value transactions, frequent withdrawals, or cross-border transfers, which can be indicative of fraud.

4. **Establish Private Channels**:
   Create private channels to enable selective sharing of sensitive transaction data. This ensures privacy while allowing collaboration between institutions for fraud detection.

5. **Transaction Validation and Consensus**:
   Implement a consensus mechanism where each transaction is validated by multiple financial entities before being recorded on the blockchain, preventing any tampering or fraudulent entries.

6. **Deploy Real-Time Analytics Tools**:
   Integrate machine learning algorithms and analytics tools on top of the Hyperledger network to continuously monitor and detect suspicious activities in real time.

Benefits and Outcomes

1. **Enhanced Security**:
   The decentralized nature of Hyperledger Fabric ensures that transaction data is tamper-proof, making it harder for fraudsters to manipulate records.

2. **Faster Fraud Detection**:
   Smart contracts automate fraud detection by flagging suspicious transactions immediately, reducing the time it takes to respond to fraud.

3. **Increased Trust Between Parties**:
   By using a consensus mechanism and maintaining an immutable record of all transactions, Hyperledger Fabric builds trust between financial institutions, allowing for secure data sharing and collaboration.

4. **Improved Transparency**:
   The shared ledger enables full visibility into transactions across all entities, making it easier to audit and trace the origins of fraudulent activities.

5. **Scalability**:
   Hyperledger Fabric's modular architecture allows the system to scale with the increasing volume of financial transactions without compromising performance.

Conclusion

Hyperledger Fabric offers an innovative solution to the challenges faced in fraud detection by providing a secure, transparent, and scalable platform for real-time monitoring of financial transactions. By using smart contracts, decentralized ledgers, and private channels, financial institutions can detect and prevent fraud more effectively while maintaining privacy and trust between participants. The case study demonstrates how Hyperledger Fabric can revolutionize fraud detection in the financial sector, ultimately leading to improved security, faster response times, and increased collaboration between organizations.

10/10/24

Blockchain Experiment No: 8

Case Study on Blockchain Platform: EOSIO

Introduction

Blockchain technology has emerged as a revolutionary paradigm that enables decentralized, secure, and transparent data management. It provides a solution to traditional centralized systems by eliminating the need for intermediaries and enhancing trust among participants. Within this ecosystem, EOSIO stands out as a powerful blockchain platform developed by Block.one, launched in 2018. Designed specifically for building decentralized applications (dApps), EOSIO aims to address significant challenges such as scalability, speed, and usability. By offering a user-friendly interface and robust performance, EOSIO facilitates the development of high-performance dApps that can support real-world applications across various sectors, including finance, gaming, and supply chain management.

Overview of EOSIO Architecture

a. Consensus Mechanism

EOSIO employs a Delegated Proof of Stake (DPoS) consensus mechanism, which allows EOS token holders to vote for block producers (BP). This system enables a small number of BPs to validate transactions and create new blocks. Each BP is responsible for producing blocks in a round-robin manner, leading to significantly faster transaction times—often exceeding 4,000 transactions per second (TPS). This scalability addresses the congestion and latency issues prevalent in traditional blockchain systems.

b. Smart Contracts

EOSIO utilizes WebAssembly (WASM) for executing smart contracts, providing a versatile environment for developers. By allowing contracts to be written in multiple programming languages, including C++ and Rust, EOSIO caters to a wider range of developers, enabling them to leverage their existing skills. This flexibility is crucial for creating complex dApps that require robust backend logic.

c. Account Model

The EOSIO account model is distinct in that it allows multiple permissions for a single account. Each account can manage different permission levels, enabling more nuanced access control. For example, a user can grant specific permissions to an application without exposing their entire account. This feature enhances security and makes it easier for developers to create user-centric applications.

d. Inter-Blockchain Communication

EOSIO supports inter-blockchain communication through its architecture, allowing different blockchains to interact seamlessly. This interoperability is vital for developing ecosystems where multiple blockchain applications can work together, sharing data and resources effectively.

Problem Statement

As the adoption of blockchain technology continues to grow across various industries, several persistent challenges hinder its broader implementation and effectiveness. The traditional blockchain platforms, while revolutionary, are grappling with the following key issues:

1. Scalability Issues

- **Limited Throughput**: Most blockchain networks struggle with processing large volumes of transactions per second (TPS). For instance, Bitcoin and Ethereum can process approximately 7 and 30 TPS, respectively. This limited throughput leads to network congestion during peak usage times, resulting in delayed transactions and a poor user experience.
- **Impact on dApps**: Decentralized applications (dApps) that require high-frequency transactions, such as gaming or financial services, are severely affected by scalability issues. As user adoption grows, many dApps on these platforms experience slow response times, leading to user frustration and abandonment.

2. High Transaction Costs

- **Variable Fees**: Traditional blockchains often employ a fee structure that varies based on network congestion. This can lead to exorbitant transaction costs during peak times, making it economically unfeasible for users to transact, especially for microtransactions.
- **Barrier to Entry**: High fees create a barrier to entry for new users and small businesses, preventing them from leveraging blockchain technology. This situation stifles innovation and hinders the development of new dApps that could benefit from lower transaction costs.

3. Complex Development Environments

- **Steep Learning Curves**: Many blockchain platforms require developers to learn specialized programming languages or frameworks, which can slow down development cycles. For instance, platforms like Ethereum rely heavily on Solidity, a language that is not as widely known as more mainstream programming languages.
- **Tooling and Resources**: Developers often encounter a lack of comprehensive tools and libraries, making it challenging to build, test, and deploy dApps efficiently. The complexity can deter new developers from entering the space and can lead to a shortage of talent.

Solutions Provided by EOSIO

EOSIO presents a comprehensive suite of features and technologies designed to tackle the major challenges faced by traditional blockchain platforms. Here's a closer look at how EOSIO effectively addresses each issue:

1. Scalability Solutions

- **Delegated Proof of Stake (DPoS)**: EOSIO utilizes the DPoS consensus mechanism, where EOS token holders elect a limited number of block producers (BPs) to validate transactions. This design allows the network to achieve high throughput, with the capability to process thousands of transactions per second (TPS). The rotation of BPs ensures that the network remains efficient and responsive, even during peak usage periods.

- **Parallel Processing**: EOSIO employs a multi-threaded architecture that enables parallel processing of transactions. This means that multiple transactions can be processed simultaneously, significantly increasing the network's overall efficiency and throughput.

2. **Low Transaction Costs**

- **Resource Model**: Unlike traditional blockchains that charge fees for each transaction, EOSIO employs a resource allocation model based on staking. Users stake EOS tokens to access network resources (CPU, NET, RAM), and transaction costs are minimal, as fees are not charged per transaction. This approach encourages frequent use of the network and lowers barriers to entry for new users and businesses.
- **Predictable Costs**: The staking model provides users with predictable costs, as they can anticipate the resources needed for their applications based on usage. This predictability helps developers and businesses budget more effectively.

3. **User-Friendly Development Environment**

- **Flexible Smart Contract Development**: EOSIO supports smart contracts written in multiple programming languages, including C++ and Rust, which are more familiar to many developers. This flexibility allows a wider range of developers to contribute to the ecosystem and accelerates the development process.
- **Comprehensive Developer Tools**: EOSIO provides a robust set of developer tools, including the EOSIO SDK, libraries, and APIs, which simplify the development, testing, and deployment of dApps. These tools enable developers to build applications efficiently without needing to navigate steep learning curves.

4. **Enhanced Security Features**

- **Immutable Smart Contracts**: Once deployed, smart contracts on EOSIO cannot be altered, which prevents malicious actors from tampering with code post-deployment. This immutability enhances trust in dApps built on the platform.
- **Robust Auditing Mechanisms**: EOSIO encourages developers to conduct thorough testing and auditing of their smart contracts before deployment. The platform also supports third-party auditing tools that help identify vulnerabilities, ensuring a more secure ecosystem.
- **Proactive Governance**: EOSIO's governance model allows the community to propose and vote on upgrades and changes to the protocol. This decentralized governance structure helps maintain security and adaptability, as the community can swiftly respond to emerging threats.

Real World Applications of EOSIO

EOSIO has been leveraged by various projects across different sectors:

- **Everipedia**: A decentralized encyclopedia that aims to create a more democratic platform for knowledge sharing. By utilizing EOSIO, Everipedia ensures that content is verified and curated by its community, enhancing the reliability of information.

- **WAX (Worldwide Asset Exchange)**: A blockchain designed for the trading of virtual goods and collectibles. WAX utilizes EOSIO's technology to enable fast and secure transactions, fostering a marketplace for gamers and collectors.
- **Uphold**: A digital wallet that supports multiple currencies and assets, allowing users to trade and transfer with minimal fees. By using EOSIO, Uphold enhances its transaction speeds and security features.

Case Study: Everipedia

Problem

Everipedia aimed to create a decentralized alternative to traditional online encyclopedias, such as Wikipedia, which have several limitations:

1. **Centralized Control**: Wikipedia's content is moderated and controlled by a small group of administrators. This centralization can lead to biased content, censorship, and a lack of diverse viewpoints.
2. **Incentivization Issues**: While Wikipedia relies on volunteer contributions, it lacks a direct incentive mechanism for contributors. Many potential contributors are discouraged by the lack of recognition or rewards for their efforts.
3. **Monetization Challenges**: Traditional platforms like Wikipedia often struggle with monetization, relying on donations. This model can limit growth and development opportunities.

Solution

Everipedia leveraged blockchain technology, specifically the EOSIO platform, to address these issues:

1. **Decentralization**: By utilizing a decentralized network, Everipedia enables users to create, edit, and manage content collaboratively without a central authority. This decentralization promotes diverse perspectives and reduces bias in content curation.
2. **Token-Based Incentives**: Everipedia introduced its own cryptocurrency, IQ tokens, to incentivize contributions. Users earn IQ tokens for creating and editing articles, which encourages active participation. This token-based model aligns the interests of contributors with the platform's growth.
3. **Smart Contracts for Governance**: Everipedia employs smart contracts to facilitate transparent governance and decision-making processes. Token holders can vote on content changes, ensuring that the community plays a crucial role in curating and validating information.

Result

The implementation of EOSIO technology and the decentralized model has led to several positive outcomes for Everipedia:

1. **Increased User Engagement**: The token-based incentive system has attracted a diverse group of contributors, leading to a more vibrant and active community. Users are more willing to contribute to the platform knowing they will be rewarded for their efforts.

2. **Diverse and Rich Content**: With decentralized governance, Everipedia has witnessed the creation of a wide variety of articles and content types, reflecting a broader spectrum of knowledge and viewpoints compared to traditional encyclopedias.
3. **Enhanced Reliability and Accountability**: The immutable nature of blockchain technology has improved content reliability. Users can trust that information is curated by the community and subject to collective oversight.

Benefits of EOSIO

EOSIO offers numerous advantages:

- **High Transaction Speed**: The DPoS mechanism supports thousands of transactions per second, making it suitable for high-demand applications.
- **Cost Efficiency**: By utilizing a staking model for resource allocation, users face minimal transaction fees, which encourages greater usage.
- **Developer-Friendly**: The platform supports various programming languages and provides extensive documentation, making it accessible to a broader developer audience.
- **Strong Community Support**: EOSIO benefits from an active community of developers and users, contributing to its ongoing development and improvement.

Challenges and Limitations

Despite its advantages, EOSIO faces several challenges:

- **Centralization Concerns**: The reliance on a limited number of block producers can lead to centralization, where a few entities hold significant power over the network.
- **Governance Issues**: The governance model can be contentious, as the voting system may lead to conflicts of interest among stakeholders.
- **Competition**: EOSIO operates in a competitive landscape with other platforms like Ethereum, Cardano, and Polkadot, each offering unique features and capabilities that can attract developers and users.

Conclusion

EOSIO represents a significant advancement in blockchain technology, providing a scalable, efficient, and developer-friendly platform for building decentralized applications. By addressing critical challenges faced by traditional blockchains, EOSIO is well-positioned to foster innovation across various industries. However, for EOSIO to achieve widespread adoption, it must navigate its challenges, particularly concerning centralization and governance. The future of EOSIO will depend on its ability to adapt and evolve in the rapidly changing blockchain landscape.

Thadomal Shahani Engineering College, Bandra, Mumbai
Department of Computer Engineering

Report on Mini Project

Subject: Block Chain (CSDC7022)
AY: 2024-25

Trust Vote

(A Blockchain-based Voting System)

Vedant Devkar - 2103034
Parth Dabholkar- 2103032
Kanav Bhatia - 2103020
Shirish Shetty - 2103164

Guided By
Prof. Nabanita Mandal

Introduction

Elections in India are the foundation of its democracy, allowing citizens to vote for their leaders at various levels of government. Managed by the Election Commission of India, these elections ensure that the voices of over a billion people are heard and represented. Despite being one of the largest and most complex electoral systems in the world, India's elections aim to be free, fair, and transparent.

Fraud and malpractices in elections:

- **Lack of Transparency:** When the election process is not transparent, it can create opportunities for manipulation. Lack of clear oversight and auditing can lead to fraudulent activities going unnoticed.
- **Political Pressure:** In some cases, political leaders or parties may use their influence to coerce or bribe voters, or to tamper with the election process to secure their positions.
- **Corruption:** Corruption at various levels, from local authorities to national officials, can facilitate fraudulent activities such as vote buying, manipulation of voter lists, or tampering with electronic voting machines.
- **Inadequate Security Measures:** Weaknesses in the security of voting machines, electronic systems, or voter registration processes can be exploited for fraud. For example, hacking into electronic voting machines or databases can alter results.
- **Lack of Voter Education:** When voters are not well-informed about the election process or their rights, they may be more susceptible to manipulation or coercion.
- **Economic Incentives:** Candidates or parties with substantial financial resources might engage in fraud to secure victory, knowing that the potential benefits of winning outweigh the risks of getting caught.
- **Inefficient Systems:** Outdated or inefficient electoral systems can create opportunities for fraud due to procedural gaps or delays in addressing irregularities.

Problem Statement

The current electoral system in India faces challenges like fraud, lengthy vote counting, and issues with voter verification. Therefore, we propose to create a decentralized blockchain-based voting system can address these issues by ensuring a secure, transparent, and efficient election process.

Blockchain Concepts

To develop the solution, we have used several blockchain-related technologies and concepts:

1. Ganache

- What it is: Ganache is a personal blockchain environment that developers use for testing and development. It simulates the Ethereum blockchain on your local machine, allowing you to deploy and interact with smart contracts without needing to connect to the actual Ethereum network.
- How it's used in the voting system:
  In the development of your Blockchain Voting System, Ganache is used as a local test network where you can deploy and test your smart contracts before going live on the Ethereum mainnet. By using Ganache, you ensure that all the features of your voting system (like casting a vote, verifying voter identity, and counting votes) work correctly in a controlled environment. Ganache provides logs of the transactions, making it easier to debug and verify the functionality of your smart contracts.

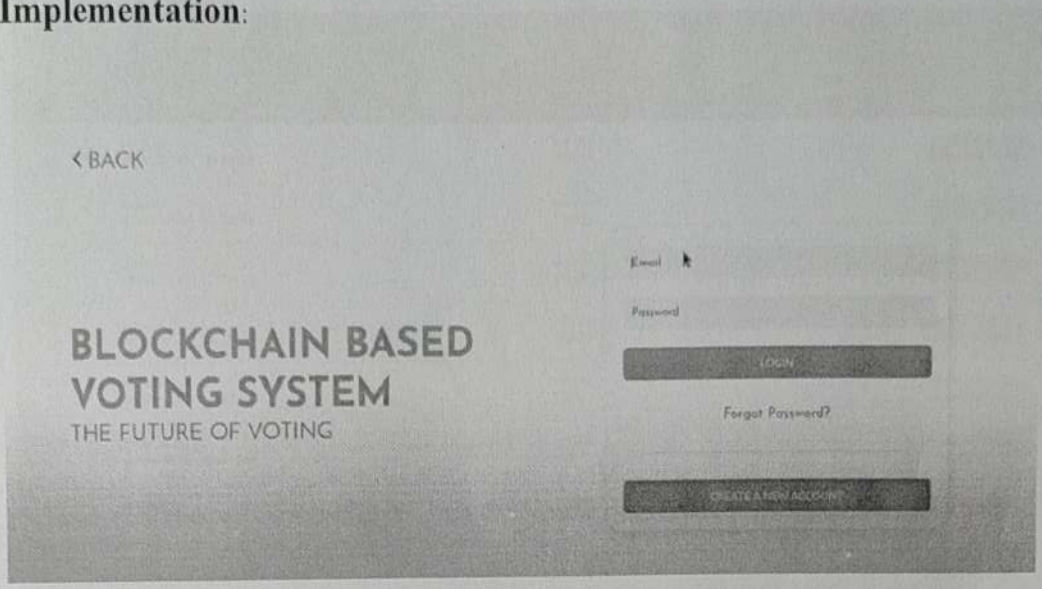
2. Solidity

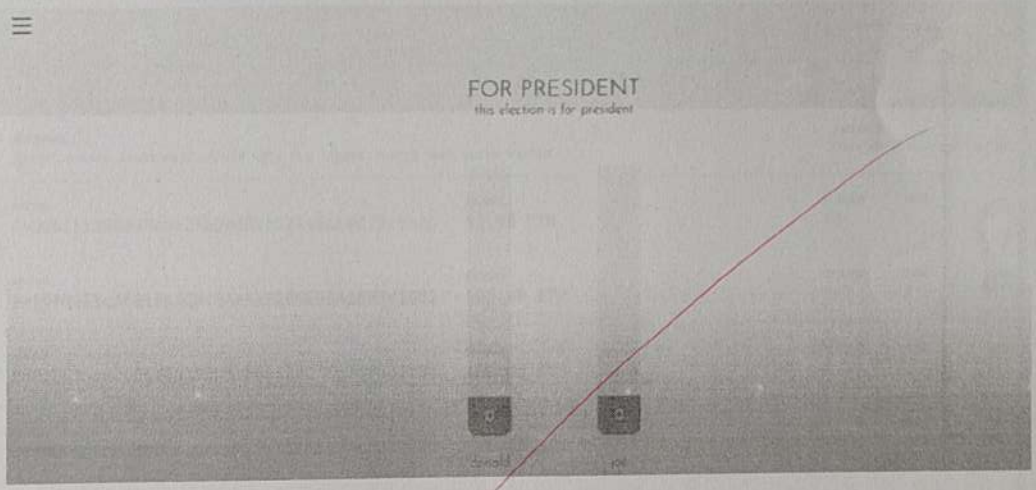
- What it is: Solidity is a programming language specifically designed for writing smart contracts that run on the Ethereum blockchain. It allows developers to define the logic and rules for the execution of contracts in a secure and efficient way.
- How it's used in the voting system:
  In your voting system, Solidity is used to write the logic for key functionalities, such as voter registration, casting a vote, and tallying votes. Solidity enables you to implement complex conditions, like ensuring that each voter can only vote once and that the vote remains anonymous while being securely recorded. With Solidity, you can also program functions that automatically close the voting process and publish the results once voting is complete.

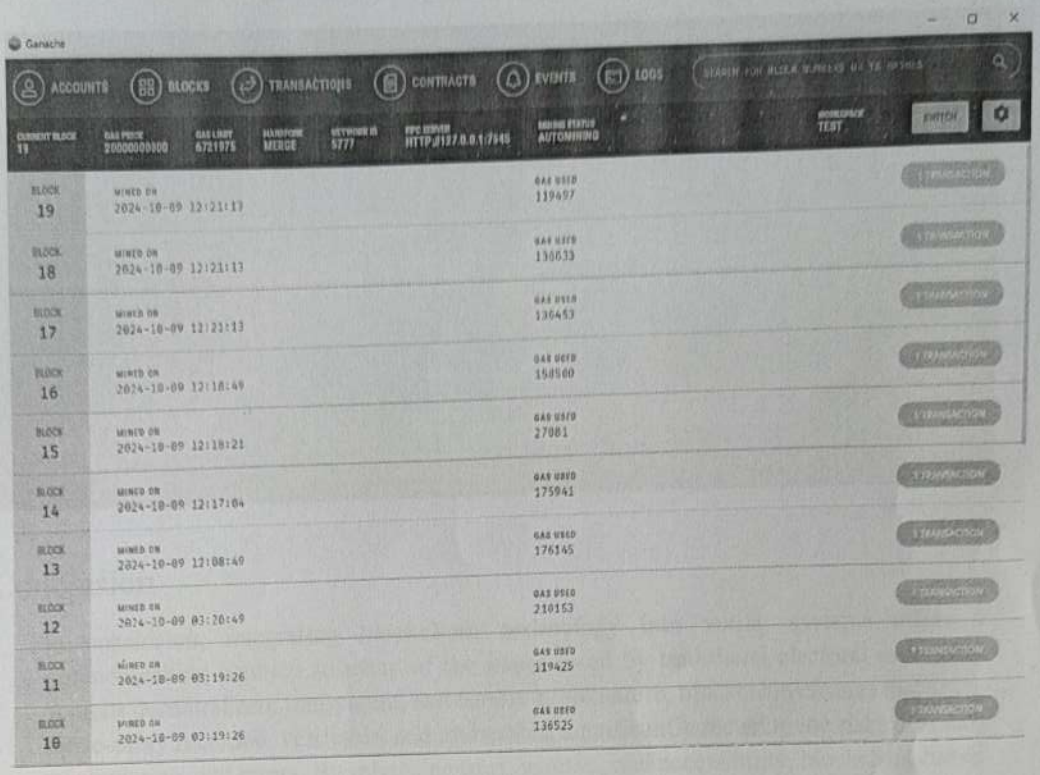
3. Smart Contracts

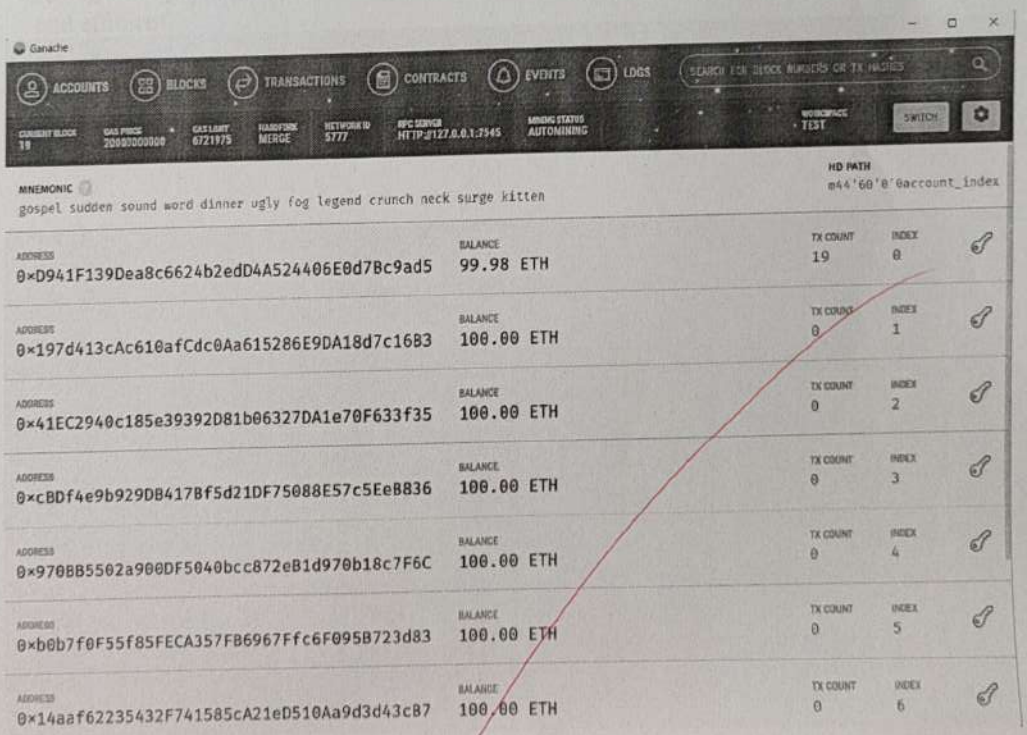
- What they are: Smart contracts are self-executing contracts with the rules and conditions of the agreement written directly into the code. Once deployed, they execute the contract's terms automatically when the predefined conditions are met.
- How they're used in the voting system:
  In your Blockchain Voting System, smart contracts are used to automate and enforce the voting process. For instance, they ensure that only eligible voters can cast a vote, that votes are recorded immutably on the blockchain, and that votes cannot be altered once they are cast. Smart contracts eliminate the need for a central authority to manage the voting process and provide a secure, tamper-proof environment. Once the election is over, the smart contract can automatically tally the votes and declare the result, ensuring that the entire process is transparent and fair.

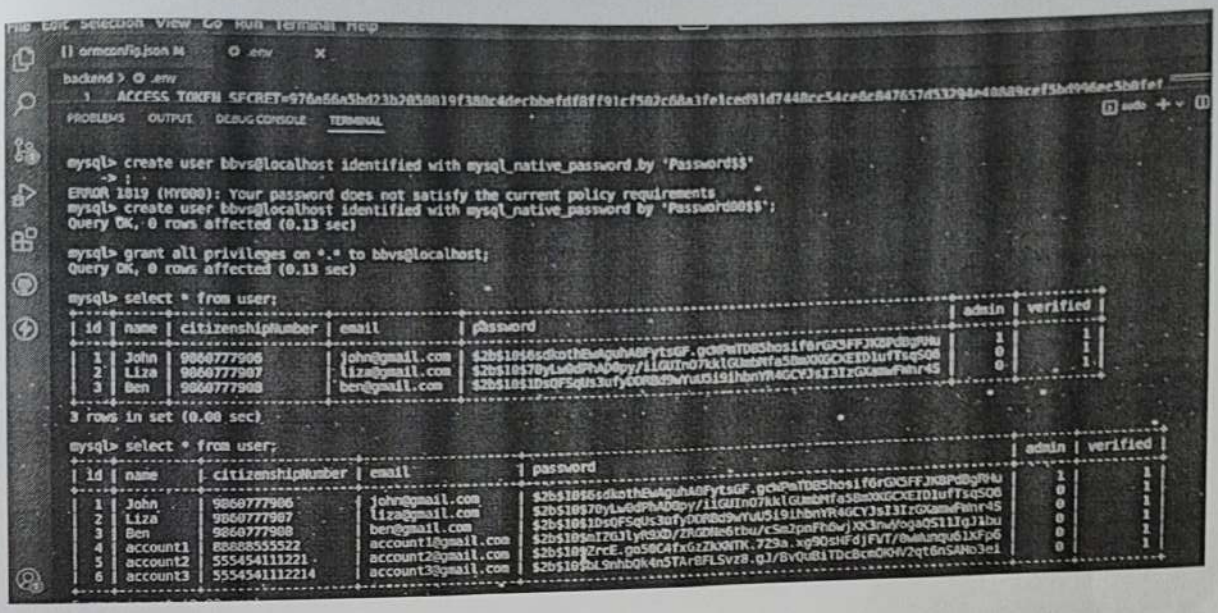
Implementation:






Ganache
ACCOUNTS
BLOCKS
TRANSACTIONS
CONTRACTS
EVENTS
LOGS
SEARCH FOR BLOCK NUMBERS OR TX HASHES
CURRENT BLOCK 19
GAS PRICE 20000000000
GAS LIMIT 6721975
HARDFORK MERGE
NETWORK ID 5777
RPC SERVER HTTP://127.0.0.1:7545
MINING STATUS AUTOMINING
WORKSPACE TEST
SWITCH
BLOCK 19 MINED ON 2024-10-09 12:21:13 GAS USED 119497 1 TRANSACTION
BLOCK 18 MINED ON 2024-10-09 12:21:13 GAS USED 136633 1 TRANSACTION
BLOCK 17 MINED ON 2024-10-09 12:21:13 GAS USED 136453 1 TRANSACTION
BLOCK 16 MINED ON 2024-10-09 12:18:49 GAS USED 150560 1 TRANSACTION
BLOCK 15 MINED ON 2024-10-09 12:18:21 GAS USED 27081 1 TRANSACTION
BLOCK 14 MINED ON 2024-10-09 12:17:04 GAS USED 175941 1 TRANSACTION
BLOCK 13 MINED ON 2024-10-09 12:08:49 GAS USED 176145 1 TRANSACTION
BLOCK 12 MINED ON 2024-10-09 03:20:49 GAS USED 210153 1 TRANSACTION
BLOCK 11 MINED ON 2024-10-09 03:19:26 GAS USED 119425 1 TRANSACTION
BLOCK 10 MINED ON 2024-10-09 03:19:26 GAS USED 136525 1 TRANSACTION


Ganache
ACCOUNTS
BLOCKS
TRANSACTIONS
CONTRACTS
EVENTS
LOGS
SEARCH FOR BLOCK NUMBERS OR TX HASHES
CURRENT BLOCK 19
GAS PRICE 20000000000
GAS LIMIT 6721975
HARDFORK MERGE
NETWORK ID 5777
RPC SERVER HTTP://127.0.0.1:7545
MINING STATUS AUTOMINING
WORKSPACE TEST
SWITCH
MNEMONIC
gospel sudden sound word dinner ugly fog legend crunch neck surge kitten
HD PATH
m/44'/60'/0'/0/account_index
ADDRESS 0x5D941F139Dea8c6624b2edD4A524406E0d7Bc9ad5 BALANCE 99.98 ETH TX COUNT 19 INDEX 0
ADDRESS 0x197d413cAc610afCdc0Aa615286E9DA18d7c16B3 BALANCE 100.00 ETH TX COUNT 0 INDEX 1
ADDRESS 0x41EC2940c185e39392D81b06327DA1e70F633f35 BALANCE 100.00 ETH TX COUNT 0 INDEX 2
ADDRESS 0xcBDf4e9b929DB417Bf5d21DF75088E57c5EeB836 BALANCE 100.00 ETH TX COUNT 0 INDEX 3
ADDRESS 0x970BB5502a900DF5040bcc872eB1d970b18c7F6C BALANCE 100.00 ETH TX COUNT 0 INDEX 4
ADDRESS 0xb0b7f0F55f85FECA357FB6967Ffc6F095B723d83 BALANCE 100.00 ETH TX COUNT 0 INDEX 5
ADDRESS 0x14aaf62235432F741585cA21eD510Aa9d3d43cB7 BALANCE 100.00 ETH TX COUNT 0 INDEX 6



```
mysql> create user bbvs@localhost identified with mysql_native_password by 'Password$$'
    -> ;
ERROR 1819 (HY000): Your password does not satisfy the current policy requirements
mysql> create user bbvs@localhost identified with mysql_native_password by 'Password00$$';
Query OK, 0 rows affected (0.13 sec)

mysql> grant all privileges on *.* to bbvs@localhost;
Query OK, 0 rows affected (0.13 sec)

mysql> select * from user;
+----+-------+-----------------+-----------------+--------------------------------------------------------------+-------+----------+
| id | name  | citizenshipNumber | email           | password                                                     | admin | verified |
+----+-------+-----------------+-----------------+--------------------------------------------------------------+-------+----------+
|  1 | John  | 9860777906      | john@gmail.com  | $2b$10$6sdkothEwAguhA0FytsGF.gOkPmTDBShosif0rGO5FFJK8PdBgNNu |     1 |        1 |
|  2 | Liza  | 9860777907      | liza@gmail.com  | $2b$10$70yLw6dPhAD0py/iiGUInO7kklGUmbNfa5BmXOGCXEIDIufTsqSO6 |     0 |        1 |
|  3 | Ben   | 9860777908      | ben@gmail.com   | $2b$10$1DsOFSqUs3ufyDDRBd9wYuU5i9ihbnYR4GCYJsI3IrGXamwFWnr4S |     0 |        1 |
+----+-------+-----------------+-----------------+--------------------------------------------------------------+-------+----------+
3 rows in set (0.00 sec)

mysql> select * from user;
+----+----------+-----------------+-----------------+--------------------------------------------------------------+-------+----------+
| id | name     | citizenshipNumber | email           | password                                                     | admin | verified |
+----+----------+-----------------+-----------------+--------------------------------------------------------------+-------+----------+
|  1 | John     | 9860777906      | john@gmail.com  | $2b$10$6sdkothEwAguhA0FytsGF.gOkPmTDBShosif0rGO5FFJK8PdBgNNu |     1 |        1 |
|  2 | Liza     | 9860777907      | liza@gmail.com  | $2b$10$70yLw6dPhAD0py/iiGUInO7kklGUmbNfa5BmXOGCXEIDIufTsqSO6 |     0 |        1 |
|  3 | Ben      | 9860777908      | ben@gmail.com   | $2b$10$1DsOFSqUs3ufyDDRBd9wYuU5i9ihbnYR4GCYJsI3IrGXamwFWnr4S |     0 |        1 |
|  4 | account1 | 88888555522     | account1@gmail.com | $2b$10$aT26JlyHSXD/ZRGDNe6tbu/cSmZpmFhOwjXX3nwVogaQS11IgJlbu |     0 |        1 |
|  5 | account2 | 5554541112221   | account2@gmail.com | $2b$10$ZrcE.ga58C4fxGzZkXNTK.7Z9a.xg9OsHFdjFVT/BwNunqu61XFp6 |     0 |        1 |
|  6 | account3 | 5554541112214   | account3@gmail.com | $2b$10$xLSnhDQk4nSTArBFLSvz8.gJ/BvQuBiTDcBcmQOW/2qt6nSANb3ei |     0 |        1 |
+----+----------+-----------------+-----------------+--------------------------------------------------------------+-------+----------+
```

Conclusion

In conclusion, integrating **blockchain technology** into voting systems offers a groundbreaking solution to many of the issues faced by traditional electoral processes. With its decentralized, transparent, and tamper-proof nature, blockchain ensures that votes are securely recorded, verifiable, and immutable, significantly reducing the risks of fraud, manipulation, and errors. By enhancing trust, security, and accessibility, blockchain-based voting can revolutionize democratic participation, making elections more fair, transparent, and efficient.

10/10/24

ASSIGNMENT NO: 1

Q] What are the problems associated with a centralized systems?

⇒ Centralised systems are type of computing architecture where all or most of the processing and data storage is done on a single central server or a group of closely connected servers.

This central server manages all operations, resources and data, acting as the hub through resources and data, where all client requests are processed.

Some key problems are as follows:

1) Security risk: Centralized systems are more susceptible to hacking and data breaches since all the data is stored in a single location once hacker breach the central server, they can access all the data.

2) Lack of Transperancy: Centralized system often lack transparency. Data and operations are controlled by a single entity making it difficult to verify and audit transactions independently.

3) Data Ownership and Privacy: Users in centralized systems typically do not have control over their data. The central authority can access, modify or even sell user data without consent.

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

4. Intermediate Cost: Centralized system often intermediaries or third parties to facilitate transaction, leading to additional cost and delay.

5. Single Point of failure: Centralized are vulnerable to attack, outages or failure. If the central server is compromised or experiencing downtime, the entire system can be distributed.

6. Limited Resilience: Centralized systems can struggle to scale and handle high volume of transactions or users. This can lead to bottle necks, shoudowns and decreased performance during peak usage terms.

Q2) Differentiate between centralized and decentralized and distributed systems.

Centralized System	Decentralize Systems	Distributed
single centre control and manager all the operations	Multiple nodes with independent contd.	Multiple inter-connected nodes working together.
single point of management.	Each nodes operates independently.	Nodes collaborate to achieve a goal.
Limited scalability and can become a bottle neck.	More scalable can add nodes independently.	High scalable, adds more node to distribute load.

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Performance is high initially but decrease.	Performance is generally good and improves with more nodes.	High performance due to parallel processing and resource sharing.
Easier to manage centrally.	More complex hence requires multiple managing node.	Complex to manage require cordination and management of many nodes.
Lower latency as operations are managed centrally.	Can vary depends on the distance b/w the nodes.	Potentially higher latency due to network.
~~Resources~~ Central server resource are heavily utilized.	Resources are spread across multiple node.	Efficient resource sharing across nodes.

7/10/24

ASSIGNMENT NO: 2

Q1) Explain transactions in Blockchain and UTXO and double spending.

⇒ Transaction in Blockchain:-
A transaction in blockchain refer to transfer of digital assets between two parties on blockchain network.

A) Initiation: A user decides to send a digital asset to another user.

B) Creation: The transaction is created specifying the sender address, reciever address and amount to be transfered.

C) Broadcast: The transaction is broadcast to blockchain network, where it awaits validation.

D) Validation: It is done through proof of work and proof of stake. Nodes validate if sender has enough funds.

E) Inclusion in Block: Once validated transaction is included in new block, the blockchain network.

F) Confirmation: A transaction is considered confirm once it is been added to blockchain network.

2) UTXO (Unspent Transaction O/p):

UTXO is model used by some blockchain like bitcoin to track the balance of digital currency.

A) Existing UTXOs: A user balance is represented by a collection of UTXOs from previous transaction that have not yet been spent.

B) Transaction creation: The transactions generate one or more output including: i) the amount sent to recipient.
ii) any change sent back to sender.

C) Broadcasting: Transaction is broadcast to network and existing await validations.

D) Inclusion in Block: The transaction is included in a block, and input UTXOs are marked as spent.

E) New UTXOs: The output of transaction become new UTXOs, which are spent in future.

3) Double Spending:

A) Transaction creation: A malicious user attempts to create two different transaction using same digital asset.

- First transaction: A malicios user attempts to create 2 different

second transaction: The user attempts to send same asset to another recipient before first transaction is confirmed.

B) Broadcast: Both transaction are broadcast to network.

c) validation: Nodes detect that same UTXO or balance is being used in both transaction

D) Consensus Mechanism: It ensures that only of these transaction can be confirmed and added to blockchain.

Q2) Explain in detail CORDA, Ripple and Quorum.

1) Cordo: Cordo is an open-source blockchain platform, designed for buisness. Unlike traditional blockchains, Cordo is tailored specifically for recording, managing and synchronizing financial agreements in a secure, private and scalable manner. It was developed by R3, a consortium of financial institution.

Key features :-

A) Permissioned Network: only authorized participans can join.

B) Smart contracts :- Legally enforcable contract.

C) Concensus: Require consensus only from transaction parties.

2) Ripple:
Ripple is a digital payment protocol that enable fast, low cost international money transfers. It was designed to replace existing systems like SWIFT by providing a more efficient and reliable solution for cross border payment.

Key features: A) RippleNet: A global network for fast low cost payments between financial institutions.
B) Scalability: Handles 1000s of transaction per second.
C) Regulatory compliance: Designed to work within existing regulatory framework.

3) Quorum: Quorum is enterprise focussed version of the ethereum blockchain, developed by JPMC. It is designed used by businesses that require high speed and high secure private transaction.

Key features:
1) Privacy and Confidentiality: Private transaction visible to only involved parties.

2) Ethereum Integration: compatible with ethereum tools, easily development of dApps.

Q4) What is smart contract? How crowd funding platform can be managed using smart contracts

A]:- A smart contract is a self executing construct with term of agreement directly into lines of code.

B]- These contracts run on Blockchain, ensuring that they are immutable and transparent.

C]- Smart contracts automatically execute any predefined conditions are met, without the need for intermediaries.

D]- Key Feature:
i) security ii) Transparency iii) Anomaly
iv) Decentralized v) Accurate vi) self verifying
vii) Self Enforcing.

1) Smart contracts can streamline and secure crowdfunding process by automatically fund collection, distribution and management.
1.) Creation of Smart Contract: A project creator define the terms of crowdfunding campaign.
2). These terms are coded into a smart contract and deployed.
3) Fund Collection: A smart contract securely hold funds contributed by backers until campaign end.
4) Milestone: Funds are released in stages as the project meet predefined milestones
5) Dispute Resolution: Automatic resolution of disputes.

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

03) Explain 0, 1 and 6 confirmation transaction.

A] Each new block added the block containing the transaction counts as an additional information.

B)- 0-confirmation:

1) A transaction with 0 confirmation means it has been broadcast to network but hast not yet been included in the block.
2) At this stage the transaction is unconfirmed and considered risky because it could potentially be altered or double spend.

C]- 1-confirmation:

1) A transaction with 1 confirmation means it has been included in one block on the Blockchain.
2) The transaction is now confirmed and part of blockchain, but still relatively new and somewhat vulnerable to chain recognition.

D]- 6-confirmation:

1) A transaction with 6 confirmation means it has be included in one block and followed by additional blocks on blockchain.
2) This generally considered the gold standard for security in most blockchain, especially in Bitcoin.

1/10/24