DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

B.E / SEM VII / REV 2019 'C SCHEME' / CSE-(AI&ML)
Academic Year: 2023-24

NAME	**SINGH SUDHAM DHARMENDRA**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	
SUBJECT	**BLOCKCHAIN LAB**
COURSE CODE	**CSDOL7022**
PRACTICAL NO.	
DOP	
DOS	

Experiment No. 01-A

Aim :- To Study Cryptography in Blockchain.

Theory :-

1. **Cryptography in Blockchain:**
   Cryptography in blockchain is a fundamental concept that plays a crucial role in ensuring the security and integrity of data within the blockchain network. Cryptography is the practice of using mathematical algorithms and techniques to secure information, making it unreadable to unauthorized users.

   By incorporating robust cryptographic techniques, blockchain networks achieve a high level of security and immutability. The decentralized nature of blockchain, combined with cryptographic mechanisms, ensures that data remains secure, transparent, and tamper-resistant, making blockchain technology suitable for a wide range of applications, from financial transactions to supply chain management and beyond.

2. **Types of cryptography:**
   Cryptography can be classified into several types based on the techniques and methodologies used to secure information. Here are some of the main types of cryptography:

   - **Symmetric Key Cryptography (Secret Key Cryptography):** In symmetric key cryptography, a single secret key is used for both encryption and decryption. The same key is shared between the sender and receiver to ensure secure communication.

   - **Asymmetric Key Cryptography (Public Key Cryptography):** Asymmetric key cryptography uses a pair of keys: a public key and a private key. The public key is freely distributed, while the private key is kept secret. Messages encrypted with the public key can only be decrypted using the corresponding private key.

   - **Hash Functions:** Hash functions are a type of one-way cryptographic function that generates a fixed-size output (hash) from variable-length input data.

   - **Digital Signatures:** Digital signatures are a cryptographic technique used to verify the authenticity and integrity of digital documents or messages. They are created using the private key of a user and can be verified using the corresponding public key.

   - **Block Ciphers:** Block ciphers are symmetric key algorithms that encrypt data in fixed-size blocks. They divide the plaintext into blocks and encrypt each block separately. Block ciphers are widely used in data encryption standards.

   - **Stream Ciphers:** Stream ciphers encrypt data in a continuous stream, one bit at a time. They are often used in scenarios where data is transmitted in real-time or as a continuous stream.

- **RSA (Rivest-Shamir-Adleman):** RSA is a widely used public-key encryption algorithm based on mathematical properties of large prime numbers. Commonly used for secure data transmission & digital signatures.

- **Elliptic Curve Cryptography (ECC):** ECC is a type of public-key cryptography that uses the mathematics of elliptic curves to generate key pairs.

- **Zero-Knowledge Proofs:** Zero-knowledge proofs are advanced cryptographic protocols that allow one party (the prover) to prove knowledge of a secret without revealing any information about the secret to another party (the verifier). They are used in privacy-enhancing applications.

Each type of cryptography serves specific purposes and has its strengths and weaknesses. By leveraging different cryptographic techniques, information can be secured, authenticated, and protected in various ways, depending on the requirements of the application or system.

3. **Cryptography Hash Function:**
   A. **Properties of cryptographic Hash:** Cryptographic hash functions possess several essential properties that make them valuable for securing data and information:

   - **Deterministic:** For a given input, a cryptographic hash function will always produce the same fixed-size output (hash). This determinism ensures consistency and predictability in the hashing process.

   - **Fast Computation:** Hash functions are computationally efficient, allowing for quick calculation of the hash value from the input data.

   - **Preimage Resistance:** It should be computationally infeasible to determine the original input from its hash value. Given the hash, it should be impossible to reverse-engineer the input data.

   - **Second Pre-image Resistance:** Given an input, it should be computationally infeasible to find another input that produces the same hash value. In other words, it should be difficult to find a different input with the same hash.

   - **Collision Resistance:** It should be computationally infeasible to find two different inputs that produce the same hash value. This property ensures that the probability of two distinct inputs generating identical hashes is negligible.

   - **Avalanche Effect:** A small change in the input should result in a significantly different hash output. This property ensures that even minor alterations to the input data produce vastly different hash values.

   - **Non-reversible:** A cryptographic hash function is one-way, meaning that it should not be possible to obtain the original input data from its hash. This property ensures that hashes cannot be "decrypted" to reveal the original content.

B. **Benefits of Hash function in Blockchain:** Cryptographic hash functions play a vital role in ensuring the security and integrity of data within a blockchain network. Here are some key benefits of using hash functions in blockchain:

- **Data Integrity:** Hash functions verify the integrity of data stored in each block. Any alteration to the data, no matter how small, will result in a completely different hash value. By comparing the computed hash with the original hash, blockchain participants can quickly detect tampering or data corruption.

- **Immutability:** The use of hash functions ensures that once a block is added to the blockchain, its contents cannot be changed without affecting the entire chain. This property establishes the immutability of blockchain data, enhancing trust and reliability.

- **Linking Blocks:** Hash functions facilitate the linking of blocks in the blockchain. Each block contains the hash of the previous block, creating a chain of blocks where any modification to a block's data will invalidate subsequent blocks, making it evident that the chain has been tampered with.

- **Secure Transactions:** Cryptographic hash functions are used in transaction verification and digital signatures within the blockchain. They ensure that transactions are securely recorded and authenticated, preventing unauthorized access and ensuring the integrity of the transaction history.

- **Efficiency and Speed:** Hash functions are computationally efficient, enabling fast validation and verification of data in the blockchain. This efficiency is crucial for maintaining the decentralized and distributed nature of blockchain networks.

- **Mining and Consensus:** In blockchain networks that use Proof-of-Work (PoW) consensus mechanisms, miners compete to solve complex cryptographic puzzles (hash computations) to validate and add new blocks. Hash functions provide a mechanism for achieving consensus and securing the network.

Overall, cryptographic hash functions are foundational to the security and functionality of blockchain technology. They play a central role in establishing trust, data integrity, and the immutability of the blockchain, making them indispensable in securing and maintaining the integrity of distributed ledger systems.

4. **Benefits of Cryptography in Blockchain:**

Cryptography plays a pivotal role in blockchain technology, providing several essential benefits that contribute to the security, privacy, and overall functionality of the blockchain network. Here are the key benefits of cryptography in blockchain:

- **Security and Data Integrity:** Cryptography ensures the security & integrity of data stored in each block. Hash functions & digital signatures are used to verify integrity of data, making it tamper-resistant.

- **Immutability:** Through cryptographic hashing and linking of blocks, blockchain achieves immutability. Once a block is added to the chain, its data cannot be altered without invalidating subsequent blocks.

- **Secure Transactions:** Cryptography enables secure transactions within the blockchain network. Digital signatures using asymmetric key cryptography ensure that only the authorized sender can initiate a transaction, and the integrity of the transaction remains intact.

- **Privacy and Confidentiality:** Blockchain can support private and confidential transactions through various cryptographic techniques. Zero-knowledge proofs and advanced cryptographic algorithms allow parties to prove the validity of a transaction without revealing sensitive information.

- **Authentication and Access Control:** Cryptographic key pairs enable secure authentication of users in the blockchain network. Public keys serve as addresses for receiving transactions, while private keys are used to sign and authorize transactions.

- **Decentralization and Trust:** Cryptography enables blockchain's decentralized nature, where data is distributed across multiple nodes. Through cryptographic protocols and consensus algorithms, participants can trust the system without the need for central authorities or intermediaries.

- **Efficiency and Scalability:** Cryptographic techniques used in blockchain are computationally efficient, allowing for fast validation and verification of transactions. This efficiency contributes to the scalability of the blockchain network, enabling it to handle a high volume of transactions.

- **Protection against Cyber Attacks:** Cryptography provides protection against various cyber attacks, including data breaches, tampering, and unauthorized access. The robust cryptographic algorithms used in blockchain are designed to withstand attacks, enhancing the security of the entire system.

In summary, cryptography is the backbone of blockchain technology, providing the necessary tools to ensure security, privacy, and trust in a decentralized and transparent manner. The use of cryptographic techniques in blockchain networks fosters a secure and reliable ecosystem for various applications, ranging from cryptocurrencies to supply chain management and beyond.

5. **Limitations of Cryptography in Blockchain:**

While cryptography is an essential component of blockchain technology, it is not without its limitations. Some of the key limitations of cryptography in blockchain include:

- **Quantum Computing Threat:** One of the emerging challenges for traditional cryptographic algorithms is the potential threat posed by quantum computing. Quantum computers have the potential to break certain widely used encryption algorithms, such as RSA and ECC, which could compromise the security of blockchain networks. To address this, research into quantum-resistant cryptographic algorithms is ongoing.

- **Key Management Complexity:** In blockchain networks, users need to manage their cryptographic key pairs securely. Mishandling or loss of private keys can result in permanent loss of access to funds or data. Ensuring robust key management practices is critical, and mistakes can be costly.

- **Algorithmic Vulnerabilities:** Cryptographic algorithms are subject to vulnerabilities over time as new cryptographic techniques are developed and computational power increases. A flaw or vulnerability discovered in a widely used cryptographic algorithm could jeopardize the security of blockchain networks.

- **Transaction Privacy Challenges:** While blockchain technology provides transparency for all transactions, it may encounter challenges in providing complete privacy for sensitive data. Balancing transparency with privacy is a complex issue in some blockchain use cases.

- **Zero-Day Attacks and Exploits:** Zero-day attacks refer to vulnerabilities in cryptographic protocols or software that are unknown to developers. Exploiting these unknown vulnerabilities can lead to security breaches and compromise the integrity of blockchain data.

- **Smart Contract Vulnerabilities:** Smart contracts, which often rely on cryptographic primitives, can be vulnerable to bugs or coding errors. These vulnerabilities can result in serious security breaches and financial losses.

- **Regulatory and Compliance Challenges:** The use of strong encryption in blockchain can present challenges for compliance with data protection regulations in some jurisdictions. Balancing privacy and regulatory requirements is a complex issue for blockchain applications.

Despite these limitations, cryptography remains a critical tool for securing blockchain networks. As blockchain technology continues to evolve, researchers and developers work to address these challenges and develop more robust cryptographic solutions to enhance the security and effectiveness of blockchain systems.

6. **Applications of Cryptography:**

Cryptography finds applications in various domains, providing essential security and privacy solutions. Some of the key applications of cryptography include:

- **Secure Communication:** Cryptography is widely used in securing communication channels, such as email encryption and secure messaging applications.
- **Data Encryption:** Cryptography is used to encrypt data at rest and in transit. Encryption transforms plaintext data into ciphertext using cryptographic algorithms, making it unreadable without the appropriate decryption key.
- **Secure Transactions:** Cryptography is essential in securing financial transactions, including online banking, digital payments, and cryptocurrencies.
- **Digital Signatures:** Digital signatures use asymmetric cryptography to verify the authenticity and integrity of digital documents, messages, and software.
- **Password Protection:** Cryptography is used in password hashing to protect user credentials. Instead of storing actual passwords, systems store their cryptographic hashes, making it difficult for attackers to reverse-engineer the original passwords.
- **Virtual Private Networks (VPNs):** VPNs use cryptographic tunneling protocols to secure internet connections, enabling users to access private networks securely over public networks.
- **Blockchain Technology:** Blockchain technology relies heavily on cryptography for securing transactions, ensuring data integrity, and enabling digital signatures and consensus mechanisms.
- **Secure IoT Communication:** Cryptography is used in securing communications between Internet of Things (IoT) devices to ensure data privacy and prevent unauthorized control.

Overall, cryptography is a foundational pillar of modern cybersecurity, enabling secure communications, data protection, and authentication across a wide range of applications and technologies.

Conclusion :- We had successfully studied Cryptography in Blockchain as cryptography stands as a formidable pillar of modern technology, playing a vital role in securing digital communications, protecting sensitive information, and ensuring the integrity of data in various applications. Its significance is especially pronounced in the realm of blockchain, where it underpins the trust, transparency, and immutability that define distributed ledger technology.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

B.E / SEM VII / REV 2019 'C SCHEME' / CSE-(AI&ML)
Academic Year: 2023-24

NAME	**SINGH SUDHAM DHARMENDRA**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	
SUBJECT	**BLOCKCHAIN LAB**
COURSE CODE	**CSDOL7022**
PRACTICAL NO.	
DOP	
DOS	

Experiment No. 01-B

Aim: **Introduction to truffle establishing local blockchain using truffle.**

Theory:

What is truffle?

A world class development environment, testing framework & asset pipeline for blockchains using Ethereum Virtual Machine (EVM), aiming to make life as a developer easier. With Truffle, you get:

- Built-in smart contract compilation, linking, deployment and binary management.
- Advanced debugging with breakpoints, variable analysis, and step functionality.
- Use console.log in your smart contracts Deployments & transactions through MetaMask with Truffle Dashboard to protect your mnemonic.
- External script runner that executes scripts within a Truffle environment.
- Interactive console for direct contract communication.
- Scriptable, extensible deployment & migrations framework.
- Network management for deploying to any number of public & private networks.
- Package management with NPM, using the ERC190 standard.
- Configurable build pipeline with support for tight integration.

Contracts:

Contracts in Solidity are similar to classes in object-oriented languages. They contain persistent data in state variables, and functions that can modify these variables. Calling a function on a different contract (instance) will perform an EVM function call and thus switch the context such that state variables in the calling contract are inaccessible. A contract and its functions need to be called for anything to happen. There is no “cron” concept in Ethereum to call a function at a particular event automatically.

Migrations:

Migrations are JavaScript files that help you deploy contracts to the Ethereum network. These files are responsible for staging your deployment tasks, and they're written under the assumption that your deployment needs will change over time. As your project evolves, you'll create new migration scripts to further this evolution on the blockchain.

Testing:

FrameworkTruffle comes standard with an automated testing framework to make testing your contracts a breeze. This framework lets you write simple & manageable tests in two different ways:

- In Javascript and TypeScript, for exercising your contracts from the outside world, just like your application.
- In Solidity, for exercising your contracts in advanced, bare-to-the-metal scenarios.

Configuration:

Your configuration file is called truffle-config.js and is located at the root of your project directory. This file is a Javascript file and can execute any code necessary to create your configuration. It must export an object representing your project configuration.It must export an object representing your project configuration like the example below.

```
module.exports = {
        networks: {
            development: {
                host: "127.0.0.1",
                port: 8545,
                network_id: "*" // Match any network id
            }
        },
        compilers: {
            solc: {
                version: "^0.8.0
        } }};
```

Output:

How to install Truffle-

```
(base) computer@computer-ThinkCentre:~$ sudo su
root@computer-ThinkCentre:/home/computer# npm install -g npm@9.8.1

removed 17 packages, changed 102 packages, and audited 250 packages in 1s

28 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
root@computer-ThinkCentre:/home/computer# npm install -g truffle
npm WARN deprecated har-validator@5.1.5: this library is no longer supported
npm WARN deprecated mkdirp-promise@5.0.1: This package is broken and no longer maintained. 'mkdirp' itself s
npm WARN deprecated request@2.88.2: request has been deprecated, see https://github.com/request/request/issu
npm WARN deprecated multibase@0.6.1: This module has been superseded by the multiformats module
npm WARN deprecated multicodec@0.5.7: This module has been superseded by the multiformats module
npm WARN deprecated cids@0.7.5: This module has been superseded by the multiformats module
npm WARN deprecated apollo-server-types@3.8.0: The `apollo-server-types` package is part of Apollo Server v2
found in the `@apollo/server` package. See https://www.apollographql.com/docs/apollo-server/previous-version
npm WARN deprecated apollo-server-plugin-base@3.7.2: The `apollo-server-plugin-base` package is part of Apol
lity is now found in the `@apollo/server` package. See https://www.apollographql.com/docs/apollo-server/prev
npm WARN deprecated apollo-server-errors@3.3.1: The `apollo-server-errors` package is part of Apollo Server
w found in the `@apollo/server` package. See https://www.apollographql.com/docs/apollo-server/previous-versi
npm WARN deprecated apollo-reporting-protobuf@3.4.0: The `apollo-reporting-protobuf` package is part of Apol
lity is now found in the `@apollo/usage-reporting-protobuf` package. See https://www.apollographql.com/docs/
npm WARN deprecated apollo-server-env@4.2.1: The `apollo-server-env` package is part of Apollo Server v2 and
d in the `@apollo/utils.fetcher` package. See https://www.apollographql.com/docs/apollo-server/previous-vers
npm WARN deprecated apollo-datasource@3.3.2: The `apollo-datasource` package is part of Apollo Server v2 and
pollo-server/previous-versions/ for more details.
npm WARN deprecated apollo-server-express@3.12.0: The `apollo-server-express` package is part of Apollo Serv
 now found in the `@apollo/server` package. See https://www.apollographql.com/docs/apollo-server/previous-ve
npm WARN deprecated apollo-server-core@3.12.0: The `apollo-server-core` package is part of Apollo Server v2
ound in the `@apollo/server` package. See https://www.apollographql.com/docs/apollo-server/previous-versions
npm WARN deprecated testrpc@0.0.1: testrpc has been renamed to ganache-cli, please use this package from now
npm WARN deprecated apollo-server@3.12.0: The `apollo-server` package is part of Apollo Server v2 and v3, wh
e `@apollo/server` package. See https://www.apollographql.com/docs/apollo-server/previous-versions/ for more
npm WARN deprecated @ensdomains/resolver@0.2.4: Please use @ensdomains/ens-contracts
npm WARN deprecated @ensdomains/ens@0.4.5: Please use @ensdomains/ens-contracts
npm WARN deprecated uuid@3.4.0: Please upgrade  to version 7 or higher.  Older versions may use Math.random(
details.
npm WARN deprecated multicodec@1.0.4: This module has been superseded by the multiformats module
npm WARN deprecated multibase@0.7.0: This module has been superseded by the multiformats module
npm WARN deprecated uuid@2.0.1: Please upgrade  to version 7 or higher.  Older versions may use Math.random(
details.

added 825 packages in 1m

107 packages are looking for funding
  run `npm fund` for details
root@computer-ThinkCentre:/home/computer# truffle version
Truffle v5.11.1 (core: 5.11.1)
Ganache v7.9.0
Solidity v0.5.16 (solc-js)
Node v18.9.0
Web3.js v1.10.0
```

Establishing Local blockchain:-

```
root@computer-ThinkCentre:/home/computer# cd Desktop/
root@computer-ThinkCentre:/home/computer/Desktop# truffle unbox metacoin [PATH/TO/DIRECTORY]

Starting unbox...
=================

✓ Preparing to download box
✓ Downloading
✓ Cleaning up temporary files
✓ Setting up box

Unbox successful, sweet!

Commands:

  Compile contracts: truffle compile
  Migrate contracts: truffle migrate
  Test contracts:    truffle test
```

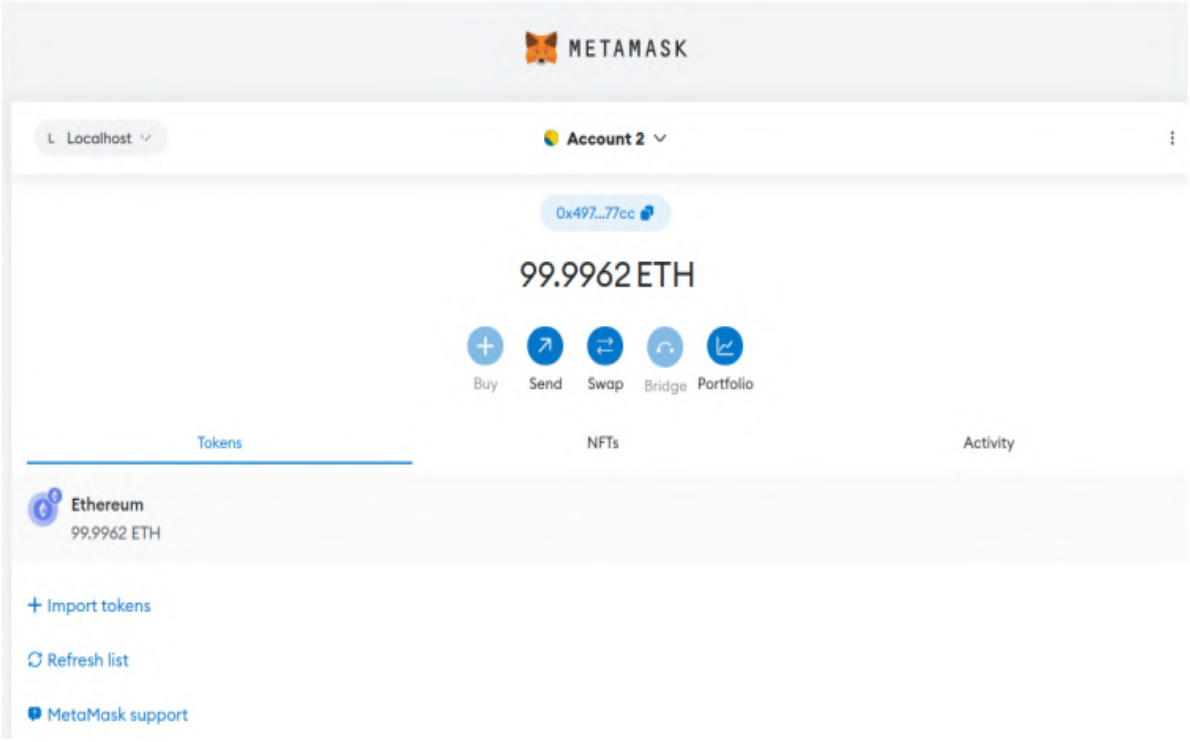
```
(base) computer@computer-ThinkCentre:~/Desktop/[PATH/TO/DIRECTORY]$ suo su
Command 'suo' not found, did you mean:
  command 'sudo' from deb sudo (1.9.9-1ubuntu2.4)
  command 'sudo' from deb sudo-ldap (1.9.9-1ubuntu2.4)
  command 'su1' from deb hxtools (20211204-1)
  command 'su' from deb util-linux (2.37.2-4ubuntu3)
  command 'sum' from deb coreutils (8.32-4.1ubuntu1)
  command 'sur' from deb subtle (0.11.3224-xi-2.2build4)
  command 'sup' from deb sup (20100519-3)
  command 'sumo' from deb sumo (1.12.0+dfsg1-1)
Try: sudo apt install <deb name>
(base) computer@computer-ThinkCentre:~/Desktop/[PATH/TO/DIRECTORY]$ sudo su
[sudo] password for computer:
root@computer-ThinkCentre:/home/computer/Desktop/[PATH/TO/DIRECTORY]# truffle compile.


Compiling your contracts...
===========================
✓ Fetching solc version list from solc-bin. Attempt #1
✓ Downloading compiler. Attempt #1.
> Compiling ./contracts/ConvertLib.sol
> Compiling ./contracts/MetaCoin.sol
> Artifacts written to /home/computer/Desktop/[PATH/TO/DIRECTORY]/build/contracts
> Compiled successfully using:
   - solc: 0.8.13+commit.abaa5c0e.Emscripten.clang
root@computer-ThinkCentre:/home/computer/Desktop/[PATH/TO/DIRECTORY]# truffle develop
Truffle Develop started at http://127.0.0.1:9545/

Accounts:
(0) 0x497867f6bc0bf2fc16f1d9d0b55487c9819277cc
(1) 0x294ef76592633ca993e6a38507db03e4f7752fd3
(2) 0x49306346ccca7892fa949863c4c951f9276d0046
(3) 0x1e58066ea6cf4bb38836739c1981ff1051f613ab
(4) 0xf53ff47196be263912fac9ea93b8f42ca90df1be
(5) 0x6de888b53b457e0b91d02ad517b3cbac7c5c6aa9
(6) 0x93fea5b9810da53711a7cf9f66b966b0dd45e521
(7) 0x9d6b8772c294b2042478f834500d22b205ed5487
(8) 0x95e0d30748fbcf6e0143b57bf5f2cefb656b375b
(9) 0x7cd1c010fdbcabc5369a65ffe829b11862201a88

Private Keys:
(0) 2e29305a624e30b61338faed20c1518fd1ef41ed15de4590c13d49e52f9fb7b6
(1) e234499cc4816a16bb89a7114e065005b20f3a662a4dd2752d61462ed8f843f1
(2) 40e974327039103c0b92ea485513a95e638f449d69e1be60843891d9f6dbf7e7
(3) 9fb97294fc07a394fcd27650c263e845e9eac8577df7492040e1f8822cc159fc
(4) cd8478018b3f88803ac36d694c2b3f3c85d20ed2b945ffbb72e52cd9ffb8ed8c
(5) 223a57f831c536e9c416b14e0132b4b6b2131dc63735cd6a90a7714b8264f6cd
(6) 1252e34dcaee2e47b4cfb65a34ec8d4073d08e0e20ecd1f1c37bbff9c29c0a0c
(7) 18862ece18c7e3e4c3df801b19dda8fce9504ea41f30ee372e1cbd7cb930458c
(8) 93f2430e78cfad8d6e8d6a2d174fe2c012a03033d80a8171bbff84e1235a2a2c
(9) c0c510eff75fb283405f9b4b2b4f1ac1badf0ee0cb942600249d1edfde257749

Mnemonic: ramp involve strategy armor nature equal essay chase canal small proud trade
```

```
truffle(development)> await web3.eth.getAccounts()
[
  '0x497867f6bC0bF2Fc16F1d9d0B55487c9819277cc',
  '0x294eF76592633CA993E6A38507Db03E4F7752Fd3',
  '0x49306346ccCA7892Fa949863C4C951F9276D0046',
  '0x1e58066ea6cF4BB38836739C1981fF1051f613ab',
  '0xF53Ff47196be263912fAC9EA93b8F42Ca90DF1Be',
  '0x6De888B53b457E0B91D02ad517b3cBac7c5c6aa9',
  '0x93fEA5B9810Da53711A7CF9f66B966B0dd45E521',
  '0x9D6B8772c294b2042478F834500d22b205Ed5487',
  '0x95E0D30748FbcF6e0143B57BF5f2Cefb656B375b',
  '0x7Cd1C010FDBCaBc5369A65fFe829b11862201A88'
]
truffle(development)> let balance = await instance.getBalance(accounts[0])
undefined
truffle(development)>  balance.toNumber()
10000
truffle(development)> let ether = await instance.getBalanceInEth(accounts[0])
Uncaught SyntaxError: Identifier 'ether' has already been declared
truffle(development)> instance.sendCoin(accounts[1], 500)
{
  tx: '0xd01a0c5ab64a1f8c58d324779ea66947b81755a15251ea769fe1a3465127297d',
  receipt: {
    transactionHash: '0xd01a0c5ab64a1f8c58d324779ea66947b81755a15251ea769fe1a3465127297d',
    transactionIndex: 0,
    blockNumber: 5,
    blockHash: '0x5e2f1eac1c0c7d7b0b1dde8dfcb8fb0bcd84e7576f68805676aa5ea91e2942fe',
    from: '0x497867f6bc0bf2fc16f1d9d0b55487c9819277cc',
    to: '0x1325084aa05a0c1d0de7cd229642f33cfb54e4dc',
    cumulativeGasUsed: 52297,
    gasUsed: 52297,
    contractAddress: null,
    logs: [ [Object] ],
    logsBloom: '0x00000000000000000000000000000000000000000000000000000000000000100000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000010000000000000000000000010000080000000000000000080000000000000000000
0000000000000000000000000000000000040000000000000000000000000000000000000000000000000000000000000000000000000000000000000000',
    status: true,
    effectiveGasPrice: 3038380000,
    type: '0x2',
    rawLogs: [ [Object] ]
  },
  logs: [
    {
      address: '0x1325084Aa05a0C1d0DE7cd229642F33cFb54e4DC',
      blockHash: '0x5e2f1eac1c0c7d7b0b1dde8dfcb8fb0bcd84e7576f68805676aa5ea91e2942fe',
      blockNumber: 5,
      logIndex: 0,
      removed: false,
      transactionHash: '0xd01a0c5ab64a1f8c58d324779ea66947b81755a15251ea769fe1a3465127297d',
      transactionIndex: 0,
      id: 'log_5a85b4b1',
      event: 'Transfer',
      args: [Result]
    }
```

```
(base) computer@computer-ThinkCentre:~/Desktop/[PATH/TO/DIRECTORY]$ truffle migrate

Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.


Starting migrations...
======================
> Network name:    'development'
> Network id:      5777
> Block gas limit: 6721975 (0x6691b7)


1_deploy_contracts.js
=====================

   Replacing 'ConvertLib'
   ----------------------
   > transaction hash:    0xa4b6024db1cf0bcf2d3138ea3a21f9762935b8127ca1200a6b171544fdb13345
   > Blocks: 0            Seconds: 0
   > contract address:    0xEA59FE81aA4971F9cbca88AC7A78d03948d18196
   > block number:        3
   > block timestamp:     1690787384
   > account:             0x497867f6bC0bF2Fc16F1d9d0B55487c9819277cc
   > balance:             99.9976029811635876
   > gas used:            157604 (0x267a4)
   > gas price:           3.1863545 gwei
   > value sent:          0 ETH
   > total cost:          0.000502182214618 ETH


   Linking
   -------
   * Contract: MetaCoin <--> Library: ConvertLib (at address: 0xEA59FE81aA4971F9cbca88AC7A78d03948d18196)

   Replacing 'MetaCoin'
   --------------------
   > transaction hash:    0x634bbdd1d4456a2823d2110cc0d9a3610e0116dcdd7902c06b172151396aa9d3
   > Blocks: 0            Seconds: 0
   > contract address:    0x8B091779E70aA979d551b7e9a35C0dA57d149cCe
   > block number:        4
   > block timestamp:     1690787384
   > account:             0x497867f6bC0bF2Fc16F1d9d0B55487c9819277cc
   > balance:             99.9963093013149786
   > gas used:            416700 (0x65bbc)
   > gas price:           3.10458327 gwei
   > value sent:          0 ETH
   > total cost:          0.001293679848609 ETH

   > Saving artifacts
```

```
}
truffle(development)> let received = await instance.getBalance(accounts[1])
undefined
truffle(development)>  received.toNumber()
500
truffle(development)> let newBalance = await instance.getBalance(accounts[0])
undefined
truffle(development)>  newBalance.toNumber()
9500
truffle(development)>
```

Metamask:-



Conclusion: We have installed Truffle and successfully established local blockchain using Truffle.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

B.E / SEM VII / REV 2019 'C SCHEME' / CSE-(AI&ML)
Academic Year: 2023-24

NAME	**SINGH SUDHAM DHARMENDRA**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	
SUBJECT	**BLOCKCHAIN LAB**
COURSE CODE	**CSDOL7022**
PRACTICAL NO.	
DOP	
DOS	

Experiment No. 02

Aim :- To Study Blockchain Platforms

Theory :-

1. **Introduction:**
   Blockchain technology has gained significant popularity since the inception of Bitcoin in 2009. Various blockchain platforms have emerged, each offering unique features and use cases. This case study aims to analyze and compare some of the most prominent blockchain platforms in the market, including Ethereum, EOS.IO, TRON, Stellar, Ripple, IBM, Hyperledger, Binance, Kaleido, Blockstream, Hashgraph, Algorand, Coinbase, and GetApp.

2. **Methodology:**
   The study involves a comprehensive analysis of the technical architecture, consensus mechanisms, scalability, security, transaction speed, governance model, ecosystem, and real-world applications of each blockchain platform. Primary sources include official whitepapers, technical documentation, and publicly available information from the respective projects.

3. **Blockchain Platforms:**

   3.1 Ethereum:

   Ethereum is a decentralized blockchain platform designed to support smart contracts and decentralized applications (DApps). It uses the Ethereum Virtual Machine (EVM) and operates on a Proof of Work (PoW) consensus mechanism. Ethereum's large developer community and extensive ecosystem have driven its adoption in various sectors, including DeFi and NFTs. However, scalability challenges have led to high transaction fees during network congestion.

   Overview: Ethereum is a decentralized, programmable blockchain platform designed to execute smart contracts and decentralized applications (DApps).

   Key Features: It uses the Ethereum Virtual Machine (EVM) and employs Proof of Work (PoW) consensus.

   Strengths: Large developer community, diverse ecosystem, and strong adoption in DeFi and NFTs.

   Weaknesses: Scalability issues and high transaction fees during periods of network congestion.

   3.2 EOS.IO:

   EOS.IO is a blockchain platform aiming to provide fast and scalable DApps development. It employs the Delegated Proof of Stake (DPoS) consensus mechanism, which enables high transaction throughput and low latency. EOS.IO is suitable for real-time applications, but there are debates about its potential impact on decentralization due to the DPoS consensus.

   Overview: EOS.IO is a decentralized blockchain platform aiming to provide fast and scalable DApps development.

   Key Features: EOS.IO uses the Delegated Proof of Stake (DPoS) consensus mechanism and offers parallel processing.

Strengths: High transaction throughput and low latency, making it suitable for real-time applications.
Weaknesses: Some argue that the DPoS consensus may compromise decentralization.

3.3 TRON:

TRON is a blockchain platform with a focus on content-sharing and entertainment applications. It utilizes the Delegated Proof of Stake (DPoS) consensus and has a vibrant ecosystem for gaming and social media. TRON's strengths lie in its presence in the entertainment industry and high transaction capacity, but concerns have been raised regarding its centralization and project plagiarism accusations.
Overview: TRON is a blockchain platform focusing on content-sharing and entertainment applications.
Key Features: It uses the Delegated Proof of Stake (DPoS) consensus and features a vibrant ecosystem for gaming and social media.
Strengths: Strong presence in the entertainment industry and high transaction capacity.
Weaknesses: Concerns regarding centralization and criticism over project plagiarism.

3.4 Stellar:

Stellar is a blockchain platform designed for fast and low-cost cross-border transactions. It uses the Stellar Consensus Protocol (SCP) and supports token issuance. Stellar is popular for financial institutions and remittances due to its fast settlement times and low fees. However, its adoption is comparatively limited in comparison to other platforms.
Overview: Stellar is a blockchain platform for fast and low-cost cross-border transactions.
Key Features: It utilizes the Stellar Consensus Protocol (SCP) and offers support for token issuance.
Strengths: Ideal for financial institutions and remittances due to its fast settlement times and low fees.
Weaknesses: Limited adoption compared to some other platforms.

3.5 Ripple:

Ripple is a blockchain platform focused on real-time gross settlement and remittances. It operates on the XRP Ledger and uses the Ripple Protocol Consensus Algorithm (RPCA). Ripple's strengths include strong partnerships with financial institutions and a focus on enterprise solutions. However, its centralized nature and legal battles with regulators have been points of criticism.
Overview: Ripple is a blockchain platform designed for real-time gross settlement and remittances.
Key Features: It uses the XRP Ledger and a consensus algorithm known as the Ripple Protocol Consensus Algorithm (RPCA).
Strengths: Partnerships with major financial institutions and its focus on enterprise solutions.
Weaknesses: Centralized nature and ongoing legal battles with regulatory authorities.

3.6 Hyperledger:

Hyperledger is an open-source collaboration hosted by the Linux Foundation, aiming to advance cross-industry blockchain technologies. It consists of various blockchain frameworks like Hyperledger Fabric and Hyperledger Sawtooth. Hyperledger focuses on enterprise use cases, modularity, and permissioned blockchains.

Overview: Hyperledger is an open-source collaborative effort by the Linux Foundation to advance cross-industry blockchain technologies.
Key Features: Various blockchain frameworks, including Hyperledger Fabric and Hyperledger Sawtooth.
Strengths: Strong focus on enterprise use cases, modularity, and permissioned blockchains.
Weaknesses: Requires substantial technical expertise to deploy and maintain.

3.7 Binance:

Binance Chain is a blockchain platform developed by the cryptocurrency exchange Binance. Binance Smart Chain (BSC) supports smart contracts and interoperability with Binance Chain. Binance's integration with the exchange, high throughput, and low transaction fees are its strengths, but concerns exist regarding centralization.
Overview: Binance Chain is a blockchain platform created by the cryptocurrency exchange Binance.
Key Features: Binance Smart Chain (BSC) supports smart contracts and interoperability with Binance Chain.
Strengths: Integration with the Binance exchange, high throughput, and low transaction fees.
Weaknesses: Concerns about centralization due to the influence of the Binance exchange.

3.8 Kaleido:

Kaleido is a blockchain platform offered as a service, simplifying enterprise blockchain deployment. It supports multiple protocols and consensus mechanisms, making it easy for businesses with minimal blockchain expertise. However, its customization options are relatively limited compared to self-hosted solutions.
Overview: Kaleido is a blockchain platform offered as a service, aiming to simplify enterprise blockchain deployment.
Key Features: It supports multiple protocols and consensus mechanisms, catering to different use cases.
Strengths: Easy deployment and management for enterprises with minimal blockchain expertise.
Weaknesses: Limited customization options compared to self-hosted solutions.

3.9 Blockstream:

Blockstream is a blockchain technology company focusing on sidechains and scaling solutions. It developed the Liquid sidechain for faster asset transfers and the Lightning Network for Bitcoin. Blockstream is known for its innovation in Bitcoin scaling and its significant role in the Bitcoin ecosystem.
Overview: Blockstream is a blockchain technology company focused on developing sidechains and scaling solutions.
Key Features: It has developed the Liquid sidechain for faster asset transfers and the Lightning Network for Bitcoin.
Strengths: Innovation in Bitcoin scaling and a prominent player in the Bitcoin ecosystem.
Weaknesses: Relatively limited use cases beyond Bitcoin-related applications.

3.10 Hashgraph:

Hashgraph is a distributed ledger technology claiming high throughput and low latency. It uses a Directed Acyclic Graph (DAG) model for consensus. Hashgraph's fast and efficient consensus algorithm shows potential for various applications, but concerns have been raised about its closed-source nature and governance model.

Overview: Hashgraph is a distributed ledger technology that claims to provide high throughput and low latency.

Key Features: Its consensus mechanism is based on the Directed Acyclic Graph (DAG) model.

Strengths: Fast and efficient consensus algorithm with potential for various applications.

Weaknesses: The closed-source nature of the technology and concerns about the fairness of its governance model.

3.11 Algorand:

Algorand is a blockchain platform aiming for scalability and decentralization. It uses the Algorand Consensus Algorithm (ACA) and Pure Proof of Stake (PPoS).

Overview: Algorand is a blockchain platform aiming to provide scalability and decentralization.

Key Features: It uses the Algorand Consensus Algorithm (ACA) and Pure Proof of Stake (PPoS).

Strengths: Scalability, fast transaction finality, and strong emphasis on security.

Weaknesses: Still building its ecosystem and facing competition from established platforms.

3.12 Coinbase and GetApp:

Coinbase is a popular cryptocurrency exchange, facilitating cryptocurrency trading and storage for users.

GetApp is a marketplace for business software, helping businesses find suitable software solutions. However, it does not offer blockchain-specific services.

Overview: Coinbase is a popular cryptocurrency exchange, while GetApp is a marketplace for business software.

Key Features: Coinbase facilitates cryptocurrency trading and storage, while GetApp helps businesses find software solutions.

Strengths: Coinbase's user-friendly interface and regulatory compliance, GetApp's extensive software selection.

Weaknesses: Coinbase's reliance on centralized custody of funds and GetApp's limitations in blockchain-specific software.

4. Comparative Analysis:

In this section, we did comparative analysis of the strengths, weaknesses, and unique features of each blockchain platform, emphasizing their target industries and real-world applications.

Conclusion:

Based on the findings, we summarize the comparative advantages of each blockchain platform and offer insights into their potential future developments and challenges.

The conclusion aims to guide decision-makers in choosing the most suitable blockchain platform for their specific use cases.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

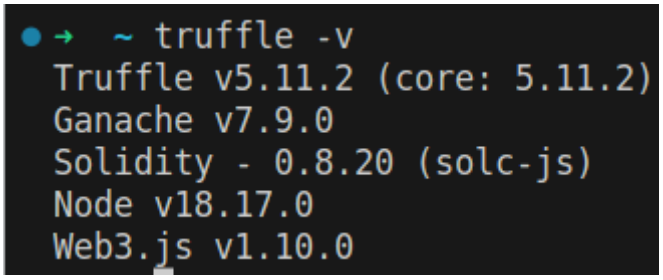
B.E / SEM VII / REV 2019 'C SCHEME' / CSE-(AI&ML)
Academic Year: 2023-24

NAME	**SINGH SUDHAM DHARMENDRA**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	
SUBJECT	**BLOCKCHAIN LAB**
COURSE CODE	**CSDOL7022**
PRACTICAL NO.	
DOP	
DOS	

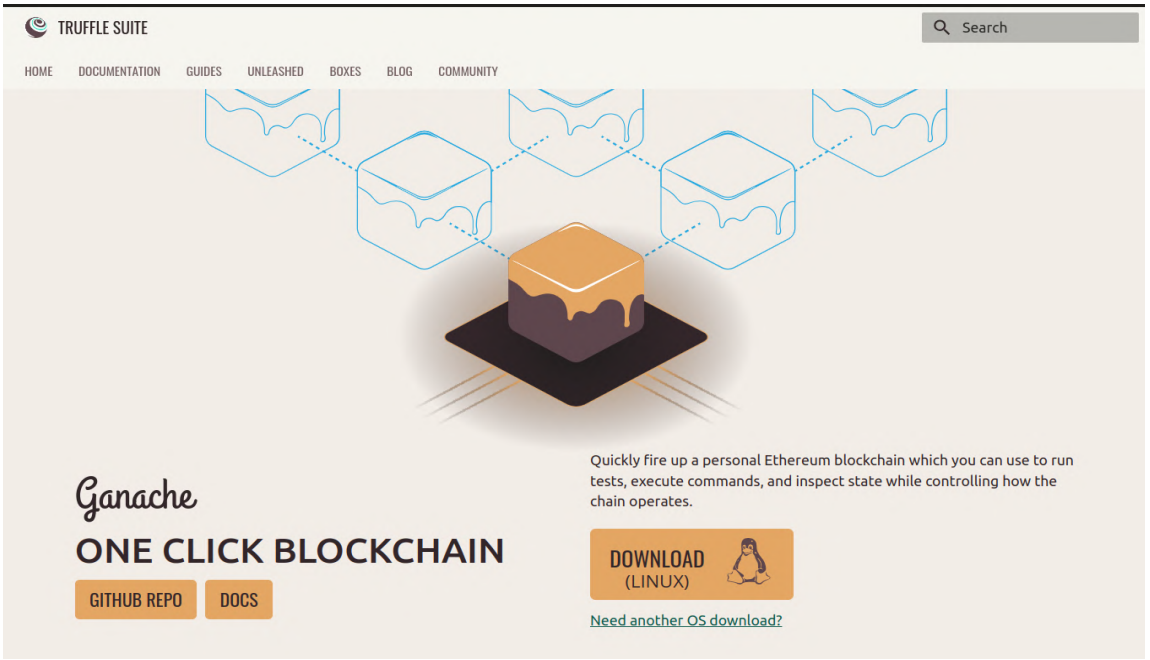
OUTPUT:-

A. CREATING SMART CONTRACTS USING SOLIDITY

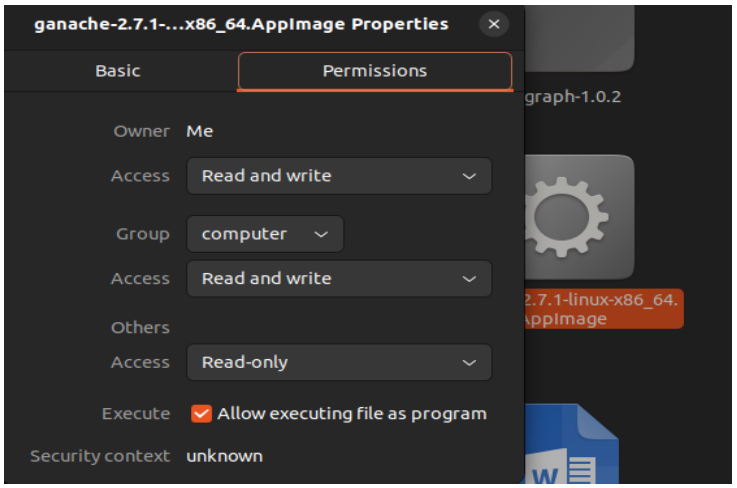
Step 1: Make sure you have truffle installed in your system



Step 2: Download Ganache - https://trufflesuite.com/ganache/

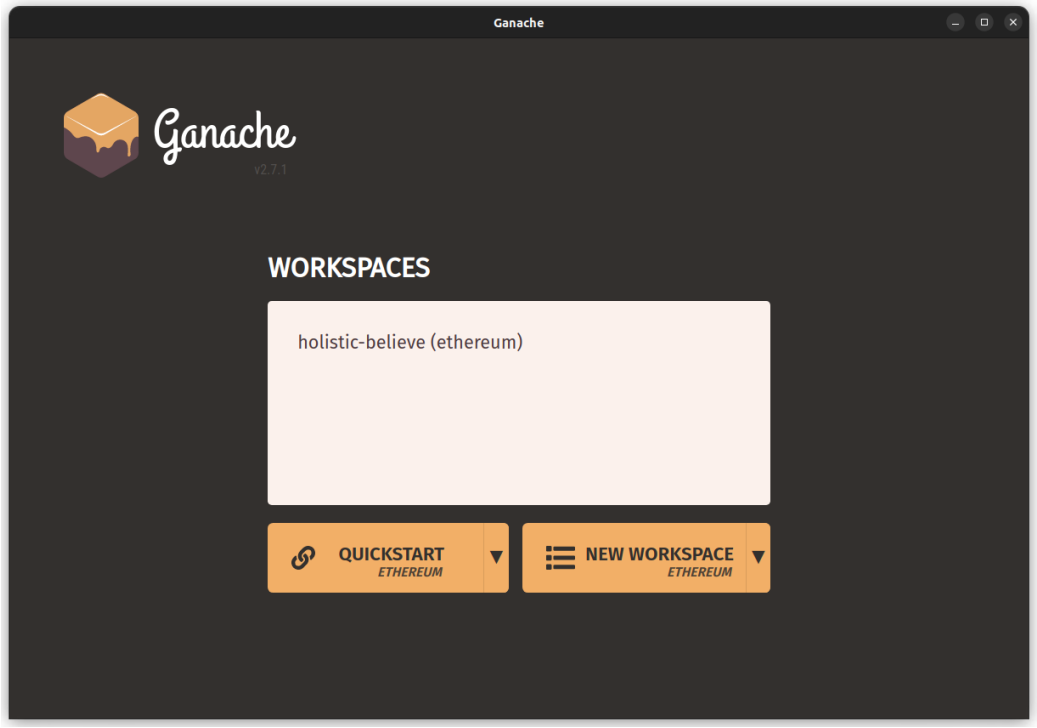


Step 3: Allow these permissions to the downloaded **ganache.x.x.AppImage** file
And make sure Restart your system.

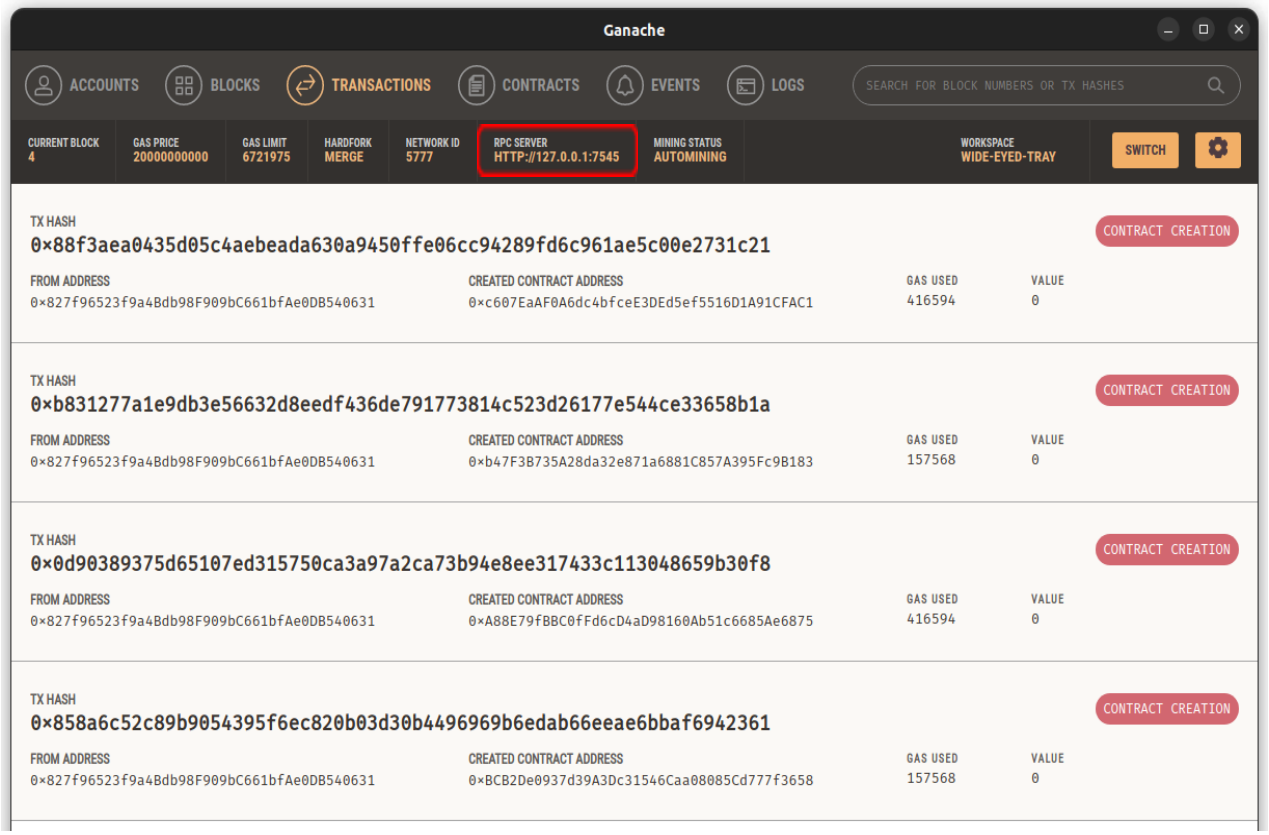


Step 4: Then click on the file to open Ganache

Step 5: Click on **NEW WORKSPACE**

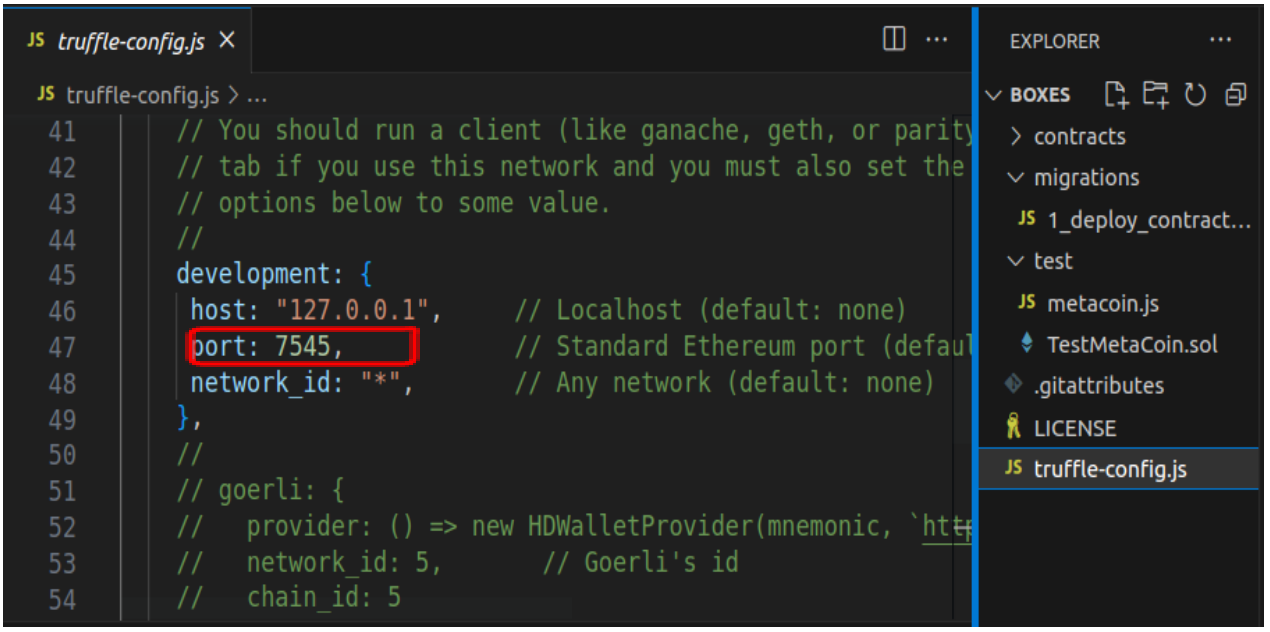


Step 6: Copy RPC SERVER's port: (**7545**)

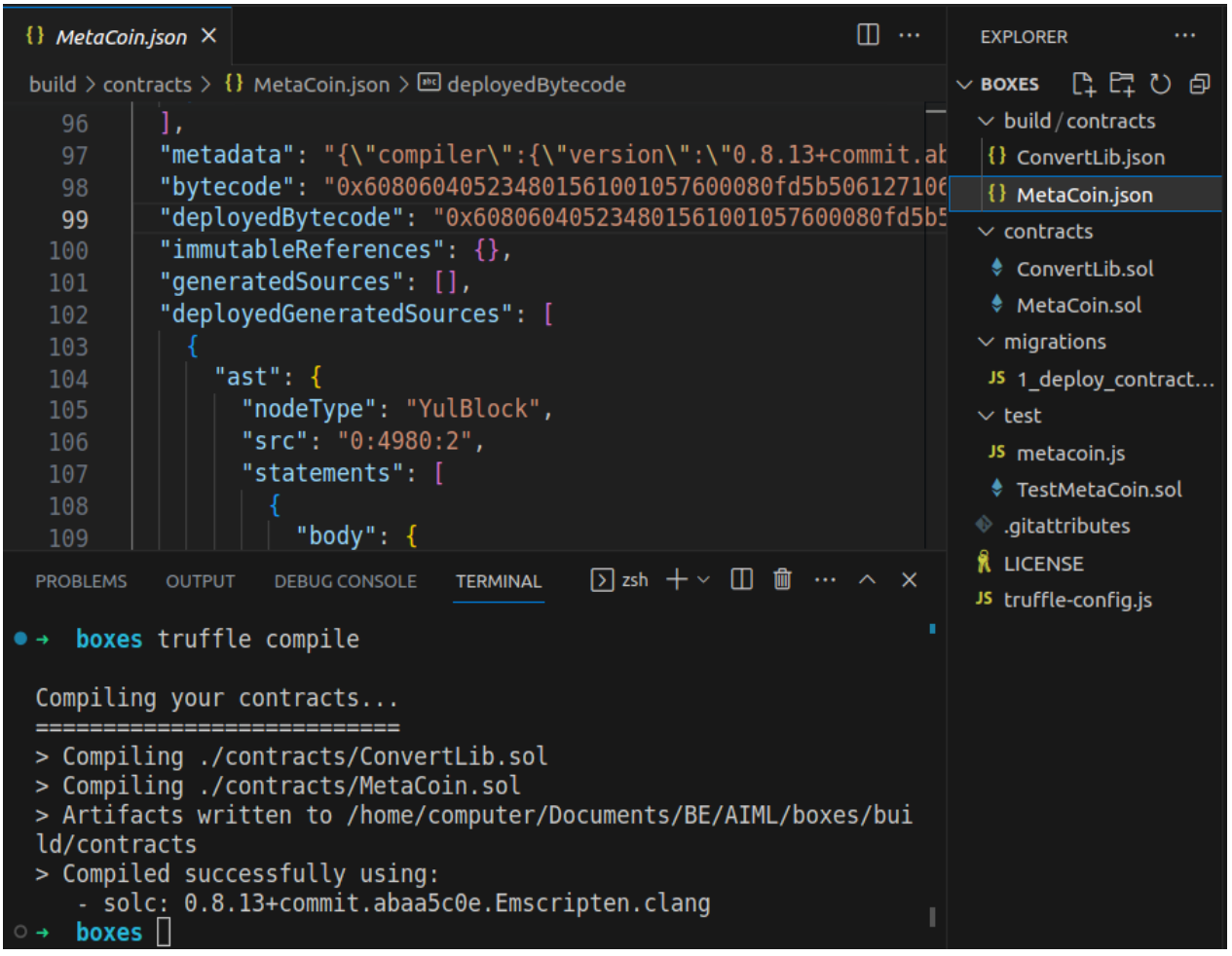


Step 7: Truffle unboxing -

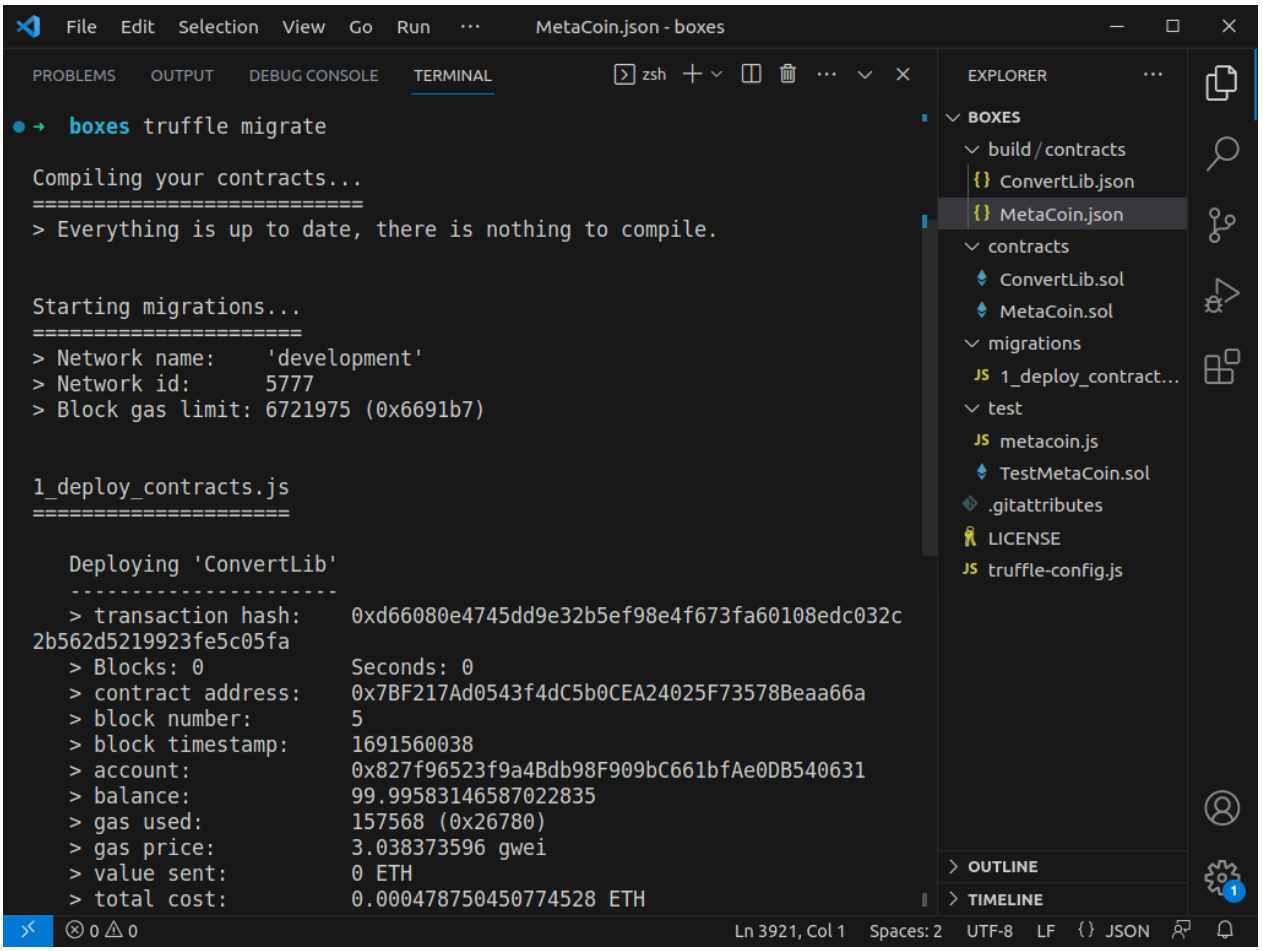
1. Now create a folder (boxes).
2. Open in terminal and run **"truffle unbox metacoin"**
   Change the port number with the previous copied ganache RPC server's port (**7545**)



3. Run the command - **"truffle compile"**

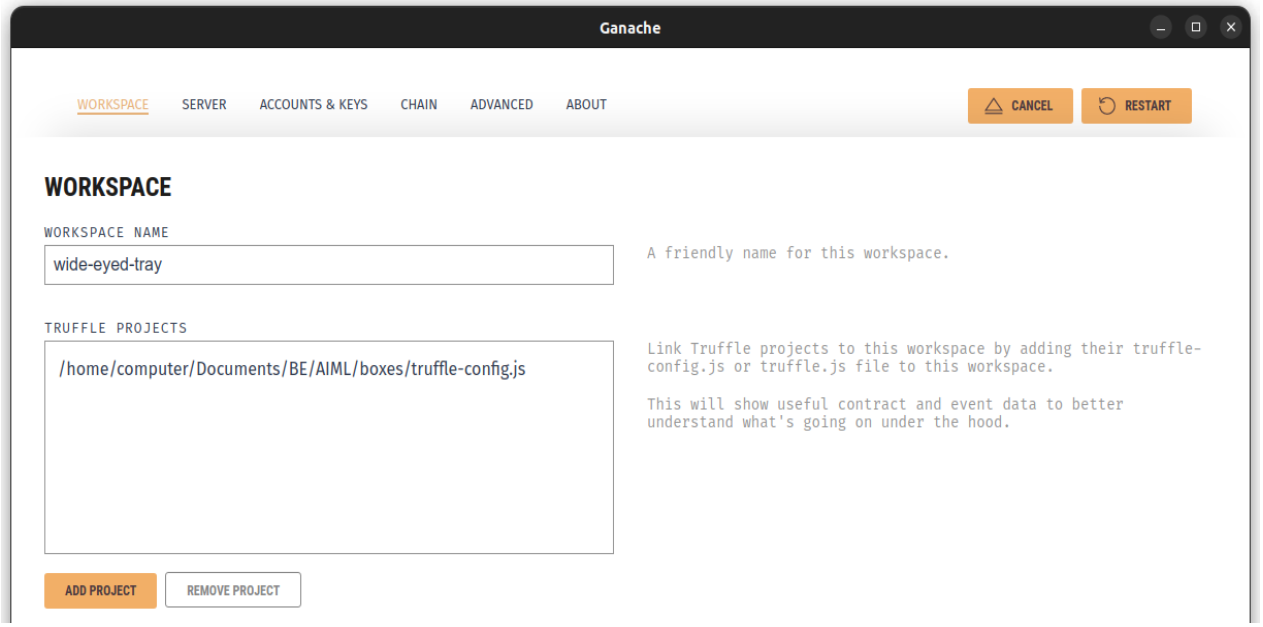


4. Run **“truffle migrate”**



5. Open Ganache and click on settings:

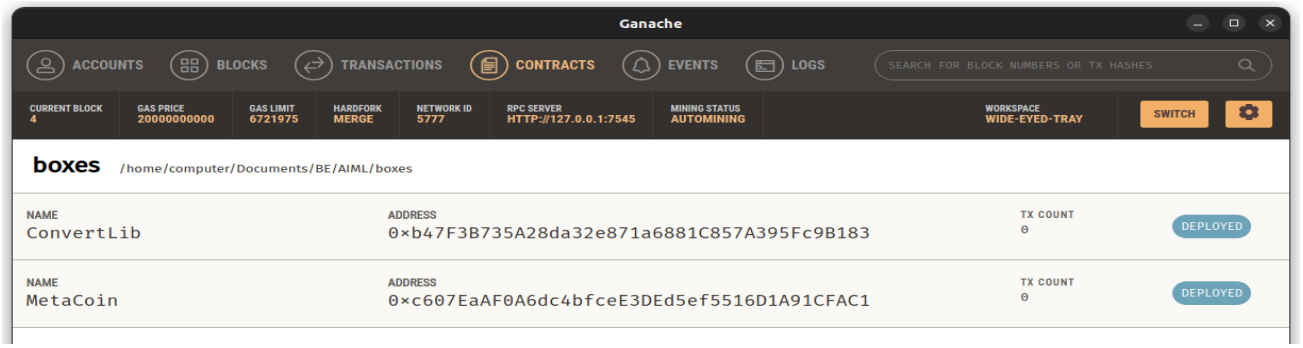
6. Add path of the **truffle-config.js** file to TRuffle Project and Click on **Add Project**



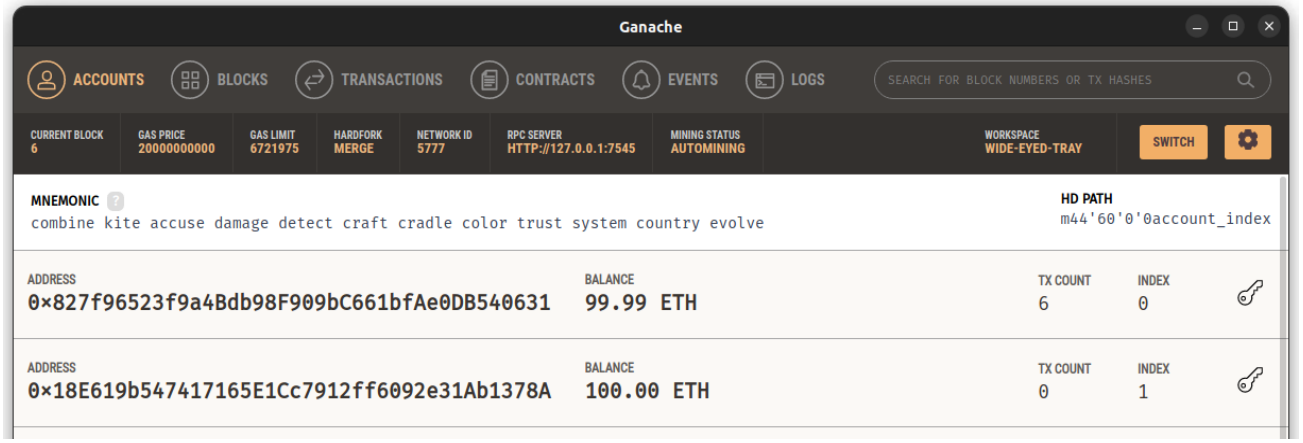
B. EMBEDDING WALLET AND TRANSACTION USING SOLIDITY

You can see the -

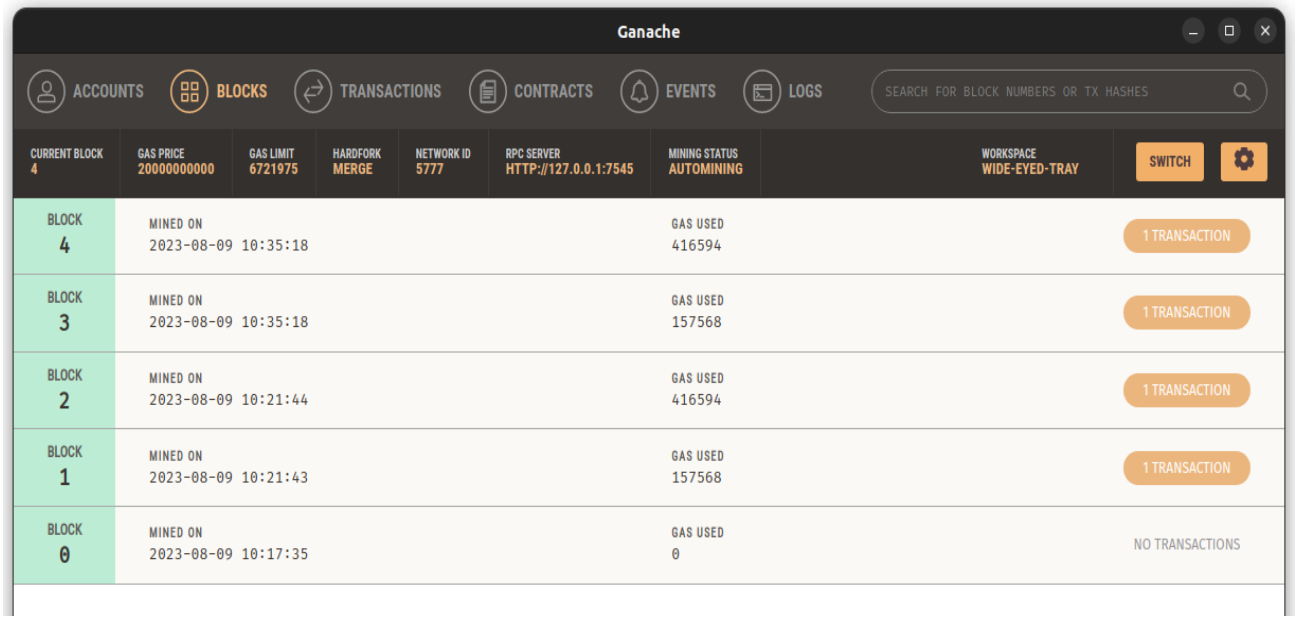
1. Contracts



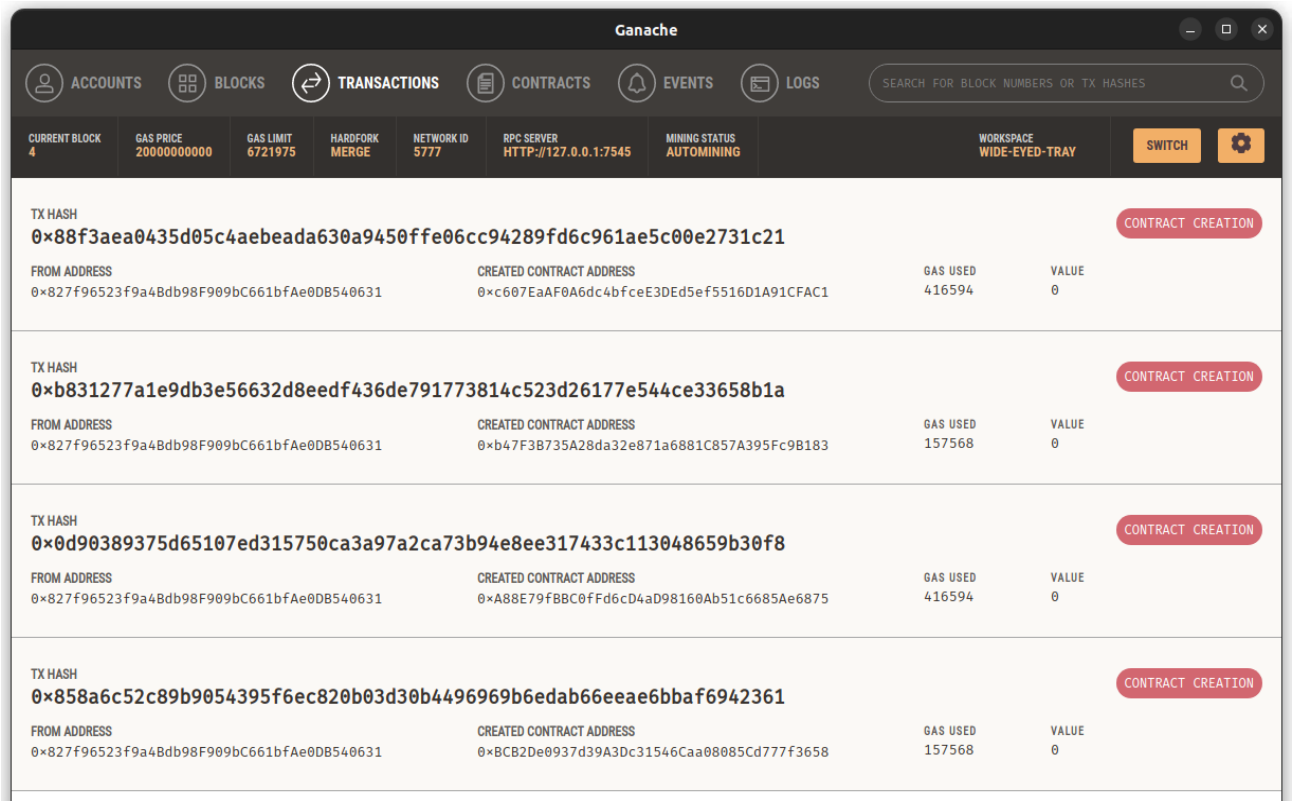
2. Accounts



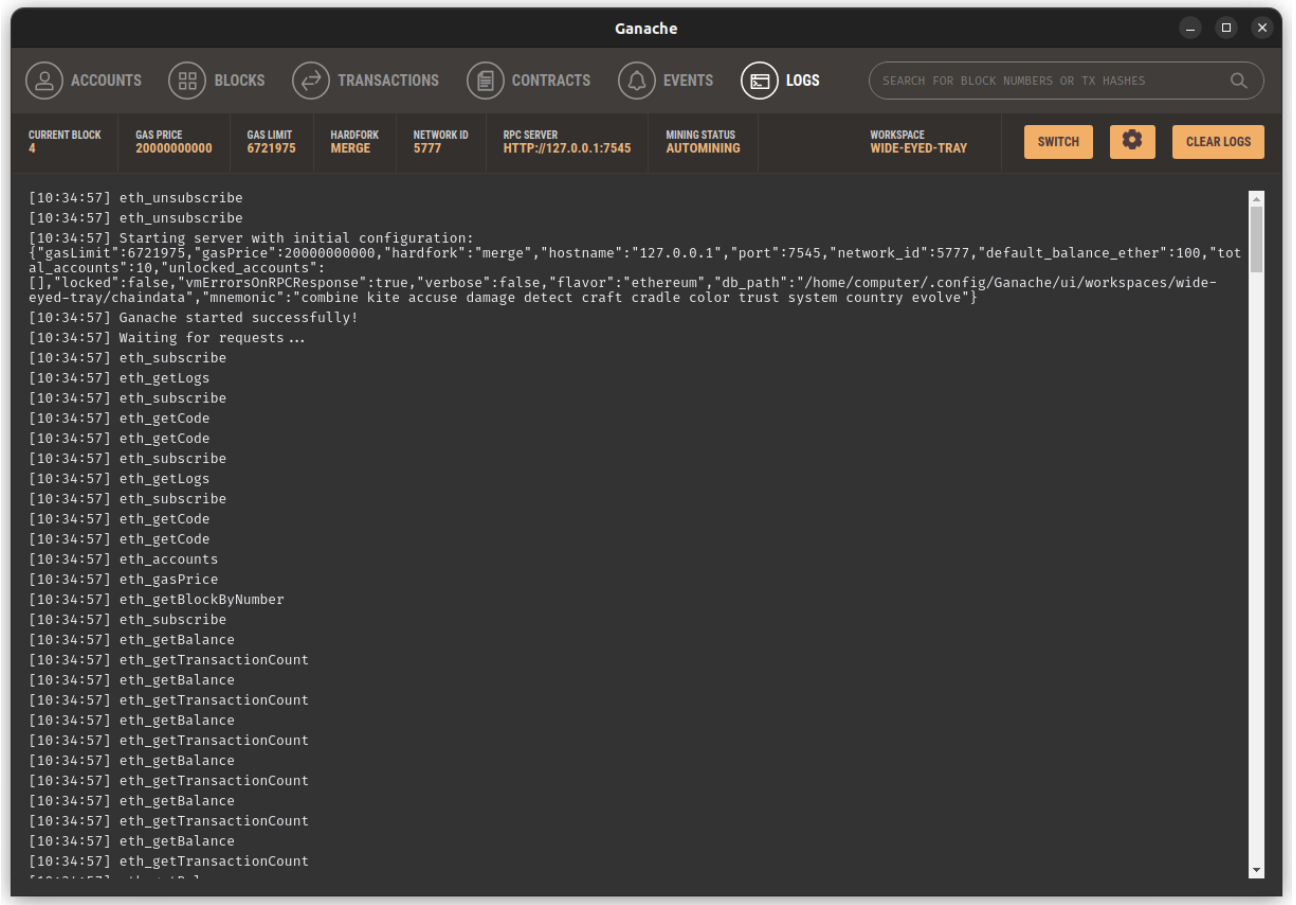
3. Blocks:



4. Transaction:



5. Logs:



Codes-

Contracts:

ConvertLib.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;
// A library is like a contract with reusable code, which can be called by other contracts.
// Deploying common code can reduce gas costs.
library ConvertLib{
	function convert(uint amount, uint conversionRate) public pure returns (uint
convertedAmount)
	{
		return amount * conversionRate;
	}
}
```

MetaCoin.sol

```
// SPDX-License-Identifier: MIT
// Tells the Solidity compiler to compile only from v0.8.13 to v0.9.0
pragma solidity ^0.8.13;

import "./ConvertLib.sol";
// This is just a simple example of a coin-like contract.
// It is not ERC20 compatible and cannot be expected to talk to other
// coin/token contracts.
contract MetaCoin {
	mapping (address => uint) balances;
	event Transfer(address indexed _from, address indexed _to, uint256 _value);
	constructor() {
		balances[tx.origin] = 10000;
	}
	function sendCoin(address receiver, uint amount) public returns(bool sufficient) {
		if (balances[msg.sender] < amount) return false;
		balances[msg.sender] -= amount;
		balances[receiver] += amount;
		emit Transfer(msg.sender, receiver, amount);
		return true;
	}
	function getBalanceInEth(address addr) public view returns(uint){
		return ConvertLib.convert(getBalance(addr),2);
	}
	function getBalance(address addr) public view returns(uint) {
		return balances[addr];
	}
}
```

Migrations:

1_deploy_contracts.js

```
const ConvertLib = artifacts.require("ConvertLib");
const MetaCoin = artifacts.require("MetaCoin");

module.exports = function(deployer) {
deployer.deploy(ConvertLib);
deployer.link(ConvertLib, MetaCoin);
deployer.deploy(MetaCoin);
};
```

Test:

metacoin.js

```
const MetaCoin = artifacts.require("MetaCoin");

contract('MetaCoin', (accounts) => {
it('should put 10000 MetaCoin in the first account', async () => {
const metaCoinInstance = await MetaCoin.deployed();
const balance = await metaCoinInstance.getBalance.call(accounts[0]);

assert.equal(balance.valueOf(), 10000, "10000 wasn't in the first account");
});
it('should call a function that depends on a linked library', async () => {
const metaCoinInstance = await MetaCoin.deployed();
const metaCoinBalance = (await metaCoinInstance.getBalance.call(accounts[0])).toNumber();
const metaCoinEthBalance = (await
metaCoinInstance.getBalanceInEth.call(accounts[0])).toNumber();

assert.equal(metaCoinEthBalance, 2 * metaCoinBalance, 'Library function returned unexpected
function, linkage may be broken');
});
it('should send coin correctly', async () => {
const metaCoinInstance = await MetaCoin.deployed();

// Setup 2 accounts.
const accountOne = accounts[0];
const accountTwo = accounts[1];
// Get initial balances of first and second account.
const accountOneStartingBalance = (await
metaCoinInstance.getBalance.call(accountOne)).toNumber();
const accountTwoStartingBalance = (await
metaCoinInstance.getBalance.call(accountTwo)).toNumber();
// Make transaction from first account to second.
```

```
const amount = 10;
await metaCoinInstance.sendCoin(accountTwo, amount, { from: accountOne });

// Get balances of first and second account after the transactions.
const accountOneEndingBalance = (await
metaCoinInstance.getBalance.call(accountOne)).toNumber();
const accountTwoEndingBalance = (await
metaCoinInstance.getBalance.call(accountTwo)).toNumber();

assert.equal(accountOneEndingBalance, accountOneStartingBalance - amount, "Amount wasn't
correctly taken from the sender");
assert.equal(accountTwoEndingBalance, accountTwoStartingBalance + amount, "Amount wasn't
correctly sent to the receiver");
});
});
```

TestMetaCoin.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

// These files are dynamically created at test time
import "truffle/Assert.sol";
import "truffle/DeployedAddresses.sol";
import "../contracts/MetaCoin.sol";

contract TestMetaCoin {
  function testInitialBalanceUsingDeployedContract() public {
    MetaCoin meta = MetaCoin(DeployedAddresses.MetaCoin());
    uint expected = 10000;
    Assert.equal(meta.getBalance(tx.origin), expected, "Owner should have 10000 MetaCoin
initially");
  }

  function testInitialBalanceWithNewMetaCoin() public {
    MetaCoin meta = new MetaCoin();
    uint expected = 10000;
    Assert.equal(meta.getBalance(tx.origin), expected, "Owner should have 10000 MetaCoin
initially");
  }
}
```

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

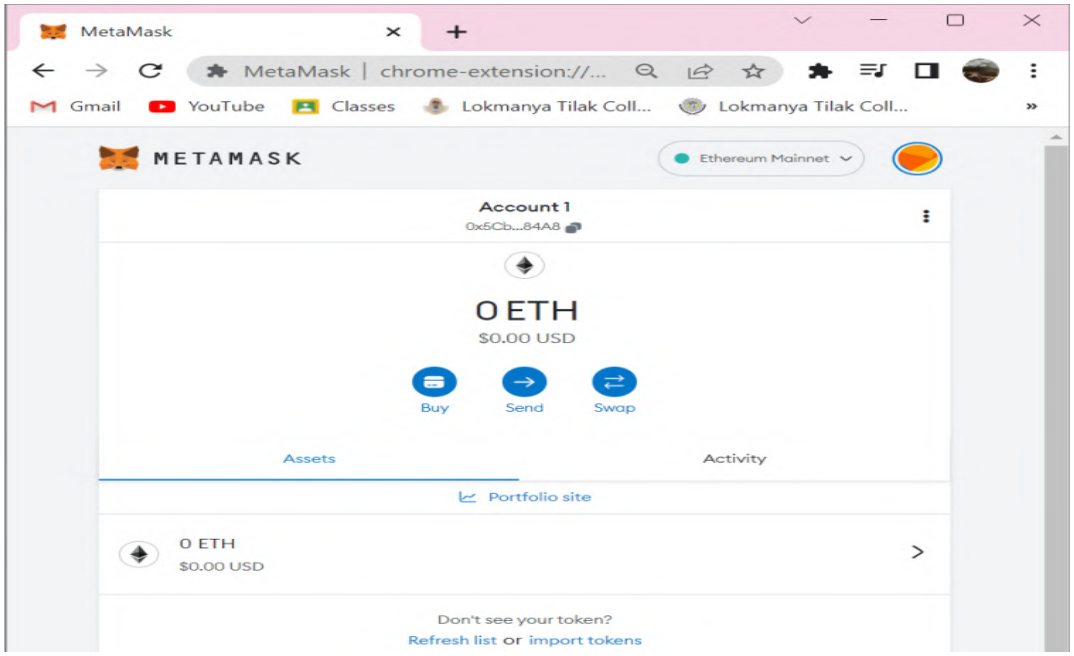
B.E / SEM VII / REV 2019 'C SCHEME' / CSE-(AI&ML)
Academic Year: 2023-24

NAME	**SINGH SUDHAM DHARMENDRA**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	
SUBJECT	**BLOCKCHAIN LAB**
COURSE CODE	**CSDOL7022**
PRACTICAL NO.	
DOP	
DOS	

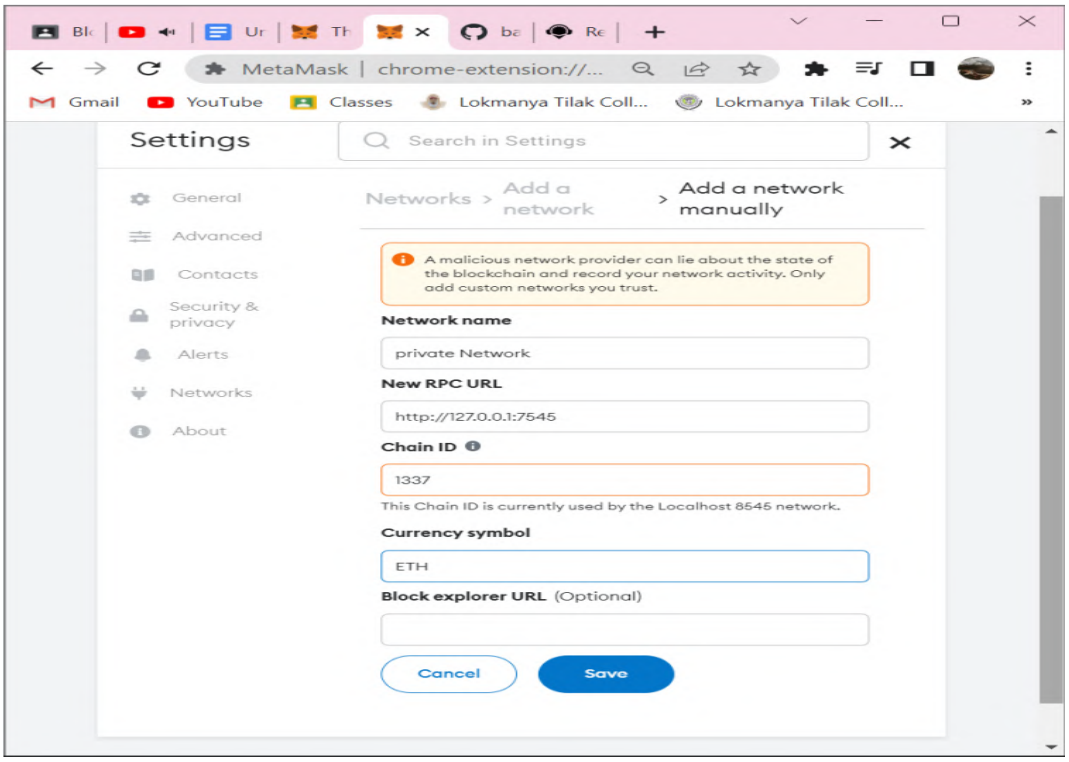
Remix IDE and Metamask :

- **Smart contract development and deployment using Metamask and Remix**
- **Design and develop Crypto currency**

Step 1: Open Chrome - go to https://metamask.io/ - add MetaMask Extension to Chrome.
Connect to Localhost 8545



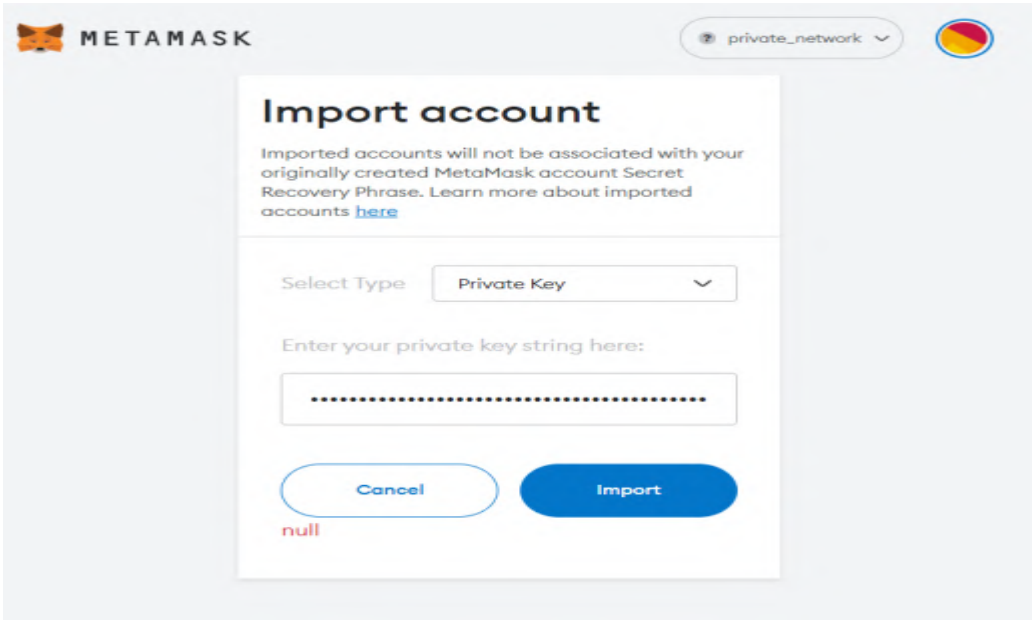
Step 2: For most networks, including Mainnet and the public testnets, the network ID and the chain ID are the same, with the network ID defaulting to the chain ID, as specified in the genesis file.
To create a private network in MetaMask:
Previously ChainID was an optional field but in the recent release of MetaMask, they made it mandatory. By default for localhost, chain ID would be 1337

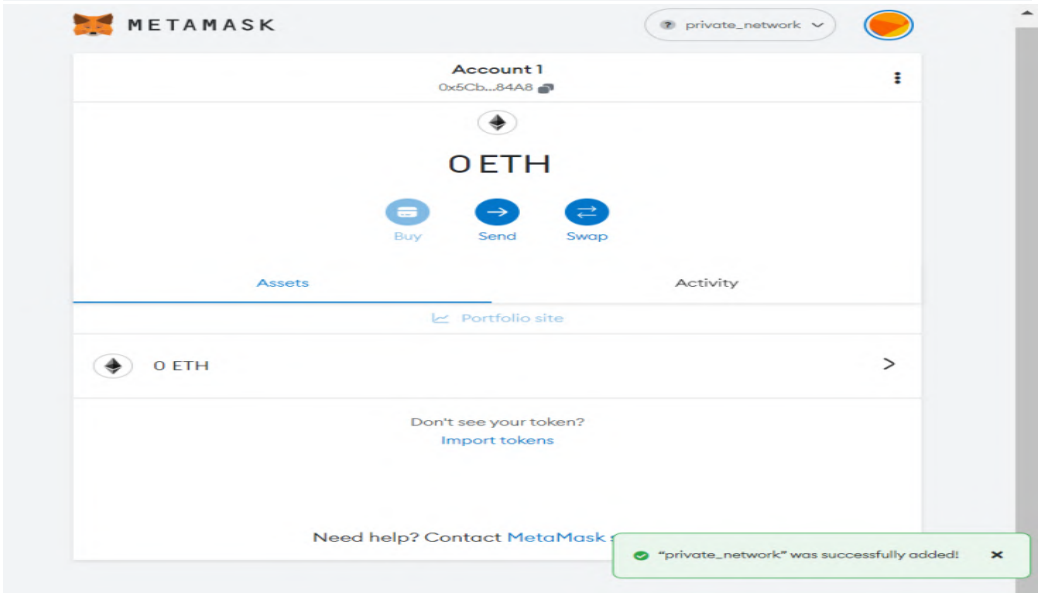


Step 3: After creating a private network;

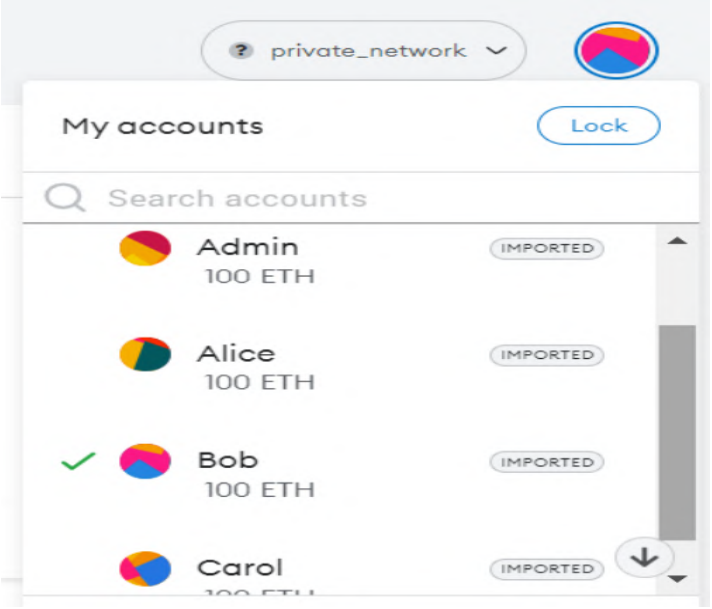
Create 4 MetaMask accounts (by importing Private Key. Get the private keys from Ganache):

- Admin - manages the ballot
- Alice - voter
- Bob - voter
- Carol - voter

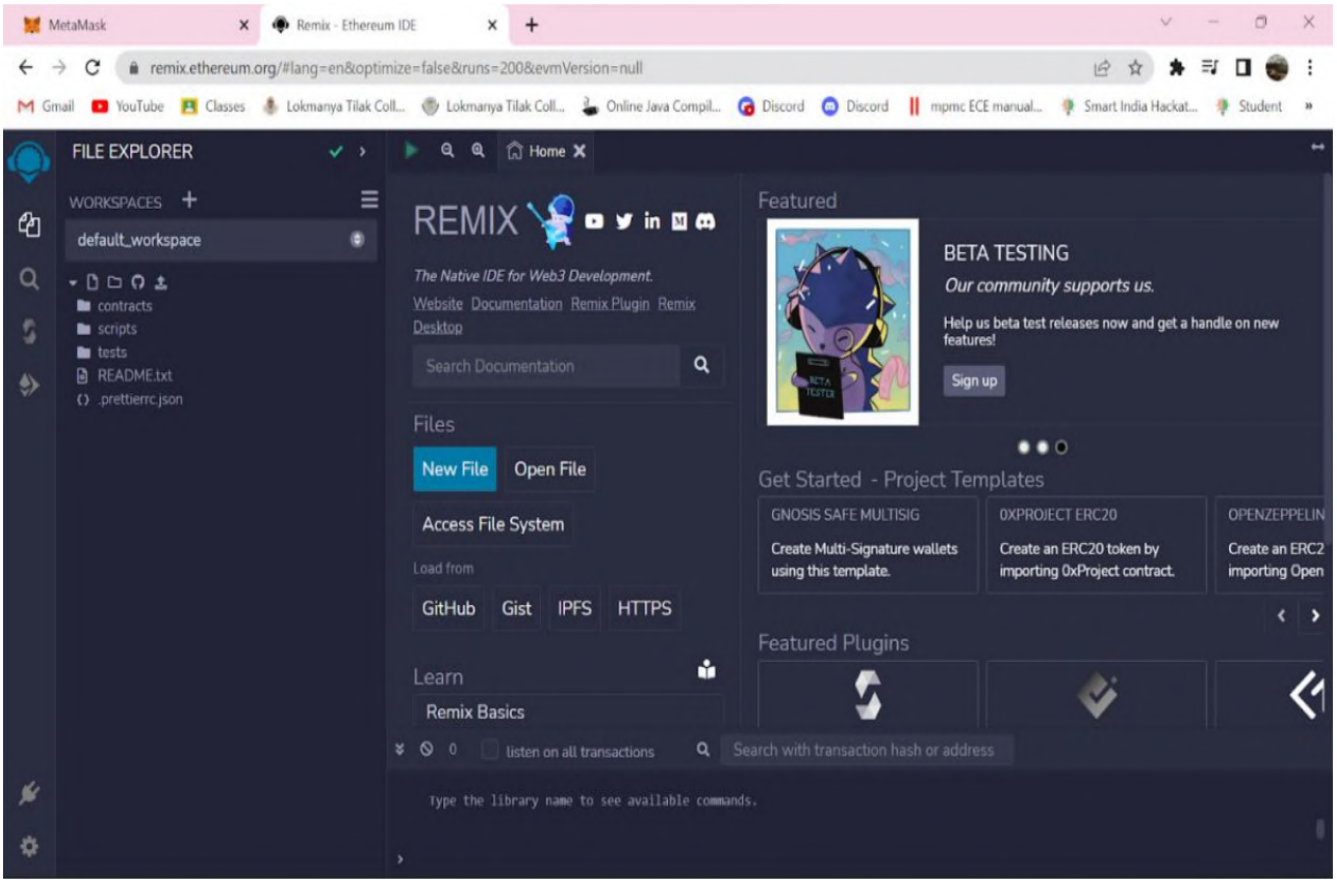




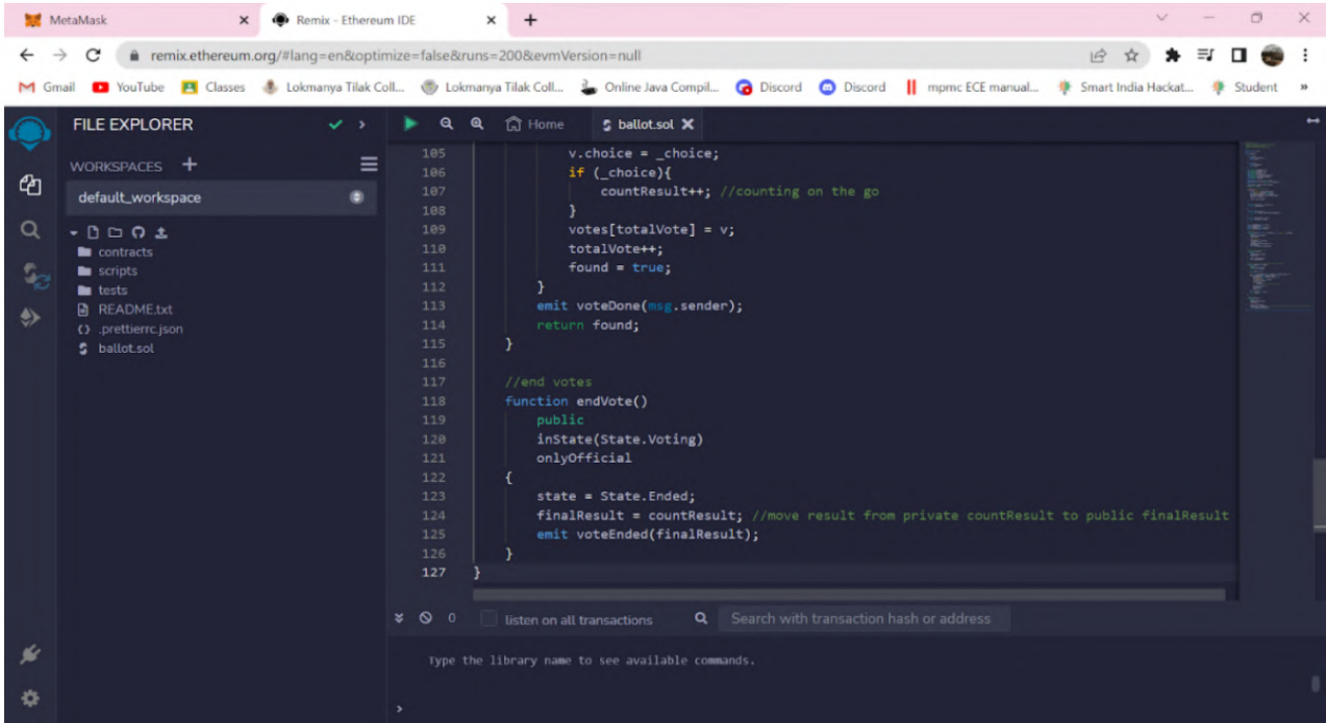
After pasting the private key click on import and MetaMask will be connected to the localhost test account to perform transactions.
Successfully, imported required accounts:
Admin - Manages the ballot
Alice - Voter
Bob - Voter
Carol - Voter



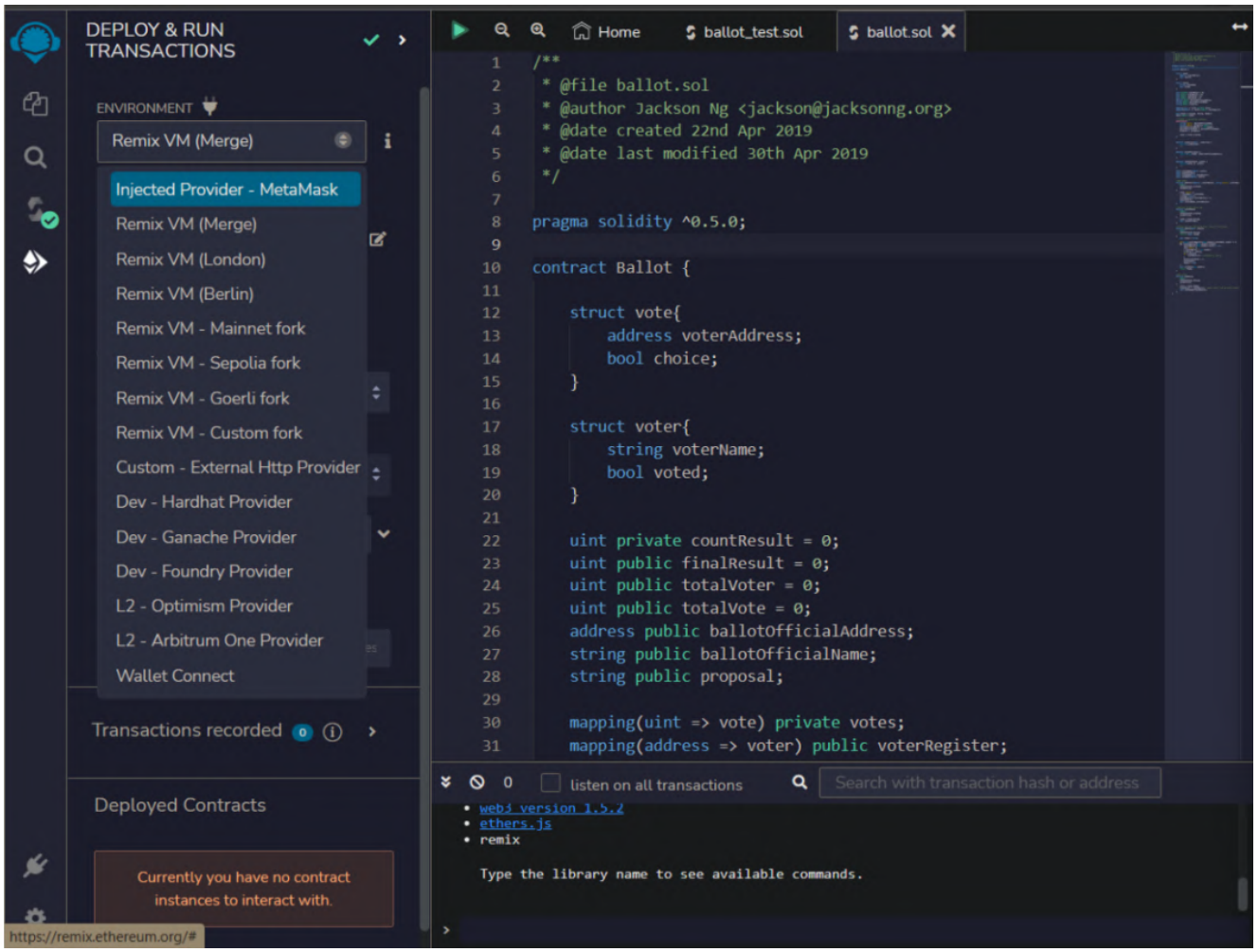
Step 4: Open Remix IDE: remix.ethereum.org

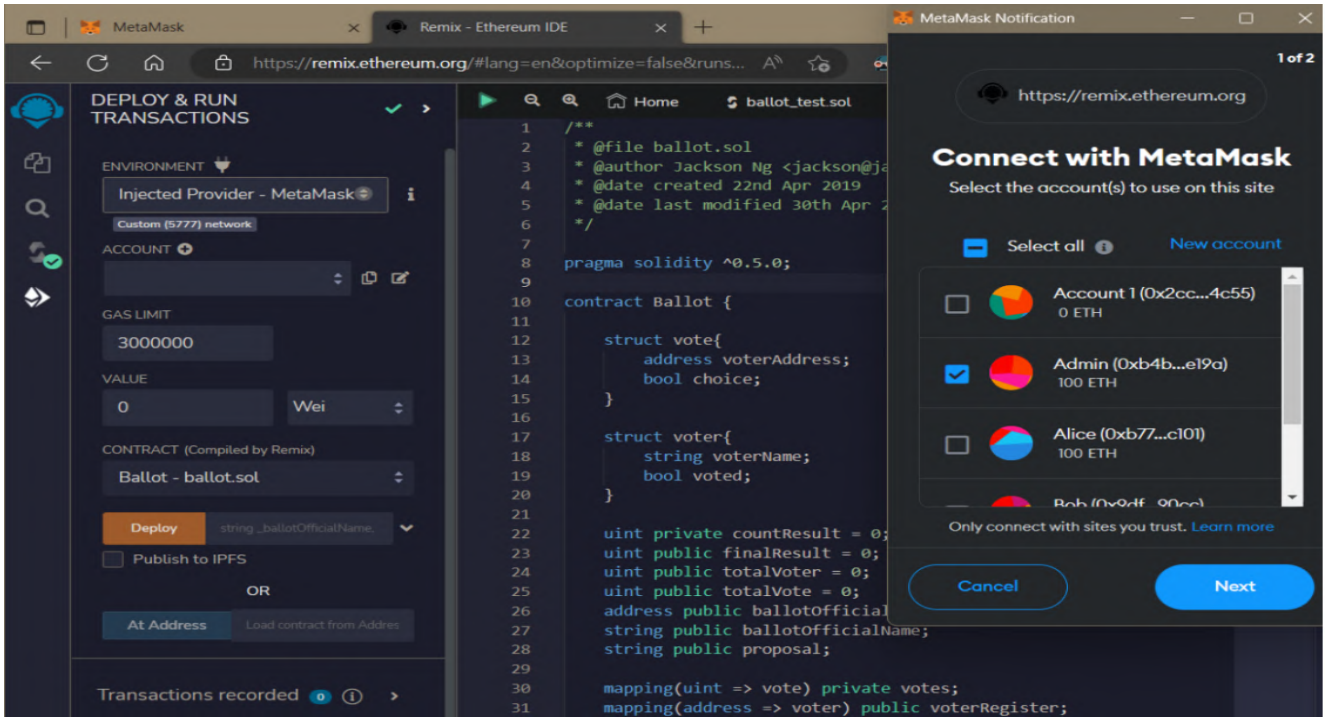


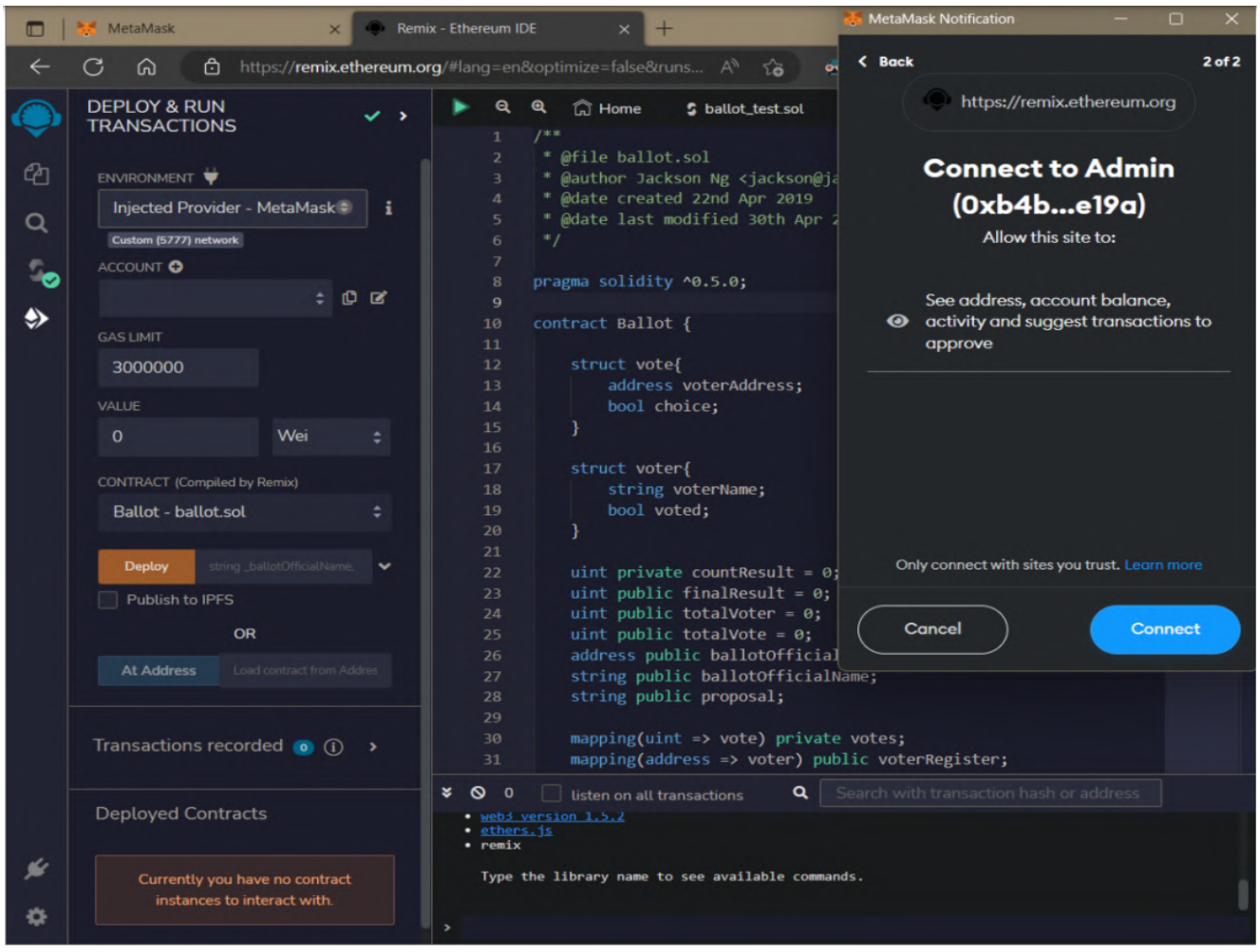
Copy and paste Ballot.sol (https://github.com/jacksonng77/ballot) to remix.ethereum.org.



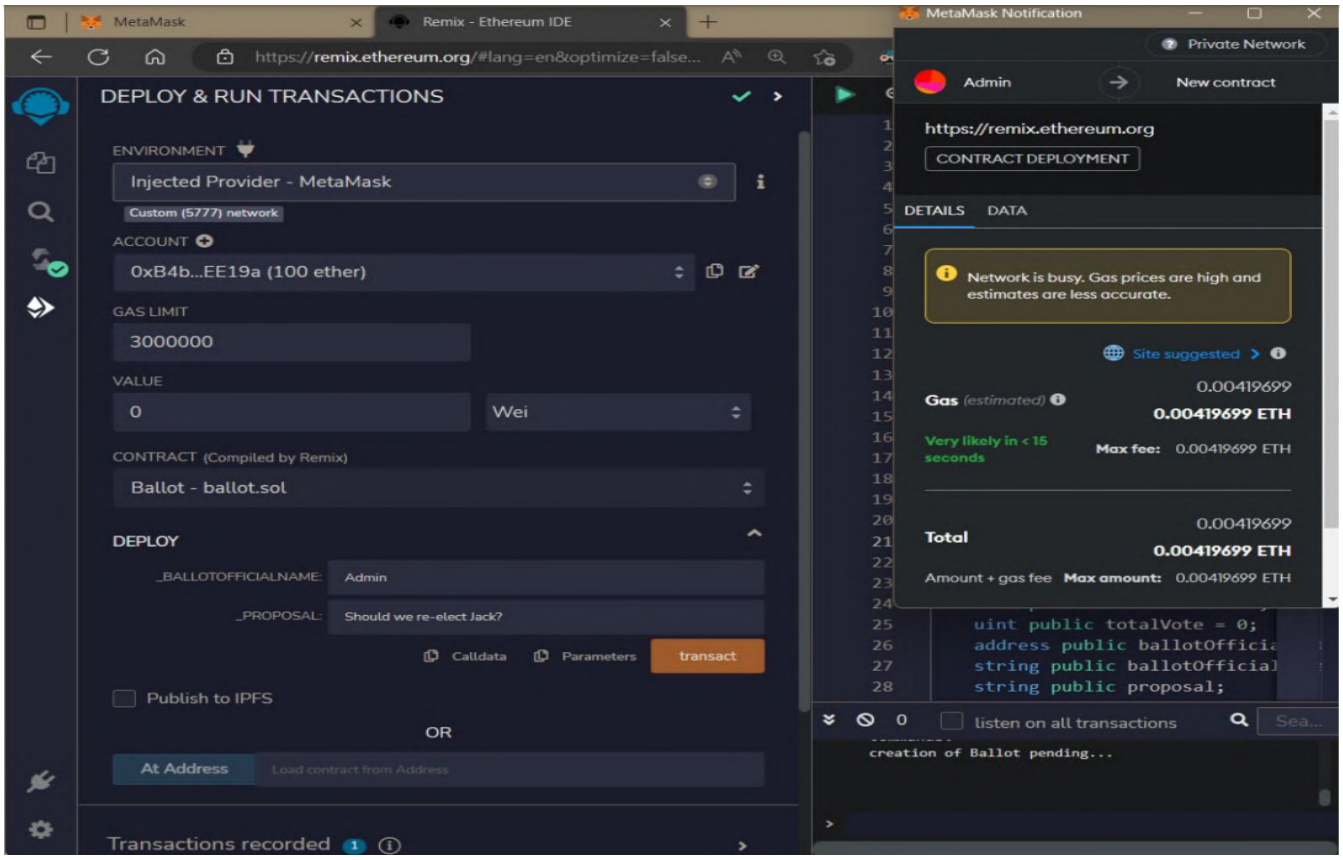
Don’t forget to switch to Admin Metamask. Compile. Set the environment to Injected Web3



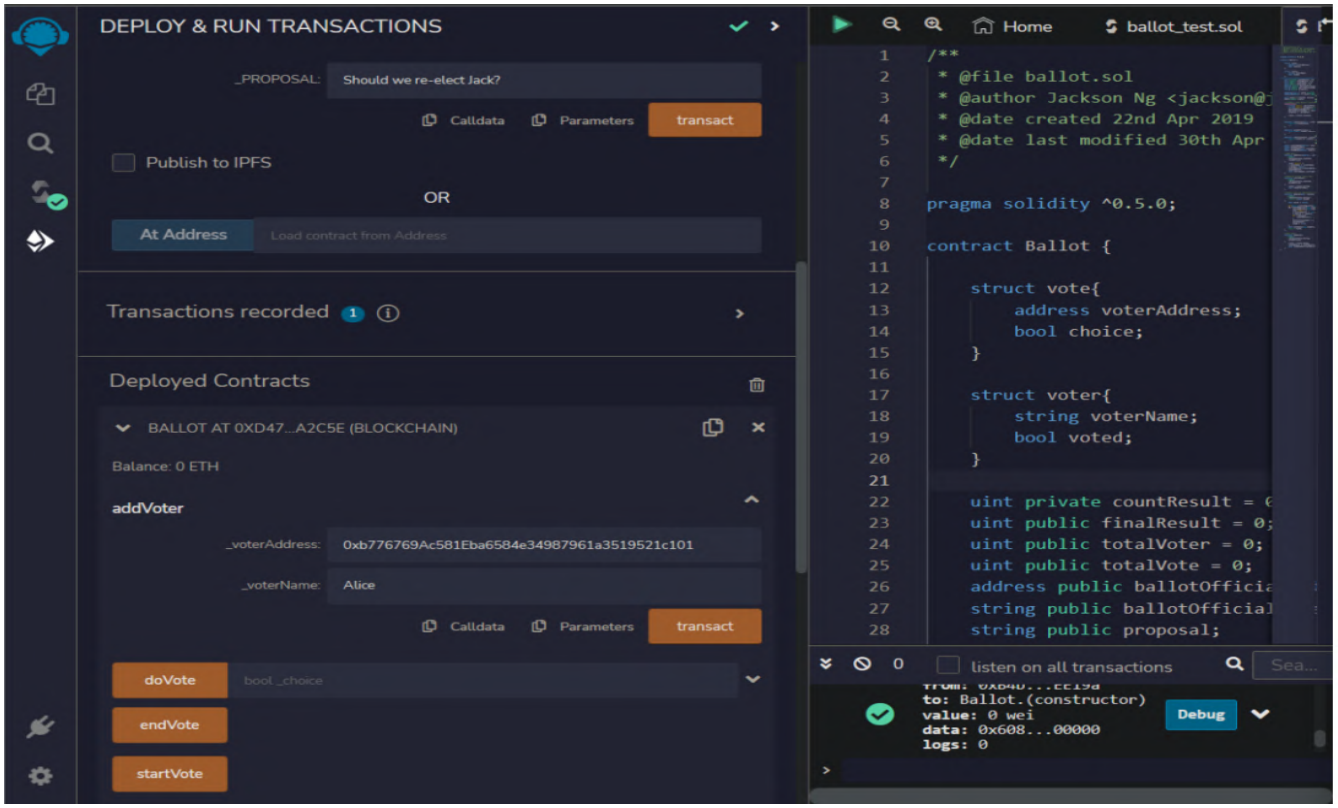

MetaMask
Remix - Ethereum IDE
https://remix.ethereum.org/#lang=en&optimize=false&runs...
DEPLOY & RUN TRANSACTIONS
ENVIRONMENT
Injected Provider - MetaMask
Custom (5777) network
ACCOUNT
GAS LIMIT
3000000
VALUE
0
Wei
CONTRACT (Compiled by Remix)
Ballot - ballot.sol
Deploy
string _ballotOfficialName
Publish to IPFS
OR
At Address
Load contract from Addres
Transactions recorded
Home
ballot_test.sol
/**
* @file ballot.sol
* @author Jackson Ng <jackson@ja
* @date created 22nd Apr 2019
* @date last modified 30th Apr
*/
pragma solidity ^0.5.0;
contract Ballot {
struct vote{
address voterAddress;
bool choice;
}
struct voter{
string voterName;
bool voted;
}
uint private countResult = 0;
uint public finalResult = 0;
uint public totalVoter = 0;
uint public totalVote = 0;
address public ballotOfficial
string public ballotOfficialName;
string public proposal;
mapping(uint => vote) private votes;
mapping(address => voter) public voterRegister;
MetaMask Notification
1 of 2
https://remix.ethereum.org
Connect with MetaMask
Select the account(s) to use on this site
Select all
New account
Account 1 (0x2cc...4c55)
0 ETH
Admin (0xb4b...e19a)
100 ETH
Alice (0xb77...c101)
100 ETH
Only connect with sites you trust. Learn more
Cancel
Next


MetaMask
Remix - Ethereum IDE
https://remix.ethereum.org/#lang=en&optimize=false&runs...
DEPLOY & RUN TRANSACTIONS
ENVIRONMENT
Injected Provider - MetaMask
Custom (5777) network
ACCOUNT
GAS LIMIT
3000000
VALUE
0
Wei
CONTRACT (Compiled by Remix)
Ballot - ballot.sol
Deploy
string _ballotOfficialName
Publish to IPFS
OR
At Address
Load contract from Addres
Transactions recorded
Deployed Contracts
Currently you have no contract instances to interact with.
Home
ballot_test.sol
/**
* @file ballot.sol
* @author Jackson Ng <jackson@ja
* @date created 22nd Apr 2019
* @date last modified 30th Apr
*/
pragma solidity ^0.5.0;
contract Ballot {
struct vote{
address voterAddress;
bool choice;
}
struct voter{
string voterName;
bool voted;
}
uint private countResult = 0;
uint public finalResult = 0;
uint public totalVoter = 0;
uint public totalVote = 0;
address public ballotOfficial
string public ballotOfficialName;
string public proposal;
mapping(uint => vote) private votes;
mapping(address => voter) public voterRegister;
0
listen on all transactions
Search with transaction hash or address
web3 version 1.5.2
ethers.js
remix
Type the library name to see available commands.
MetaMask Notification
Back
2 of 2
https://remix.ethereum.org
Connect to Admin (0xb4b...e19a)
Allow this site to:
See address, account balance, activity and suggest transactions to approve
Only connect with sites you trust. Learn more
Cancel
Connect

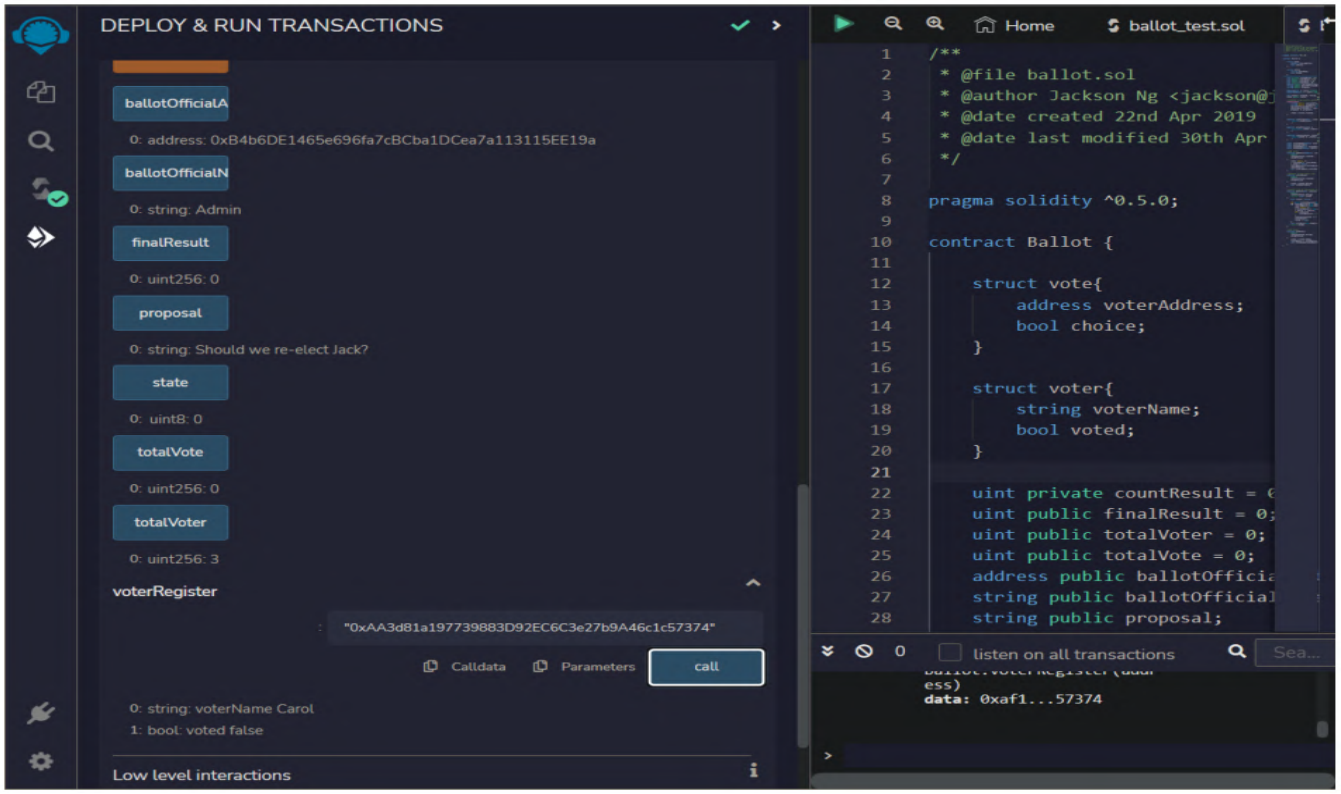
In **Remix**, switch to the Admin's **MetaMask** wallet account to deploy the Ballot contract.

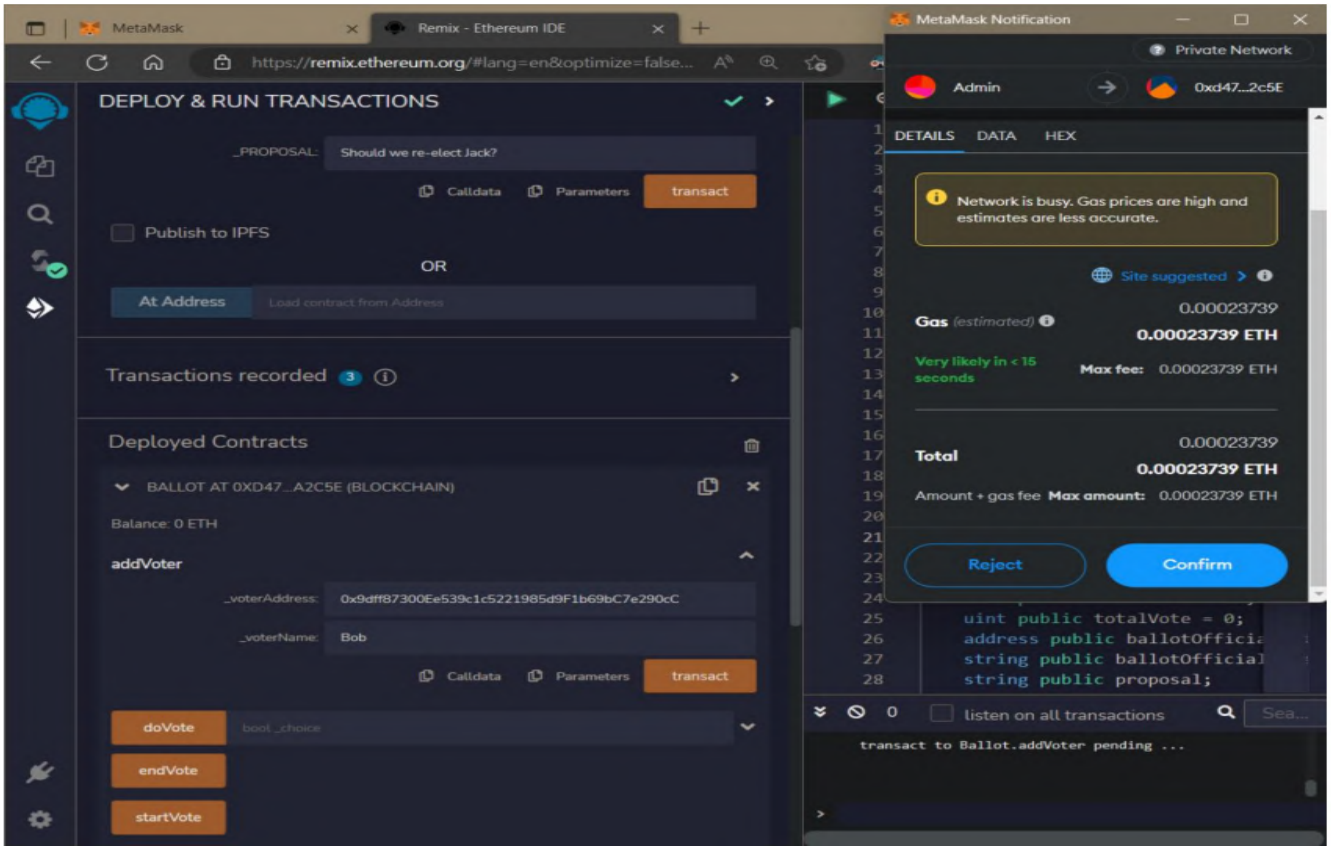


The Admin sets up a proposal in which he will like his group of voters to vote, for example "Should we re-elect Jack?".



In Remix, switch to the Admin's wallet address. Then add the wallet addresses and names of Alice, Bob and Carol using the addVoter function. At any time you can click on the respective buttons below to read the ballot contract's public variables which include information about who the Admin is, what the proposal says, the current state of the contract and the names of people in the voters register and whether or not they have voted. This ensures transparency



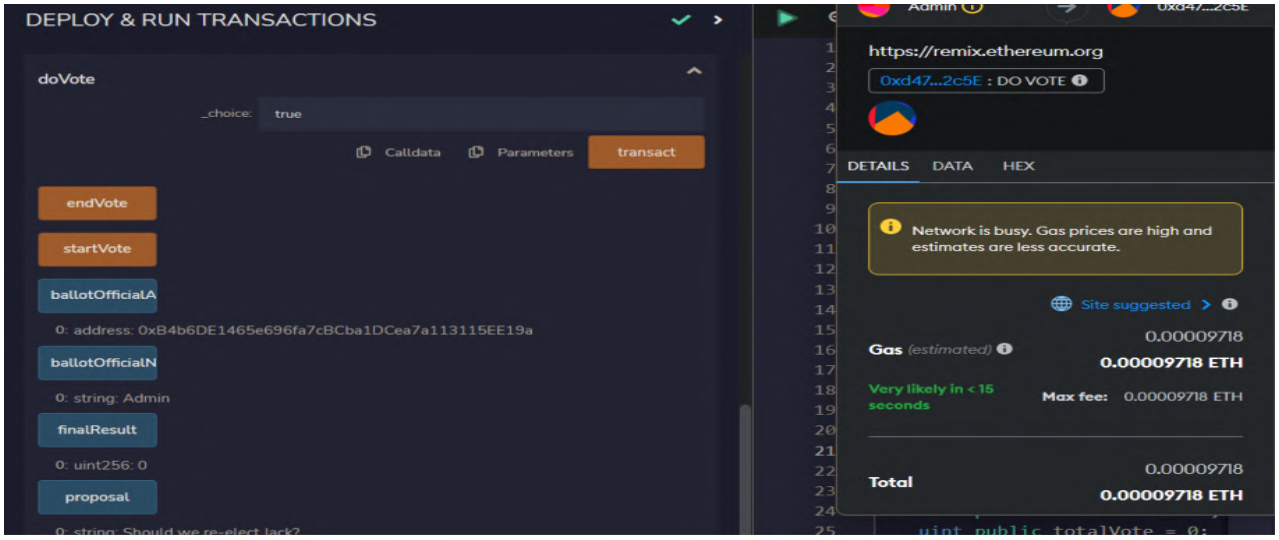


The Admin triggers voting to start. At this point, no new voters can be added and the proposal is cast in stone. The Admin sends the ballot contract address to eligible votes.

In Remix, make sure that you are still using the Admin's wallet address. Press [startVote] to let voting begin (the contract will not allow anyone else other than the person who created it to startVote).

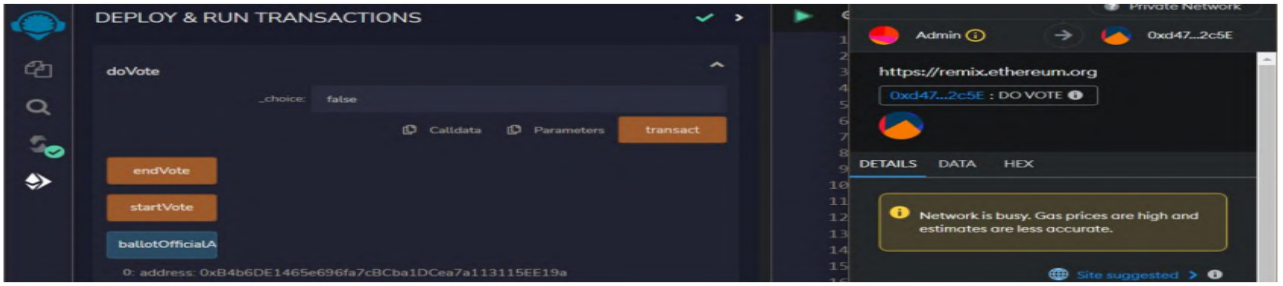
Alice attempts to vote. The contract first compares Alice's MetaMask wallet address to her record in the voters' array. If Alice's contract address is not found in the array, she's deemed an ineligible voter and her vote will not be accepted. If Alice's wallet address is found in the voters' array, the contract checks the vote array to confirm if she has voted previously. If she has, the ballot contract will refuse to accept her vote. If she hasn't, her vote is accepted, and her status is updated to say that she has voted. Once she has voted, the contract will no longer allow Alice to vote again. To vote yes, type "true" at "doVote" and click [transact]. Note that the state of the contract now says "1", which means that voting is now ongoing.

Alice Votes

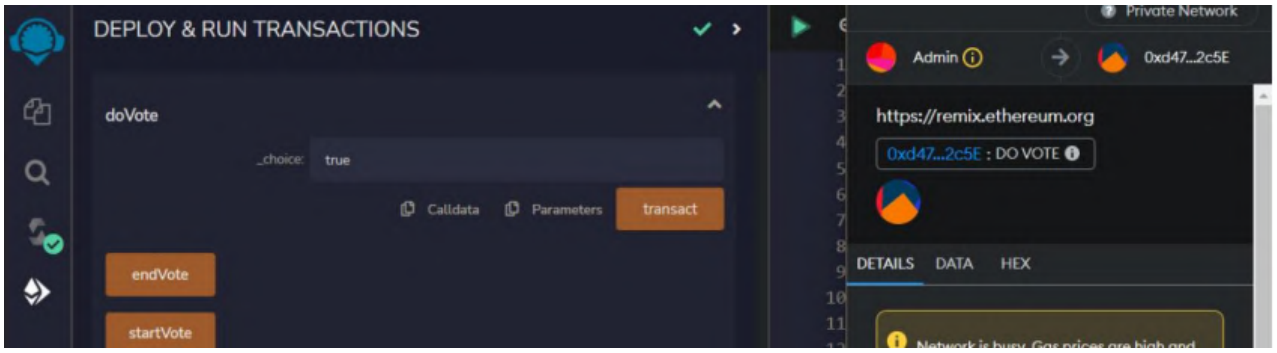


As we can notice, the State count has changed to '1'

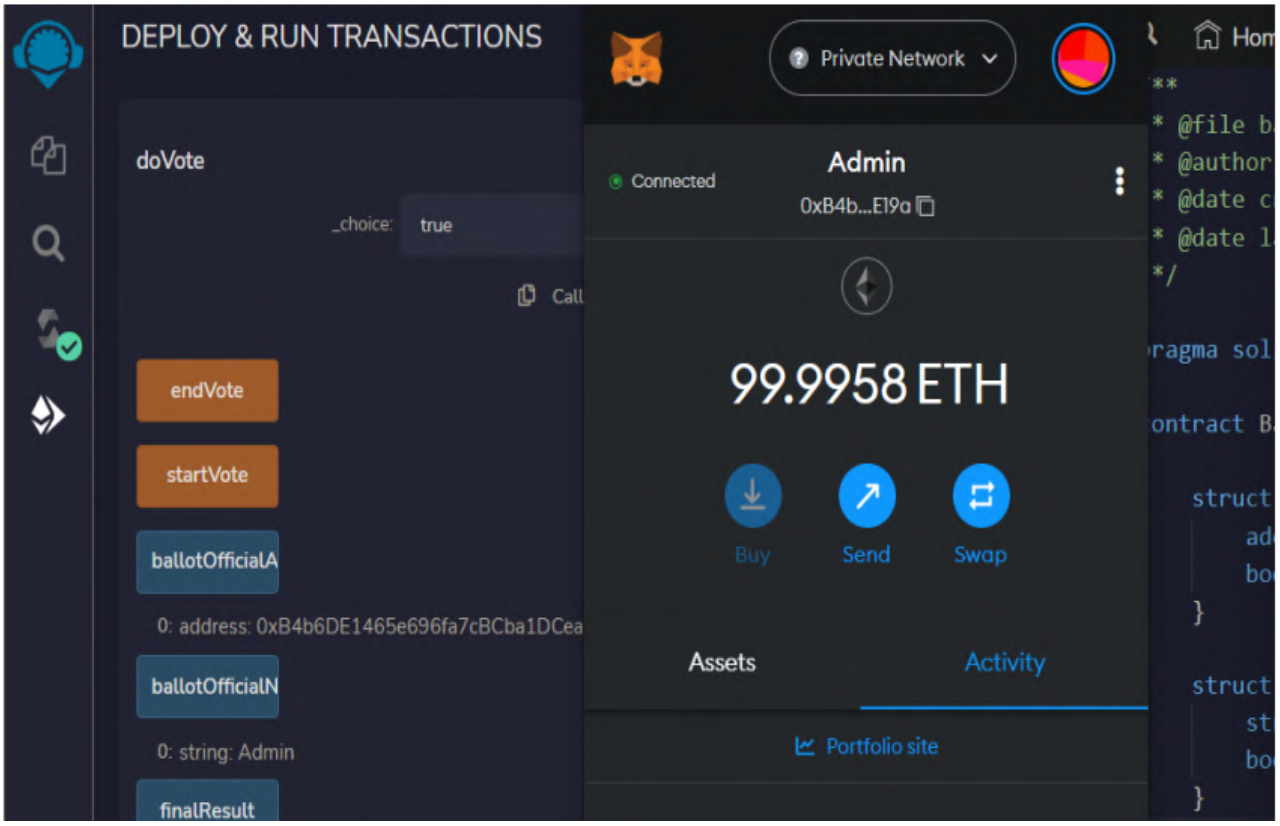
Bob Votes



Carol Votes



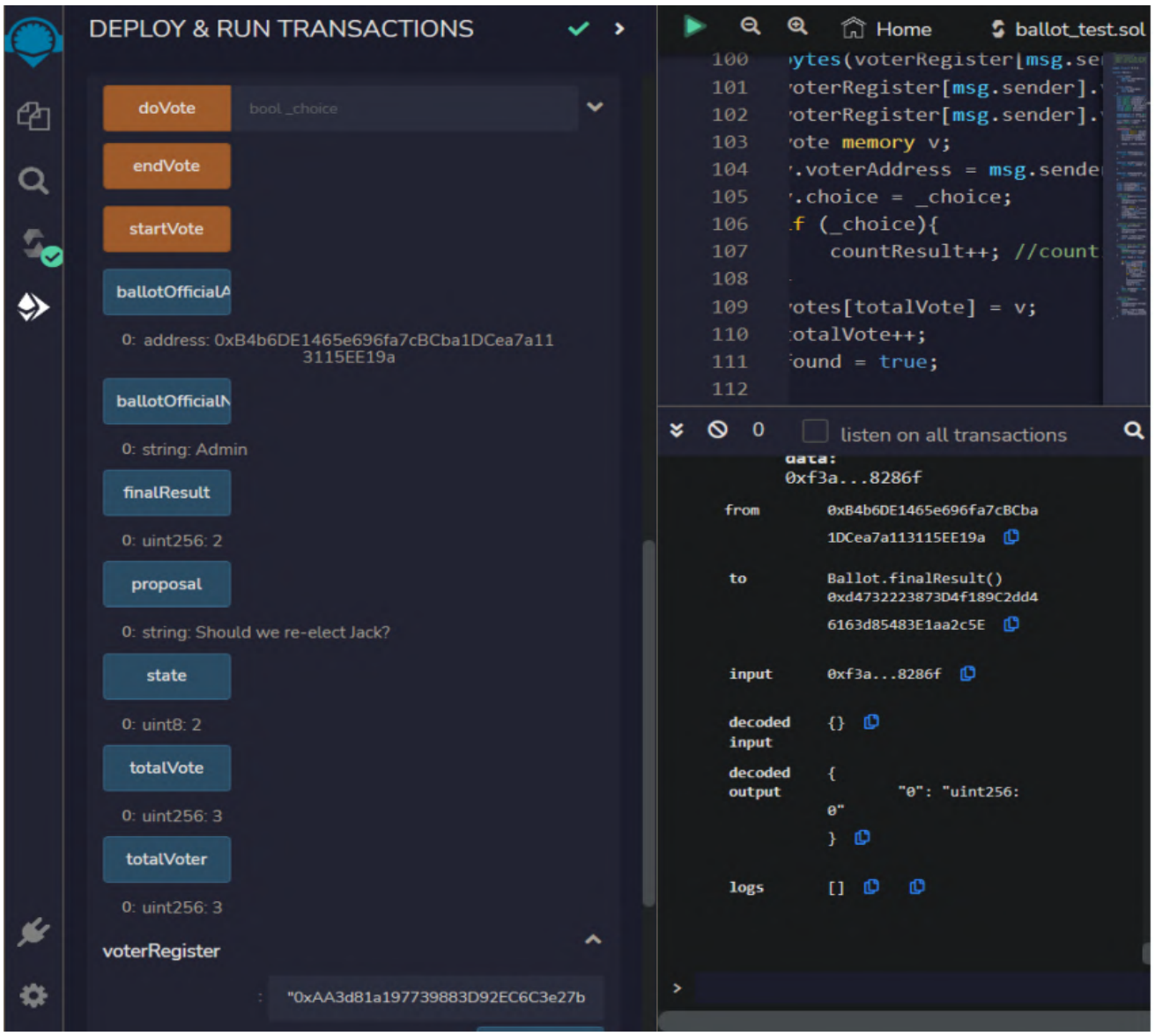
Admin Ends Vote



Closing the ballot is a significant step because it stops allowing anyone who hasn't voted to vote. Counting of votes is not possible unless the ballot closes. The Ballot contract now counts all the votes it has received and records the total number of "yes" votes.

Once this is done, everyone can view the result. The balloting process has ended and no one, not even the chairman is allowed to change any properties of the ballot contract.

To do so in Remix, switch back to the Admin's wallet account and press [endVote]. The contract will not allow anyone other than the contract creator (which is the Admin) to execute this step. Once counting is done, results are released and the contract's state becomes 2 which means that voting has ended.



The final result = 2 means there are two people who agreed to reelect Jack (Alice and Carol, who voted ‘true’). The state = 2 means the ballot is over. Total vote = 3

Note: For every transaction done, we will be charged the gas fee and there will be notification from MetaMask.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

B.E / SEM VII / REV 2019 'C SCHEME' / CSE-(AI&ML)
Academic Year: 2023-24

NAME	**SINGH SUDHAM DHARMENDRA**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	
SUBJECT	**BLOCKCHAIN LAB**
COURSE CODE	**CSDOL7022**
PRACTICAL NO.	
DOP	
DOS	

Hyperledger Fabric :

- **Chaincode Deployment**
  - ➔ **Install cURL.**
  - ➔ **Install Go Language.**
  - ➔ **Install Docker and Docker Compose.**
  - ➔ **Install Git.**
  - ➔ **Install Pip**
  - ➔ **Install Hyperledger Fabric on Ubuntu LTS 16.0.4**

Once we are done with installing prerequisite, we will be going to hands on with:
1. Build & Deploy your first-network on Hyperledger Fabric
2. Build & Deploy a demo example Fabcar on Hyperledger Fabric

We are using Ubuntu LTS 16.04 for this hands on tutorial:

Step 1: Press Ctrl + Alt + T to open a terminal

$ **sudo su**

Enter your password

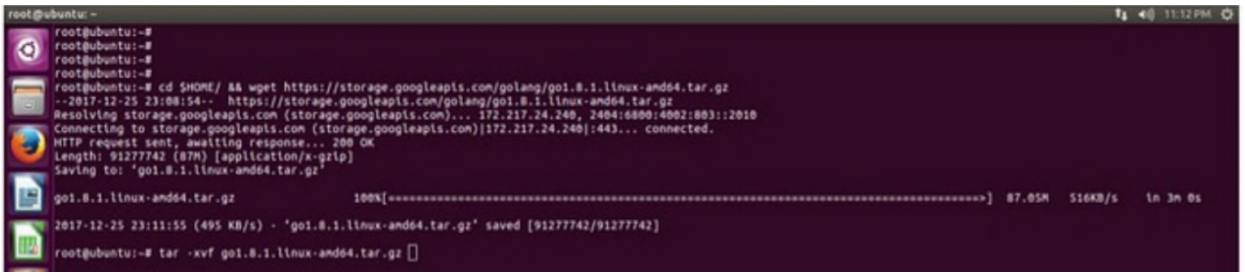
$ **cd**

Step 2: Install Google golang

$ **cd $HOME/ &&**

wget https://storage.googleapis.com/golang/go1.8.1.linux-amd64.tar.gz

$ **tar -xvf go1.8.1.linux-amd64.tar.gz**



Set the go path

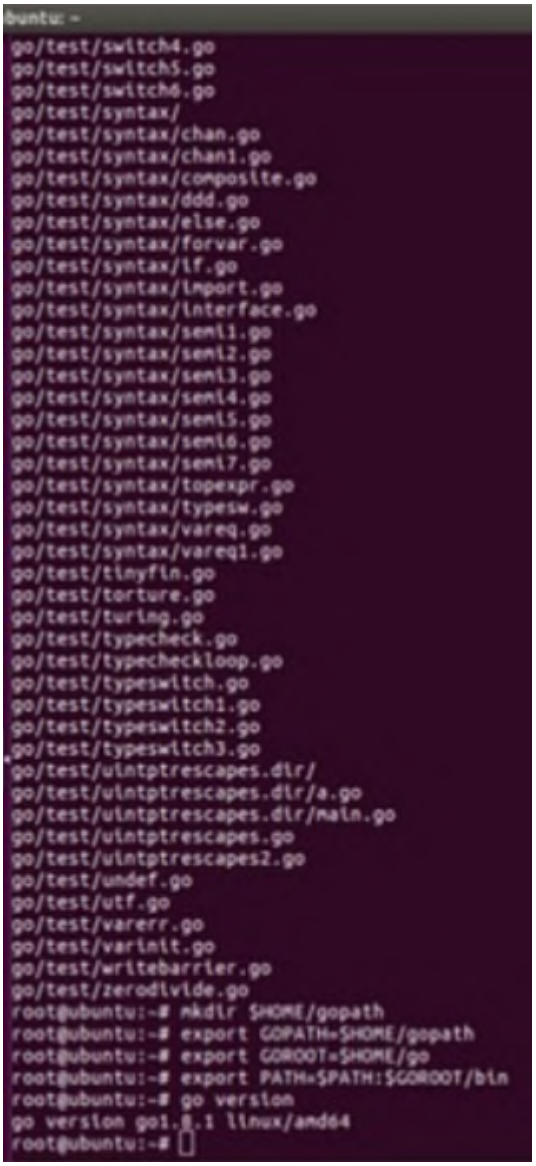
$ **mkdir $HOME/gopath**

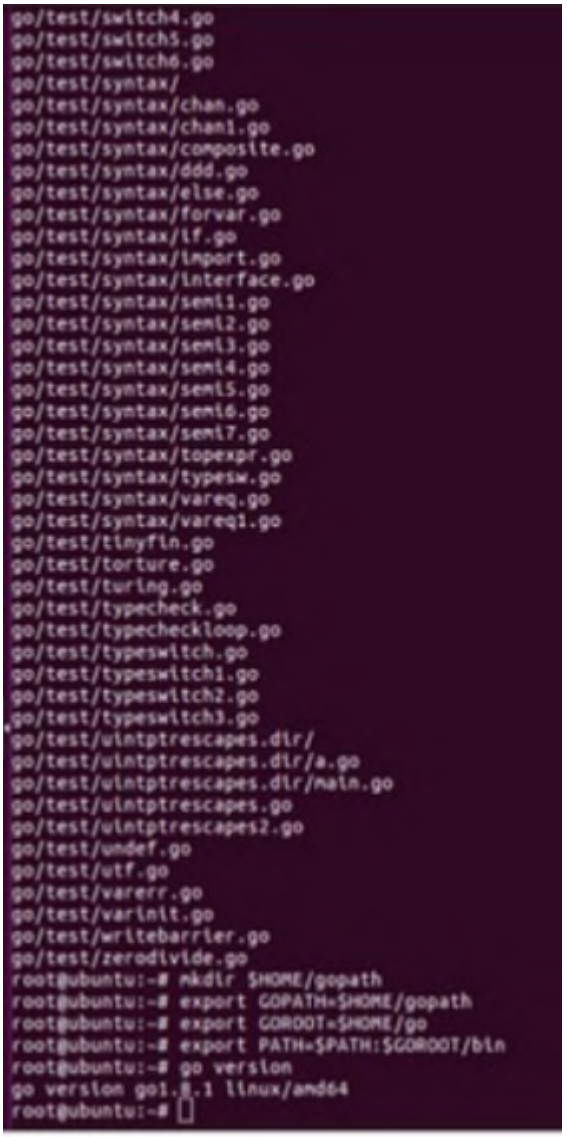
$ **export GOPATH=$HOME/gopath**

$ **export GOROOT=$HOME/go**

$ **export PATH=$PATH:$GOROOT/bin**

$ **go version**





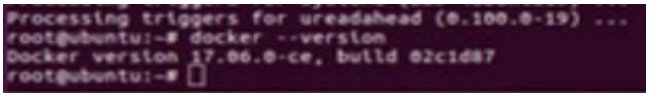
Step 3: Step 3: Install libltdl-dev

$ **apt-get install libltdl-dev**

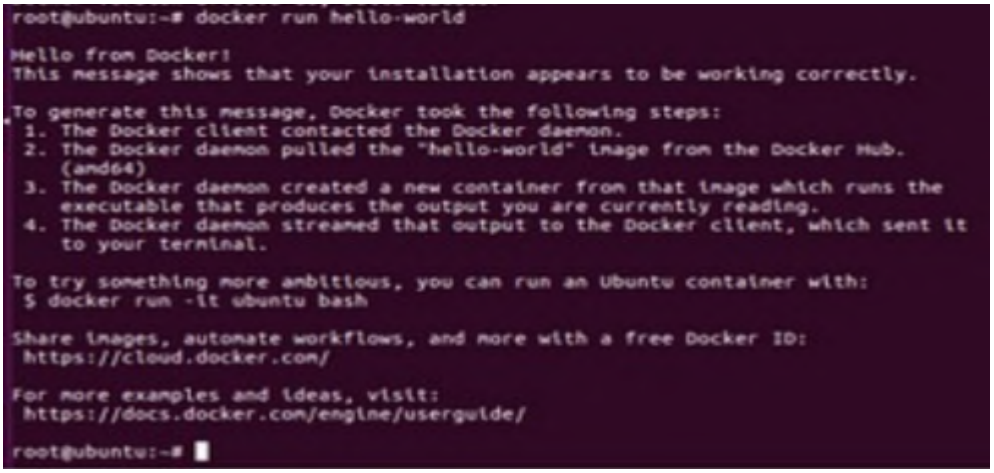
Step 4: Install docker-ce

$**wget**
https://download.docker.com/linux/ubuntu/dists/xenial/pool/stable/amd64/docker-ce_17.06.0~ce-0~ubuntu_amd64.deb

$ **dpkg i docker-ce_17.06.0~ce-0~ubuntu_amd64.deb**
$ **docker - -version**



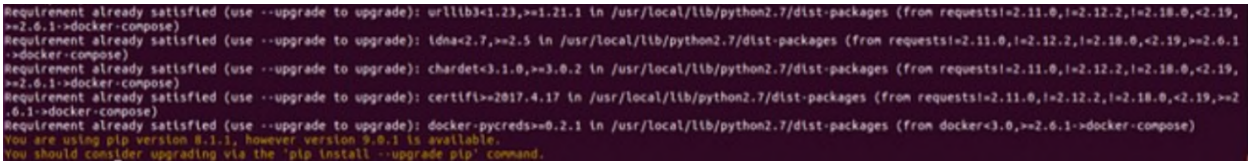
Note: The above approach simply leverages the Docker images that the Hyperledger Fabric project publishes to Docker Hub
$ **docker run hello-world**



Step 5: Check versions if following exists

1. Python -> $ apt-get install python-pip
   pip - -version

2. Docker -> $ pip install docker-compose
   $ docker-compose - -version



3. Git -> $ apt-get install git
   git - -version

```
root@ubuntu:~# git --version
git version 2.7.4
```

4. Curl -> $ apt-get install curl
   curl - -version

```
root@ubuntu:~# curl --version
curl 7.47.0 (x86_64-pc-linux-gnu) libcurl/7.47.0 GnuTLS/3.4.10 zlib/1.2.8 libidn/1.32 librtmp/2.3
Protocols: dict file ftp ftps gopher http https imap imaps ldap ldaps pop3 pop3s rtmp rtsp smb smbs smtp smtps telnet tftp
Features: AsynchDNS IDN IPv6 Largefile GSS-API Kerberos SPNEGO NTLM NTLM_WB SSL libz TLS-SRP UnixSockets
```

5. Node js and npm -> $ curl sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
   node - -version
   npm - -version

```
root@ubuntu:~# node --version
v8.9.3
root@ubuntu:~# npm --version
5.5.1
```

Step 6: Clone the fabric-samples from the github

$ **git clone** https://github.com/hyperledger/fabric-samples.git

```
root@ubuntu:~# git clone https://github.com/hyperledger/fabric-samples.git
Cloning into 'fabric-samples'...
remote: Counting objects: 924, done.
remote: Compressing objects: 100% (38/38), done.
remote: Total 924 (delta 8), reused 52 (delta 7), pack-reused 870
Receiving objects: 100% (924/924), 328.98 KiB | 180.00 KiB/s, done.
Resolving deltas: 100% (346/346), done.
Checking connectivity... done.
```

Step 7: Enter the fabric-samples directory and install the platform specific binaries

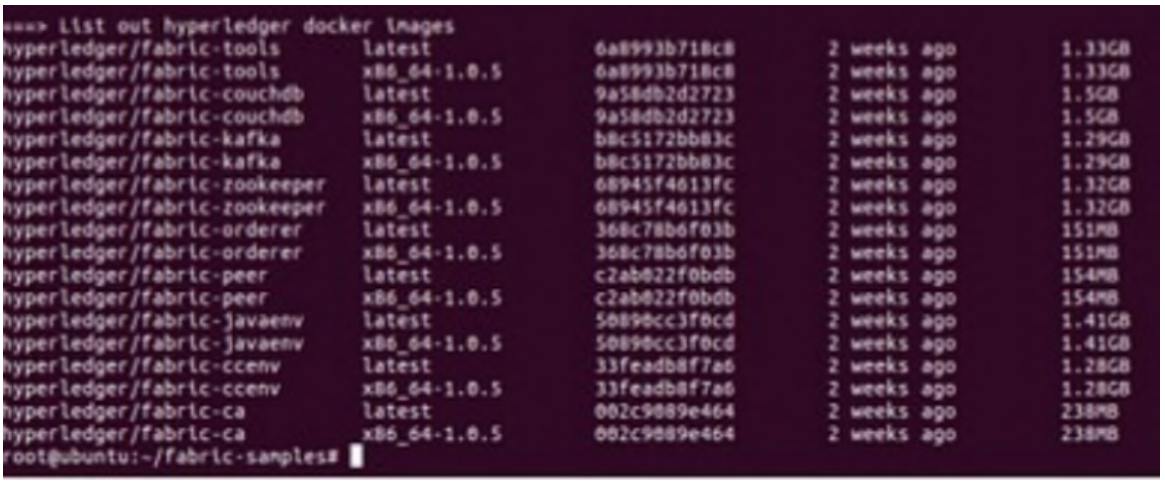
$ **cd fabric-samples**

$ **curl -sSL https://goo.gl/byy2Qj | bash -s 1.0.5**

```
root@ubuntu:~# cd fabric-samples/
root@ubuntu:~/fabric-samples# curl -sSL https://goo.gl/byy2Qj | bash -s 1.0.5
```

```
+ apt-get update
Hit:1 https://deb.nodesource.com/node_8.x xenial InRelease
Get:2 http://security.ubuntu.com/ubuntu xenial-security InRelease [102 kB]
Hit:3 http://us.archive.ubuntu.com/ubuntu xenial InRelease
Get:4 http://us.archive.ubuntu.com/ubuntu xenial-updates InRelease [102 kB]
Get:5 http://us.archive.ubuntu.com/ubuntu xenial-backports InRelease [102 kB]
Fetched 306 kB in 2s (127 kB/s)
Reading package lists... Done

## Run `apt-get install nodejs` (as root) to install Node.js v8.x and npm

root@ubuntu:~# apt-get install nodejs
Reading package lists... Done
Building dependency tree
Reading state information... Done
nodejs is already the newest version (8.9.3-1nodesource1).
0 upgraded, 0 newly installed, 0 to remove and 262 not upgraded.
root@ubuntu:~# node --version
v8.9.3
root@ubuntu:~# npm --version
5.5.1
root@ubuntu:~# git clone https://github.com/hyperledger/fabric-samples.git
Cloning into 'fabric-samples'...
remote: Counting objects: 924, done.
remote: Compressing objects: 100% (38/38), done.
remote: Total 924 (delta 8), reused 52 (delta 7), pack-reused 870
Receiving objects: 100% (924/924), 328.98 KiB | 180.00 KiB/s, done.
Resolving deltas: 100% (346/346), done.
Checking connectivity... done.
root@ubuntu:~# cd fabric-samples/
root@ubuntu:~/fabric-samples# curl -sSL https://goo.gl/byy2Qj | bash -s 1.0.5
===> Downloading platform binaries
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 22.7M  100 22.7M    0     0  78731      0  0:05:02  0:05:02 --:--:-- 24742
===> Pulling fabric Images
===> FABRIC IMAGE: peer

x86_64-1.0.5: Pulling from hyperledger/fabric-peer
d5c6f90da05d: Downloading [>                                                  ]  474.3kB/47.26MB
130088308705: Download complete
```

$ **docker images**



If everything goes well you will see the above output on your screen.

Step 8: To have a look on the download binaries execute the following from your terminal:

$ **cd bin**

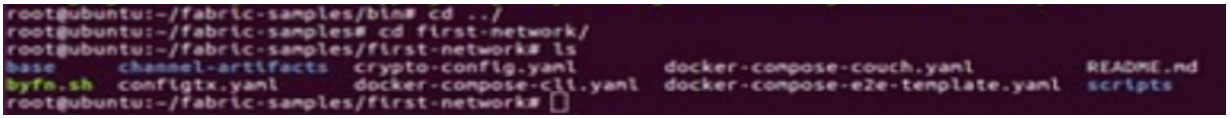
$ **ls**

Step 9: Enter into the first-network directory

$ **cd ../**

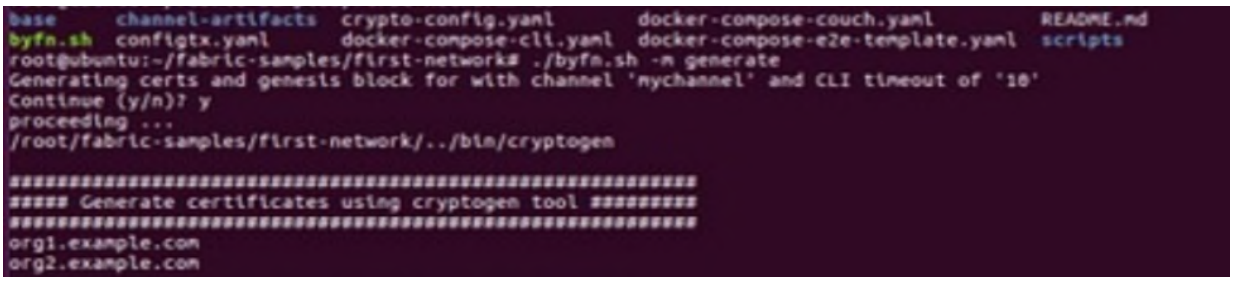
$ **cd first-network**

$ **ls**



Step 10: Generate the required certificates and articates for your first network

$ **./byfn.sh -m generate**



Step 11: To see the generate certificates use the following command:

$ ls

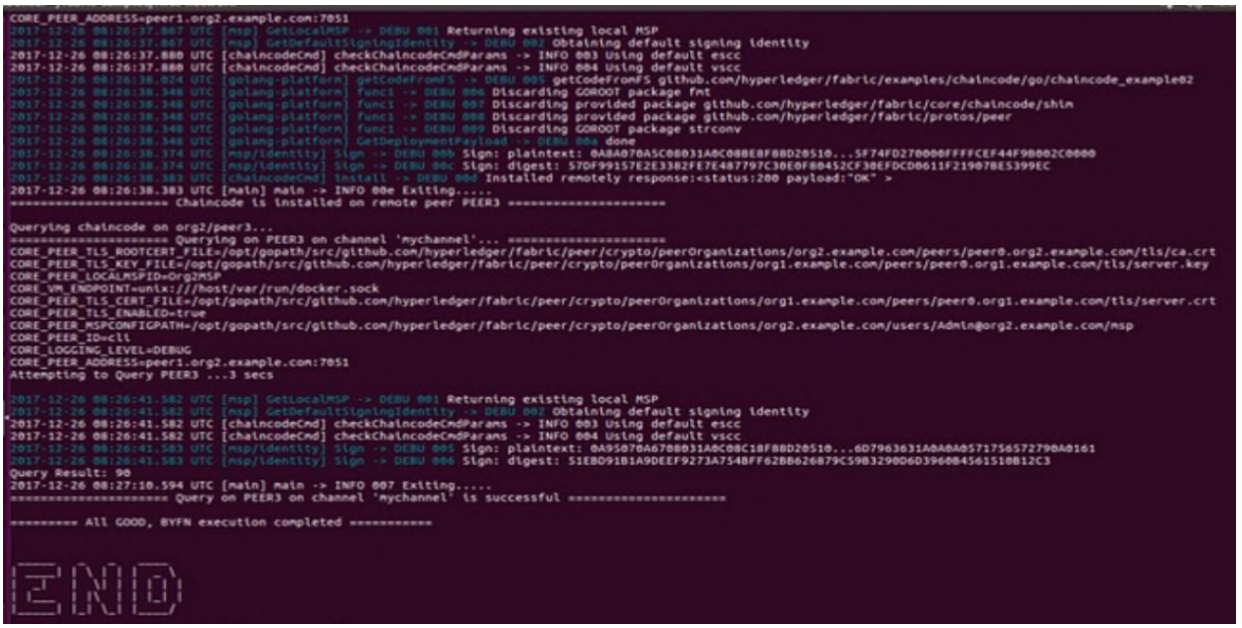
$ **cd crypto-config**

$ **ls**

Step 12: Create your first network using the following command

$ **cd ../**

$ **./byfn.sh -m up**

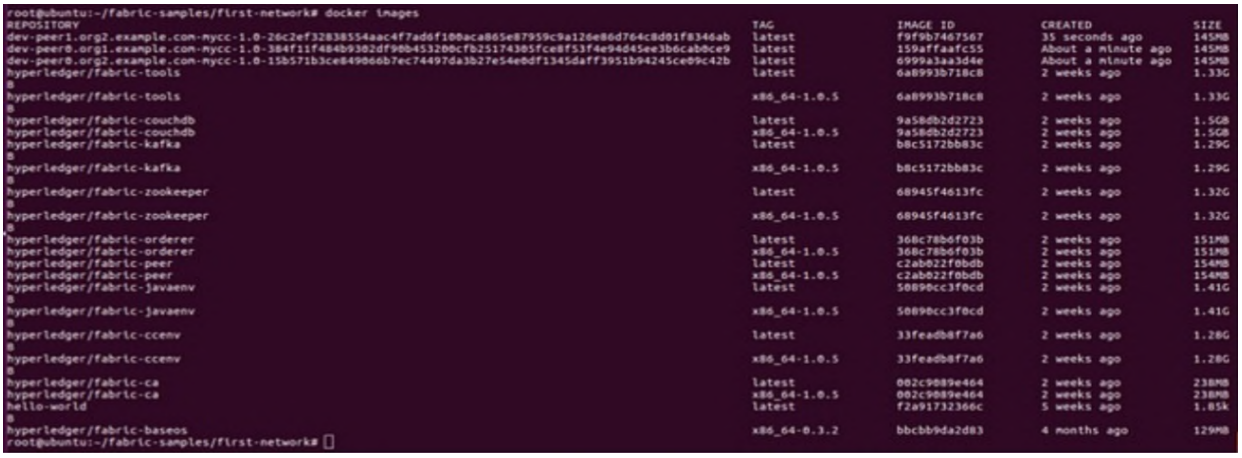


You will see above message once your first-network is created using Hyperledger Fabric.

Step 13: Check the generates images and running containers using the following command:

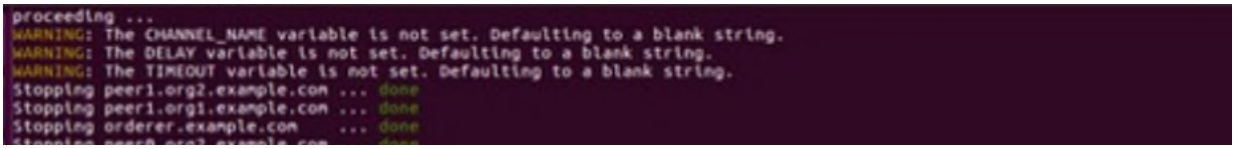
$ **docker images**

$ **docker ps**



Step 14: To bring down the created network executed the following command

$ **./byfn.sh -m down**



```
Stopping peer0.org2.example.com ... done
Stopping peer0.org1.example.con ... done
Removing cli                    ... done
Removing peer1.org2.example.con ... done
Removing peer1.org1.example.con ... done
Removing orderer.example.con    ... done
Removing peer0.org2.example.con ... done
Removing peer0.org1.example.con ... done
Removing network net_byfn
WARNING: The CHANNEL_NAME variable is not set. Defaulting to a blank string.
WARNING: The DELAY variable is not set. Defaulting to a blank string.
WARNING: The TIMEOUT variable is not set. Defaulting to a blank string.
Removing network net_byfn
WARNING: Network net_byfn not found.
841e86aee0be
0aff4054e712
f02c51439e49
781c7b4ae6ff
5b2bbf56005c
Untagged: dev-peer1.org2.example.com-mycc-1.0-26c2ef32838554aac4f7ad6f100aca865e87959c9a126e86d764c8d01f8346ab:latest
Deleted: sha256:f9f9b74675673e73e16c0959fe4bfbb525eaac3f88e9f2c065368d0d7cb3bbee
Deleted: sha256:3dac17fbe9dbdcb53661250aa42d0a85edb0c04bd756bc696ffdfd0fa9ba26b6
Deleted: sha256:a3100a7650f43b9bbbb070053e17acbec0c2e4191421f22c369b13037a563c91
Deleted: sha256:db1072e49f7123155e12ea93ae2a7ff959d8d0e05b137d13739202e7847bba33
Deleted: sha256:bdb8eeb4e0b3b8e0c668e6dd9666ba1bc3b9541a9f5377dad722bc82318d6d8a
Deleted: sha256:361fb38519f3e28e4c28b53466818d713bdbda0fcbf10c81d7b485cf698b554f
Deleted: sha256:b8a16639eefae2e331b0caa7f36dfe24db4aa125ccfebbbac21c8c1302deca50
Untagged: dev-peer0.org1.example.com-mycc-1.0-384f11f484b9302df90b453200cfb25174305fce8f53f4e94d45ee3b6cab0ce9:latest
Deleted: sha256:159affaafc555035338e99896a6d9cc692be34cdce1e751b4dc314bf007db7b6
Deleted: sha256:e888e7bce008fb0dbaf7424d47c83bab462e72c3382fdaf3844620f16ccf740c
Deleted: sha256:5932c737bf2e68bd1979053984cc6192ce79d5abbce0f1e637163d14ba354295
Deleted: sha256:27efcd7a4b7a908c7dcfc5179de605ce7a350d3edae79a17edc626d5ccb19e68
Deleted: sha256:bc13f79b8d25915a189075f064393b2177798918554c6f206f3cd52a0d85c439
Deleted: sha256:d5ab86037477f3a9c0a9d1c74a988678a969ded120622af4a64c4d2e099bb5d6
Deleted: sha256:d28c987ce62a0e94e361457f5ee644abf9c8edd776a1f657d8dd8b1a182085c6
Untagged: dev-peer0.org2.example.com-mycc-1.0-15b571b3ce849066b7ec74497da3b27e54e0df1345daff3951b94245ce09c42b:latest
Deleted: sha256:6999a3aa3d4ed91698bbbd278ad8f078bf1e02bfb641941a819d56f271caae72
Deleted: sha256:d4adebd7c6a9a283558ac10127c2671299a4596ba8a2beea5c92c5d48ad99eb9
Deleted: sha256:9a0119f2a8e8305387c22224b8a04922eb7e8b56f7b6356515206a9c38f81557
Deleted: sha256:2515cb467549e4bd7c26422f58d93f4855bbf5a9d569f959ba6f2b19b25aebc3
Deleted: sha256:dab319454d232a04e979a9b4be34ee41575683a9c75de37806a8cfb2baebad24
Deleted: sha256:eceed525a12a3dcbe6c2dbd8c861f11307ec07e22ea007ddc3ae87b7a90d2a9a
Deleted: sha256:2322cb8eb9d8c93f402c4564e2655ba6be1f08e5c8b8bf27afd8537219aad9ed
root@ubuntu:~/fabric-samples/first-network#
```

Step 15: You can check the created images have been removed using the following:

$ **docker images**

```
root@ubuntu:~/fabric-samples/first-network# docker images
REPOSITORY                    TAG                 IMAGE ID            CREATED             SIZE
hyperledger/fabric-tools      latest              6a8993b718c8        2 weeks ago         1.33GB
hyperledger/fabric-tools      x86_64-1.0.5        6a8993b718c8        2 weeks ago         1.33GB
hyperledger/fabric-couchdb    latest              9a58db2d2723        2 weeks ago         1.5GB
hyperledger/fabric-couchdb    x86_64-1.0.5        9a58db2d2723        2 weeks ago         1.5GB
hyperledger/fabric-kafka      latest              b8c5172bb83c        2 weeks ago         1.29GB
hyperledger/fabric-kafka      x86_64-1.0.5        b8c5172bb83c        2 weeks ago         1.29GB
hyperledger/fabric-zookeeper  latest              68945f4613fc        2 weeks ago         1.32GB
hyperledger/fabric-zookeeper  x86_64-1.0.5        68945f4613fc        2 weeks ago         1.32GB
hyperledger/fabric-orderer    latest              368c78b6f03b        2 weeks ago         151MB
hyperledger/fabric-orderer    x86_64-1.0.5        368c78b6f03b        2 weeks ago         151MB
hyperledger/fabric-peer       latest              c2ab022f0bdb        2 weeks ago         154MB
hyperledger/fabric-peer       x86_64-1.0.5        c2ab022f0bdb        2 weeks ago         154MB
hyperledger/fabric-javaenv    latest              50890cc3f0cd        2 weeks ago         1.41GB
hyperledger/fabric-javaenv    x86_64-1.0.5        50890cc3f0cd        2 weeks ago         1.41GB
hyperledger/fabric-ccenv      latest              33feadb8f7a6        2 weeks ago         1.28GB
hyperledger/fabric-ccenv      x86_64-1.0.5        33feadb8f7a6        2 weeks ago         1.28GB
hyperledger/fabric-ca         latest              002c9089e464        2 weeks ago         238MB
hyperledger/fabric-ca         x86_64-1.0.5        002c9089e464        2 weeks ago         238MB
hello-world                   latest              f2a91732366c        5 weeks ago         1.85kB
hyperledger/fabric-baseos     x86_64-0.3.2        bbcbb9da2d83        4 months ago        129MB
root@ubuntu:~/fabric-samples/first-network#
```

We have successfully created our first-network using Hyperledger Fabric.

```
root@ubuntu:~/fabric-samples/fabcar# ls
enrollAdmin.js  hfc-key-store  invoke.js  node_modules  package.json  package-lock.json  query.js  registerUser.js  startFabric.sh
root@ubuntu:~/fabric-samples/fabcar# gedit query.js

** (gedit:27915): WARNING **: Set document metadata failed: Setting attribute metadata::gedit-position not supported
root@ubuntu:~/fabric-samples/fabcar# node query.js
Store path:/root/fabric-samples/fabcar/hfc-key-store
Successfully loaded user1 from persistence
Query has completed, checking results
Response is  [{"Key":"CAR0", "Record":{"colour":"blue","make":"Toyota","model":"Prius","owner":"Tomoko"}},{"Key":"CAR1", "Record":{"colour":"red","make":"Ford","model":"
Mustang","owner":"Brad"}},{"Key":"CAR10", "Record":{"colour":"Red","make":"Chevy","model":"Volt","owner":"Nick"}},{"Key":"CAR2", "Record":{"colour":"green","make":"Hyund
ai","model":"Tucson","owner":"Jin Soo"}},{"Key":"CAR3", "Record":{"colour":"yellow","make":"Volkswagen","model":"Passat","owner":"Max"}},{"Key":"CAR4", "Record":{"colour
":"black","make":"Tesla","model":"S","owner":"Adriana"}},{"Key":"CAR5", "Record":{"colour":"purple","make":"Peugeot","model":"205","owner":"Michel"}},{"Key":"CAR6", "Rec
ord":{"colour":"white","make":"Chery","model":"S22L","owner":"Aarav"}},{"Key":"CAR7", "Record":{"colour":"violet","make":"Fiat","model":"Punto","owner":"Pari"}},{"Key":"
CAR8", "Record":{"colour":"Indigo","make":"Tata","model":"Nano","owner":"Valeria"}},{"Key":"CAR9", "Record":{"colour":"brown","make":"Holden","model":"Barina","owner":"S
hotaro"}}]
root@ubuntu:~/fabric-samples/fabcar#
```

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

B.E / SEM VII / REV 2019 'C SCHEME' / CSE-(AI&ML)
Academic Year: 2023-24

NAME	**SINGH SUDHAM DHARMENDRA**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	
SUBJECT	**BLOCKCHAIN LAB**
COURSE CODE	**CSDOL7022**
PRACTICAL NO.	
DOP	
DOS	

Experiment No. 06

Aim :- Case Study on Hyperledger.

Theory :-

1. **Introduction to Hyperledger**
   Hyperledger is an open-source collaborative effort and umbrella project under the Linux Foundation that aims to advance the development of blockchain and distributed ledger technologies for various industries. It was launched in December 2015 with the goal of fostering cross-industry collaboration to create enterprise-grade, open-source blockchain solutions.

2. **Key features and components of Hyperledger include:**
   - **Diverse Frameworks and Tools:** Hyperledger offers a variety of frameworks and tools that cater to different use cases and requirements. Some of the most notable Hyperledger projects include:
     1) Hyperledger Fabric: A permissioned blockchain framework designed for developing enterprise-level applications. It provides flexibility, scalability, and confidentiality features.
     2) Hyperledger Sawtooth: Another permissioned blockchain platform that focuses on ease of development and supports pluggable consensus algorithms.
     3) Hyperledger Besu: An Ethereum client designed to be compatible with the Ethereum public network while offering permissioned blockchain features.
     4) Hyperledger Indy: A project focused on decentralized identity management, enabling individuals and entities to have control over their digital identities.
     5) Hyperledger Aries: A toolkit for building interoperable identity solutions.

   - **Permissioned Blockchains:** Many Hyperledger projects are permissioned, meaning that they require participants to be authenticated and authorized to join the network.

   - **Modularity:** Hyperledger projects are designed to be modular, allowing organizations to customize and extend the technology to suit their specific needs.

   - **Enterprise Focus:** Hyperledger projects are tailored for enterprise use cases, emphasizing scalability, performance, and interoperability.

   - **Open Source:** Hyperledger projects are open-source, meaning that their source code is freely available and can be used, modified, and redistributed by anyone. This fosters collaboration and innovation among a broad community of developers and organizations.

   - **Education and Training:** Hyperledger provides educational resources and training to help individuals and organizations understand and implement blockchain technology effectively.

   Hyperledger's diverse set of projects and frameworks allow businesses and organizations to explore and implement blockchain solutions tailored to their specific needs, making it a valuable platform for the development of enterprise-grade blockchain applications.

3. **Title: Enhancing Supply Chain Transparency with Hyperledger Fabric**

   A. Background:

   Company X is a global food distributor known for its commitment to food safety and quality. They source and distribute a wide range of products, including fresh produce, meat, and dairy. Maintaining the integrity of their supply chain is crucial to ensuring the safety and trust of their customers.

   B. Challenge:

   Company X faced several challenges in their supply chain:

   1) Lack of transparency: The existing supply chain system lacked transparency, making it challenging to trace the origins of products and verify their authenticity.
   2) Food safety: Ensuring the safety of perishable goods is critical. Rapid identification and containment of contaminated products are essential to avoid health hazards and reputational damage.
   3) Compliance: Company X needed to adhere to various industry regulations and standards, such as FDA's Food Safety Modernization Act (FSMA).

   Solution:

   Company X decided to implement a blockchain-based solution using Hyperledger Fabric to address these challenges.

   C. Implementation:

   I. **Hyperledger Fabric Network Setup:** Company X established a private Hyperledger Fabric blockchain network with multiple participants, including suppliers, distributors, and retailers.

   II. **Smart Contracts:** Smart contracts were developed to automate processes such as product tracking, quality control, and compliance checks. These contracts ensured that every step in the supply chain was recorded on the blockchain.

   III. **Data Integration:** Integration with IoT devices, such as temperature sensors and GPS trackers, was implemented to collect real-time data about the conditions of goods during transportation.

   IV. **Identity Management:** Hyperledger Indy was used for identity management, allowing each participant to maintain control over their identity and data.

   V. **User Interface:** A user-friendly web interface was created, enabling stakeholders to access real-time data and trace the journey of products through the supply chain.

   D. Results:

   - **Improved Transparency:** Hyperledger Fabric provided a transparent and tamper-proof ledger of all transactions and product movements. This allowed stakeholders to trace the origin of products and verify their authenticity easily.

   - **Enhanced Food Safety:** Real-time data from IoT devices enabled immediate identification of temperature fluctuations or other issues that could compromise food safety. This helped prevent foodborne illnesses and costly recalls.

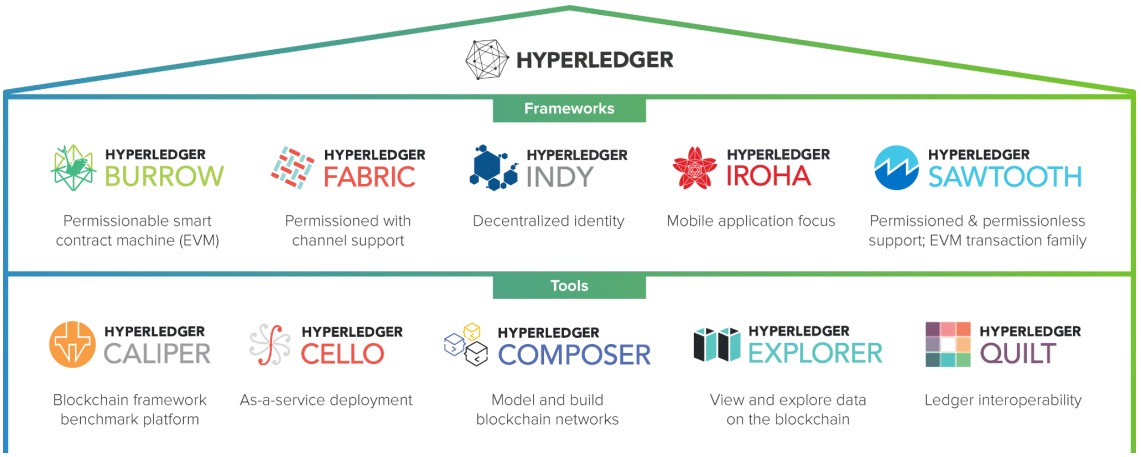
- **Regulatory Compliance:** The blockchain system helped Company X meet regulatory requirements more efficiently. Auditors could access the immutable blockchain records for compliance verification.

- **Cost Savings:** Automation of supply chain processes and the elimination of intermediaries reduced administrative costs and improved overall efficiency.

- **Customer Trust:** Company X's commitment to transparency and safety enhanced their brand reputation, leading to increased customer trust and loyalty.

4. Hyperledger Framework:

Hyperledger is important for several reasons, and its significance extends to various industries and use cases. Here are some of the key reasons why Hyperledger is considered important:



- **Hyperledger Fabric:** Hyperledger Fabric is one of the most well-known and widely used blockchain frameworks within the Hyperledger ecosystem. It is designed for building enterprise-level applications. Key features of Hyperledger Fabric include:
  1) Permissioned Network: Fabric supports permissioned networks, where participants must be authenticated and authorized to join. This is important for businesses and organizations that require control over their blockchain networks.
  2) Modular Architecture: Fabric's modular architecture allows developers to plug in various components, including consensus mechanisms, identity management, and data storage, making it highly customizable.
  3) Private Data: Fabric enables private data collections, allowing certain data to be shared only with authorized parties while preserving data confidentiality.
  4) Smart Contracts (Chaincode): Fabric supports the execution of smart contracts, known as chaincode, which are written in programming languages like Go, Node.js, or Java.
  5) High Performance: It offers high throughput and scalability, making it suitable for applications with demanding performance requirements.

- **Hyperledger Sawtooth:** Hyperledger Sawtooth is another blockchain platform that emphasizes simplicity, modularity, and ease of development. Key features of Sawtooth include:
  1) Modular Design: Sawtooth's modular design allows for the easy addition of custom logic and consensus algorithms, making it flexible for different use cases.
  2) Smart Contracts (Transaction Families): Sawtooth uses a unique concept called "transaction families" to allow developers to define custom transaction logic for various applications.
  3) Pluggable Consensus: It supports pluggable consensus algorithms, enabling organizations to choose the most appropriate consensus mechanism for their network.
  4) Permissioned and Permissionless: Sawtooth can be configured for both permissioned and permissionless (public) blockchain networks.

- **Hyperledger Besu:** Hyperledger Besu is an Ethereum client that is part of the Hyperledger project. It is designed to be compatible with the Ethereum public network while offering permissioned blockchain features. Key features of Besu include:
  1) Ethereum Compatibility: Besu supports Ethereum's JSON-RPC and Web3 RPC interfaces, making it compatible with Ethereum-based applications and tools.
  2) Permissioning: It offers permissioning features for network access control, allowing organizations to create private, permissioned networks.
  3) Java Implementation: Besu is implemented in Java, which can be advantageous for organizations with Java-based infrastructure.

- **Hyperledger Iroha:** Hyperledger Iroha is designed for applications that require a simple and user-friendly blockchain framework. Key features of Iroha include:
  1) Ease of Use: Iroha is designed to be easy to understand and use, making it suitable for developers with varying levels of experience.
  2) Support for Mobile and IoT: It is well-suited for mobile and IoT (Internet of Things) applications due to its lightweight architecture.
  3) Customizable Consensus: Iroha supports multiple consensus algorithms, allowing users to choose the most appropriate one for their use case.

In summary, Hyperledger is important because it offers enterprise-grade blockchain solutions that are secure, customizable, and designed for interoperability. It facilitates innovation, cost reduction, and compliance in various industries, making it a valuable platform for organizations looking to harness the benefits of blockchain technology.

Conclusion :- Implementing Hyperledger Fabric allowed Company X to overcome supply chain challenges and achieve greater transparency, safety, and efficiency in their operations. The blockchain-based solution not only improved internal processes but also strengthened relationships with suppliers and customers while ensuring compliance with industry regulations. This case study demonstrates the practical benefits of Hyperledger in supply chain management.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

B.E / SEM VII / REV 2019 'C SCHEME' / CSE-(AI&ML)
Academic Year: 2023-24

NAME	**SINGH SUDHAM DHARMENDRA**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	
SUBJECT	**BLOCKCHAIN LAB**
COURSE CODE	**CSDOL7022**
PRACTICAL NO.	
DOP	
DOS	

A) Blockchain platform ethereum using Geth.

- Follow the next steps to **build** your **Private Blockchain.**
  1. Install Geth
  2. Define Genesis Block
  3. Start the first blockchain node
  4. Start the second blockchain node
  5. Create the peer-to-peer connection
  6. Mining Blocks and Create Transactions

- **Installing Geth (Go Ethereum):-**
  Geth is a CLI with some resources to connect you to the Ethereum network. It will be used to start our private network in a local environment.
  To install Geth in Ubuntu/Debian follow the following steps:
  sudo add-apt-repository -y ppa:ethereum/ethereum
  sudo apt-get update
  sudo apt-get install ethereum

  You can see installation to other platforms here
  https://geth.ethereum.org/docs/getting-started/installing-geth
  Check if geth was installed successfully with the following command.
  geth -h
  This shows all available commands using geth

- **The Genesis Block:-**
  All blockchains start with the genesis block. This block defines the initial configuration to a blockchain. The configuration to genesis block is defined in genesis.json file.
  So, let's create a folder to start the private network and create the genesis.json file.
  mkdir my-blockchain
  cd my-blockchain
  touch genesis.json
  Add the following content to genesis.json file:

```
{
    "config": {
            "chainId": 1234,
            "homesteadBlock": 0,
            "eip150Block": 0,
            "eip155Block": 0,
            "eip158Block":      0,
            "byzantiumBlock":   0,
            "constantinopleBlock":      0,
            "petersburgBlock":  0,
            "ethash":       {}
            },
            "difficulty":    "4",
            "gasLimit":    "8000000",
            "alloc":        {}
}
```

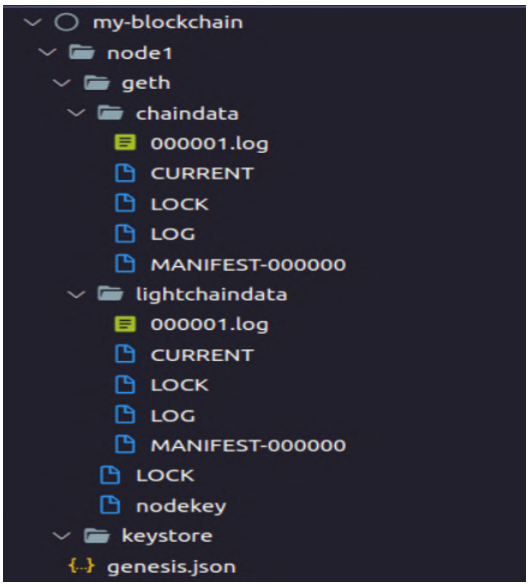
- **Explanation:**
  1. chainId: An integer greater than zero that defines a unique id for your private network. The main network has ID 1. If you supply your own custom network ID which is different than the main network, your nodes will not connect to other nodes and form a private network.. You can find a community-run registry of Ethereum networks at https://chainid.network.
  2. homesteadBlock: Homestead is the first official stable version of the Ethereum protocol. If you plan to use this release, the attribute should be set to 0.
  3. difficulty: Determines the difficulty of generating blocks.
  4. gasLimit: Indicates the current network-wide gas consumption limit per unit. Gas is the fuel that is used to pay transaction fees on the Ethereum network.
  5. alloc: Is used to start some accounts with positive balance in the network. In this case we have not defined any account.
  6. The other attributes refer to the protocols and attributes that will be used on the network, we set the default values for the ethereum network.

- **Start Database (The Bootnode)**
  The bootnode is the first node created when the blockchain is started. To start the database to first node execute:
  geth init --datadir node1 genesis.json
  The folder node1 will be created with the database to bootnode.

  The terminal result must be something like this:

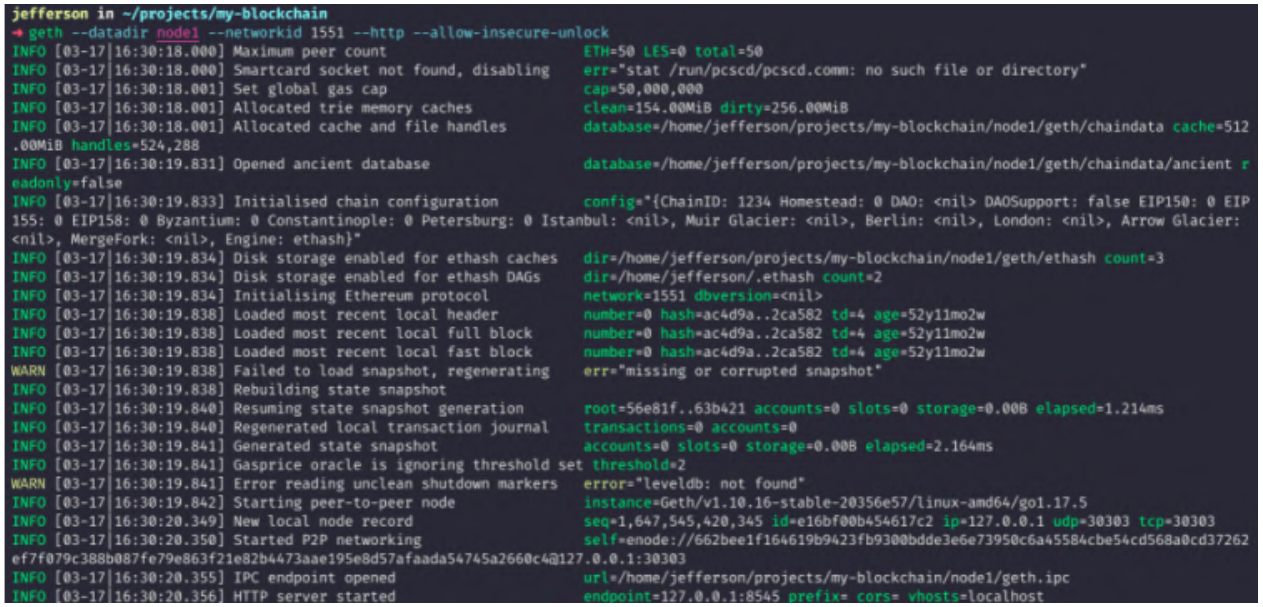
```
jefferson in ~/projects/my-blockchain
➜ geth init --datadir node1 genesis.json
INFO [03-17|16:10:25.866] Maximum peer count                       ETH=50 LES=0 total=50
INFO [03-17|16:10:25.866] Smartcard socket not found, disabling    err="stat /run/pcscd/pcscd.comm: no such file or directory"
INFO [03-17|16:10:25.867] Set global gas cap                       cap=50,000,000
INFO [03-17|16:10:25.867] Allocated cache and file handles         database=/home/jefferson/projects/my-blockchain/node1/geth/chaindata cache=16.
00MiB handles=16
INFO [03-17|16:10:26.322] Writing custom genesis block
INFO [03-17|16:10:26.323] Persisted trie from memory database      nodes=0 size=0.00B time="32.042µs" gcnodes=0 gcsize=0.00B gctime=0s livenodes=
1 livesize=0.00B
INFO [03-17|16:10:26.324] Successfully wrote genesis state         database=chaindata hash=ac4d9a..2ca582
INFO [03-17|16:10:26.325] Allocated cache and file handles         database=/home/jefferson/projects/my-blockchain/node1/geth/lightchaindata cach
e=16.00MiB handles=16
INFO [03-17|16:10:26.812] Writing custom genesis block
INFO [03-17|16:10:26.813] Persisted trie from memory database      nodes=0 size=0.00B time="22.667µs" gcnodes=0 gcsize=0.00B gctime=0s livenodes=
1 livesize=0.00B
INFO [03-17|16:10:26.814] Successfully wrote genesis state         database=lightchaindata hash=ac4d9a..2ca582
```

  The directory result should be something like this:



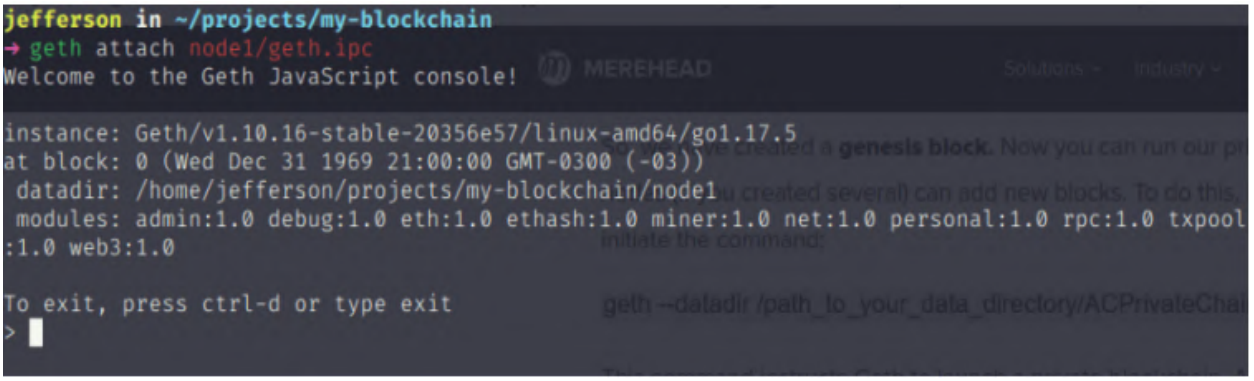
- **Start Node:**
  geth --datadir node1 --networkid 1234 --http --allow-insecure-unlock --nodiscover
  This command will be start the node in private network with id 1234. The flag --http is used to enable Web App Access, we will use this in next post to connect Metamask with that private network. The flag --allow-insecure-unlock is used to permit execute transfers without a web application, we will use this in tests to this network. The flag --nodiscover is used to prevent node from trying to connect to others automatically

  The terminal result must be something like this:



  Done!! The blockchain in the private network is running. Let's go do some tests to verify this.

  Open another terminal window in the same folder my-blockchain and execute the following command:
  geth attach node1/geth.ipc
  This opens an interactive javascript terminal to execute some tasks to this node. The terminal result should be something like this:



  To see node informations execute the following command in this javascript terminal opened:
  admin.nodeInfo
  This command show all informations about this node, and you can verify the node is active.
  Keep this window open for some more tests ahead.

- **Adding member peers:**
  Let's start the second node to our blockchain. To do this execute the following commands:
  geth init --datadir node2 genesis.json
  geth --datadir node2 --networkid 1234 --port 30304
  The commands are very similar to the previous ones with some minor differences:
  1. The folder to database in this case is node2.
  2. The flags --http and --allow-insecure-unlock are not necessary.
  3. The flag --port it's necessary to differentiate this node from the previous one. The first node is running on default port 30303.

  Open a new javascript terminal to second node created and verify the informations executing:
  geth attach node2/geth.ipc
  admin.nodeInfo
  The result should be something like this:

- **Connecting Peers:**
  Now, let's create the peer-to-peer connection between the two nodes in blockchain.
  Copy the **enode** from the nodeInfo of the **second** node, then execute the following command in javascript terminal of the **first** node:

admin.addPeer("enode://431fa50a676b35dd750a68656cf8a822edb2c083ddf2359b6c246216dfbef0f1dad517b53b9ba0a35f4d2f1d44274ae4fc4b2bdf69e774af892b097d082eff1c@127.0.0.1:30304?discport=0")

And to see all peers connected execute to both nodes the following command:

admin.peers

Any node must have one peer connected.

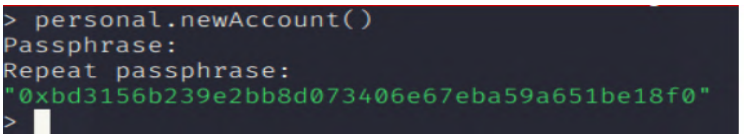
Note the **30304** port referencing the second node in the network. The result should be something like this:

Done!! Your blockchain is created with multiple nodes running and connected. Let's create accounts and start mining to see data updated in both nodes.

- **Mining:**

  To mine blocks it's necessary to have a base account. Let's do this. In javascript terminal of the first node execute the command to create a new account and define the password required to this account, the public address from the created account should be presented.

  personal.newAccount()



  You can see account balance executing:

  eth.getBalance("0xbd3156b239e2bb8d073406e67eba59a651be18f0")

  Now, you can start mining in the first node. Before that run the command to see the current height of the blocks, must be 0, because no blocks have been mined yet:

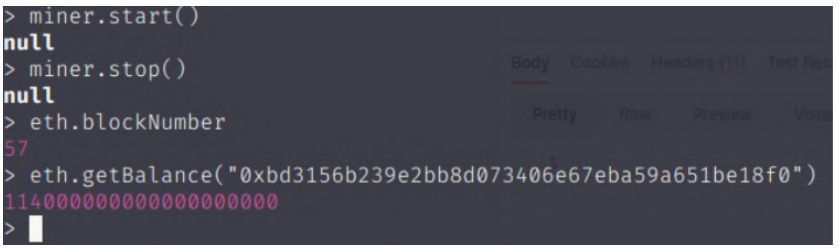
  eth.blockNumber

  Start the mining, and stop after some seconds to see the updated block.

  miner.start() // Start Mining

  miner.stop() // Stop Mining

  Now Verify again the **blockNumber** and **balance** of account created. Some blocks must have been created and values must have been earned as a reward



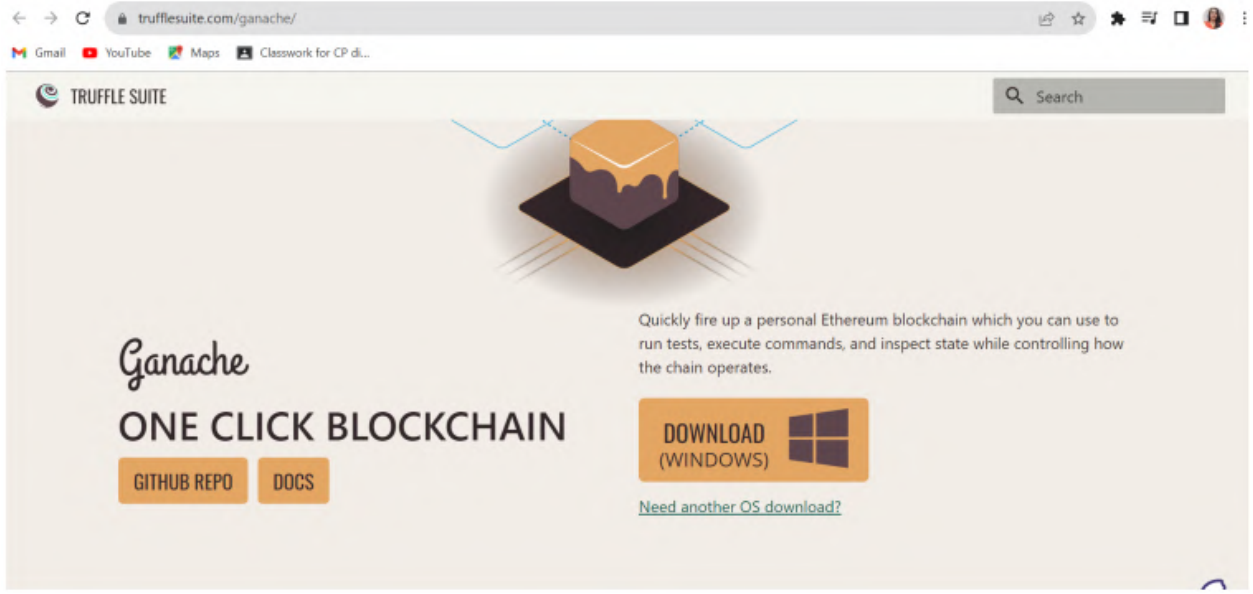
  You can verify the blockNumber also in the javascript terminal of the second node after some seconds, the value must be the same presented in the first node. This means that in fact the nodes are connected and are updated.

B) Blockchain platform Ganache

Step 1: Download and install Ganache

Download the application for your operating system from the official Ganache website.

Run the installation file after downloading it, then install the application on your computer by adhering to the on-screen prompts. Ganache is available for Windows, Mac and Linux operating systems in all its versions.



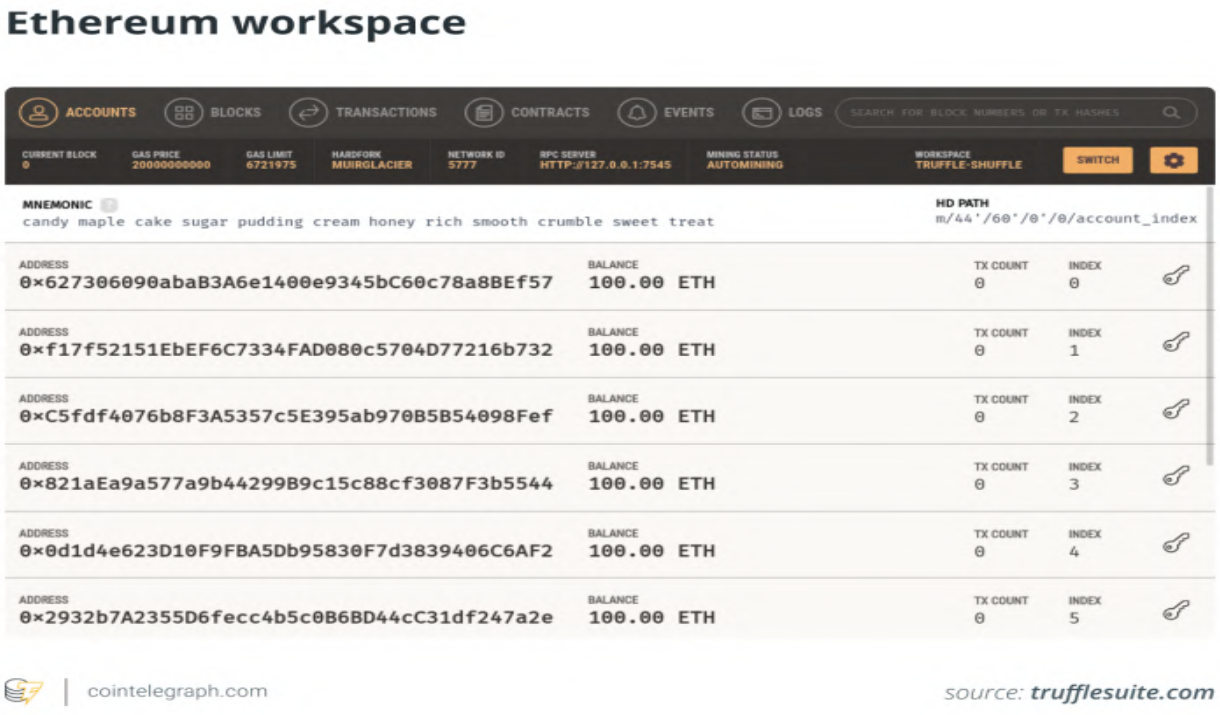
Step 2: Create a new workspace

To create a new workspace, open the Ganache application and select “New Workspace.”

Users can set up the network parameters for their unique Ethereum blockchain in the workspace settings, including the number of accounts, the gas limit and the starting balance of each account.

An Ethereum workspace is a set of settings and user accounts that establish the parameters for a customized Ethereum blockchain network built using Ganache.

Developers may quickly set up a private Ethereum network for testing and development purposes using workspaces.



Step 3: Start the personal Ethereum blockchain network

After configuring the network settings, click “Start” to begin your own private Ethereum blockchain network. For each of the accounts you set up in the workspace settings, Ganache will generate a set of private keys. Then, copy the remote procedure call (RPC) server address from the top of the screen, as you’ll need this to connect your development tool.

Using the RPC communication protocol, client software can invoke a server-side process from a distance. As a result, it is feasible to activate a procedure or function in another address space or process without the programmer worrying about the specifics of the underlying network transport or communication protocols. It enables programs to communicate with other systems on a network.

Step 4: Connect your development tool to the Ganache network It is necessary to link one’s development tool, such as Truffle Suite, to the Ganache network to deploy and test smart contracts on the private Ethereum blockchain. To do so, follow these steps:

- Open your development tool and find the settings or configuration menu.
- Search for a provider or network selection option, then type the RPC server address you copied from Ganache.
- To ensure your development tool uses a new network, save your modifications and restart it.

Step 5: Test and deploy smart contracts

- After configuring the network, users can deploy and test their smart contracts on the private Ethereum blockchain. Using the Truffle command line interface, they can compile and deploy their contracts to the Ganache network. Once the contracts are deployed, the Truffle CLI can interact with them and test their functionality.
- It allows developers to interact with their smart contracts and the underlying blockchain network using various commands. Using the Truffle CLI, developers can automate the building and deployment of smart contracts, making it easier to develop and deploy DApps.

- When a smart contract is deployed to the mainnet, it must be submitted to the network, and a fee in cryptocurrency is paid to cover the cost of running the contract on the blockchain. When a contract is deployed, it becomes unchangeable and immutable. To guarantee that the smart contract works as intended and is secure, testing it properly before deployment is crucial.
  An example of a simple contract deployment using Truffle CLI
  Step 1: Go to the directory where one wishes to build a project by opening the terminal or command prompt.
  Step 2: To start a new Truffle project, enter the following command:

Command to start a new Truffle project

```
truffle init
```

cointelegraph.com

"Truffle init" is a command that initializes a new Truffle project with a basic directory structure and configuration files.

Step 3: Under the contracts directory, add a new Solidity contract file. Here's an example of a simple contract that stores a string:

An example of a simple contract that stores a string

```
// SPDX-License-Identifier:
MITpragma solidity >=0.4.22 <0.9.0;

contract MyContract {
   string public myString = "Hello, world!";
}
```

cointelegraph.com

The above code is a smart contract written in the Solidity programming language. One declared variable, a public string variable called "myString," is present in the contract named "MyContract." Everybody on the blockchain can access the string variable, which is initialized to "Hello, world!" With a tool like Ganache, this contract can be set up on a private blockchain or an Ethereum network. Once installed, it can be used to interact with transactions sent to its blockchain address.

Step 4: A contract can be compiled by running the following command:

Command to compile a contract

```
truffle compile
```

cointelegraph.com

“Truffle compile” is a command that compiles the contract code and generates an application binary interface (ABI) and bytecode. The ABI serves as the interface between smart contracts and applications, while bytecode is a smart contract’s compiled version that may be run on the Ethereum Virtual Machine (EVM).

Step 5: Run the following command to deploy the contract to a local blockchain network like Ganache:

Command to deploy the contract to a local blockchain network

```
truffle migrate
```

cointelegraph.com

“Truffle migrate” is a command used to deploy the contract to the local network and create a new migration script in the “migrations” directory.

Step 6: Run the following command to interact with the deployed contract using the Truffle console:

Command to interact with the deployed contract

```
truffle console
```

cointelegraph.com

“Truffle console” opens up a console with the web3.js library and contract artifacts loaded, allowing interaction with a blockchain network.

Step 7: By establishing an instance of their contract and calling its functions once they are on the console, users can communicate with their contract. For instance, the following commands can be used to retrieve the value of myString:

Commands to retrieve the value of myString

```
let instance = await MyContract.deployed()
let result = await instance.myString()
console.log(result)
```

cointelegraph.com

The value of a string variable (myString) is then retrieved from the deployed instance of a smart contract (MyContract) using the above code. The output “Hello, world!” is printed to the console using “console.log(result).”

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

B.E / SEM VII / REV 2019 'C SCHEME' / CSE-(AI&ML)
Academic Year: 2023-24

NAME	**SINGH SUDHAM DHARMENDRA**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	
SUBJECT	**BLOCKCHAIN LAB**
COURSE CODE	**CSDOL7022**
PRACTICAL NO.	
DOP	
DOS	

Experiment No:- 08

Aim: To create a blockchain application

We're going to create the first baby blockchain in Go. You can also learn how to create it in Python, and JavaScript. These prototypes can help you understand the concepts we described earlier. We'll complete this in four steps:

1. **Create a block.**
2. **Add the data (header and body) to the block.**
3. **Hash the block.**
4. **Chain the blocks together.**

Step 1: CREATE A BLOCK

We'll start with creating a folder with two files in it, **main.go** and **block.go**

Folder structure

go // the folder
main.go // file 1
block.go // file 2

Main.go

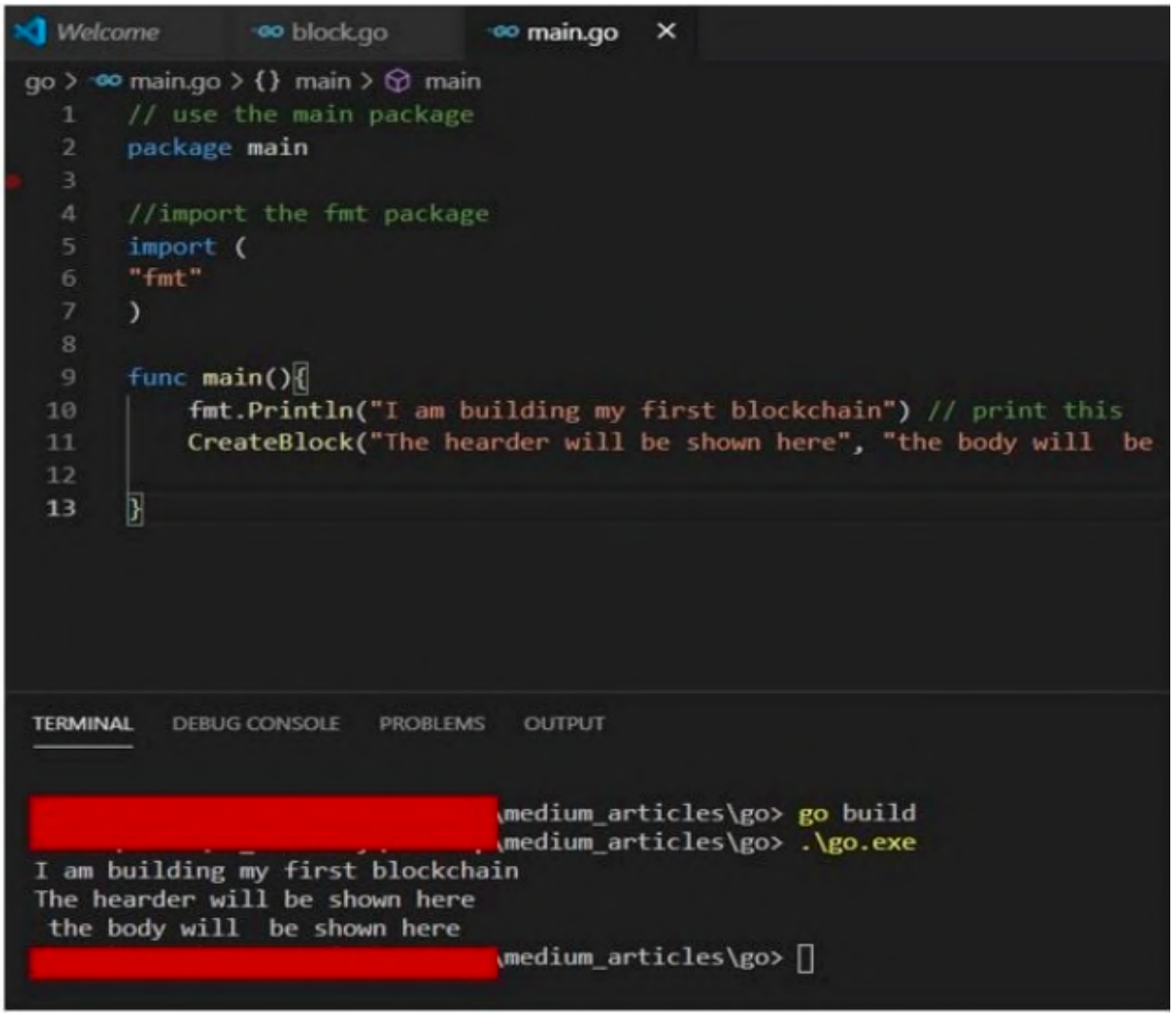
```
// use the main package
package main
//import the fmt package
import (
	"fmt"
)
func main(){
	fmt.Println("I am building my first blockchain") // print this
	CreateBlock("The hearder will be shown here", "the body will be shown here") // call the function that will create the block and pass some parameters in it.
}
```

If you run this program it will show an error because the **CreateBlock** function is not defined yet, so go ahead and create it in **block.go**.

```
package main
import (
	"fmt" // this will help us to print on the screen
)
func CreateBlock(Header, Body string){
	fmt.Println(Header ,"\n", Body) // Show me the block content
}
```

Step 2: ADD DATA TO YOUR BLOCKS.

The beauty of Go is that you don't have to import or export functions, just declare them with capital letters, and Go will find them for you. Now, open a terminal and move to your created folder, and **run go build**, then run **.\go** on Windows, or **./go** on Linux and Macbook.



We just created a simple Go program that calls a function and passes some string data. Let's add two more files, **blockchain.go** and **structures.go**. Now we have four files: **main.go**, **block.go**, **structures.go**, and **blockchain.go**

Step 3: HASH YOUR BLOCK

Structures.go

```
package main //Import the main package
// Create the Block data structure
// A block contains this info:
type Block struct {
        Timestamp int64 // the time when the block was created
        PreviousBlockHash []byte // the hash of the previous block
        MyBlockHash []byte // the hash of the current block
        AllData []byte // the data or transactions (body info)
}
// Prepare the Blockchain data structure :
type Blockchain struct {
        Blocks []*Block // remember a blockchain is a series of blocks
}
```

Block.go

```
package main
import (
	// We will need these libraries:
	"bytes" // need to convert data into byte in order to be sent on the network,
	computer understands better the byte(8bits)language
	"crypto/sha256" //crypto library to hash the data
	"strconv" // for conversion
	"time" // the time for our timestamp
)
// Now let's create a method for generating a hash of the block
// We will just concatenate all the data and hash it to obtain the block hash
func (block *Block) SetHash() {
	timestamp := []byte(strconv.FormatInt(block.Timestamp, 10)) // get the time and
	convert it into a unique series of digits
	headers := bytes.Join([][]byte{timestamp, block.PreviousBlockHash, block.AllData},
	[]byte{}) // concatenate all the block data
	hash := sha256.Sum256(headers) // hash the whole thing
	block.MyBlockHash = hash[:] // now set the hash of the block
}

// Create a function for new block generation and return that block
func NewBlock(data string, prevBlockHash []byte) *Block {
	block := &Block{time.Now().Unix(),prevBlockHash,[]byte{},[]byte(data)} //block  is received
	block.SetHash() // the block is hashed
	return block // the block is returned with all the information in it
}

/* let's now create the genesis block function that will return the first block. The genesis block
is the first block on the chain */
func NewGenesisBlock() *Block {
	return NewBlock("Genesis Block", []byte{}) // genesis block is made with some data in it
}
```

Blockchain.go

```
package main
// create the method that adds a new block to a blockchain
func (blockchain *Blockchain) AddBlock(data string) {
	PreviousBlock := blockchain.Blocks[len(blockchain.Blocks)-1] // the previous block is
	needed, so let's get it
	newBlock := NewBlock(data, PreviousBlock.MyBlockHash) // create a new block
	containing the data and the hash of the previous block
	blockchain.Blocks = append(blockchain.Blocks, newBlock) // add that block to the
	chain to create a chain of blocks
}

/* Create the function that returns the whole blockchain and add the genesis to it first. the  genesis
block is the first ever mined block, so let's create a function that will return it since it
does not exist yet */
func NewBlockchain() *Blockchain { // the function is created
	return &Blockchain{[]*Block{NewGenesisBlock()}} //genesis block is added first  to chain
}
```

Main.go

```
//Time to put everything together and test
package main
import (
	"fmt" // just for printing something on the screen
)
func main() {
	newblockchain := NewBlockchain() // Initialize the blockchain
	// create 2 blocks and add 2 transactions
	newblockchain.AddBlock("first transaction") // first block containing one tx
	newblockchain.AddBlock("Second transaction") // second block containing one tx
	// Now print all the blocks and their contents
	for _, block := range newblockchain.Blocks { // iterate on each block
		fmt.Printf("Hash of the block %x\n", block.MyBlockHash) // print hash of  the block
		fmt.Printf("Hash of the previous Block: %x\n", block.PreviousBlockHash) //
		print the hash of the previous block
		fmt.Printf("All the transactions: %s\n", block.AllData) // print the transactions
	} // our blockchain will be printed
}
```

Let's run it now, go build then .\go

```
ndaysinai@Ndays-MacBook-Pro go % go build
ndaysinai@Ndays-MacBook-Pro go % ./go
Block ID : 0
Timestamp : 1610770863
Hash of the block : 58d43603575ae717828752ce1ebc5bcef86231de9a282a9122842c79dd73a4db
Hash of the previous Block :
All the transactions : Genesis Block
Block ID : 1
Timestamp : 1610770864
Hash of the block : 6465ae7dde0d7b5359518a26d0a198c6bcb8964b6c84d4a85ceaf3fdc9ea5fe0
Hash of the previous Block : 58d43603575ae717828752ce1ebc5bcef86231de9a282a9122842c79dd73a4db
All the transactions : first transaction
Block ID : 2
Timestamp : 1610770865
Hash of the block : 5962a66cff9c8d05ed3b02a901cf0ea539d715c911458acac7d5df293d582453
Hash of the previous Block : 6465ae7dde0d7b5359518a26d0a198c6bcb8964b6c84d4a85ceaf3fdc9ea5fe0
All the transactions : Second transaction
ndaysinai@Ndays-MacBook-Pro go %
```

Step 4: CHAIN YOUR BLOCKS TOGETHER

Oops, our blocks in the blockchain don't have any IDs and timestamps. So, let's add that information by modifying the **main.go** file, add these two lines in the **for loop**:

```
fmt.Printf("Block ID : %d \n", i)
fmt.Printf("Timestamp : %d \n", block.Timestamp+int64(i))
//Time to put everything together and test
package main
import (
	"fmt" // just for printing something on the screen
)
func main() {
	newblockchain := NewBlockchain() // Initialize the blockchain
	// create 2 blocks and add 2 transactions
	newblockchain.AddBlock("first transaction") // first block containing one tx
	newblockchain.AddBlock("Second transaction") // second block containing one tx
```

```
        // Now print all the blocks and their contents
        for i, block := range newblockchain.Blocks { // iterate on each block
                fmt.Printf("Block ID : %d \n", i) // print the block ID
                fmt.Printf("Timestamp : %d \n", block.Timestamp+int64(i)) // print the
                timestamp of the block, to make them different, we just add a value i
                fmt.Printf("Hash of the block : %x\n", block.MyBlockHash) // print hash of the block
                fmt.Printf("Hash of the previous Block : %x\n", block.PreviousBlockHash) //
                print the hash of the previous block
                fmt.Printf("All the transactions : %s\n", block.AllData) // print the transactions
        } // our blockchain will be printed
}
```

Let's save the code and run it again, go build then ./go.

As you can see, the blockchain is well structured. Except for the genesis block, each block contains its hash and the hash of the previous block, which makes it immutable. If the data in the block is altered, the hash will automatically change and the block will be discarded. The genesis block doesn't have any previous hash because it's the first one. There is no previous block.

Conclusion: We have successfully created blockchain application.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

B.E / SEM VII / REV 2019 'C SCHEME' / CSE-(AI&ML)
Academic Year: 2023-24

NAME	**SINGH SUDHAM DHARMENDRA**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	
SUBJECT	**BLOCKCHAIN LAB**
COURSE CODE	**CSDOL7022**
PRACTICAL NO.	
DOP	
DOS	

Experiment No. 09

Aim: Mini-project on Design and Development of 'Snippet-Adaptable Y'

Abstract:

Snippet-Adaptable Y is based on wallet to wallet cryptocurrency computers or mobile devices such as phones or tablets. They use an internet connection to access the blockchain network for the cryptocurrency you're using.Cryptocurrencies are not "stored" anywhere—they are bits of data stored in a database.
These bits of data are scattered all over the database; the wallet finds all of the bits associated with your public address and sums up the amount for you in the app's interface.You can send or receive cryptocurrency from your wallet using various methods. Typically, you enter the recipient's wallet address, choose an amount to send, sign the transaction using your private key, add an amount to pay the transaction fee, and send it.

Introduction:

The world of blockchain and Web3 technology has seen rapid advancements in recent years. Wallets aims to take these advancements to the next level, offering a comprehensive platform that seamlessly connects users to transfer cryptocurrency through wallets.

- **Key Features**

  1) Decentralized Transactions: Wallet-to-wallet transfers are decentralized, meaning they occur directly between two parties without the need for intermediaries like banks or payment processors. This decentralization is a fundamental feature of most cryptocurrencies.

  2) Peer-to-Peer (P2P): These transfers are peer-to-peer, allowing users to send and receive cryptocurrencies directly with one another. This eliminates the need for third-party involvement.

  3) Security: Cryptocurrencies employ cryptographic techniques to secure transactions. Public and private keys are used to authenticate and verify each transaction, making it difficult for unauthorized parties to tamper with the transfer.

  4) Transparency: Cryptocurrency transactions are recorded on a public ledger known as the blockchain. This transparency allows anyone to verify the details of a transaction, such as the sender, recipient, and amount.

  5) Speed: Transactions within most blockchain networks can be quite fast compared to traditional banking systems. The exact speed depends on the cryptocurrency and the network's capacity. Some cryptocurrencies, like Bitcoin, may have longer confirmation times than others.

  6) Low Transaction Costs: Cryptocurrency transactions typically have lower fees than traditional financial systems, especially for cross-border transfers. The fees vary depending on the cryptocurrency and network congestion.

7) Cross-Border Capabilities: Cryptocurrencies borderless, which means that wallet-to-wallet transfers can be made across international boundaries without the need for currency conversion or the involvement of banks.
8) User Control: Users have full control over their cryptocurrency wallets, which means they can initiate transfers at any time, 24/7, without relying on traditional banking hours.

- **Cryptocurrency Wallet Types**

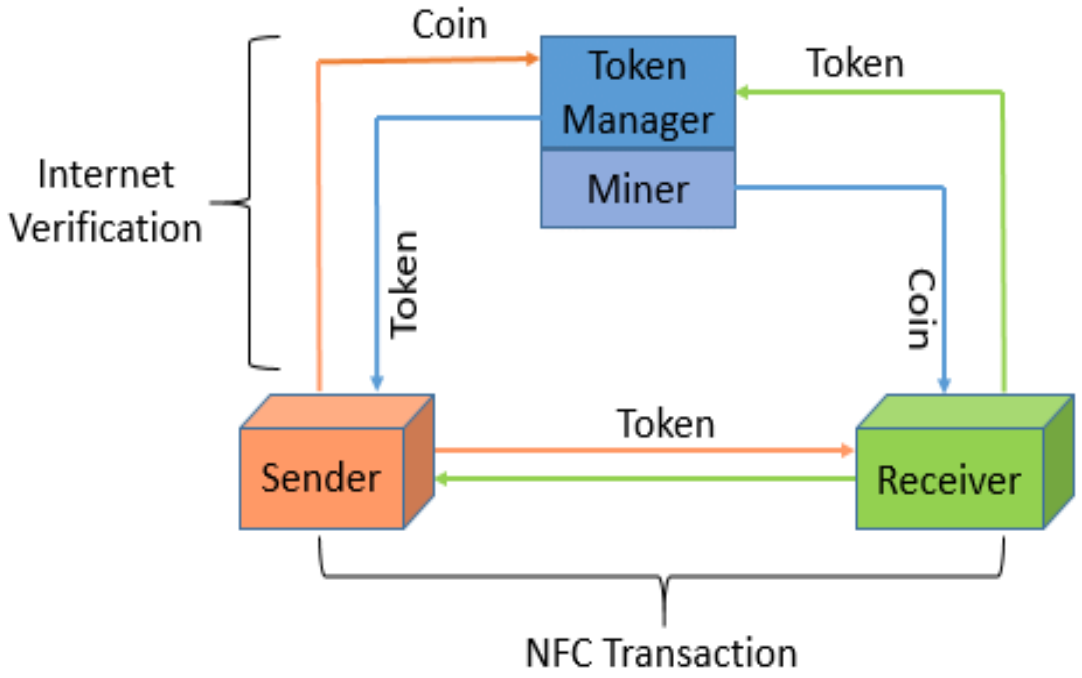
There are two main types of wallets, custodial and noncustodial.
1) Custodial wallets are hosted by a third party that stores your keys for you. This could be a company that provides enterprise-level data security systems businesses use to preserve and secure data. Some cryptocurrency exchanges offer custodial wallets for their customers.
2) Noncustodial wallets are wallets in which you take responsibility for securing your keys. This is the type that most cryptocurrency wallets on devices are.

There are two subcategories of wallets, hot and cold. A hot wallet has a connection to the internet or to a device that has a connection, and a cold wallet has no connection. Lastly, there are three subcategories of wallets—software, hardware, and paper. Each of these types is considered either a hot or cold wallet.
So, you can have a noncustodial software hot wallet, a noncustodial hardware cold or hot wallet, or a custodial hardware cold wallet. These are the most common types, but you may also encounter other combinations.

- **Software Wallets:**

Software wallets include applications for desktops and mobile devices. These wallets are installed on a desktop or laptop computer and can access your cryptocurrency, make transactions, display your balance, and much more. Some software wallets also include additional functionality, such as exchange integration if you're using a wallet designed by a cryptocurrency exchange.
Many mobile wallets can facilitate quick payments in physical stores through near-field communication (NFC) or by scanning a QR code. Mobile wallets tend to be compatible with iOS or Android devices. Trezor, Electrum, and Mycelium are examples of wallets that you can use. Software wallets are generally hot wallets.

Flowchart:



Implementation:

The implementation of wallet-to-wallet cryptocurrency transfers involves several key components and steps. Here's a high-level overview of how it's typically done.

Steps:

1)Setting Up Wallets:

Both the sender and recipient need to have cryptocurrency wallets. These wallets can be software-based (mobile, desktop, web) or hardware wallets (physical devices).

2)Choosing a Cryptocurrency:

Both parties should use the same cryptocurrency for the transfer. For example, if the sender wants to send Bitcoin, the recipient should have a Bitcoin wallet.

3) Obtaining Recipient's Public Address:

The sender needs to obtain the recipient's public wallet address. This is usually a long string of characters unique to the recipient's wallet and is used to identify the destination.

4)Initiating the Transfer:

The sender opens their wallet software or app, initiates a transaction, and enters the recipient's public address, as well as the amount they want to send.

5)Digital Signature:

Sender's wallet software uses sender's private key to create digital signature for transaction.

6)Notification:

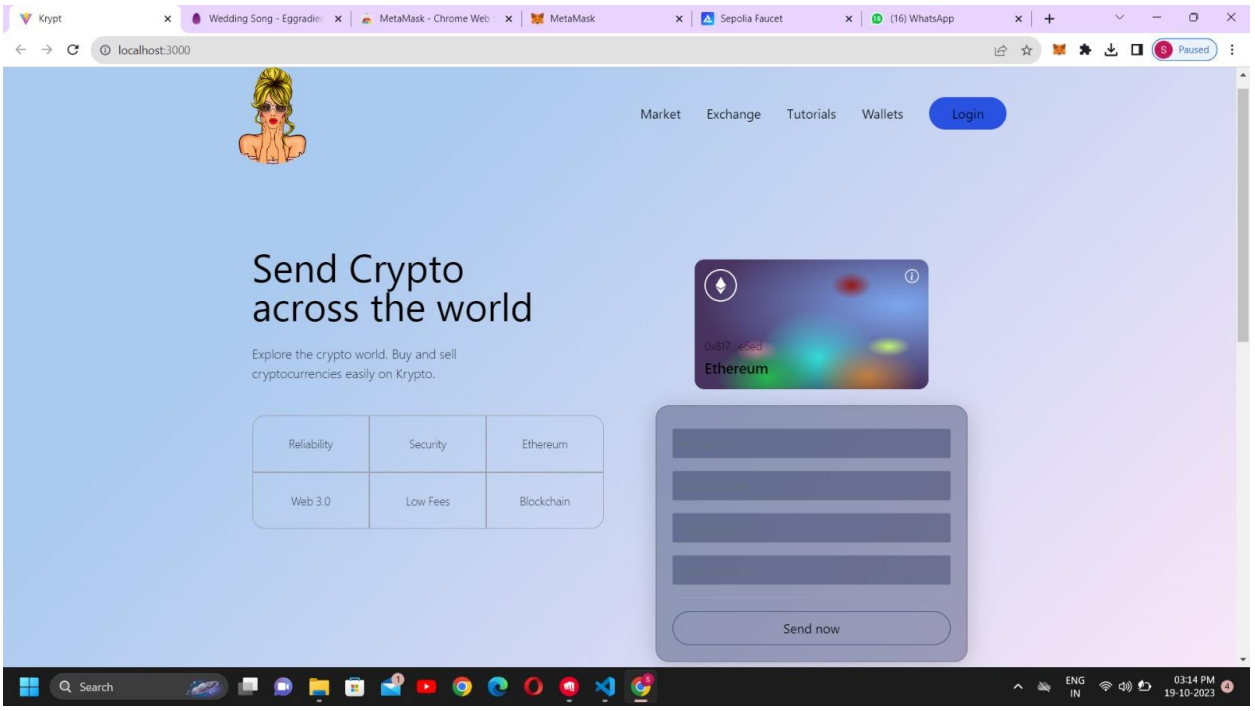
Some wallet software or blockchain explorers can send notifications to both the sender and recipient once the transaction is confirmed.

7)Finality:

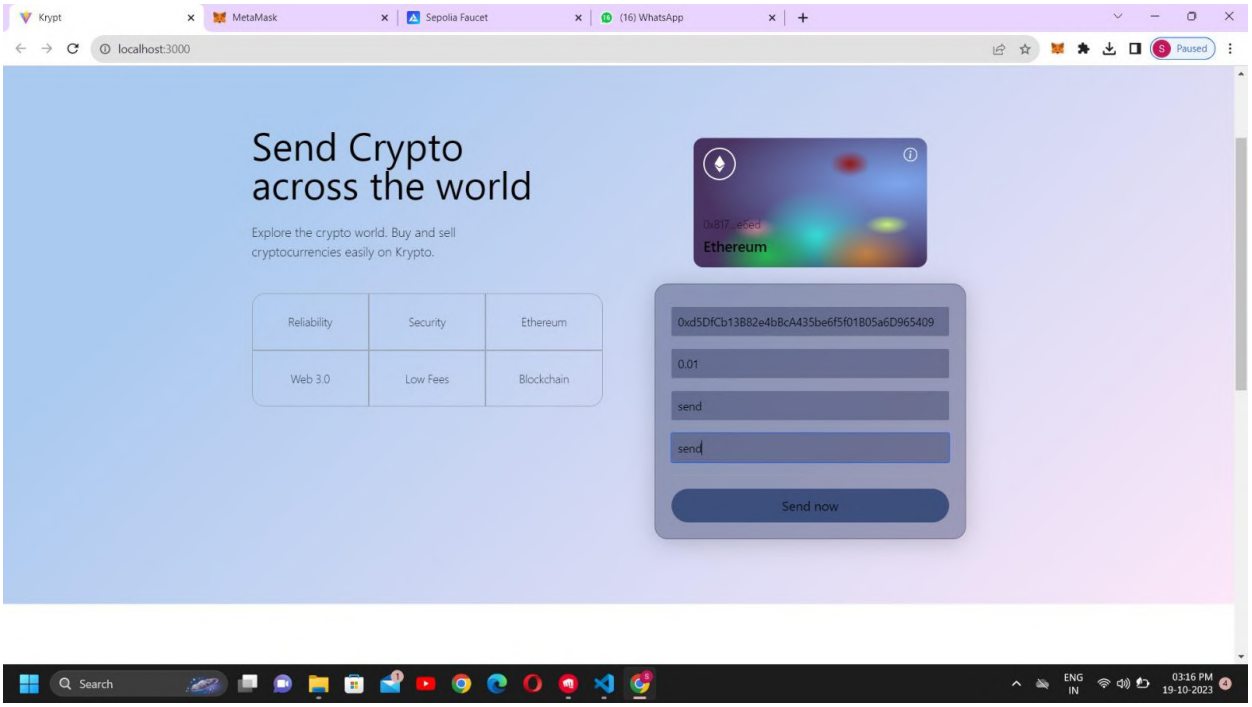
Once the transaction is confirmed and added to the blockchain, it is irreversible. It becomes a permanent part of the public ledger.

Output:

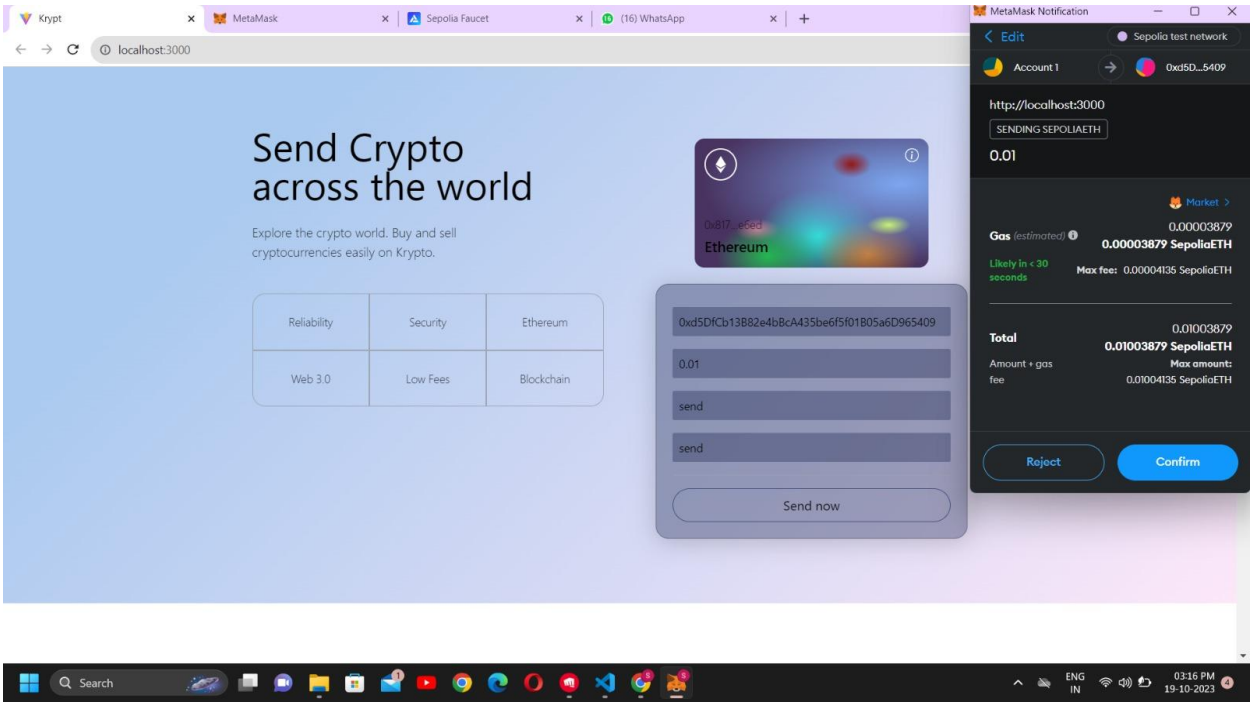
1) Home Page



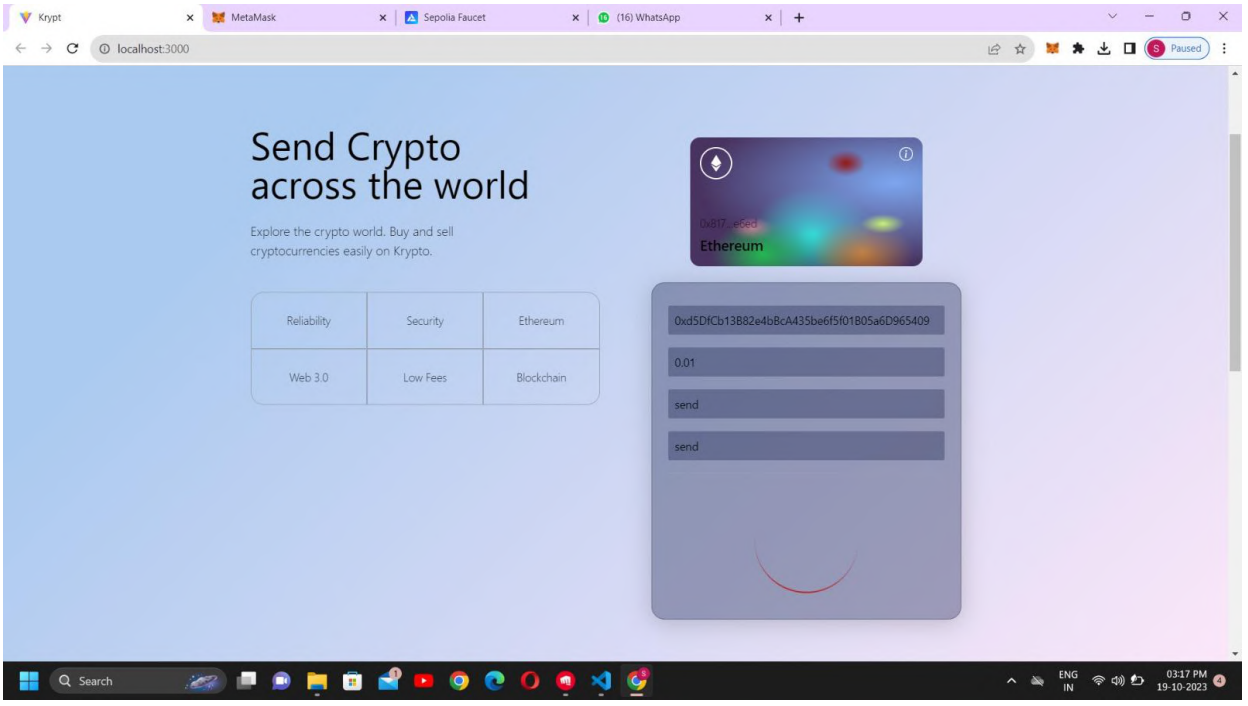
2) Receiver's Address with Crypto Amount



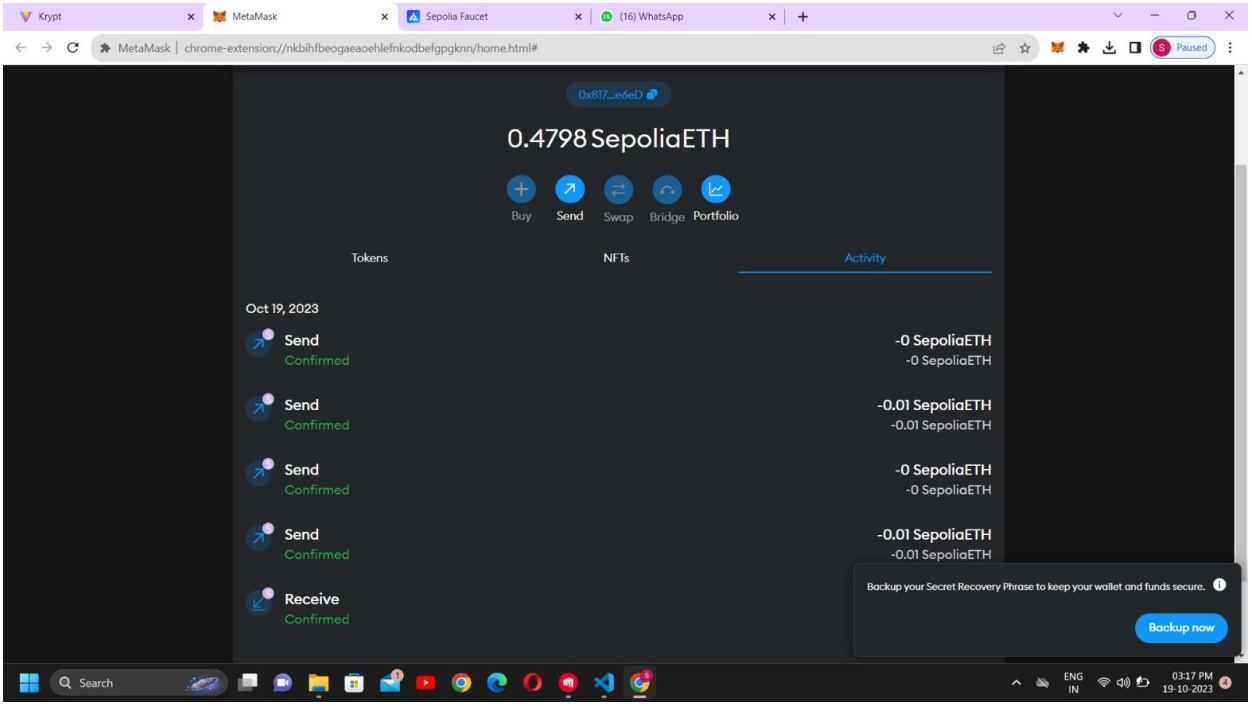
3) Confirmation of transaction



4) Fetching Receiver's Address



5) Transaction History



Conclusion:

In conclusion, wallet-to-wallet cryptocurrency transfers are a fundamental aspect of the cryptocurrency ecosystem. They enable users to send and receive digital assets directly to and from one another without the need for intermediaries, such as banks or payment processors. The process involves several key components and steps, including setting up wallets, choosing the right cryptocurrency, obtaining the recipient's public address, initiating the transfer, creating a digital signature, broadcasting the transaction, validation, confirmation, and finality.

It's important for individuals engaging in cryptocurrency transfers to use secure wallet software, protect their private keys, and be aware of the specific details and features of the cryptocurrency they are using. While wallet-to-wallet cryptocurrency transfers offer many advantages, users should also exercise caution to prevent potential security risks and scams.

Name :- Singh Sudham Dharmendra

Branch :- CSE (AI & ML)

Roll no :- AIML57

Subject :- Blockchain Technology

Topic :- Assignment No :- 1

Date of submission :-

Signature :-

Q.1] What is blockchain? Explain features of Blockchain.

→ A blockchain is constantly growing ledger which keeps a permanent record of all the transaction that have taken place in a secure, chronological and immutable way.
A blockchain is a chain of blocks with which contain information.

(i) Immutable :- Once a transaction is a recorded on the blockchain an immutable that provides a high degree of security & trust.

(ii) Distributed :- All network participants have a copy of the ledger for complete transparency. A public ledger will provide complete information about all the transaction & participants on the N/w.

(iii) Decentralized :- It means there is no central authority controlling the N/w. The N/w is made up of a large number of nodes that work together to verify & validate transaction. Each & every node in the ~~together~~ blockchain N/w will have the same copy of ledger.

(iv) Secure :- All the records in the blockchain are individually encrypted. There is no central authority. Every information on the blockchain is hashed cryptographically which means that every piece of data has a unique identity on the N/w.

(v) Consensus :- Consensus is a decision making algorithm for the group of nodes active on the N/w to reach on agreement quickly and faster and for the smooth functionally of the system. Nodes might not trust each other but they can trust the algorithm that runs at the core of the N/w to make decisions.

(vi) Unanimous :- All the N/w participants agree to the validity of the records before they can be added to the N/w. A node cannot simply add, update or delete records is updated simultaneously and the updation:

Q.2] What are the components of Blockchain.

→ (i) Node :- A member of blockchain, It is of two types :-
(a) Full Node (b) Partial Node / light weight Node

(a) Full node :- It maintain a full copy of all the transaction. It has the capacity to validate, accept & reject the transaction.

(b) Partial Node / light-weight Node :- It doesn't maintain the whole copy of the blockchain ledger. It maintains only hash value of transaction

(ii) Ledger :- It is a digital database of information. Here digital means cryptocurrency which is used to exchange blw different nodes.
These are three types of ledger.

(a) Public ledger :- It is open & transparent to all

(b) Distributed ledger :- all nodes have local copy of database.

(c) Decentralized Ledger :- Single or group of node not having the central control. Every node participate in execution of the job.

(iii) Wallet :- It is a digital wallet that allows user to store their crypto. Every node is the blockchain has a wallet mainly of two types :-

(a) Hot wallet : It is used for online day-to-day transaction connection to N/w
Hot wallet further classified into two types:
- online/web wallet - runs on cloud platform, eg- My ether wallet
- Software wallet - consist of desktop wallets & mobile wallets eg:- Ethereum

(b) Cold wallet : These wallets aren't connected to internet, it is very safe
- Paper wallet - offline wallet in which a piece of paper is used.
- Hardware wallet - physical electronic device that uses random numbers generator.

(iv) Nonce :- A nonce is an abbreviation for numbers only used nonce. which is a numbers added to hashed or encrypted data block in a blockchain. It is 32 bit no. generated randomly only once which make blockchain secure

(v) Hash :- The data is wrapped to the fixed sized using hashing. It plays a very important role in cryptography.
Properties of hash function are :- • Collision resistant, • hiding
• Puzzle friendliness

(vi) Block :- Each block contains a specific amount of data & they are linked together in a cronological sequence.

(vii) Consensus algorithm :- The consensus algorithm can guarantee that allows all the blocks coming into a multi-node blockchain N/w are verified, providing safety.

(viii) Master Node :- Selective blockchain N/w have master nodes. Master node are more capable than normal node. They take in active node (24*7). It is not responsible for adding new block to the N/w.

(ix) Virtual Machine :- A compiler progcum that contain instruction written in a programming language create a virtual machine. Physical machine change its state according to programmer's instruction

(x) Asset :- Asset having a value recognized by the nodes in the N/w. It may be used as a payment method on the blockchain, & also to facilitate a rewards mechanism for all N/w participants

(xi) State databases :- This component of blockchain is a key-value database representing the current state of the N/w

Q.3] What are the different types of Blockchain.

→

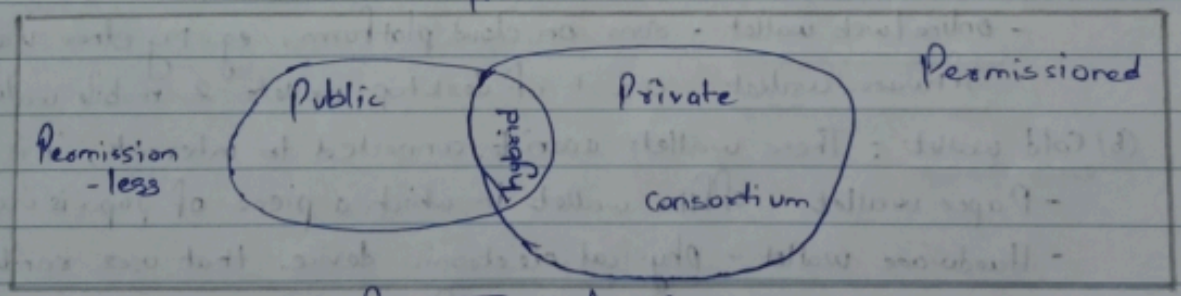


Fig. Types of Blockchain.

(i) Public Blockchain :- As the name is public, this blockchain is open to the public, which means it is not owned by anyone. Anyone can perform verification of transaction or records.

- advantages :- Trustable, secure, anonymous nature, Decentralized
- disadvantages :- Processing, energy consumption, acceptance
- Example :- Bitcoin & Ethereum.

(ii) Private Blockchain :- only selected nodes can participate in process. These are not as open as public blockchain, operated in closed N/w.
- advantages : high speed, scalability, privacy & balanced.
- disadvantages : centralized & cant.
- Example :- Hyperledger & Coeda

(iii) Hybrid Blockchain :- It is a combination of both public & private blockchain. Permission-based & permissionless systems are used.
- advantages : hybrid nature, less cost, transparency.
- disadvantages : efficiency maintainence, transparency, incentives.

(iv) Consortium blockchain / Federated blockchain :- This is an innovative method to solve the organization needs some part of public & private.
- advantages :- fast speed, decentralized authority, privacy.
- disadvantages :- approval → less flexible, transparency.

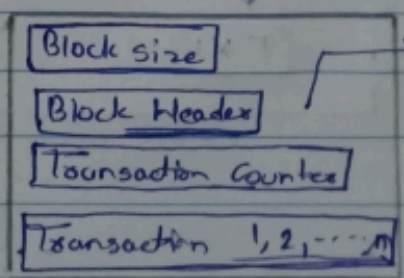
Q.4] Explain structure of block in blockchain.

→

Field	Size	Description.
Block size	4 Bytes	the size of the block
Block Header	80 Bytes	Several field forms the block Header.
Transaction counter	1-9 Bytes (Var Int)	how many transaction
Transaction	Variable	Transaction recorded in the block.

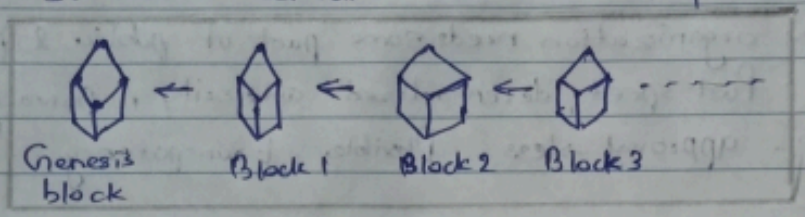
- Block Header :- It contains following fields :-
  1] Version 2] Previous block hashs 3] Merkle root
  4] time stamps 5] difficulty target 6] Nonce.

* Merkle root :- A merkle root uses mathematical formula to check if the data is not corrupted, hacked or manipulated.

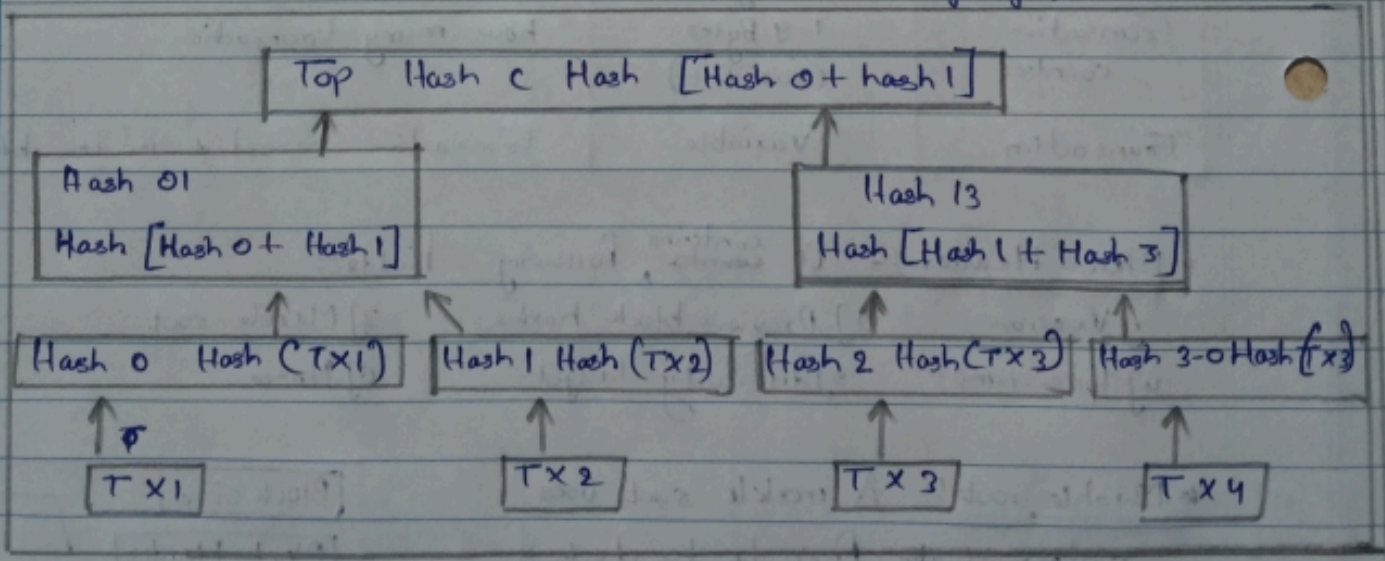


* Block Header Hash :- It is not actually included inside the blocks data structure, neither when the block's transmitted on the n/w or - nor, when it is stored as part of blockchain
* Block Height :- Second way to identify a block is by its position in the blockchain, called the block height. Block height 1 is the genesis block [First in the N/w]

* The Genesis block :-
First block in any blockchain based protocols sometimes refer to 0.
It doesn't contain hash value of 'previous block.



* Merkle tree :- Merkle root is stored in the block header [Bottom-up]
It is a data structure tree in which leaf node labelled with hash of a data block & non-leaf node labelled with cryptographic hash child....



Q.5] Explain -

→ (a) Genesis Block :-

- It is the first block in any blockchain based application protocol.
- This block is sometime refers to Block 0.
- Genesis Block is hard coded into the blockchain application s/w.
- It doesn't contain hash value of previous block.

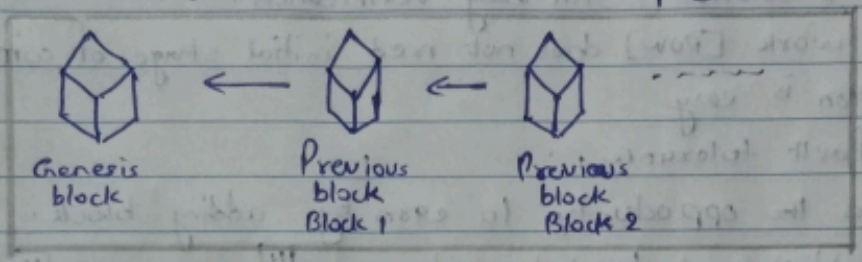


fig. Genesis block.

- Characteristics

→ Unique identifier. → No previous transaction
→ Bootstraping the chain → N/w parameters → Immutability.

(b) Merkle Tree :-

- Merkle root is stored in the Blockchain.
- It is a DS tree in which leaf node labelled with hash of data block & non-leaf node labelled with cryptographic hash of child node.
- Merkle tree are constructed in bottom-up approach.

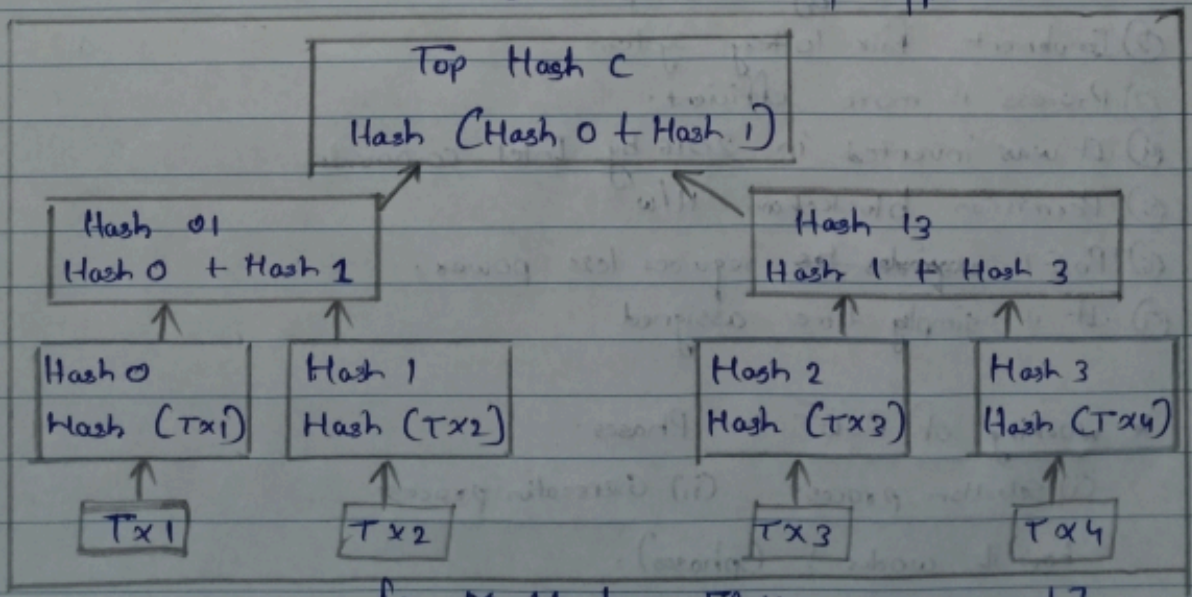


fig. Merkle tree [Bottom-up approach]

Q.6] Explain Proof of Work Algorithm.

→ Proof of Work [PoW] is a consensus algorithm used in blockchain N/w to secure and validate transaction & create new Blocks.

* Block Hash < Hash Difficulty

Advantages :-

(i) Hard to find solution, still easy verification.
(ii) Proof of work [PoW] does not need initial stage of coins before mining.
(iii) implementation is easy
(iv) It is a Fault tolerance
(v) Give miners the opportunity to earn by adding block.
(vi) It is the oldest most trusted, and most populae consensus protocol

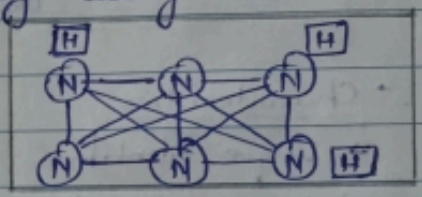


fig. PoW diag

Disadvantage :-

(i) More computational power needed.
(ii) More calculation needed to add new blak.

Q.7] Explain PoET algorithm.

→ (1) PoET is a N/w consensus algorithm that provides high resources utilization of ~~utilization~~ & energy consumption.
(2) Implement fair lottery system
(3) Process is more efficient.
(4) It was invented in 2016 by Intel corporation
(5) Permission blockchain N/w
(6) PoET ~~requires less~~ requires less power.
(7) It is simply time assigned

* Working of PoET :- Phases.

(i) Selection process (ii) Generation process.

how it works ! (phases).

(a) Certificate sharing (SGX) } Selection process
(b) SGX provide times object to all node }
(c) time done [new block] write up } Generation process
and also primary key }

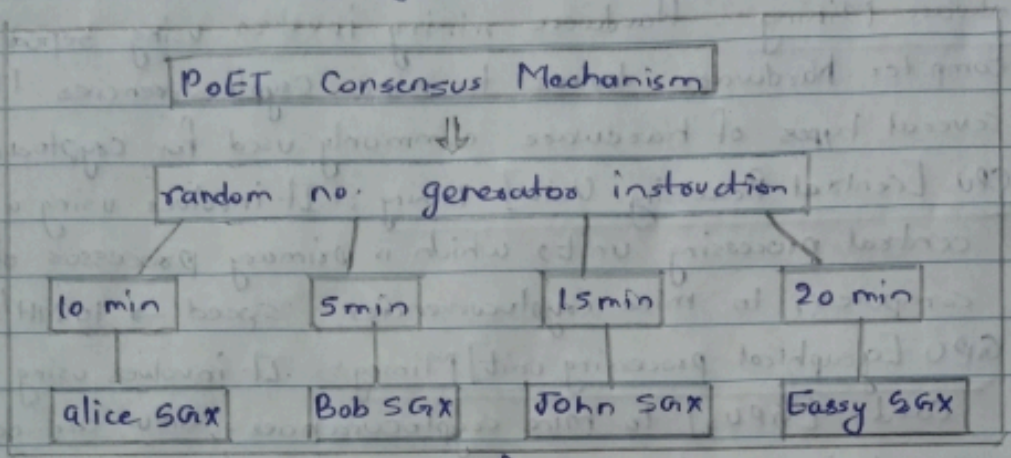
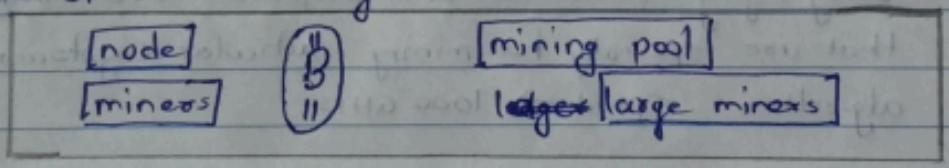


Fig. working of POET

Q.8] Explain types of Bitcoin mining.

→ Bitcoin mining is a process of creating bitcoins by solving extremely complicated maths problems.

Components of bitcoin mining are :-



Types : (i) Software mining (ii) Hardware Mining.

* Software mining :- slw mining refers to the process of using computers slw to mine cryptocurrencies, such as Bitcoin or Ethereum. slw mining can be performed in two primary ways :-

(i) Poor Mining :- Method where multiple miners combine their computing power to increase their chances of successfully mining a block & earning rewards.

(ii) Cloud Mining :- Cloud Mining is a service provided by companies or individuals that allow users to rent or purchase mining between and computational power remotely.

* Hardware Mining :- Hardware mining involves using sepcial specialized computer hardware devices to mine cryptocurrencies like Bitcoin. several types of hardware commonly used for cryptocurrency mining

(i) CPU [Central Processing Unit] Mining :- It involves using a computer's central processing unit, which is primary processor of the computer, to mine cryptocurrencies ; speed → 10 MH/s.

(ii) GPU [graphical processing unit] Mining :- It involves using graphic cards [GPUs] to mine cryptocurrencies. GPUs are desigred for rendering graphics ; speed → 1 GH/s

(iii) FPGA [Field-programmable Gate Array] Mining :- It involes using specialized hardware that can be configured and programmed to peaform specific mining tasks. Speed → 100 GH/s

(iv) ASIC [application-specific integrated circuit] Mining :- It involves using highly specialized and puopose-built hardware devices that are specially for mining particular cryptocurrency's Pow algorithm. Speed → 1000 GH/s.

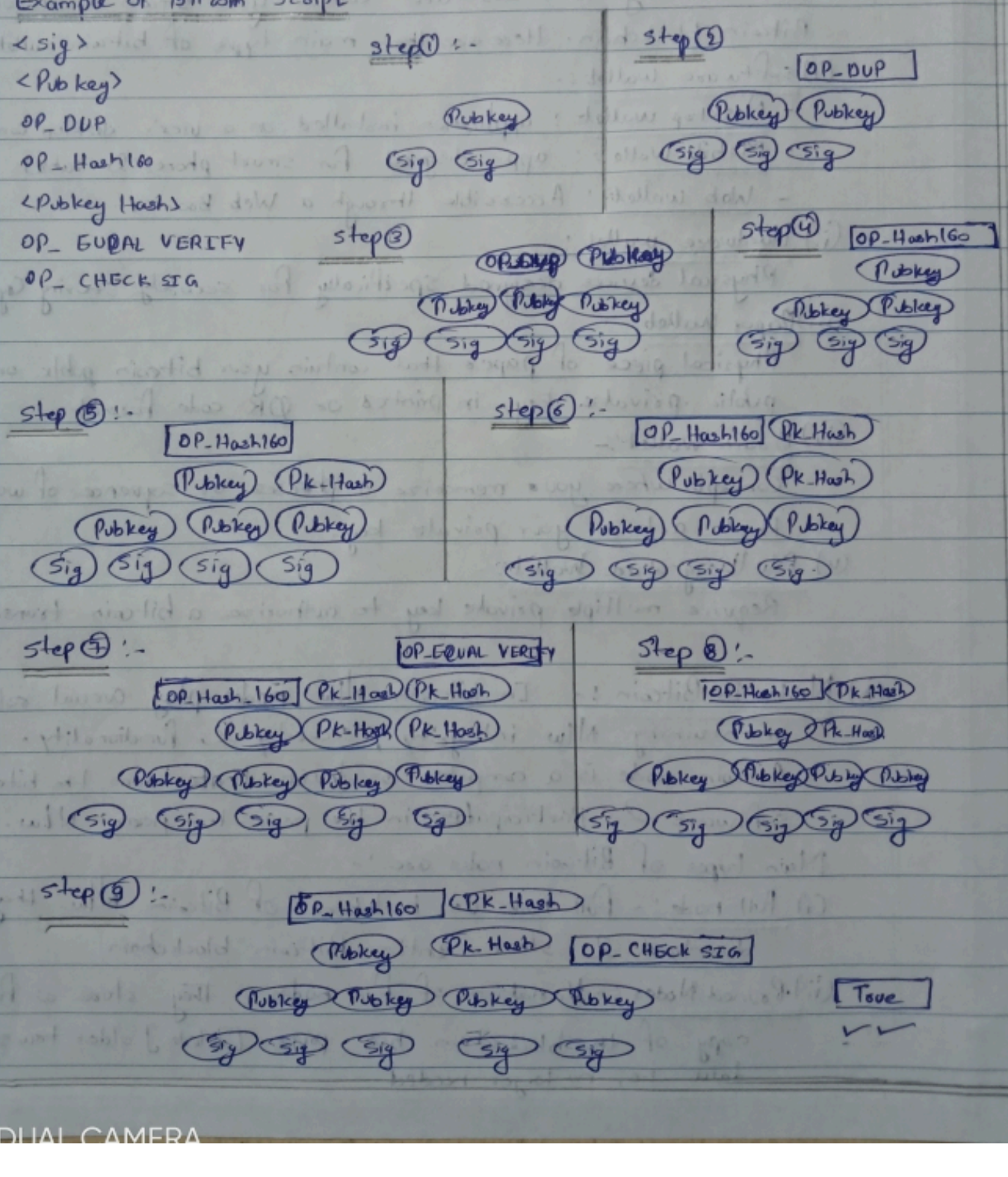
Q.9] Write a short note on Bitcoin script with example.

→
- A bitcoin scoipt is a programming language is used to validate a transaction.
- Forth' is used for script
- It supports :
  - → cryptographic operation.
  - → conditional statements [if & if-else]
  - → stack based programming, processed left to right.

- So, it is list of instruction recorded with each transaction, and it describes how the next person can gain access to the Bitcoin if that person wants to spend them.

Example of Bitcoin Script

<sig>
<Pubkey>
OP_DUP
OP_Hash160
<Pubkey Hash>
OP_EQUAL VERIFY
OP_CHECK SIG



Q.10] Explain

→ (a) Wallet in Bitcoin :- Bitcoin wallet is a digital tool that allows users to store, manage and interact with their bitcoin holdings. It serves as a secure way to stop private key stores, which are necessary to access and control your Bitcoin assets on the Bitcoin blockchain. Here are the main types of bitcoin wallets :-

(i) Software Wallet :-
- Desktop wallets : application installed on a user's desktop computer
- Mobile Wallet : apps disigned for smart phones & tablets
- Web wallet : Accessible through a Web Browser.

(ii) Hardware wallet :
Physical devices designed specifically for securely storing Cryptocurrencies

(iii) Paper Wallet :-
Physical piece of papers that contain your bitcoin public or public private keys in printed or QR code format.

(iv) Brain Wallet :-
Concept where you memorize a phrases or sequence of words that act as your private key.

(v) Multisignature Wallet :-
Require multiple private key to authorize a bitcoin transaction.

(b) Nodes in Bitcoin :- In bitcoin Nlw, nodes play crucial role in maintaining Nlw integrity, security & functionality. A Bitcoin node is a computer or device that runs the bitcoin software & participates in the peer-to-peer Nlw.

Main types of Bitcoin nodes are :-

(i) full node :- full node are backbone of Bitcoin Nlw they download & store entire Bitcoin blockchain.

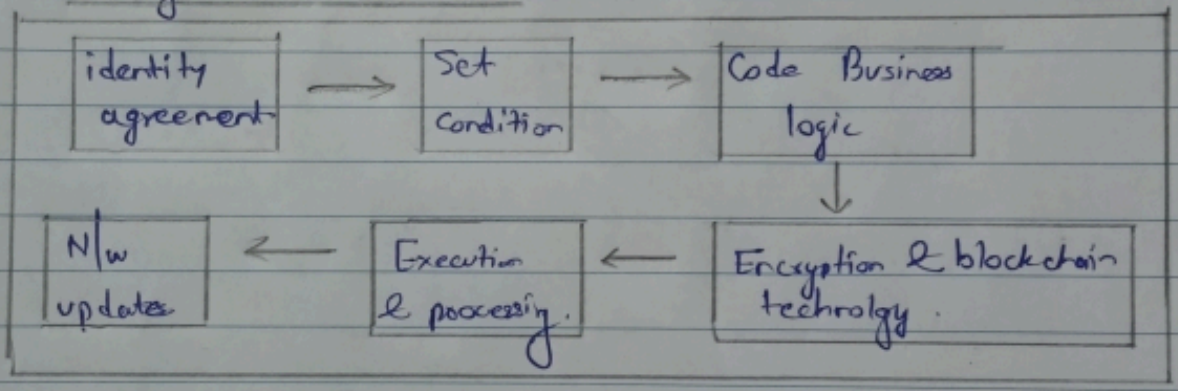
(ii) Pruned Nodes :- Variation of full nodes. They store a full copy of the blockchain but prune [delete] older transaction data i.e, no longer needed.

(iii) Mining node :- responsible for creating new blocks.
(iv) Lightweighted node [SPV nodes] :- only download & verify block headers & subset.
(v) Economic node :- operated by individuals, business or org. with interest.

Q.11] Write a short note on smart contract.

→ • A smart contract is a computer program working on top of blockchain
• Smart contracts are simply programs stored on a blockchain that run when predetermined condition are met.

* Working of smart contract :-



* Characteristics of smart Contracts :-
(i) Distributed -> point-to-point connection.
(ii) deterministic -> predefined => accordingly transaction works.
(iii) immutable -> can't change
(iv) Verify itself & execute itself => auto executable -> condition checked.
(v) Customizable ~~&~~ transparent => decision when initializing
(vi) lower transaction cost => absence of third party.
(vii) Provide a high degree of seccurity -> immutable.
(viii) Transparent