LAB TASK-1

AIM: To show demo hash, block, blockchain, distributed blockchain, tokens, coin bases before and after mining and verification

DESCRIPTION:

Elements of distributed computing:

A distributed system is a system whose components are located on different networked computers, which communicate and coordinate their actions by passing messages to one another. The three basic components of a distributed system include primary system controller, system data store, and database. In a non- clustered environment, optional components consist of user interfaces and secondary controllers.

1. Primary system controller

The primary system controller is the only controller in a distributed system and keeps track of everything. It's also responsible for controlling the dispatch and management of server requests throughout the system. The executive and mailbox services are installed automatically on the primary system controller. In a non- clustered environment, optional components consist of a user interface and secondary controllers.

2. Secondary controller

The secondary controller is a process controller or a communications controller.

It's responsible for

regulating the flow of server processing requests and managing the system's translation load. It also governs communication between the system and VANs or trading partners.

3. User-interface client

The user interface client is an additional element in the system that provides users with important system information. This is not a part of the clustered environment, and it does not operate on the same machines as the controller. It provides functions that are necessary to monitor and control the system.

1. System datastore

Each system has only one data store for all shared data. The data store is usually on the disk vault, whether clustered or not. For non-clustered systems, this can be on one machine or distributed across several devices, but all of these computers must have access to this datastore.

2. Database

In a distributed system, a relational database stores all data. Once the data store locates the data, it shares it among multiple users. Relational databases can be found in all data systems and allow multiple users to use the same information simultaneously.

Elements of cryptography:

Blockchain technology relies heavily on cryptography to secure and maintain the integrity of the distributed ledger. Some key elements of cryptography used in blockchain include:

HASH: A hash is a function that meets the encrypted demands needed to secure information. Hashes are of a fixed length, making it nearly impossible to guess the hash if someone was trying to crack a blockchain.

\
The same data will always produce the same hashed value. Hashes are one of the backbones of the blockchain network.

Digital signatures: Digital signatures are used to verify the authenticity of a message or transaction. In blockchain, each transaction is digitally signed by the sender's private key, which verifies that the transaction is coming from the intended sender. Digital signatures also provide non-repudiation, meaning that the sender cannot deny having sent the transaction.

Public-key cryptography: Public-key cryptography, also known as asymmetric cryptography, is used to generate a pair of keys: a public key and a private key.
The public key can be shared with anyone and is used to encrypt messages, while the private key is kept secret and is used to decrypt messages. In blockchain, public keys are used as addresses to identify users and their transactions.

Zero-knowledge proofs: Zero-knowledge proofs allow a user to prove possession of a specific piece of information, without revealing the actual information. In blockchain, zero-knowledge proofs are used to enhance privacy and protect the identity of users.

Multi-signature: Multi-signature, also known as multi-sig, is a method of requiring more than one key to authorize a transaction. In blockchain, multi-sig is used to provide an additional layer of security for transactions, by requiring multiple parties to sign off on a transaction before it is executed.

Merkle tree: Merkle tree is a data structure that allows for efficient and secure verification of the contents of large datasets. In blockchain, Merkle tree is used to organize transactions in a block, and to prove the integrity of the transactions without the need to transmit the entire block.

All of these elements of cryptography work together to ensure the security and integrity of the blockchain network. The use of encryption, digital signatures, and public-key cryptography ensures that only authorized parties can access and make changes to the blockchain. Hash functions and Merkle trees ensure the integrity of the data. And zero-knowledge proofs and multi-sig provide added security and privacy for users.

MINING:
Blockchain "mining" is a metaphor for the computational work that nodes in the network undertake in hopes of earning new tokens. In reality, miners are essentially getting paid for their work as auditors. They are doing the work of verifying the legitimacy of Bitcoin transactions.

Cryptographic Hashing:

Cryptographic hashing is a technique used to create a fixed-length output, known as a hash, from a variable-length input. The hash is a unique digital fingerprint of the input and can be used for a variety of purposes, such as data integrity and data security. Cryptographic hashing is a key component of many modern cryptographic systems, including blockchain technology.

Properties Of Crytographic Hashing:

- One of the key properties of cryptographic hashing is that it is a one-way function, meaning that it is easy to compute the hash of an input, but it is computationally infeasible to determine the original input from the hash. This makes it an effective tool for creating digital fingerprints of data, as the hash can be used to verify the integrity of the data without revealing the data itself.
- Another important property of cryptographic hashing is that it is deterministic, meaning that the same input will always produce the same hash. This makes it possible to verify the integrity of data by comparing the hash of the original data to the hash of the data that is being received.
- Cryptographic hashing is also designed to be collision-resistant, meaning that it is computationally infeasible to find two inputs that produce the same hash. This is important because it ensures that each hash value is unique and cannot be used to forge a different message.
- Cryptographic hashing is also very fast and efficient, making it practical to use in large-scale systems such as blockchain. This is important because it allows the system to quickly and efficiently verify the integrity of large amounts of data without slowing down the overall system.
- Some of the key algorithms that are commonly used for cryptographic hashing include SHA-256, SHA-512, and MD5. These algorithms have been widely studied and have been found to be secure and collision-resistant, making them suitable for use in a wide range of applications.

Cryptographic Hashing Algorithms:

MD5:

The “Message Digest 5” algorithm was made in 1991 and produces a 128-bit hash value.
MD5 has been found to have severe vulnerabilities and is not recommended for security based uses. This is because it is possible to find hash collisions with MD5 pretty easily.
Despite its weaknesses, MD5 is still applied for non critical tasks due to its low computational resource use compared to modern hashing algorithms.

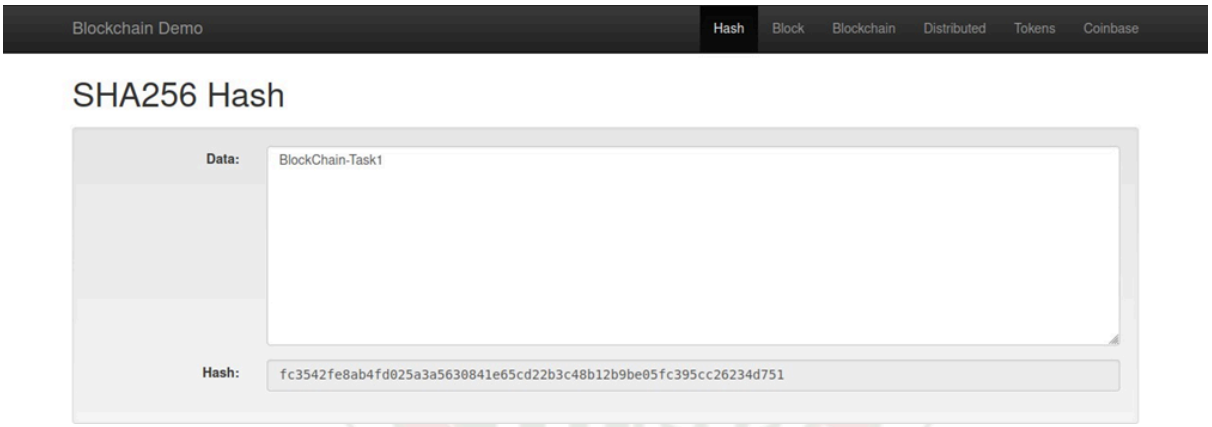
SHA:

The “Secure Hash Algorithm”s are a family of cryptographic hash functions published as a standard by the National Institute of Standards and Technology (NIST).
These algorithms are the current standard for generating secure hashes and are widely used.
There are a wide range of SHA algorithms with varying sizes and complexities based on the requirement.
SHA-0, SHA-1, SHA-2, SHA-3 are a few.

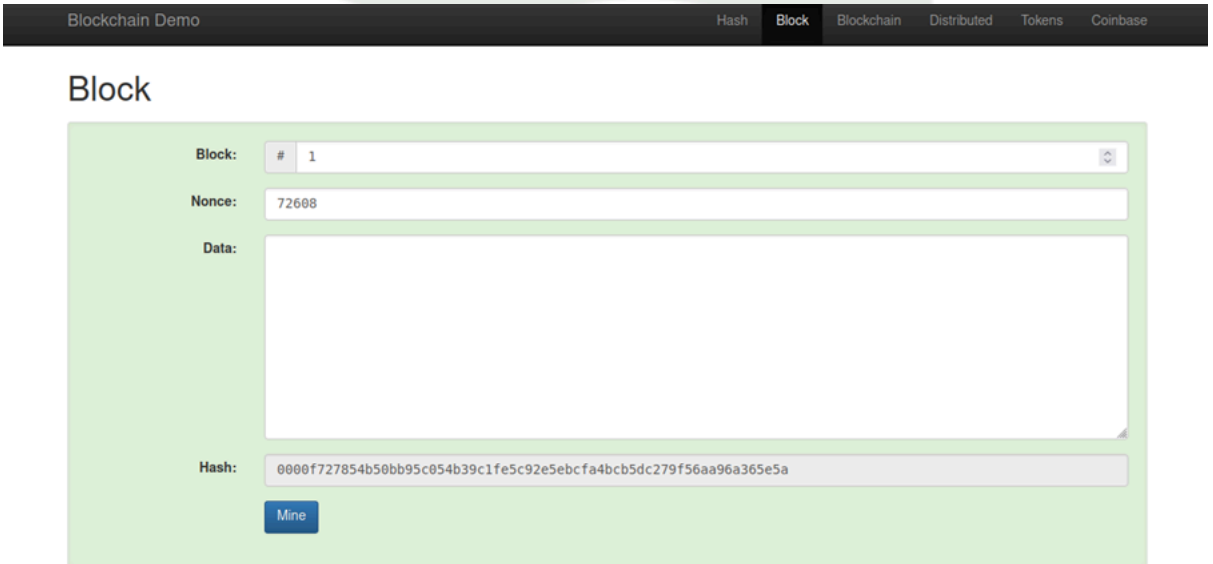
RIPEMD:

The "RIPE Message Digest" is a family of cryptographic hash functions developed by the RIPEMD Consortium originally in the 1992 and later in 1996. While RIPEMD functions are less popular than SHA-1 and SHA-2, they are used, among others, in Bitcoin and other cryptocurrencies based on Bitcoin.

HASH:

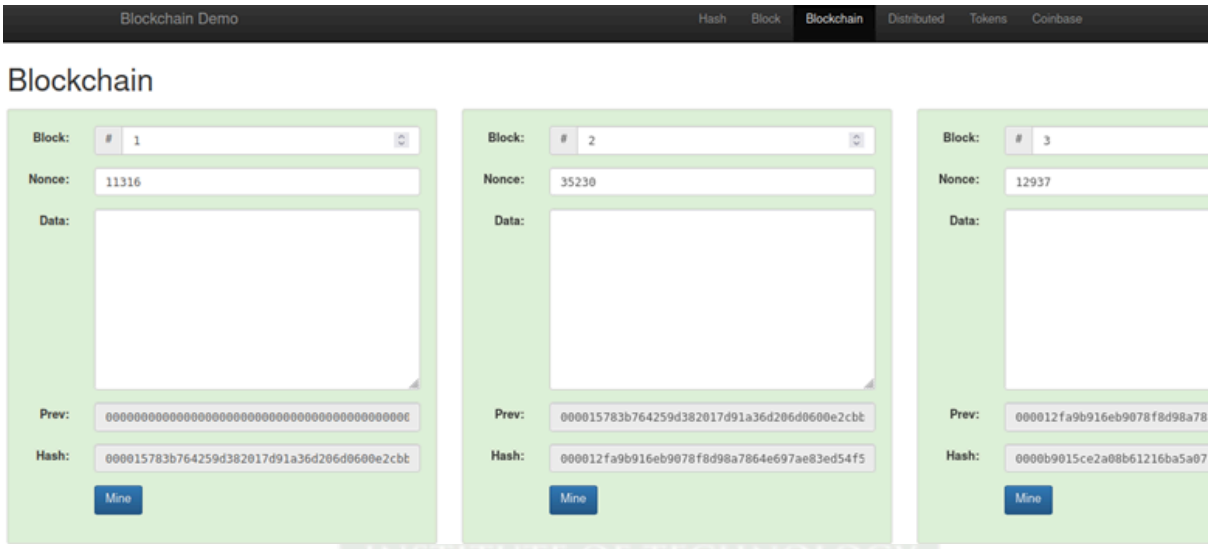


BLOCK: Acting as a database, transaction data is permanently recorded, stored and encrypted onto the "blocks" that are then "chained" together.

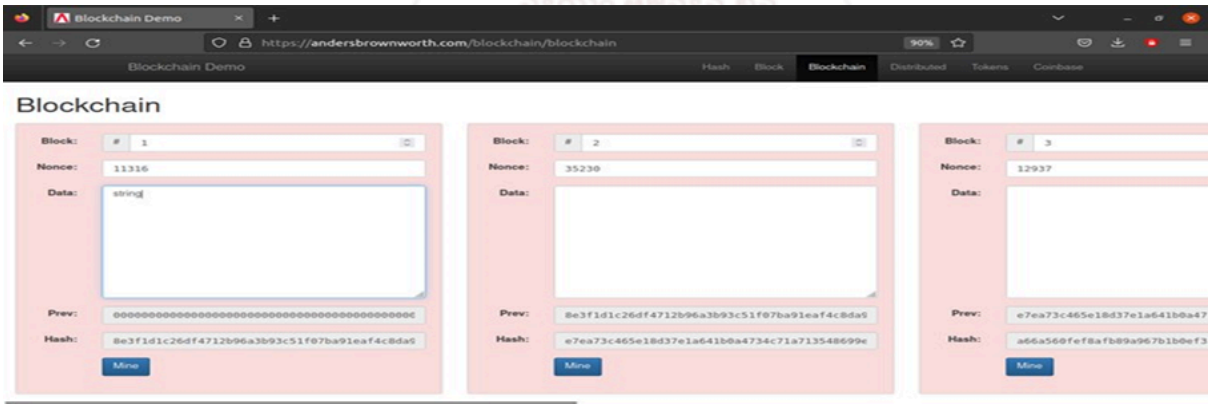


BLOCKCHAIN: The goal of blockchain is to allow digital information to be recorded and distributed, but not edited. In this way, a blockchain is the foundation for immutable ledgers, or records of transactions that cannot be altered, deleted, or destroyed.

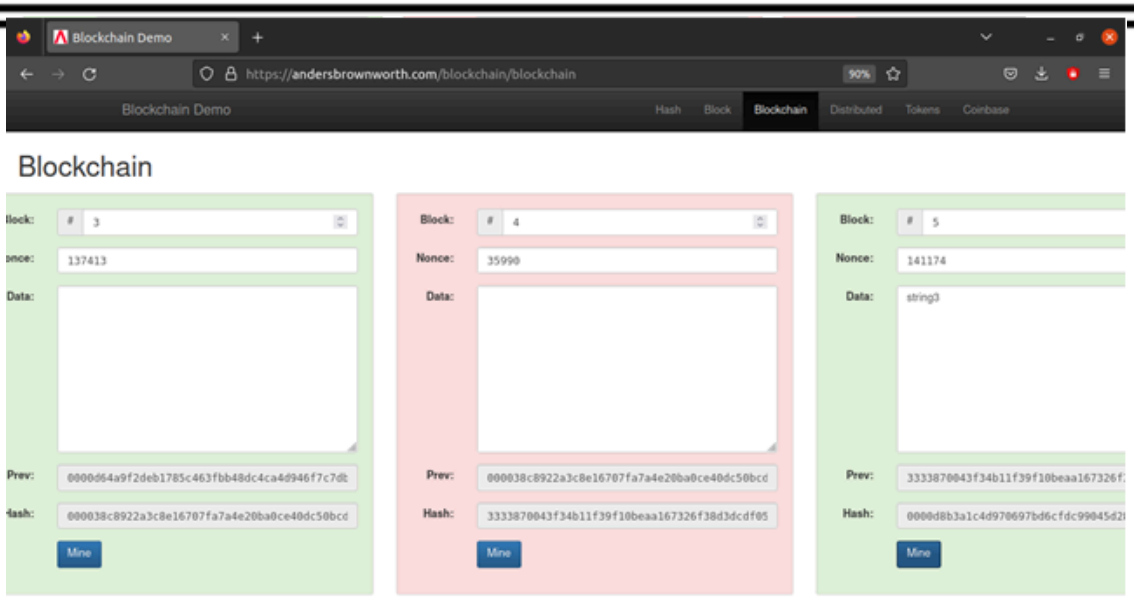
Blockchain is a time-stamped decentralized series of fixed records that contains data of any size is controlled by a large network of computers that are scattered around the globe and not owned by a single organization. Every block is secured and connected with each other using hashing technology which protects it from being tampered by an unauthorized person.



before mining:

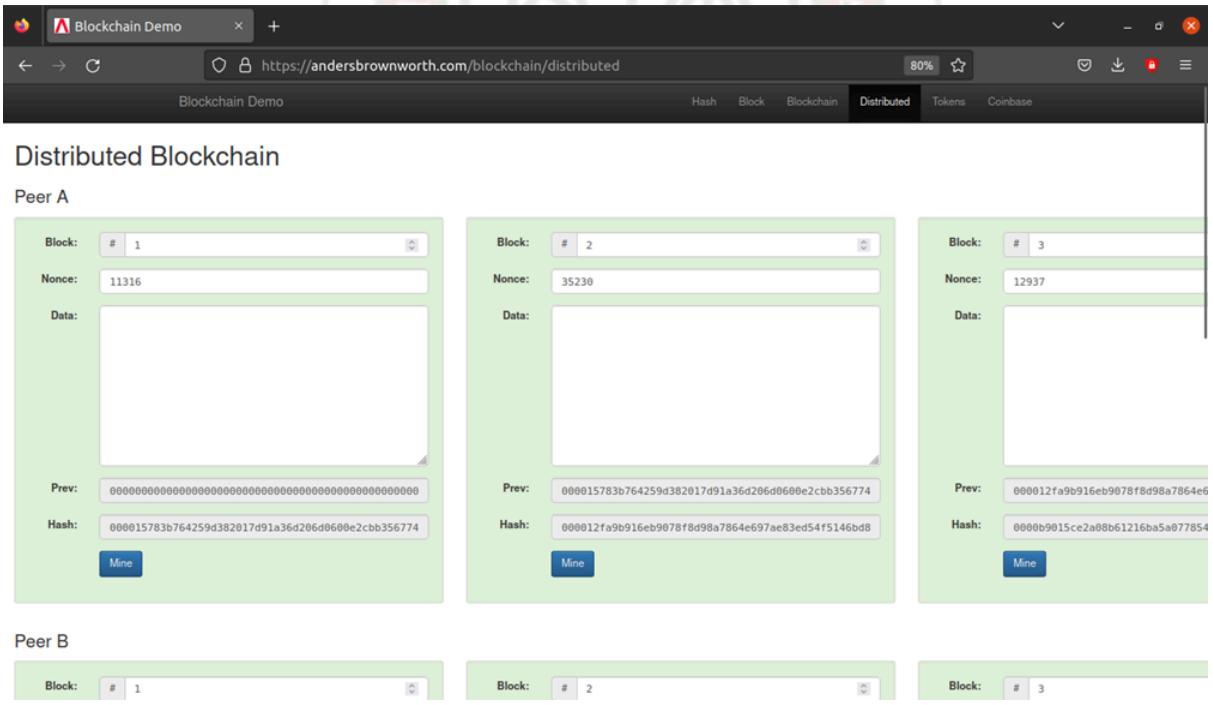


after mining:

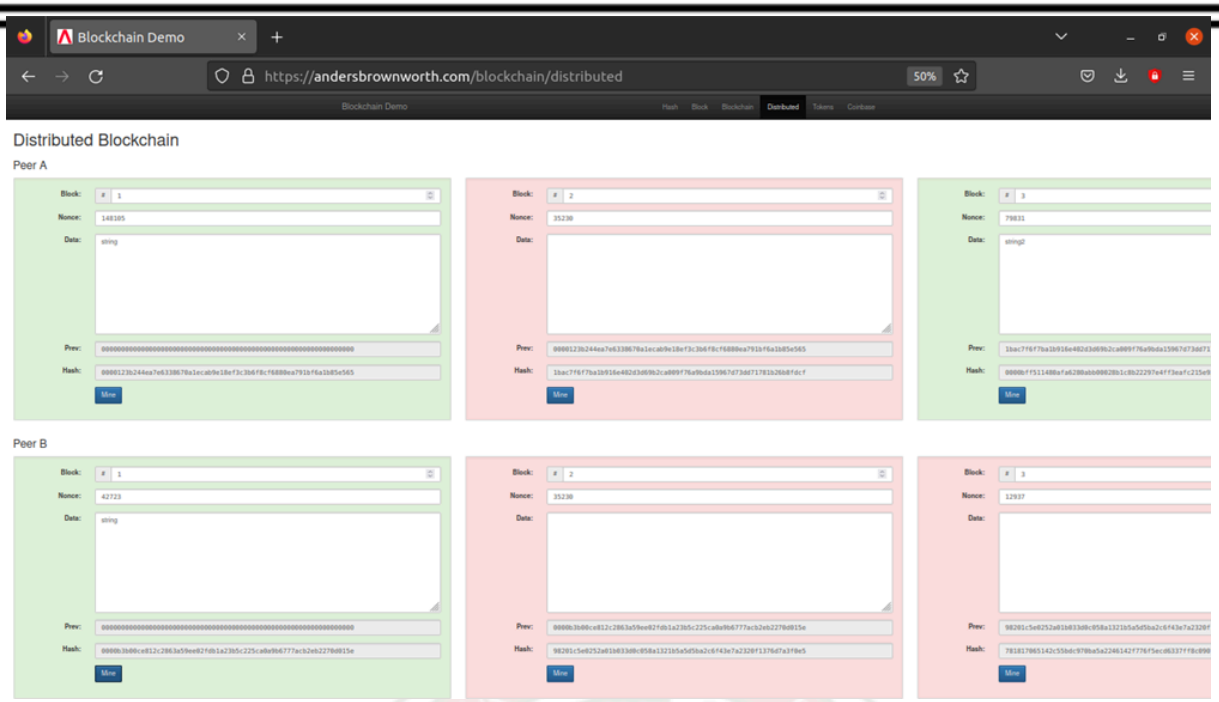


DISTRIBUTED:

Blockchain is one type of a distributed ledger. Distributed ledgers use independent computers (referred to as nodes) to record, share and synchronize transactions in their respective electronic ledgers (instead of keeping data centralized as in a traditional ledger).

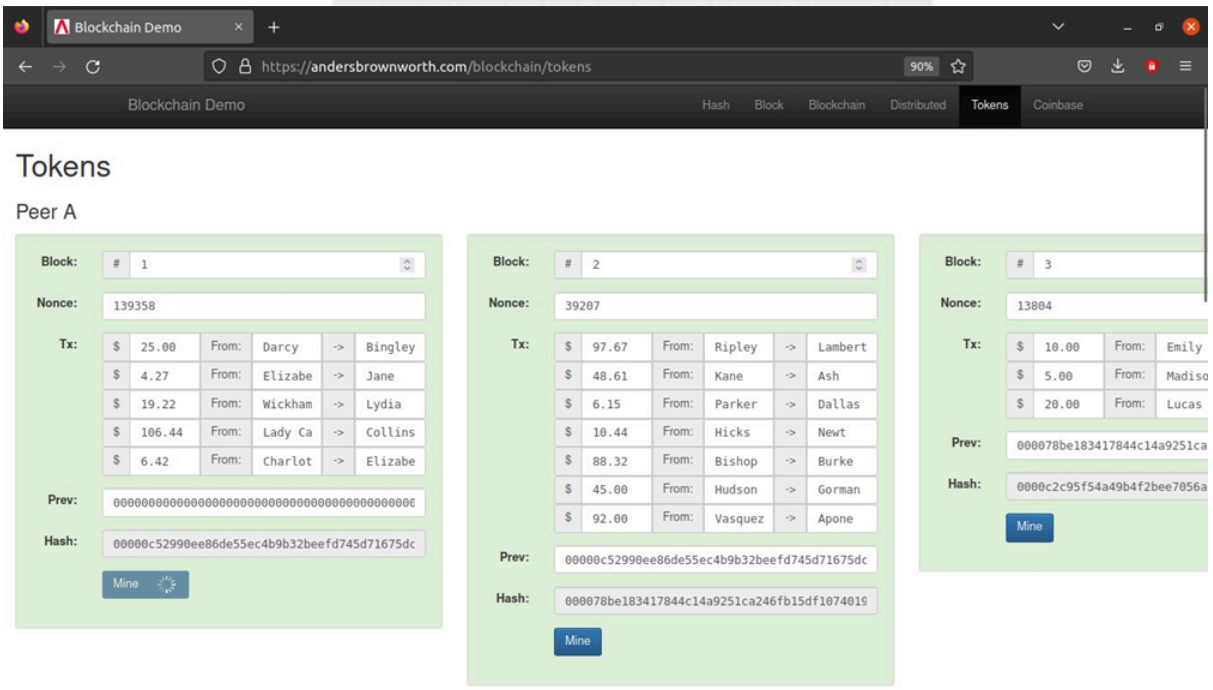


after mining:

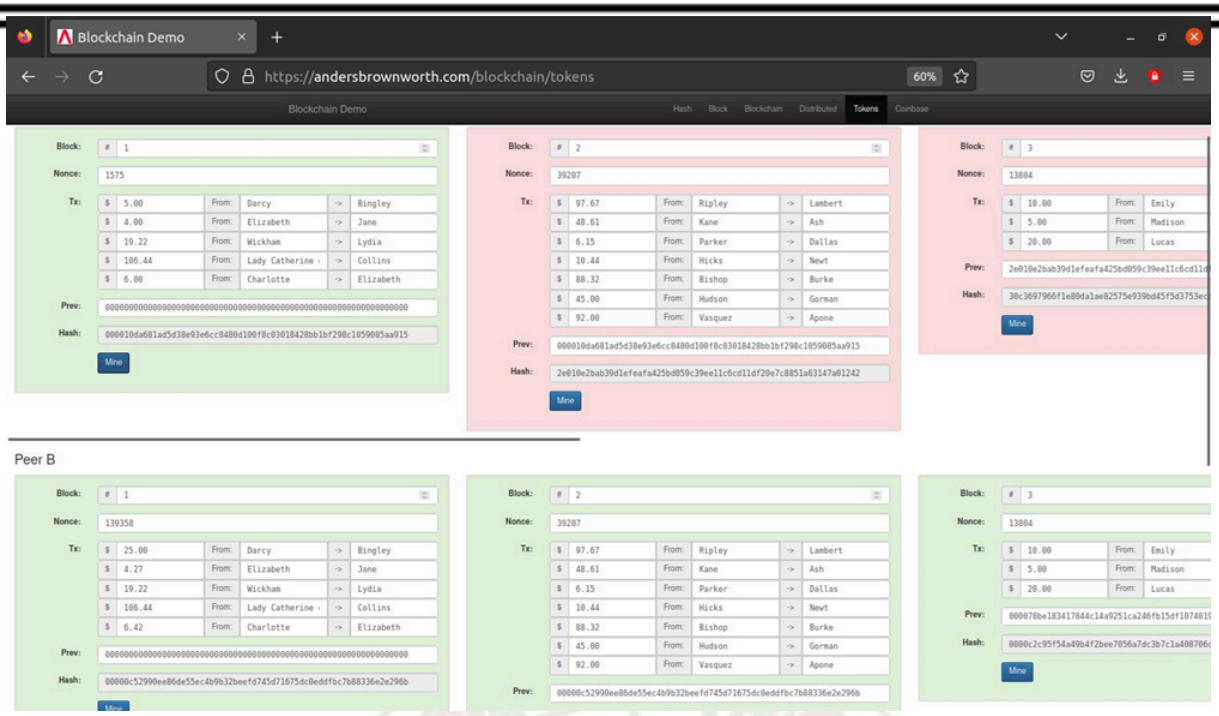


TOKENS:

A crypto token is a representation of an asset or interest that has been tokenized on an existing cryptocurrency's blockchain. Crypto tokens and cryptocurrencies share many similarities, but cryptocurrencies are intended to be used as a medium of exchange, a means of payment, and a measure and store of value.



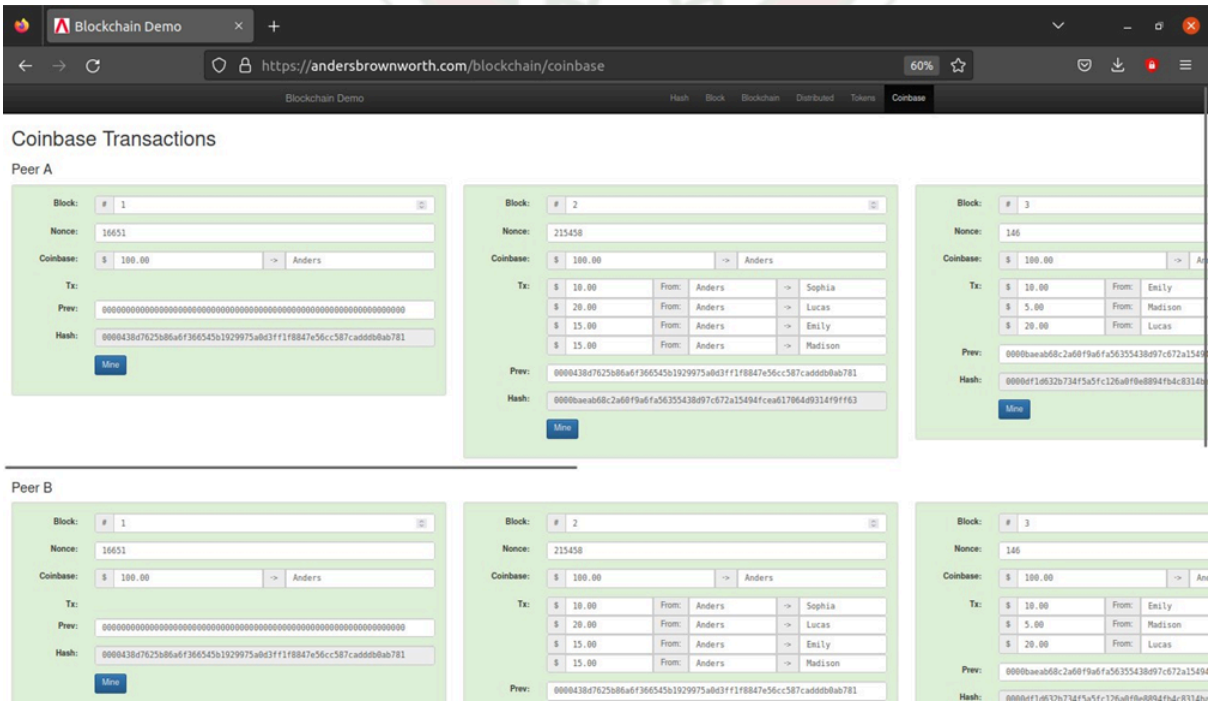
after mining:

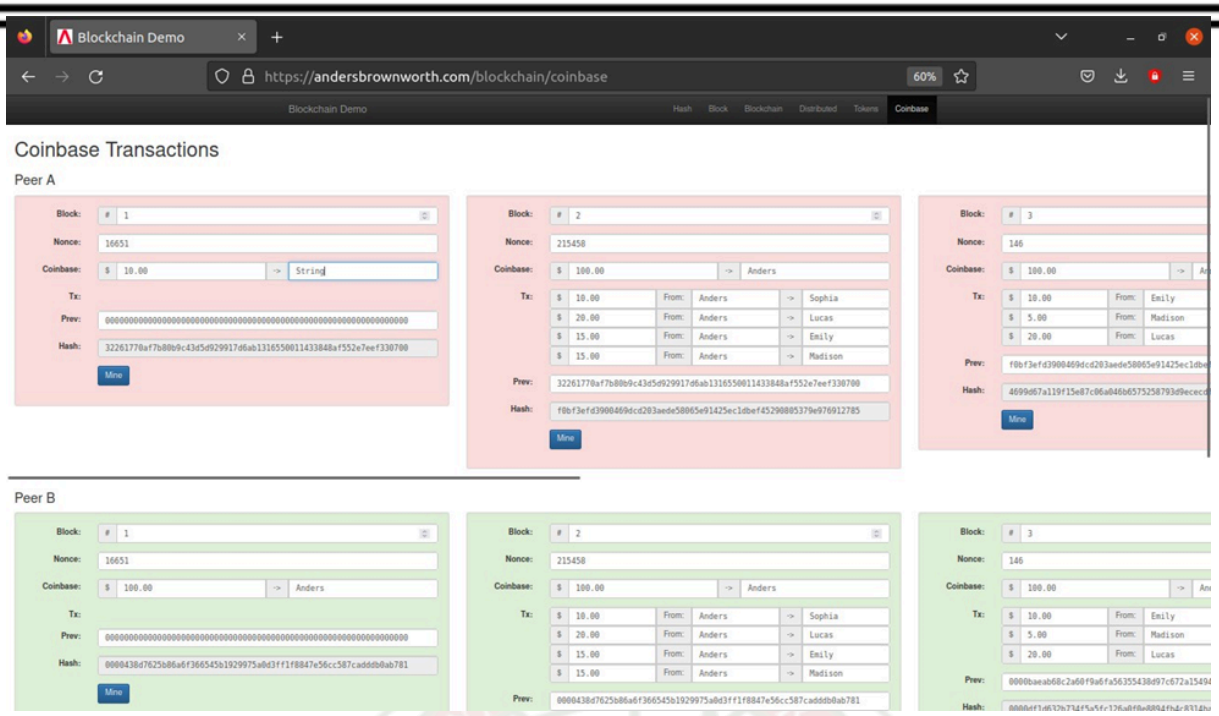


COINBASE:

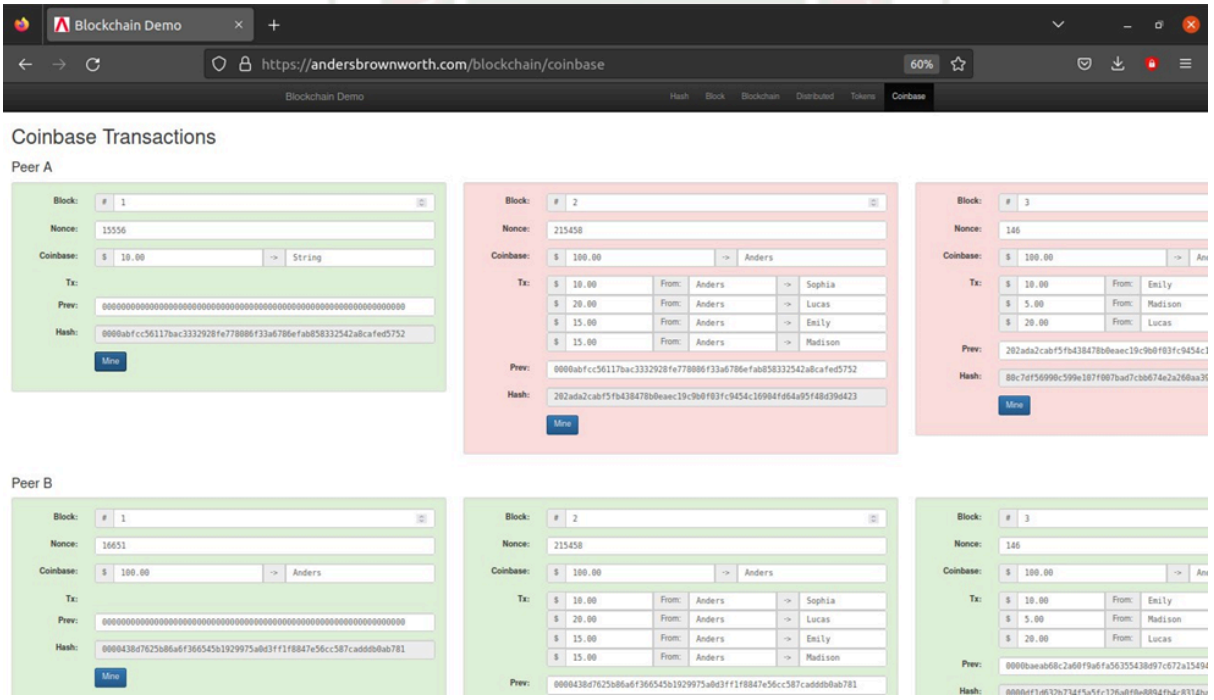
A coinbase transaction is the first transaction in a block. It is a unique type of bitcoin transaction that can be created by a miner. The miners use it to collect the block reward for their work and any other transaction fees collected by the miner are also sent in this transaction.

before mining:





after mining:



LAB TASK 2

AIM: To Show demo of BlockChain Public / Private Key Pairs,Signatures and Transaction

DESCRIPTION:

Encryption:

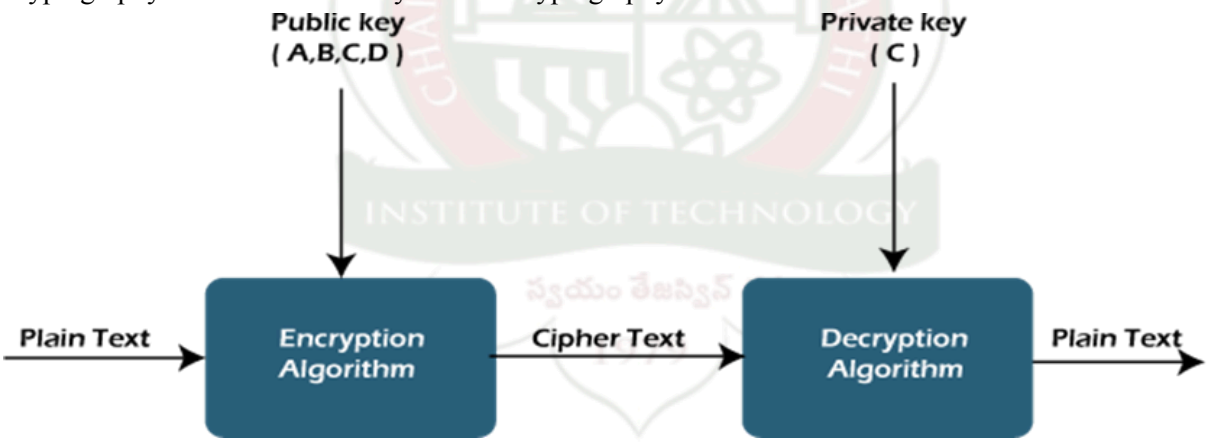
The process of changing the plaintext into the ciphertext is referred to as encryption. The encryption process consists of an algorithm and a key. The key is a value independent of the plaintext.

The security of conventional encryption depends on the major two factors:

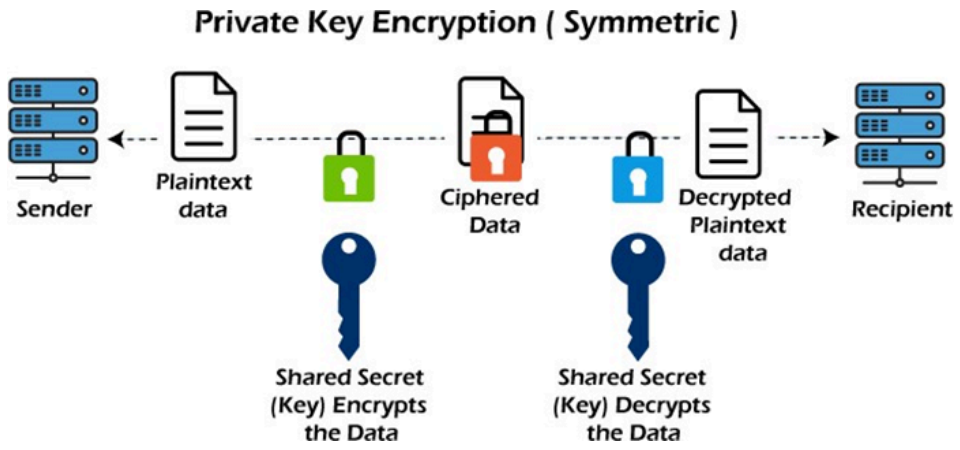
1. The Encryption algorithm
2. Secrecy of the key

Decryption:

The process of changing the ciphertext to the plaintext that process is known as decryption.
It is an encryption technique that uses a pair of keys (public and private key) for secure data communication. In the pair of keys, the public key is for encrypting the plain text to convert it into ciphertext, and the private key is used for decrypting the ciphertext to read the message.
The private key is given to the receiver while the public key is provided to the public. Public Key Cryptography is also known as asymmetric cryptography



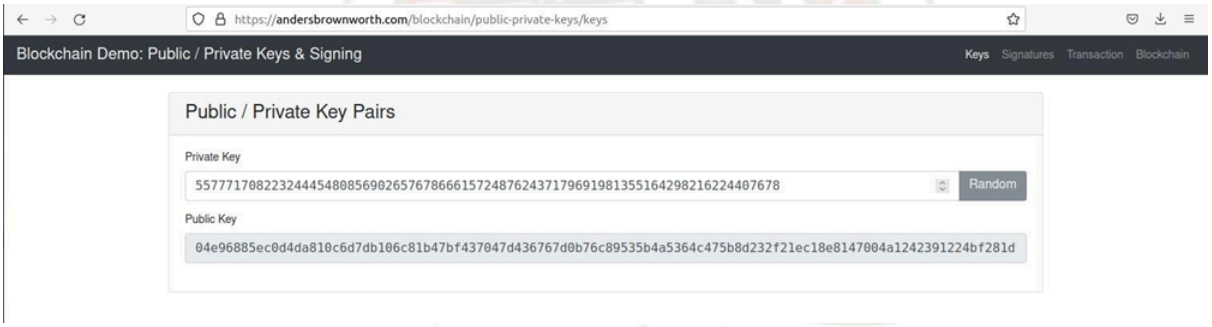
Public-key encryption is slower than secret-key encryption. In secret key encryption, a single shared key is used to encrypt and decrypt the message, while in public-key encryption, different two keys are used, both related to each other by a complex mathematical process. Therefore, we can say that encryption and decryption take more time in public-key encryption.
In private key, the same key (or secret key) is used by both the parties, i.e., the sender and receiver, for Encryption/Decryption technique.



encryption algorithm for encryption, whereas for decryption, the receiver uses this key and decryption algorithm. In Secret Key Encryption/Decryption technique, the algorithm used for encryption is the inverse of the algorithm used for decryption.

Public / Private Key Pairs DEMO:

KEYS:



DIGITAL SIGNATURE:

Digital Signing in Blockchain is a process to verify the user's impressions of the transaction. It uses the private key to sign the digital transaction, and its corresponding public key will help to authorize the sender.

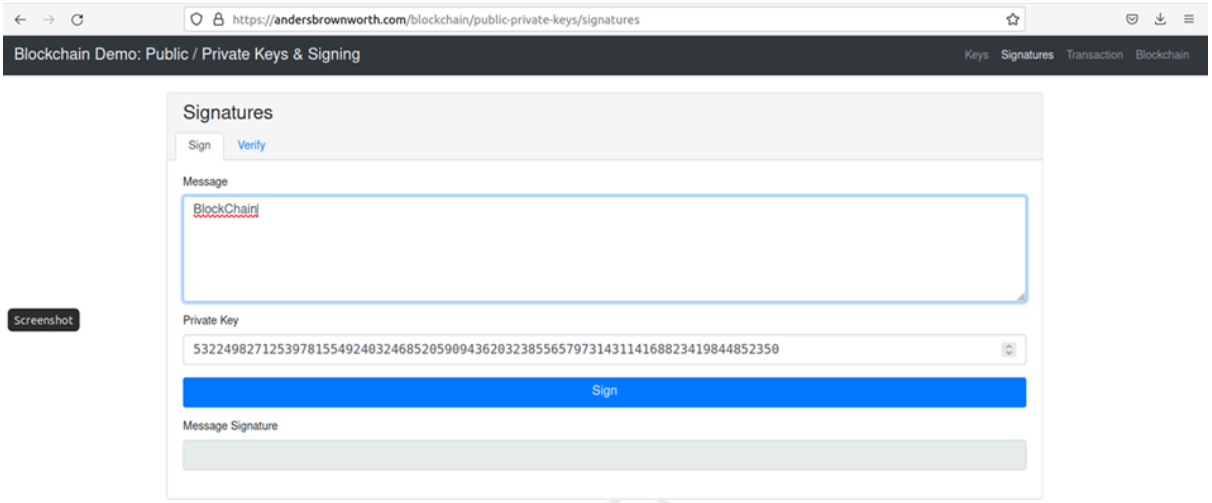
In the case of digital signing, the roles of public keys and private keys are reversed. That means the public keys can decrypt the message, and private keys can encrypt the message.

Authentication: Verifies the identity of the sender who sent the message.

Confidentiality: Ensures the security and privacy of the message.

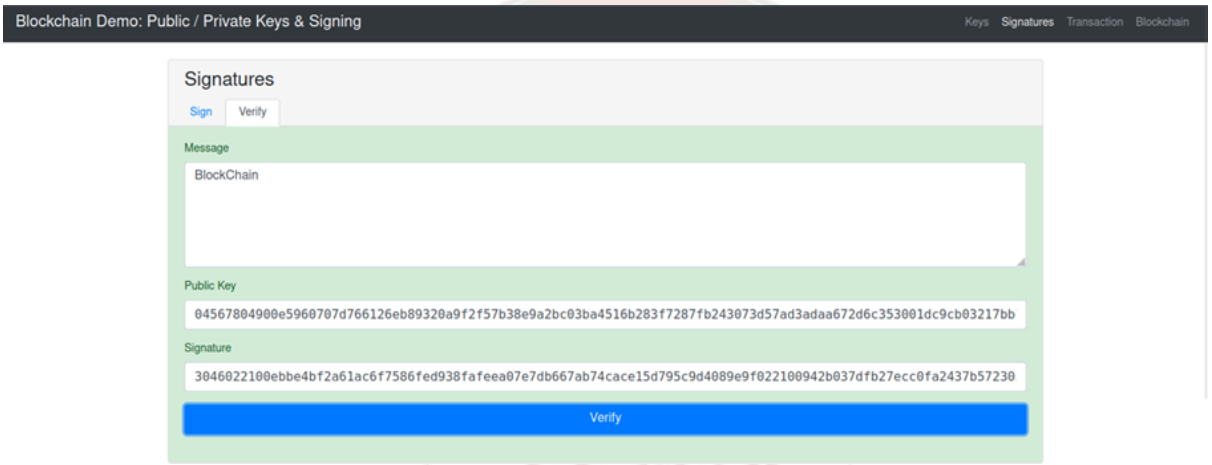
SIGNATURES:

Digital Signing in Blockchain is a process to verify the user's impressions of the transaction. It uses the private key to sign the digital transaction, and its corresponding public key will help to authorize the sender. However, in this way, anyone with the sender's public key can easily decrypt the document.

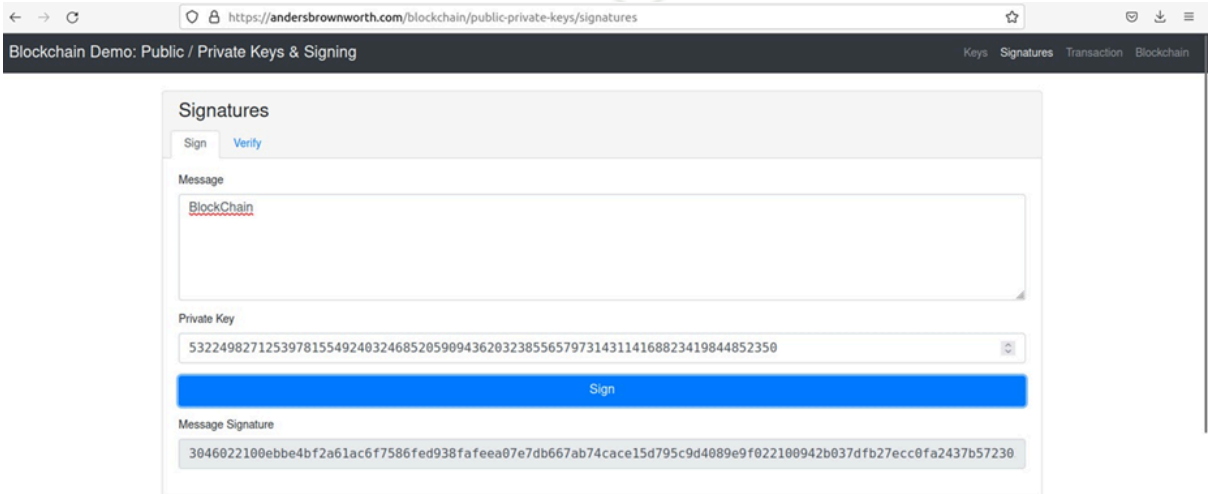


Screenshot

AFTER SIGN:

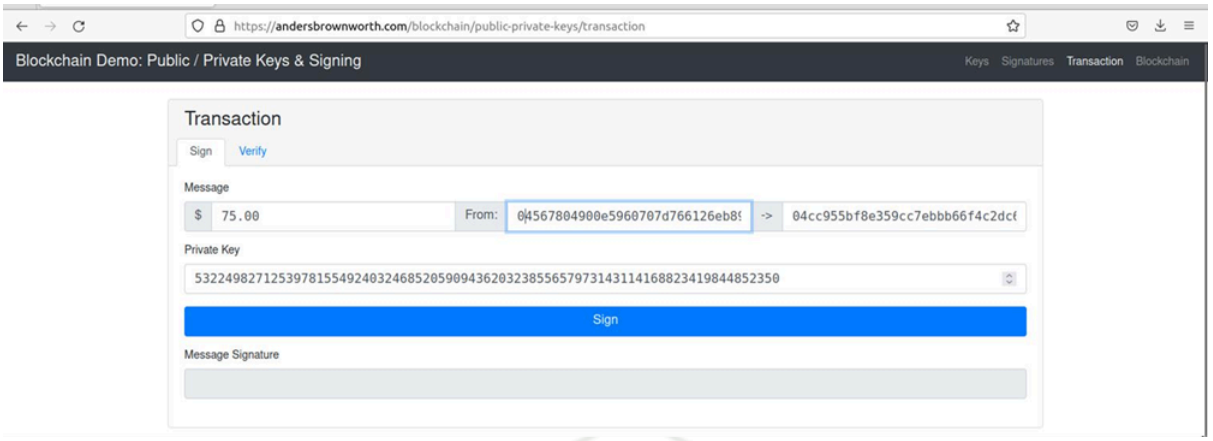


VERIFY:



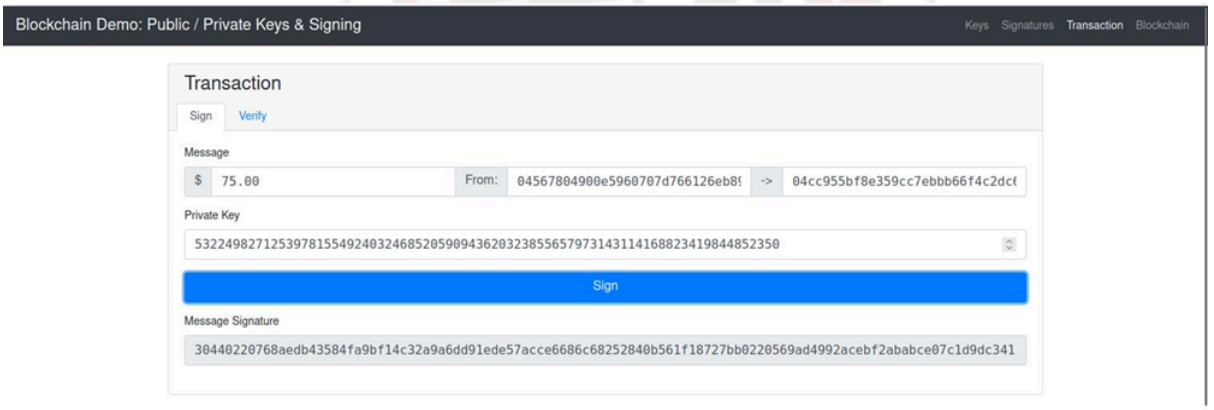
TRANSACTIONS:

For a public blockchain, the decision to add a transaction to the chain is made by consensus. This means that the majority of "nodes" (or computers in the network) must agree that the transaction is

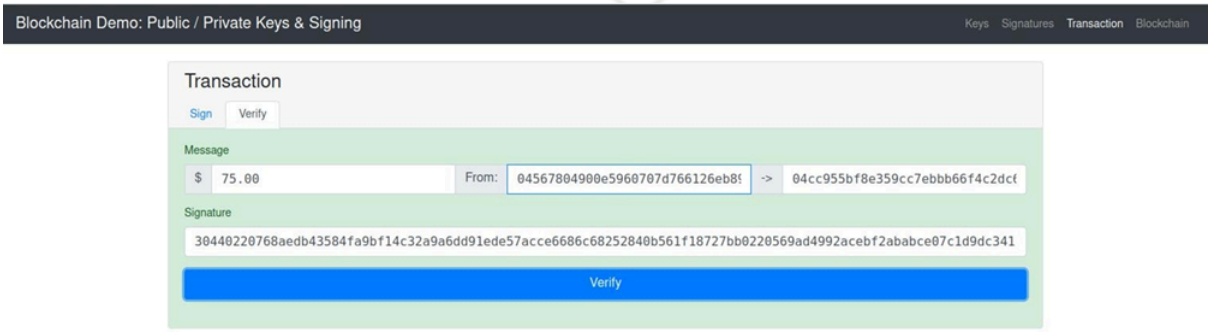


valid. The people who own the computers in the network are incentivised to verify transactions through rewards.

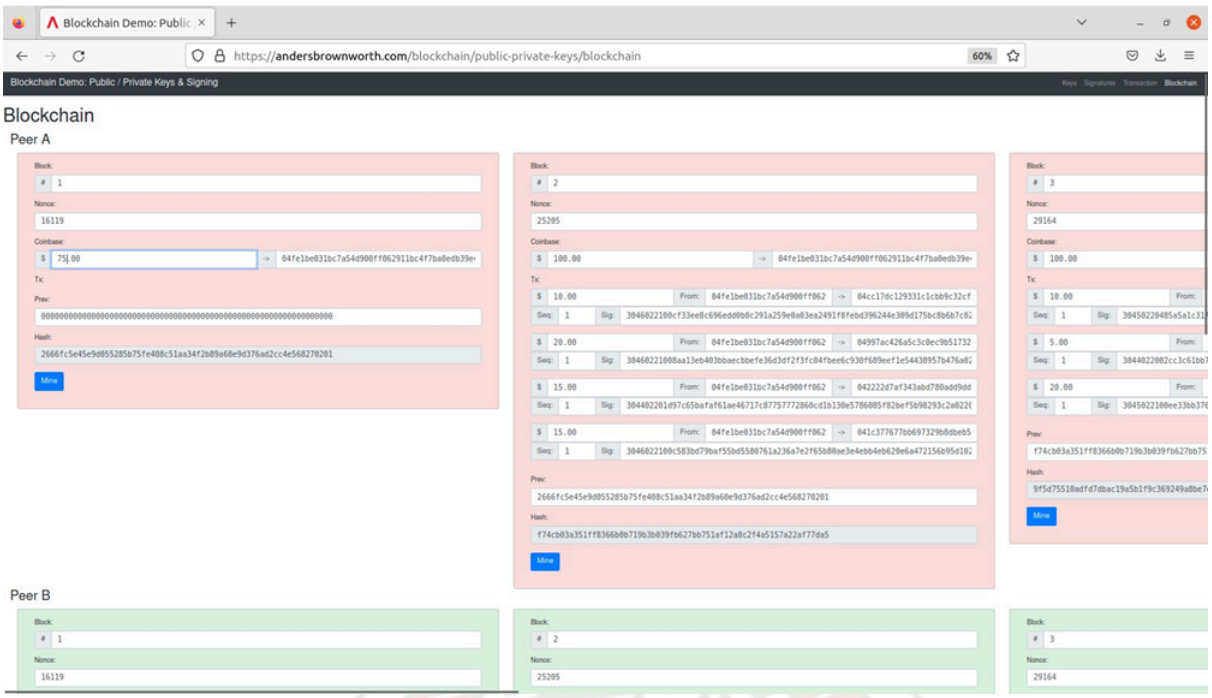
AFTER SIGNING:



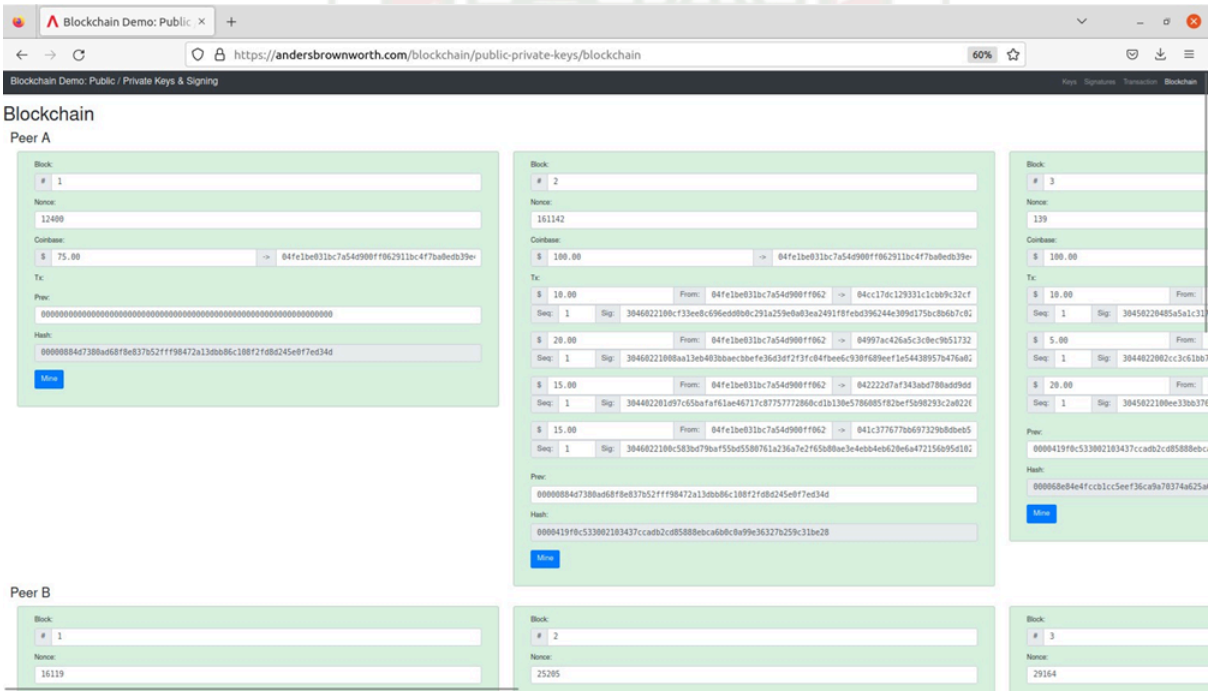
VERIFY:



OUTPUT-- BLOCK CHAIN:



after mining:



AIM: To create a block chain wallet and add ethers in it.

DESCRIPTION:

Blockchain Wallet:

A blockchain wallet is a digital wallet that allows users to store and manage their Bitcoin, Ether, and other cryptocurrencies. Blockchain Wallet can also refer to the wallet service provided by Blockchain A blockchain wallet allows transfers in cryptocurrencies and the ability to convert them back into a user's local currency.

A blockchain wallet is a cryptocurrency wallet that is used to manage cryptocurrencies like Bitcoin and Ethereum. It helps to exchange funds easily and the transactions are more secure as they are cryptographically signed. The privacy and the identity of users are maintained and it provides all the features that are necessary for secure and safe transfers and exchange of cryptocurrencies.

The blockchain wallet stores the private keys and public keys for a transaction. The wallet enables users to sell, and purchase goods using cryptocurrencies.

METAMASK:

MetaMask is a popular cryptocurrency wallet that supports a broad range of Ethereum-based tokens and non-fungible tokens (NFTs) on supported blockchains. While experienced crypto investors may appreciate the speed and simplicity of the wallet, new investors may find it difficult to navigate. In addition, the wallet does not support Bitcoin, making it a turn-off for investors whose primary investment is Bitcoin.

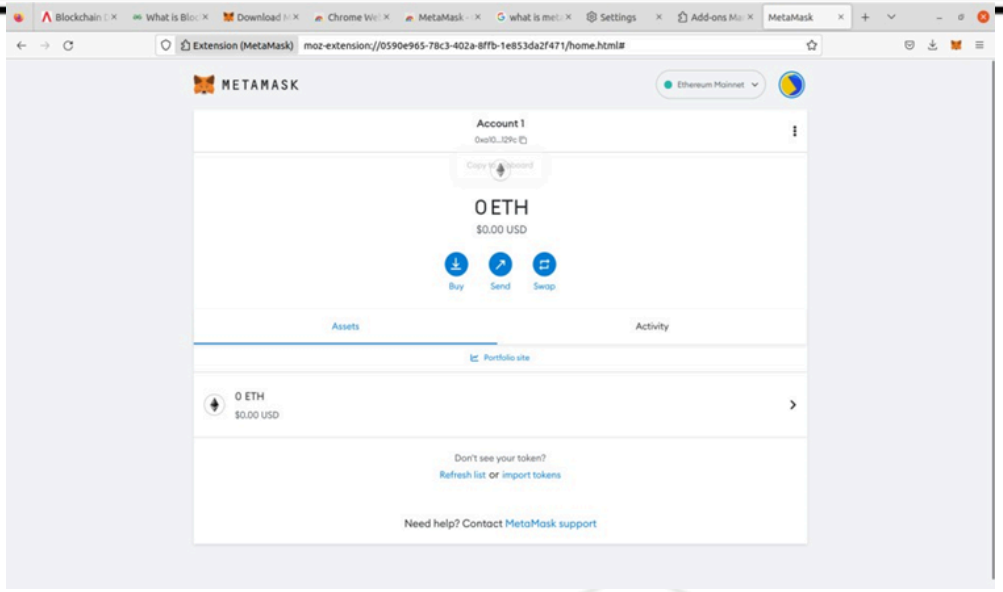
STEPS TO ADD ETHERS IN WALLET:

1. Install METAMASK
2. Go To Settings--> Advanced and turn on the “Show test networks” switch
3. Note the secret phase and WALLET is created
4. Add ethers to it and copy the address 5.Amount of ETHERS are displayed

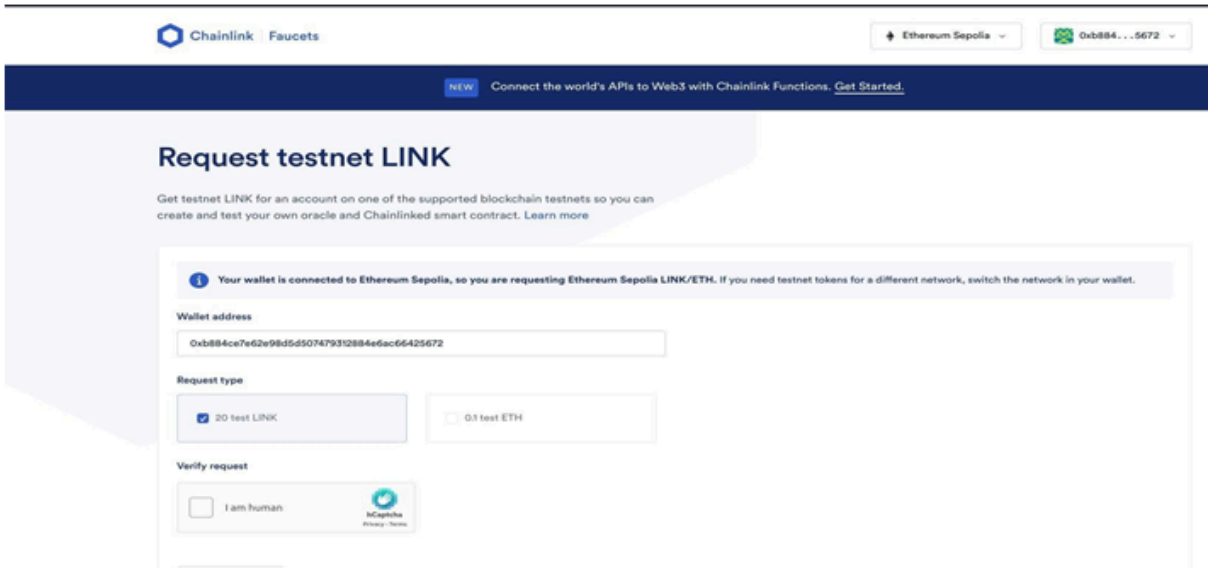
SECRET PHRASE:

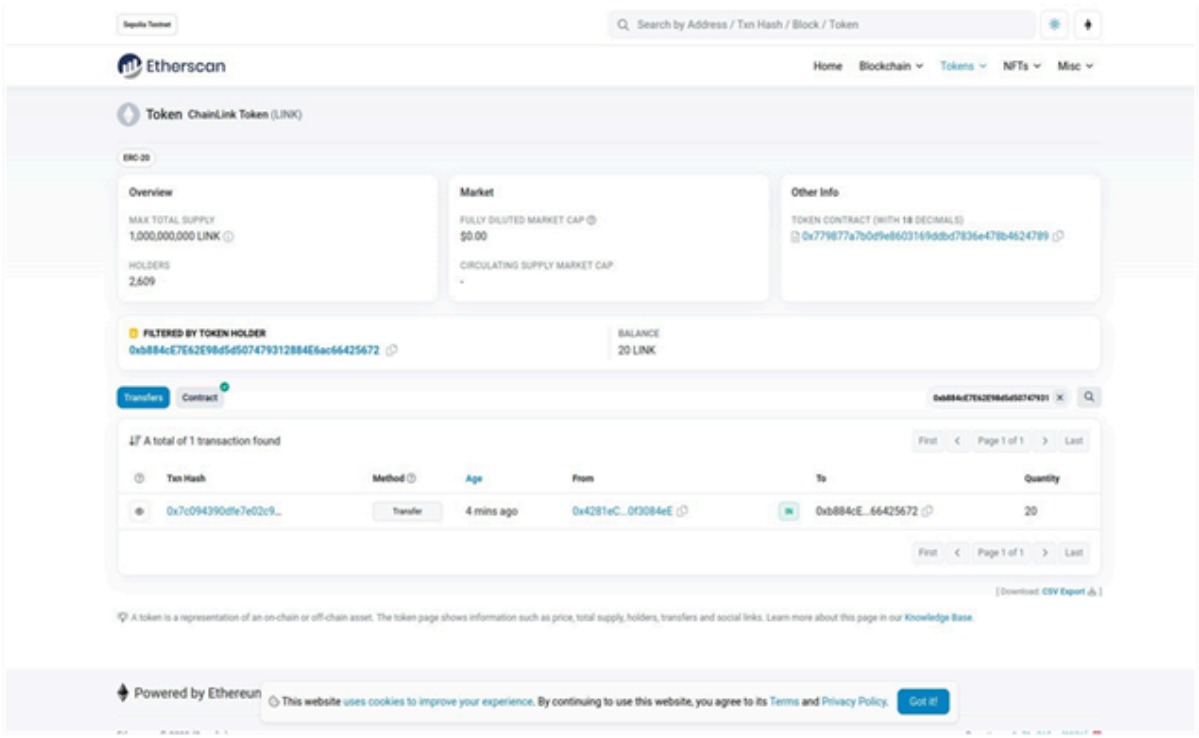
real harvest old benefit kidney slender stove civil civil wet royal citizen

OUTPUT:

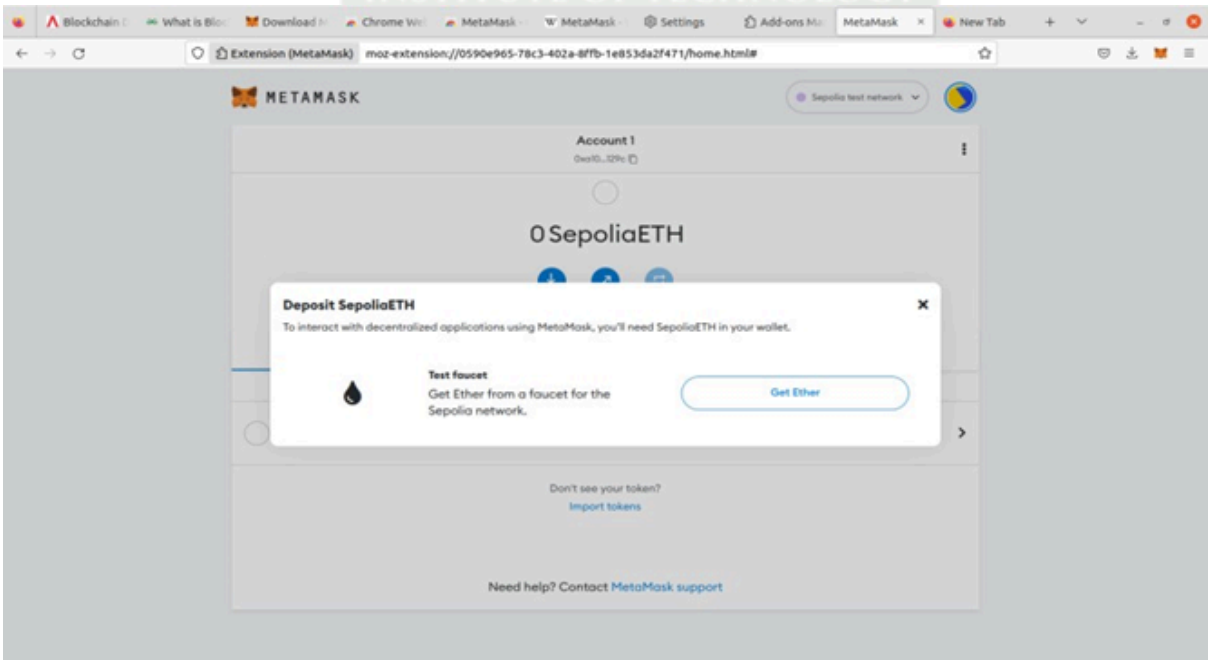


Go to faucets.chain.link, select the type of faucet, connect it to MetaMask wallet and then we can request for 20 tokens which are transferred to our MetaMask wallet whose transaction details are in the image.

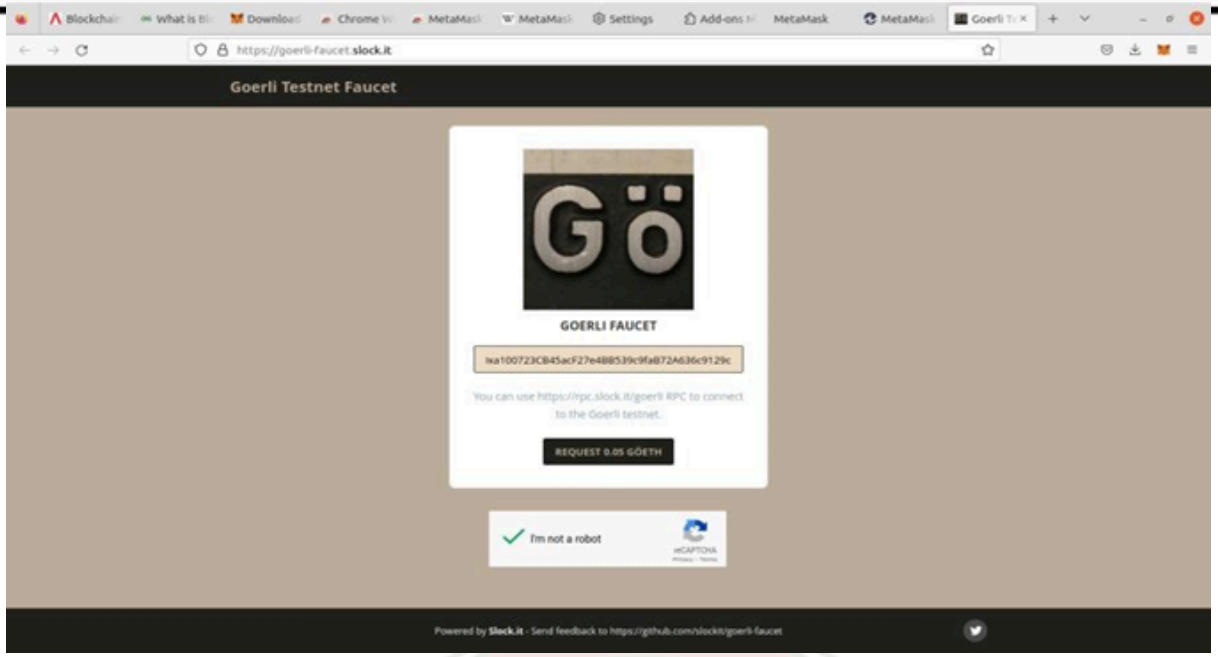




USING SEPHOLIA ETHER:



USING GORELLI FAUCET:


Goerli Testnet Faucet
GOERLI FAUCET
REQUEST 0.05 GÖETH
I'm not a robot

LAB WEEK-3

AIM: TO CREATE SMART CONTRACTS FOR THE FOLLOWING PROPGRAMS

DESCRIPTION:

Smart contracts are simply programs stored on a blockchain that run when predetermined conditions are met. They typically are used to automate the execution of an agreement so that all participants can be immediately certain of the outcome, without any intermediary's involvement or time loss. They can also automate a workflow, triggering the next action when conditions are met.

Smart contracts are digital contracts stored on a blockchain that are automatically executed when predetermined terms and conditions are met

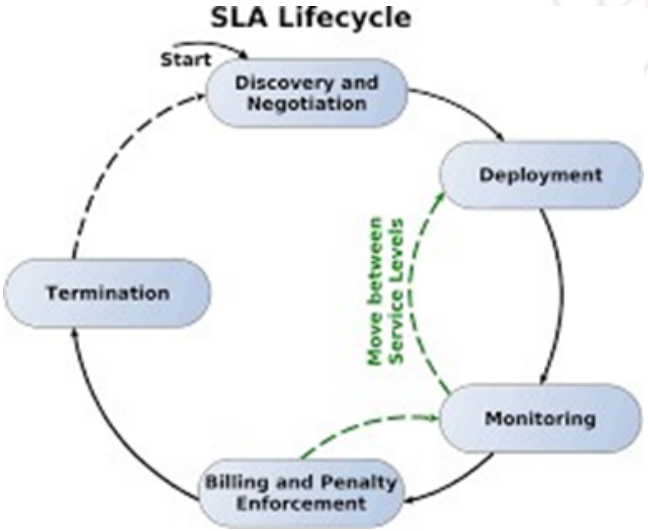
WORKING PRINCIPLE:

Smart contracts work by following simple "if/when…then…" statements that are written into code on a blockchain. A network of computers executes the actions when predetermined conditions have been met and verified. These actions could include releasing funds to the appropriate parties, registering a vehicle, sending notifications, or issuing a ticket. The blockchain is then updated when the transaction is completed. That means the transaction cannot be changed, and only parties who have been granted permission can see the results.

Within a smart contract, there can be as many stipulations as needed to satisfy the participants that the task will be completed satisfactorily. To establish the terms, participants must determine how transactions and their data are represented on the blockchain, agree on the "if/when...then…" rules that govern those transactions, explore all possible exceptions, and define a framework for resolving disputes.

Then the smart contract can be programmed by a developer – although increasingly, organizations that use blockchain for business provide templates, web interfaces, and other online tools to simplify structuring smart contracts.

Solidity is an object-oriented, high-level programming language developed specifically for creating and implementing smart contracts on various blockchain development platforms, particularly Ethereum. Initially developed in 2014, Solidity has attracted the attention of businesses who want to switch from traditional to "smart" contracts.



Solidity is a statically-typed curly-braces programming language designed for developing smart contracts that run on Ethereum.
Solidity is an object-oriented programming language created specifically by the Ethereum Network team for constructing and designing smart contracts on Blockchain platforms.

- It's used to create smart contracts that implement business logic and generate a chain of transaction records in the blockchain system.
- It acts as a tool for creating machine-level code and compiling it on the Ethereum Virtual Machine (EVM).

Pragma is generally the first line of code within any Solidity file. pragma is a directive that specifies the compiler version to be used for current Solidity file. Solidity is a new language and is subject to continuous improvement on an on-going basis.
Programs written in Solidity run on the Ethereum Virtual Machine.

Ganache, Truffle and Node.js:
Ganache provides an easy one-click solution for creating a local blockchain testnet onto which a tool such as Truffle which runs on top of node.js can be used to deploy and interact with Smart Contracts.
Procedure:

1. Download ganache for your OS.
2. Install latest version of Node.js
   For debian-based OS use the command:
   sudo snap install node
3. Install truffle using npm
   npm install truffle -g
4. Launch ganache, click on the Quickstart button and note the port number used.
5. Run the following command in your workspace folder. truffle init
6. Edit truffle-config.js and uncomment the following lines: Under “network”:

Replace the port with the one displayed in ganache.

7. Write your smart contracts using your favourite text editor in the contracts directory.
8. Create a new .js file in the migrations directory and add the following lines for each smart contract file in contracts/.

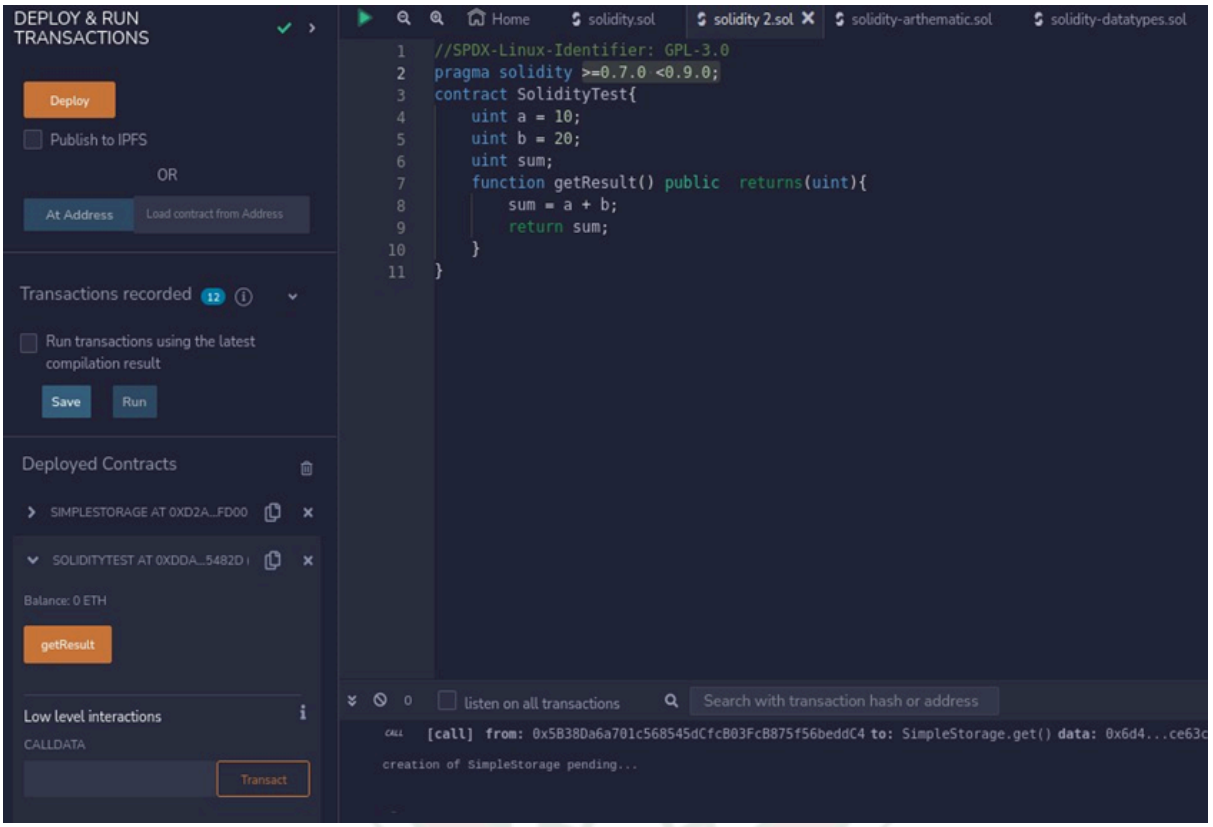
- var MyContract = artifacts.require("MyContract");
- module.exports = function(deployer) {
- // deployment steps
- deployer.deploy(MyContract);
- };

9. Deploy the contracts using “truffle migrate” or “truffle deploy” or “truffle compile”
10. Interact with the smart contracts by running “truffle console”.

The use of #pragma once can reduce build times, as the compiler won't open and read the file again after the first #include of the file in the translation unit. It's called the multiple-include optimization.

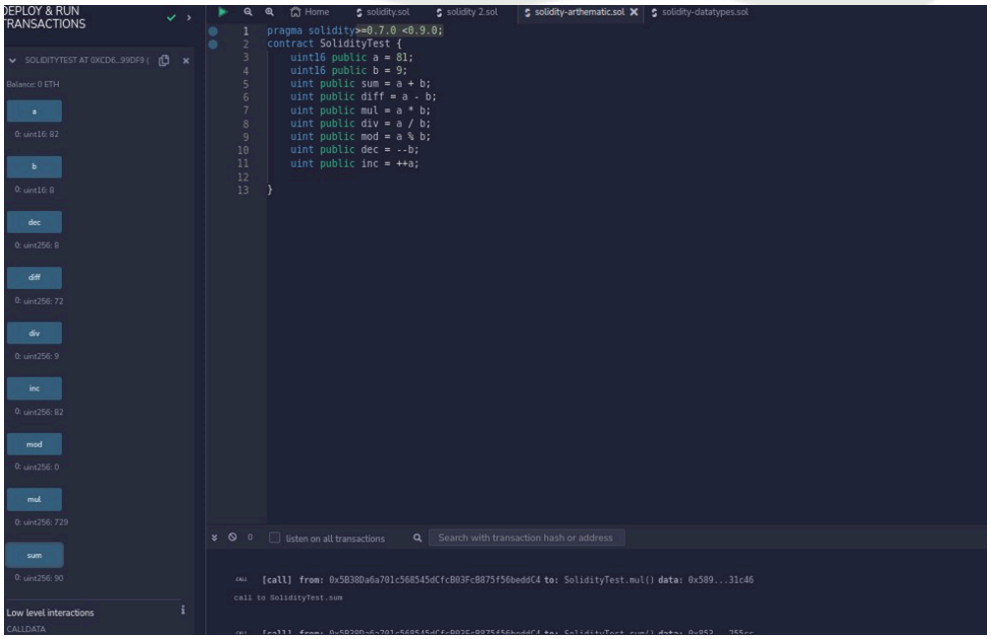
AIM: ADDITION OF TWO NUMBERS

CODE AND OUTPUT:



AIM: TO DISPLAY ARTHEMATIC OPERATIONS

CODE AND OUTPUT:



AIM: TO DISPLAY DATATYPES

CODE:

LAB WEEK-4

AIM : TO WRITE A PROGRAM TO SHOW SIGNIFICANCE OF ADDRESS IN SOLIDITY

DESCRIPTION:

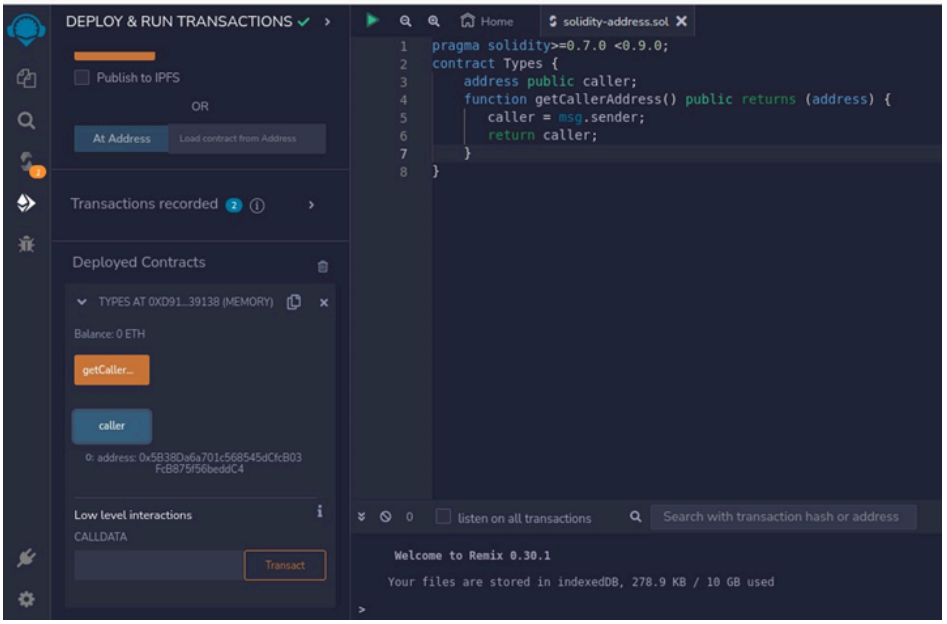
Address is a value type and it creates a new copy while being assigned to another variable.
On the Ethereum blockchain, every account and smart contract has an address and it's stored as 20- byte values. It is used to send and receive Ether from one account to another account. You can consider it as your public identity on the Blockchain. To make it more clear, you can think of it as a bank account number, if you want to send some money to me, you need my bank account number, similarly, in the Blockchain, you need an address to send and receive cryptocurrency or make transactions.

Also, when you deploy a smart contract to the blockchain, one address will be assigned to that contract, by using that address you can identify and call the smart contract.In Solidity, address type comes with two flavors, address and address payable. Both address and address payable stores the 20-byte values, but address payable has additional members, transfer and send.
Solidity functions are methods used to perform some specific task. They get invoked during the execution of a smart contract.

CODE:

```
pragma solidity>=0.7.0 <0.9.0; contract Types {
address public caller;
function getCallerAddress() public returns (address) { caller = msg.sender;
return caller;
}
}
```

OUTPUT:



2. AIM: WAP TO SHOW STATE VARIABLES IN SOLIDITY

DESCRIPTION:

Solidity supports three types of variables.

- State Variables − Variables whose values are permanently stored in a contract storage.
- Local Variables − Variables whose values are present till function is executing.
- Global Variables − Special variables exists in the global namespace used to get information about the blockchain.

Solidity is a statically typed language, which means that the state or local variable type needs to be specified during declaration. Each declared variable always have a default value based on its type. There is no concept of "undefined" or "null".

State Variable are those Variables whose values are permanently stored in a contract storage.

CODE:

```
pragma solidity>=0.7.0 <0.9.0; contract SimpleStorage {
uint public num;
function set(uint _num) public { num = _num;
}
function get() public view returns (uint) { return num;
}
}
```

OUTPUT:

3. AIM: WAP TO USE LOCAL VARIABLES IN SOLIDITY

DESCRIPTION:

Solidity supports three types of variables.

- State Variables − Variables whose values are permanently stored in a contract storage.
- Local Variables − Variables whose values are present till function is executing.
- Global Variables − Special variables exists in the global namespace used to get information about the blockchain.

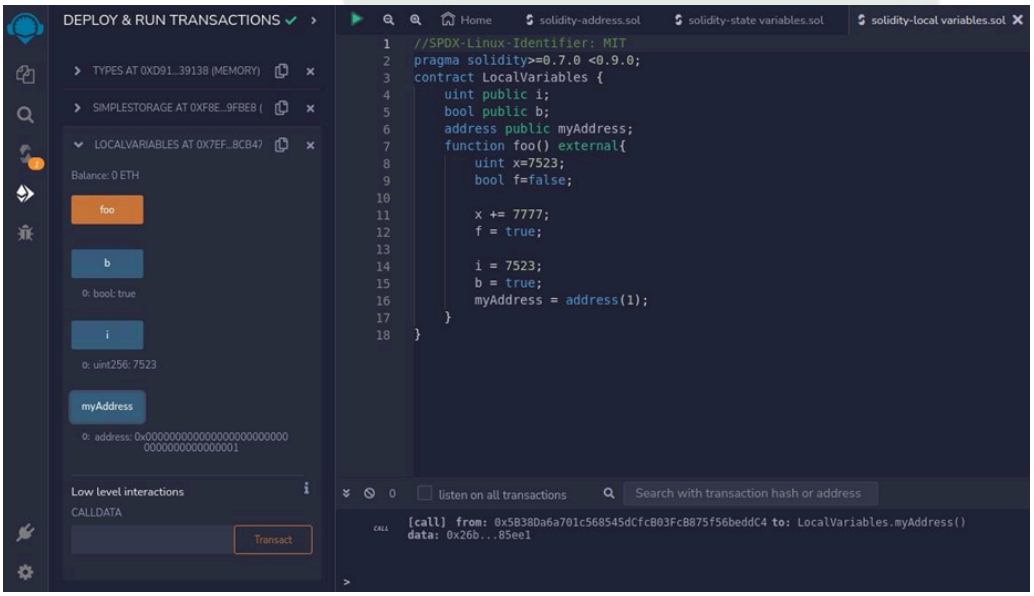
Local Variable are Variables whose values are available only within a function where it is defined.
Function parameters are always local to that function.
Are declared in a function and used in the function they are referenced in

- Local variables are temporary and not stored on the blockchain in contract storage.
- Usually these are the variables that we create temporarily to hold values in calculating or processing something
- Local variables of array, mapping or struct type reference storage by default

CODE:

```
pragma solidity>=0.7.0 <0.9.0;
contract LocalVariables { uint public i;
bool public b;
address public myAddress; function foo() external{ uint x=7523;
bool f=false; x += 7777;
f = true; i = 7523;
b = true;
myAddress = address(1);
}
}
```

OUTPUT:



4. AIM: WAP TO USE GLOBAL VARIABLES IN SOLIDITY

DESCRIPTION:

Solidity supports three types of variables.

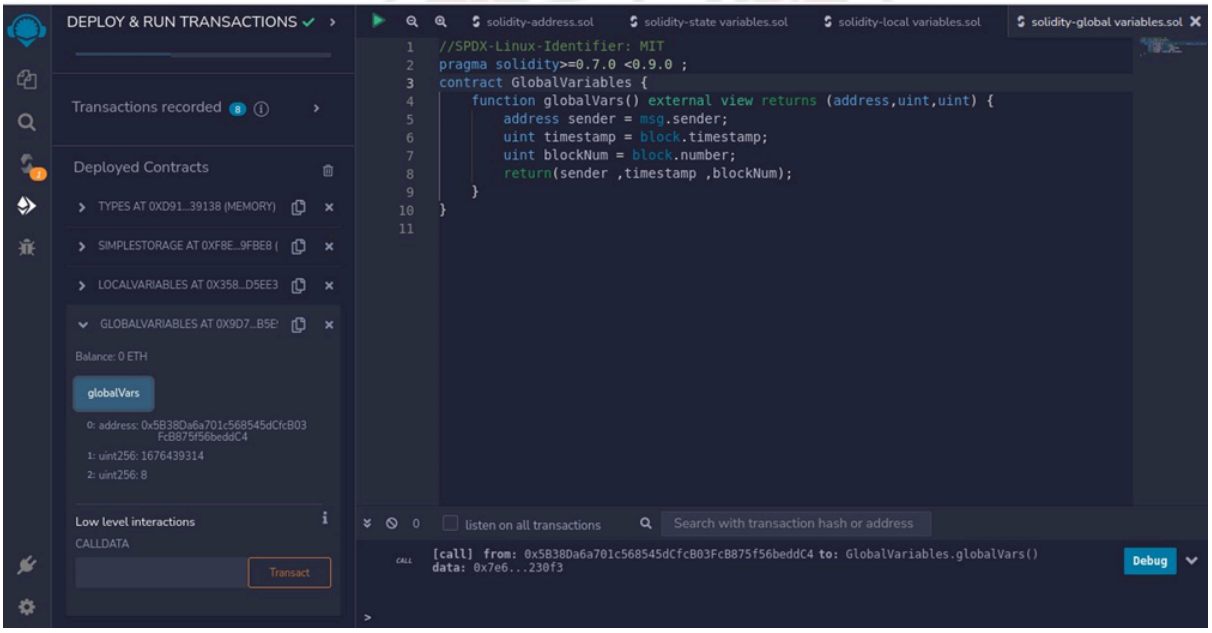
- State Variables − Variables whose values are permanently stored in a contract storage.
- Local Variables − Variables whose values are present till function is executing.
- Global Variables − Special variables exists in the global namespace used to get information about the blockchain.

Global Variables are these special variables which exist in global workspace and provide information about the blockchain and transaction properties.

CODE:

```
pragma solidity>=0.7.0 <0.9.0 ; contract GlobalVariables {
function globalVars() external view returns (address,uint,uint) { address sender = msg.sender;
uint timestamp = block.timestamp; uint blockNum = block.number; return(sender ,timestamp
,blockNum);
}
}
```

OUTPUT:



5. AIM: WAP to Demonstrate various Operators in Solidity

DESCRIPTION: The functionality of Solidity is also incomplete without the use of operators. Operators allow users to perform different operations on operands. Solidity supports the following types of operators based upon their functionality.

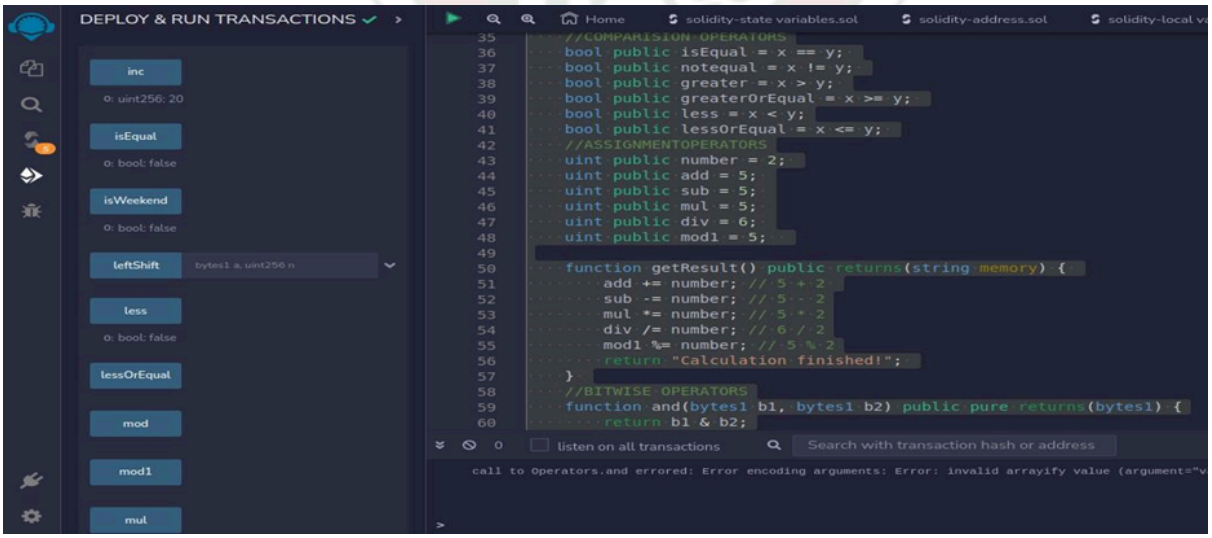
1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Bitwise Operators
5. Assignment operators
6. Conditional Operator

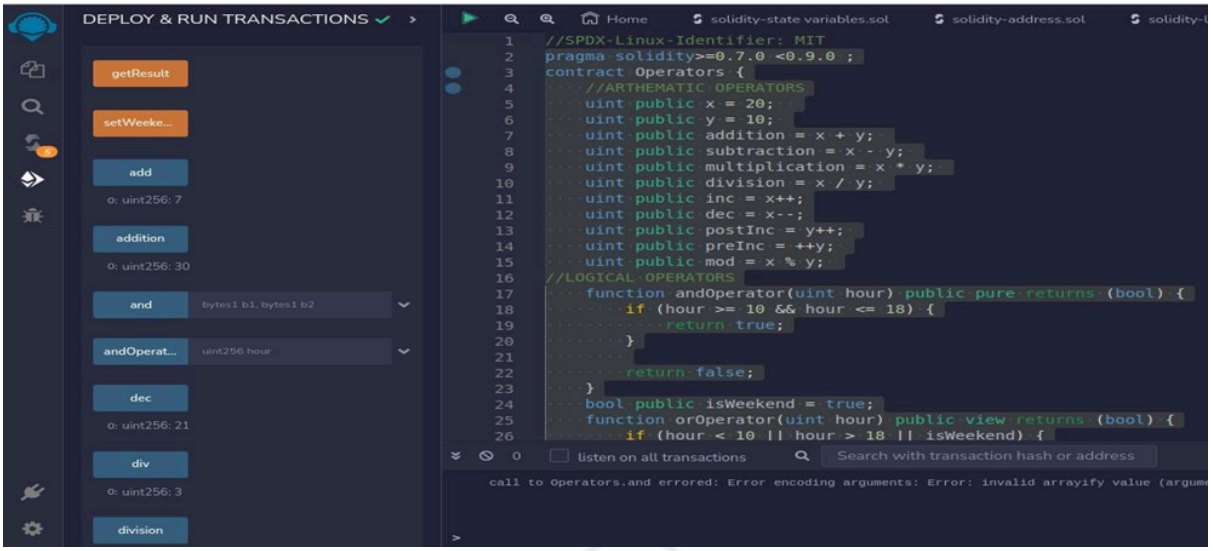
CODE:

```
pragma solidity>=0.7.0 <0.9.0 ;
contract Operators {
//ARTHEMATIC OPERATORS
uint public x = 20; uint public y = 10;
uint public addition = x + y; uint public subtraction = x - y;
uint public multiplication = x * y; uint public division = x / y;
uint public inc = x++; uint public dec = x--;
uint public postInc = y++; uint public preInc = ++y; uint public mod = x % y;
//LOGICAL OPERATORS
function andOperator(uint hour) public pure returns (bool) { if (hour >= 10 && hour <= 18) {
return true;
}
return false;
}
bool public isWeekend = true;
function orOperator(uint hour) public view returns (bool) { if (hour < 10 || hour > 18 || isWeekend) {
return true;
}
return false;
}
function setWeekendValue() public { isWeekend = false;
}
//COMPARISION OPERATORS
bool public isEqual = x == y; bool public notequal = x != y; bool public greater = x > y;
bool public greaterOrEqual = x >= y; bool public less = x < y;
bool public lessOrEqual = x <= y;
//ASSIGNMENTOPERATORS
uint public number = 2; uint public add = 5; uint public sub = 5; uint public mul = 5; uint public div = 6;
uint public mod1 = 5;
function getResult() public returns(string memory) { add += number; // 5 + 2
sub -= number; // 5 - 2 mul *= number; // 5 * 2 div /= number; // 6 / 2
mod1 %= number; // 5 % 2 return "Calculation finished!";
}
```

```
//BITWISE OPERATORS
function and(bytes1 b1, bytes1 b2) public pure returns(bytes1) { return b1 & b2;
}
function or(bytes1 b1, bytes1 b2) public pure returns(bytes1) { return b1 | b2;
}
function xor(bytes1 b1, bytes1 b2) public pure returns(bytes1) { return b1 ^ b2;
}
function not(bytes1 b1) public pure returns(bytes1) { return ~b1;
}
function leftShift(bytes1 a, uint n) public pure returns (bytes1) { return a << n;
}
function rightShift(bytes1 a, uint n) public pure returns (bytes1) { return a >> n;
}
//CONDITIONAL OPERATORS
function getMax(uint x, uint y) public pure returns(uint) { return x >= y ? x: y;
}
function _getMax(uint x, uint y) internal pure returns(uint) { if (x >= y) {
return x;
}
return y;
}
}
```

OUTPUT:




DEPLOY & RUN TRANSACTIONS
getResult
setWeeke...
add
o: uint256: 7
addition
o: uint256: 30
and
bytes1 b1, bytes1 b2
andOperat...
uint256 hour
dec
o: uint256: 21
div
o: uint256: 3
division
Home
solidity-state variables.sol
solidity-address.sol
//SPDX-Linux-Identifier: MIT
pragma solidity>=0.7.0 <0.9.0 ;
contract Operators {
//ARTHEMATIC OPERATORS
uint public x = 20;
uint public y = 10;
uint public addition = x + y;
uint public subtraction = x - y;
uint public multiplication = x * y;
uint public division = x / y;
uint public inc = x++;
uint public dec = x--;
uint public postInc = y++;
uint public preInc = ++y;
uint public mod = x % y;
//LOGICAL OPERATORS
function andOperator(uint hour) public pure returns (bool) {
if (hour >= 10 && hour <= 18) {
return true;
}
return false;
}
bool public isWeekend = true;
function orOperator(uint hour) public view returns (bool) {
if (hour < 10 || hour > 18 || isWeekend) {
listen on all transactions
Search with transaction hash or address
call to Operators.and errored: Error encoding arguments: Error: invalid arrayify value (argum

LAB TASK 5

1.AIM: WAP TO USE IF LOOP IN SOLIDITY

DESCRIPTION: The if statement is the fundamental control statement that allows Solidity to make decisions and execute statements conditionally.

The syntax for a basic if statement is as follows −

if (expression) {

Statement(s) to be executed if expression is true

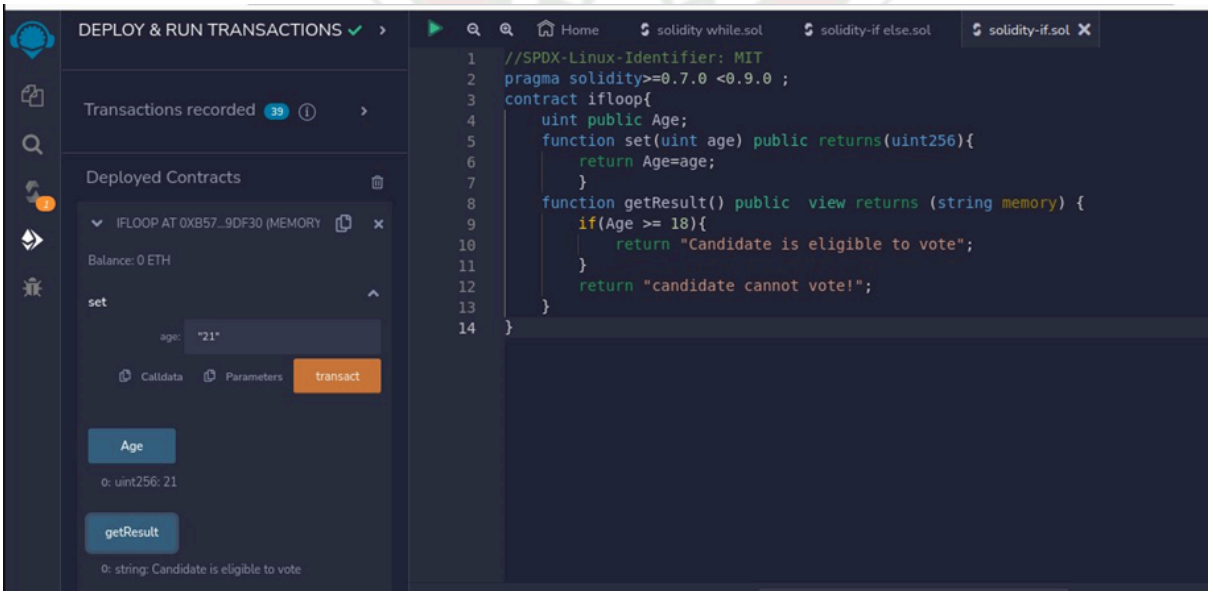
}

Here a Solidity expression is evaluated. If the resulting value is true, the given statement(s) are executed. If the expression is false, then no statement would be not executed. Most of the times, you will use comparison operators while making decisions.

CODE:

```
pragma solidity>=0.7.0 <0.9.0 ;
contract ifloop{ uint public Age;
function set(uint age) public returns(uint256){ return Age=age;
}
function getResult() public view returns (string memory) { if(Age >= 18){
return "Candidate is eligible to vote";
}
return "candidate cannot vote!";
}
}
```

OUTPUT:



2. AIM: WAP TO USE IF-ELSE IN SOLIDITY

DESCRIPTION: The 'if...else' statement is the next form of control statement that allows Solidity to execute statements in a more controlled way.

Syntax:-

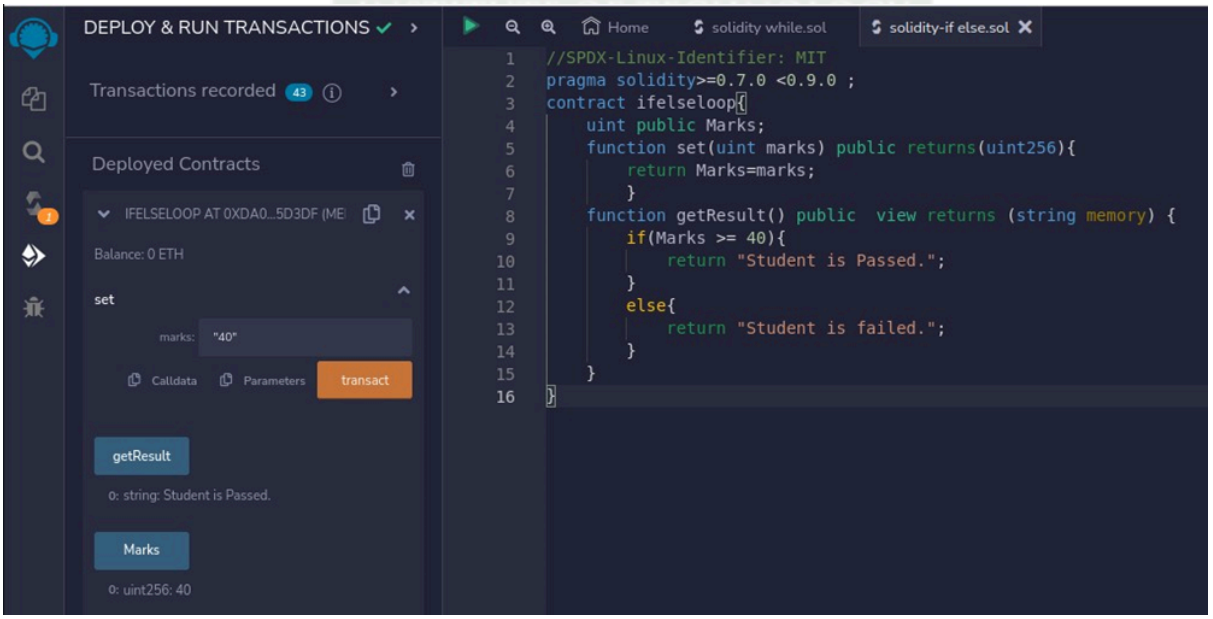
if (expression) {
Statement(s) to be executed if expression is true
} else {
Statement(s) to be executed if expression is false
}

Here Solidity expression is evaluated. If the resulting value is true, the given statement(s) in the 'if' block, are executed. If the expression is false, then the given statement(s) in the else block are executed.

CODE:

```
pragma solidity>=0.7.0 <0.9.0 ; contract ifelseloop{
uint public Marks;
function set(uint marks) public returns(uint256){ return Marks=marks;
}
function getResult() public view returns (string memory) { if(Marks >= 40){
return "Student is Passed.";
}
else{
return "Student is failed.";
}
}
}
```

OUTPUT:



3. AIM: WAP TO USE IF-ELSE-IF IN SOLIDITY

DESCRIPTION: The if...else if... statement is an advanced form of if...else that allows Solidity to make a correct decision out of several conditions.

The syntax of an if-else-if statement is as follows −

```
if (expression 1) {
Statement(s) to be executed if expression 1 is true
} else if (expression 2) {
Statement(s) to be executed if expression 2 is true
} else if (expression 3) {
Statement(s) to be executed if expression 3 is true
} else {
Statement(s) to be executed if no expression is true
}
```

There is nothing special about this code. It is just a series of if statements, where each if is a part of the else clause of the previous statement. Statement(s) are executed based on the true condition, if none of the conditions is true, then the else block is executed.

CODE:

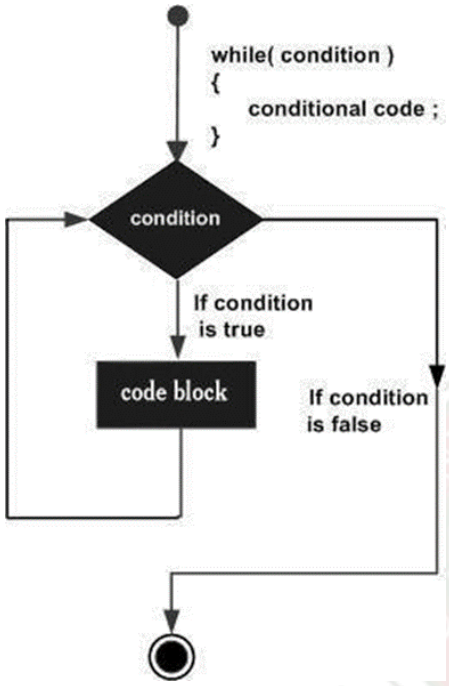
```
pragma solidity>=0.7.0 <0.9.0 ;
contract ifloop{ uint public Marks;
function set(uint marks) public returns(uint256){ return Marks=marks;
}
function getResult() public view returns (string memory) { if(Marks >= 80){
return "Student got A Grade";
}
else if(Marks>=40 && Marks<80){ return "Student got B Grade";
}
else{
return "Student has failed!";
}
}
}
```

OUTPUT:

4. AIM: WAP TO USE WHILE LOOP IN SOLIDITY

DESCRIPTION:

The most basic loop in Solidity is the while loop which would be discussed in this chapter. The purpose of a while loop is to execute a statement or code block repeatedly as long as an expression is true. Once the expression becomes false, the loop terminates. The flow chart of while loop looks as follows −



Syntax

The syntax of while loop in Solidity is as follows −

```
while (expression) {
  Statement(s) to be executed if expression is true
}
```

CODE:

```
pragma solidity >= 0.5.0;contract Types { uint[] data;
uint8 j = 0; function loop(
) public returns(uint[] memory){while(j < 5) {
j++;
data.push(j);
}
return data;
}
}
```

OUTPUT:

CODE:

```
pragma solidity >= 0.5.0;contract Types { uint[] data;
uint8 j = 1; uint8 fact = 1;
function loop(uint num) public returns(uint[] memory){while(j <= num) {
fact = fact * j; j++;
data.push(fact);
}
return data;
}
}
```

OUTPUT:

5. Aim: Solidity program to show the use of do while loop.

Description:

This loop is very similar to while loop except that there is a condition check which happens atthe end of loop i.e. the loop will always execute at least one time even if the condition is false.

Syntax: do
{
block of statements to be executed
} while (condition);

Code:

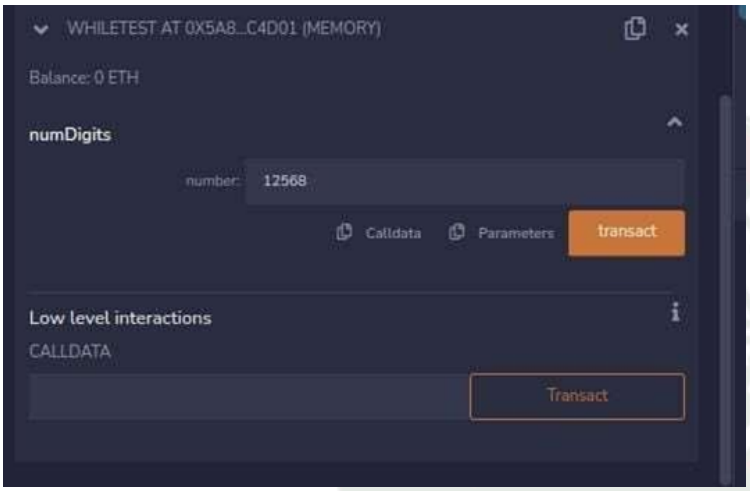
```
pragma solidity >= 0.5.0;    contract whileTest { uint8 i = 0; uint8 result = 0;

function sum(
) public returns(uint data){ do{
i++;
result=result+i;
}
while(i < 3) ; return result;
}
}
```

Output:

CODE:

```
pragma solidity >= 0.5.0;    contract whileTest { uint8 digits
= 0;

function     numDigits(uint  number)          public   re- turns(uint ) {do{
number /= 10; digits++;
}while(number != 0); return digits;
}

}
```

OUTPUT:



6. Aim: Solidity program to show the use of for loop.

Description:

This is the most compact way of looping. It takes three arguments separated by a semi-colon to run. The first one is 'loop initialization' where the iterator is initialized with starting value, this statement is executed before the loop starts. Second is 'test statement' which checks whether the condition is true or not, if the condition is true the loop executes else terminates. The third one is the 'iteration statement' where the iterator is increased or decreased. Below isthe syntax of for loop:

Syntax:

```
for (initialization; test condition; iteration statement) {
statement or block of code to be executed if the condition is True
}
```

CODE

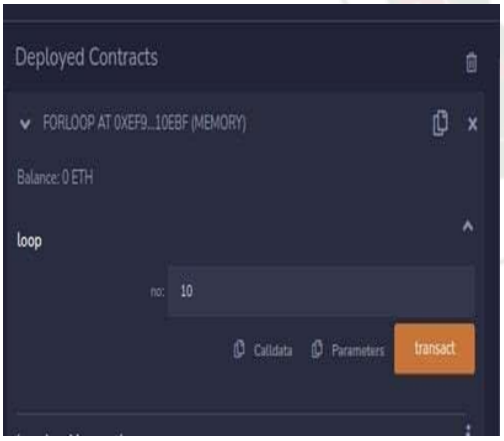
```
contract dowhile{ uint8 max; uint8 i; uint8 lcm;
function loop(uint8 a,uint8 b ) public re- turns(uint){if(a>b){
max=a;
}
else{

max=b;
}
for(i=0;i<max;i++)
{
if(max%a==0 && max%b==0)
{


lcm=max; break;
}
max++;
}
return lcm;
}
}
```

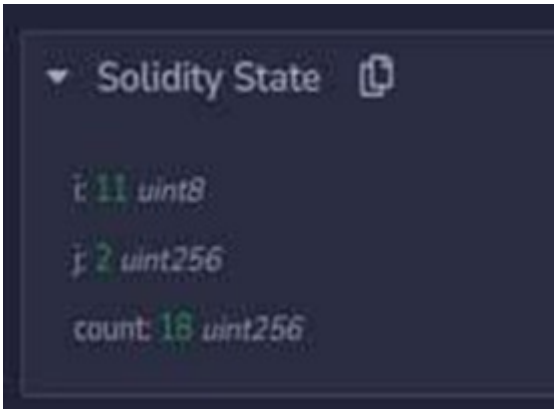
OUTPUT:

CODE:

```
contract     for-
loop{        uint8 i=1;
uint j=2;
uint count=0;
function loop(uint8 no ) public returns(uint){ for(i=1;i<=no;i++){
for(j=2;j<no;j++){ if(i%j == 0){
break;
}
else{
count++;
}



}}
return count;

}}
```

OUTPUT:





LAB WEEK 6

AIM: TO DEPLOY A SMART CONTRACT USING GANACHE AND TRUFFLE

DESCRIPTION:

Ganache provides an easy one-click solution for creating a local blockchain testnet onto which a tool such as Truffle which runs on top of node.js can be used to deploy and interact with Smart Contracts.

Procedure:

1.Download ganache for your OS.

2.Install latest version of Node.js

For debian-based OS use the command:

sudo snap install node

3.Install truffle using npm

npm install truffle -g

4.Launch ganache, click on the Quickstart button and note the port number used.

5.Run the following command in your workspace folder. truffle init

6.Edit truffle-config.js and uncomment the following lines: Under “network”: Replace the port with the one displayed in ganache.

7.Write your smart contracts using your favourite text editor in the contracts directory.

8.Create a new .js file in the migrations directory and add the following lines for each smart contract file in contracts/.

- var MyContract = artifacts.require("MyContract");
- module.exports = function(deployer) {
- // deployment steps
- deployer.deploy(MyContract);
- };

9.Deploy the contracts using “truffle migrate” or “truffle deploy” or “truffle compile”

10.Interact with the smart contracts by running “truffle console.

Deployment:

```
4  contracts truffle migrate

Compiling your contracts...
===========================
> Compiling ./contracts/third.sol
> Compilation warnings encountered:

    Warning: SPDX license identifier not provided in source file. Before
publishing, consider adding a comment containing "SPDX-License-Identifier:
<SPDX-License>" to each source file. Use "SPDX-License-Identifier:
UNLICENSED" for non-open-source code. Please see https://spdx.org for more
information.
--> project:/contracts/third.sol


> Artifacts written to /home/abhinav/Documents/BPAlab/build/contracts
> Compiled successfully using:
   - solc: 0.8.18+commit.87f61d96.Emscripten.clang


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


1_bpa.js
========

   Deploying 'Types'
   -----------------
   > transaction hash:
0x94255b7685f2ec4c1a909dcab1a01249de8d4e39907dc8145c00ea36cab94d3c
   > Blocks: 0            Seconds: 0
   > contract address:    0x3a88C3997cb4bBcBf66A4A40a4F64Cba97629e35
   > block number:        5
   > block timestamp:     1678208893
   > account:             0x6a588925B47CF7a6ee9C826486179685cF9D154A
    > balance:                99.997238648174534077
    > gas used:               395151 (0x6078f)
    > gas price:              3.023515325 gwei
    > value sent:             0 ETH
    > total cost:             0.001194745104189075 ETH

    > Saving artifacts
    -------------------------------------------------------------------
    > Total cost:      0.001194745104189075 ETH

Summary
=======
> Total deployments:   1
> Final cost:          0.001194745104189075 ETH
```

Interacting with the smart contract:

```
➜  contracts truffle console
truffle(development)> let a=await Types.deployed()
undefined
truffle(development)> a.decision_making()
{
  tx:
'0x163b86e82ae781ef55ee8e984a4dc6c54b92664c9972db2869d7973fd1bcce07',
  receipt: {
    transactionHash:
'0x163b86e82ae781ef55ee8e984a4dc6c54b92664c9972db2869d7973fd1bcce07',
```

LAB WEEK -7

1)AIM: TO USE STRUCTS USING SOLIDITY

DESCRIPTION: Struct types are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book −

- Title
- Author
- Subject
- Book ID Defining a Struct

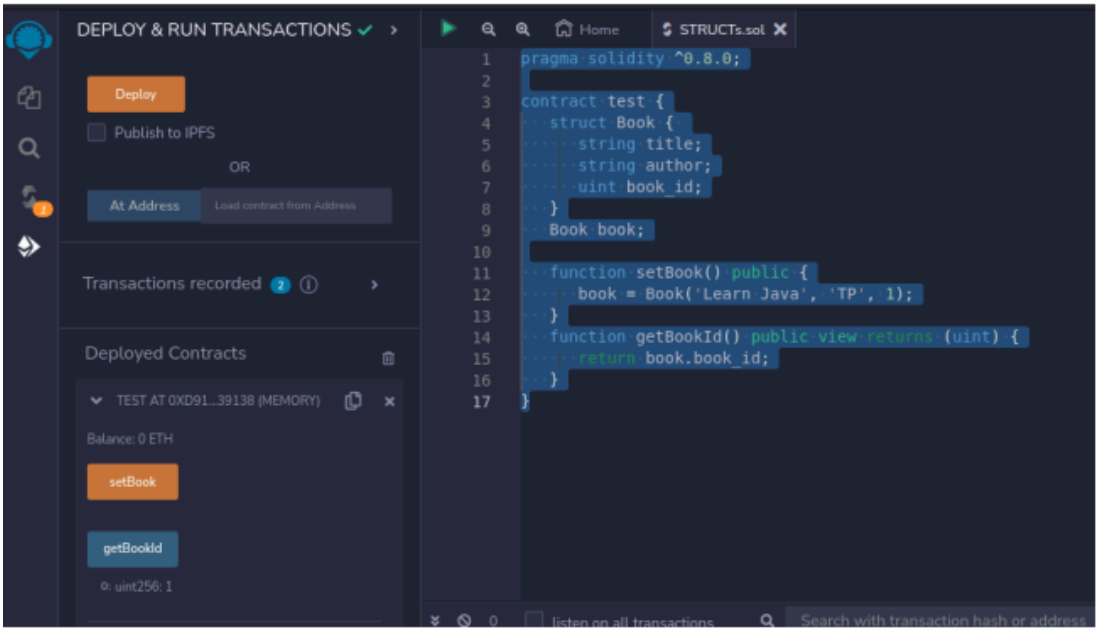
To define a Struct, you must use the struct keyword. The struct keyword defines a new data type, with more than one member. The format of the struct statement is as follows –

struct struct_name {
type1 type_name_1; type2 type_name_2; type3 type_name_3;
}

CODE:

```
pragma solidity ^0.8.0; contract test {
struct Book { string title; string author; uint book_id;
}
Book book;
function setBook() public {
book = Book('Learn Java', 'TP', 1);
}
function getBookId() public view returns (uint) { return book.book_id;
}
}
```

OUTPUT:



2) AIM: TO USE MAPPING WITH SOLIDITY

DESCRIPTION: Mapping is a reference type as arrays and structs. Following is the syntax to declare a mapping type:-

mapping(_KeyType => _ValueType)

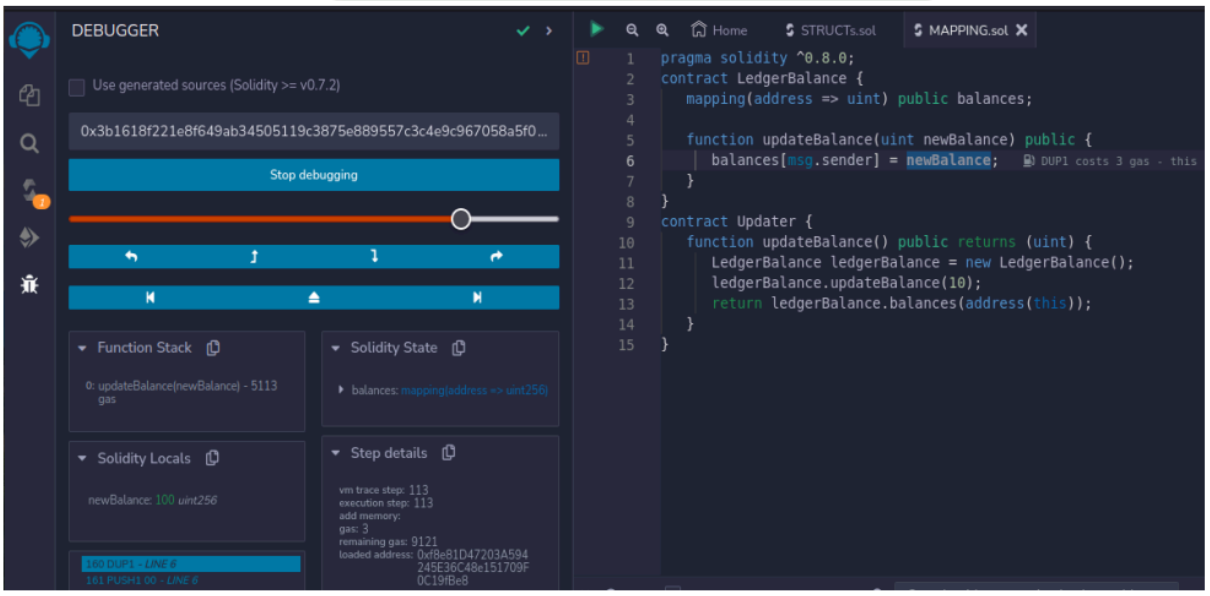
- _KeyType − can be any built-in types plus bytes and string. No reference type or complex objects are allowed.
- _ValueType − can be any type.

Mapping can only have type of storage and are generally used for state variables. Mapping can be marked public. Solidity automatically create getter for it.

CODE:

```
pragma solidity ^0.8.0; contract LedgerBalance {
mapping(address => uint) public balances;

function updateBalance(uint newBalance) public { balances[msg.sender] = newBalance;
}
}
contract Updater {
function updateBalance() public returns (uint) { LedgerBalance ledgerBalance = new LedgerBalance();
ledgerBalance.updateBalance(10);
return ledgerBalance.balances(address(this));
}
}
```

OUTPUT:



3) AIM:TO USE SOLIDITY WITH SPECIAL VARIABLES

DESCRIPTION: Special variables are globally available variables and provides information about the blockchain. Following is the list of special variables −

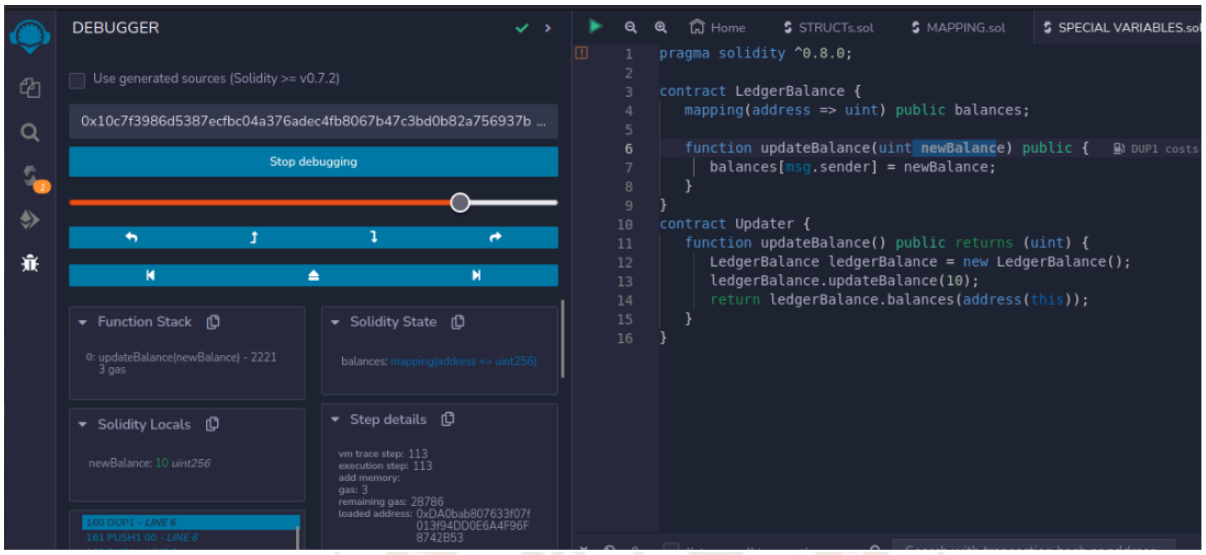
Sr.No.	Special Variable & Description
1	blockhash(uint blockNumber) returns (bytes32) Hash of the given block - only works for 256 most recent, excluding current, blocks.
2	block.coinbase (address payable) Current block miner's address.
3	block.difficulty (uint) current block difficulty.
4	block.gaslimit (uint) Current block gaslimit.
5	block.number (uint) Current block number.
6	block.timestamp Current block timestamp as seconds since unix epoch.
7	gasleft() returns (uint256) Remaining gas.
8	msg.data (bytes calldata) Complete calldata.
9	msg.sender (address payable) Sender of the message (current call).
10	msg.sig (bytes4) First four bytes of the calldata (i.e. function identifier)
11	msg.value (uint) Number of wei sent with the message.
12	now (uint) Current block timestamp (alias for block.timestamp).
13	tx.gasprice (uint) Gas price of the transaction.
14	tx.origin (address payable) Sender of the transaction (full call chain

CODE:

```
pragma solidity ^0.8.0; contract LedgerBalance {
mapping(address => uint) public balances;

function updateBalance(uint newBalance) public { balances[msg.sender] = newBalance;
}
}
contract Updater {
function updateBalance() public returns (uint) { LedgerBalance ledgerBalance = new LedgerBalance();
ledgerBalance.updateBalance(10);
```

```
return ledgerBalance.balances(address(this));
}
}
```

OUTPUT:



4) AIM: TO USE SOLIDITY FOR MATHEMATICAL FUNCTIONS

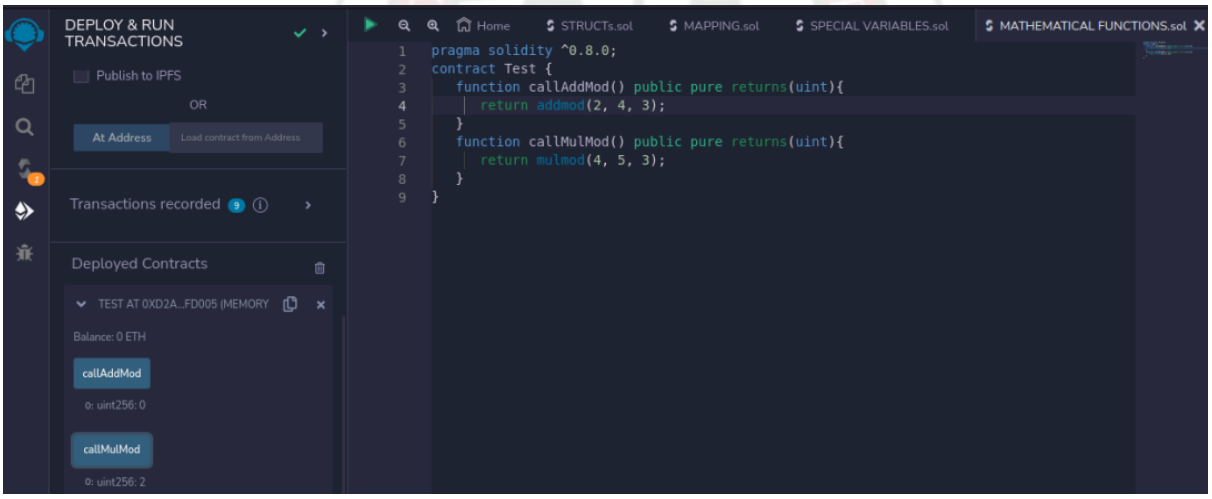
DESCRIPTION: Solidity provides inbuilt mathematical functions as well. Following are heavily used methods −

- addmod(uint x, uint y, uint k) returns (uint) − computes (x + y) % k where the addition is performed with arbitrary precision and does not wrap around at 2256.
- mulmod(uint x, uint y, uint k) returns (uint) − computes (x * y) % k where the addition is performed with arbitrary precision and does not wrap around at 2256.

CODE:

```
pragma solidity ^0.8.0;

contract Test {
function callAddMod() public pure returns(uint){ return addmod(4, 5, 3);
}
function callMulMod() public pure returns(uint){ return mulmod(4, 5, 3);
}
}
```

OUTPUT:



5) AIM: TO USE CRYPTOGRAPHIC FUNCTIONS WITH SOLIDITY

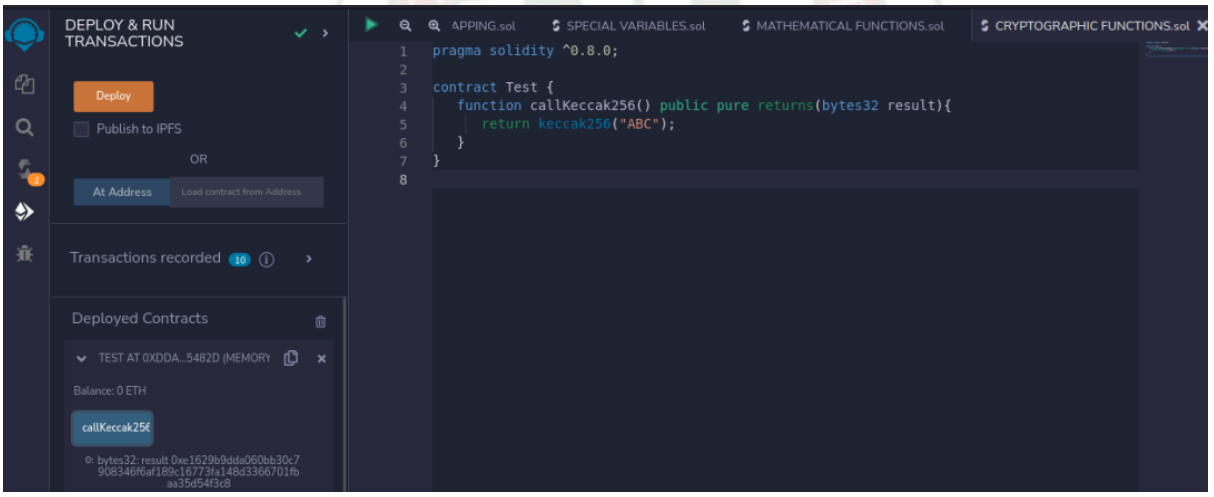
DESCRIPTION: Solidity provides inbuilt cryptographic functions as well. Following are important methods −

- keccak256(bytes memory) returns (bytes32) − computes the Keccak-256 hash of the input.
- ripemd160(bytes memory) returns (bytes20) − compute RIPEMD-160 hash of the input.
- sha256(bytes memory) returns (bytes32) − computes the SHA-256 hash of the input.
- ecrecover(bytes32 hash, uint8 v, bytes32 r, bytes32 s) returns (address) − recover the address associated with the public key from elliptic curve signature or return zero on error. The function parameters correspond to ECDSA values of the signature: r - first 32 bytes of signature; s: second 32 bytes of signature; v: final 1 byte of signature. This method returns an address.

CODE:

```
pragma solidity ^0.8.0;

contract Test {
function callKeccak256() public pure returns(bytes32 result){ return keccak256("ABC");
}
}
```

OUTPUT:



6) AIM: TO USE ARRAYS USING SOLIDITY

DESCRIPTION: In Solidity, an array can be of compile-time fixed size or of dynamic size. For storage array, it can have different types of elements as well. In case of memory array, element type can not be mapping and in case it is to be used as function parameter then element type should be an ABI type.
All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

Declaring Arrays

To declare an array of fixed size in Solidity, the programmer specifies the type of the elements and the number of elements required by an array as follows −

type arrayName [arraySize];

This is called a single-dimension array. The arraySize must be an integer constant greater than zero and type can be any valid Solidity data type. For example, to declare a 10-element array called balance of type uint, use this statement −

uint balance[10];

To declare an array of dynamic size in Solidity, the programmer specifies the type of the elements as follows −

type[] arrayName;

Initializing Arrays

You can initialize Solidity array elements either one by one or using a single statement as follows − uint balance[3] = [1, 2, 3];

The number of values between braces [ ] can not be larger than the number of elements that we declare for the array between square brackets [ ]. Following is an example to assign a single element of the array –

If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write −

uint balance[] = [1, 2, 3];

You will create exactly the same array as you did in the previous example. balance[2] = 5;

The above statement assigns element number 3rd in the array a value of 5.

Creating dynamic memory arrays

Dynamic memory arrays are created using new keyword.

uint size = 3;

uint balance[] = new uint[](size);

Accessing Array Elements

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example −

uint salary = balance[2];

The above statement will take 3rd element from the array and assign the value to salary variable. Following is an example, which will use all the above-mentioned three concepts viz. declaration, assignment and accessing arrays −

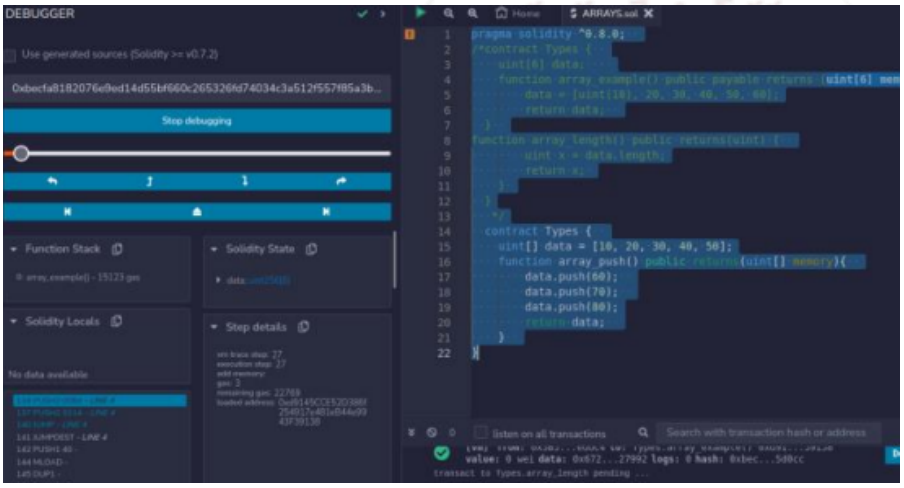
Members

- length − length returns the size of the array. length can be used to change the size of dynamic array be setting it.
- push − push allows to append an element to a dynamic storage array at the end. It returns the new length of the array.

CODE:

```
pragma solidity ^0.8.0;
/*contract Types { uint[6] data;
function array_example() public payable returns (uint[6] memory){ data = [uint(10), 20, 30, 40, 50, 60];
return data;
}
function array_length() public returns(uint) { uint x = data.length;
return x;
}
}
*/
contract Types {
uint[] data = [10, 20, 30, 40, 50];
function array_push() public returns(uint[] memory){ data.push(60);
data.push(70); data.push(80); return data;
}
}
```

OUTPUT:



LAB WEEK -8

1) AIM: To set up Hyper Ledger fabric Network

DESCRIPTION:

Hyperledger Fabric platform is an open source blockchain framework hosted by The Linux Foundation. It has an active and growing community of developers. Permissioned. Fabric networks are permissioned, meaning all participating member's identities are known and authenticated.

A Fabric permissioned blockchain network is a technical infrastructure that provides ledger services to application consumers and administrators. In most cases, multiple organizations come together as a consortium to form the network and their permissions are determined by a set of policies that are agreed to by the consortium when the network is originally configured. Moreover, network policies can change over time subject to the agreement of the organizations in the consortium.

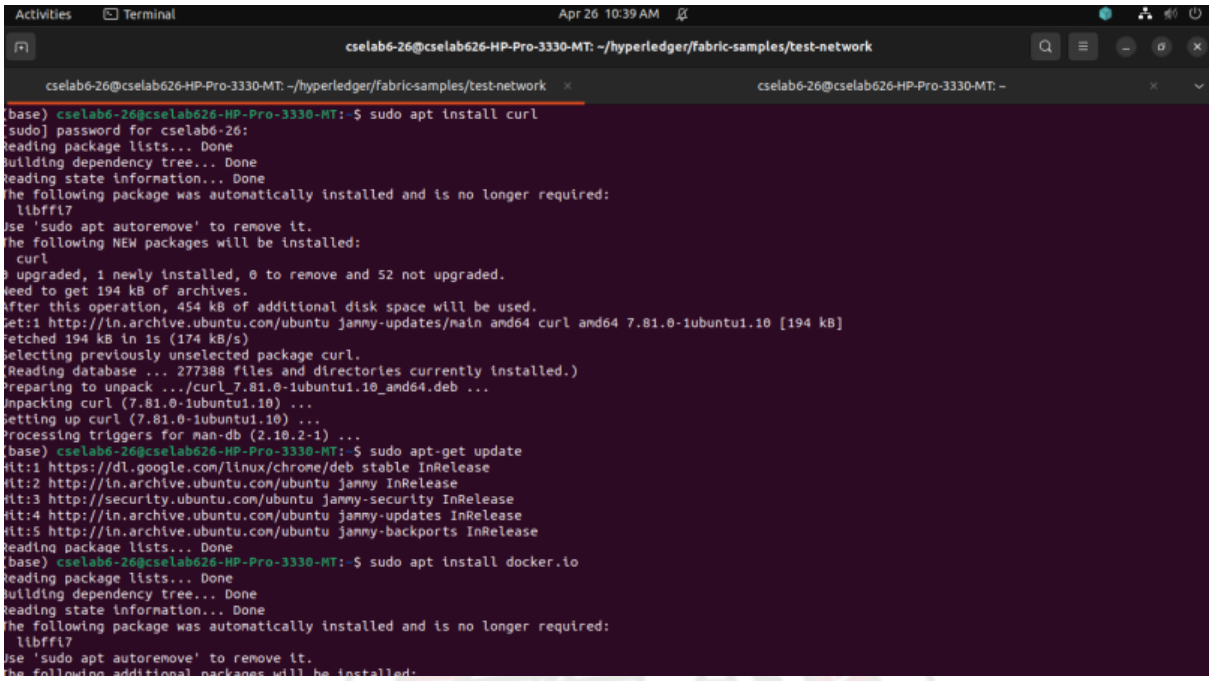
Components of a Network:-

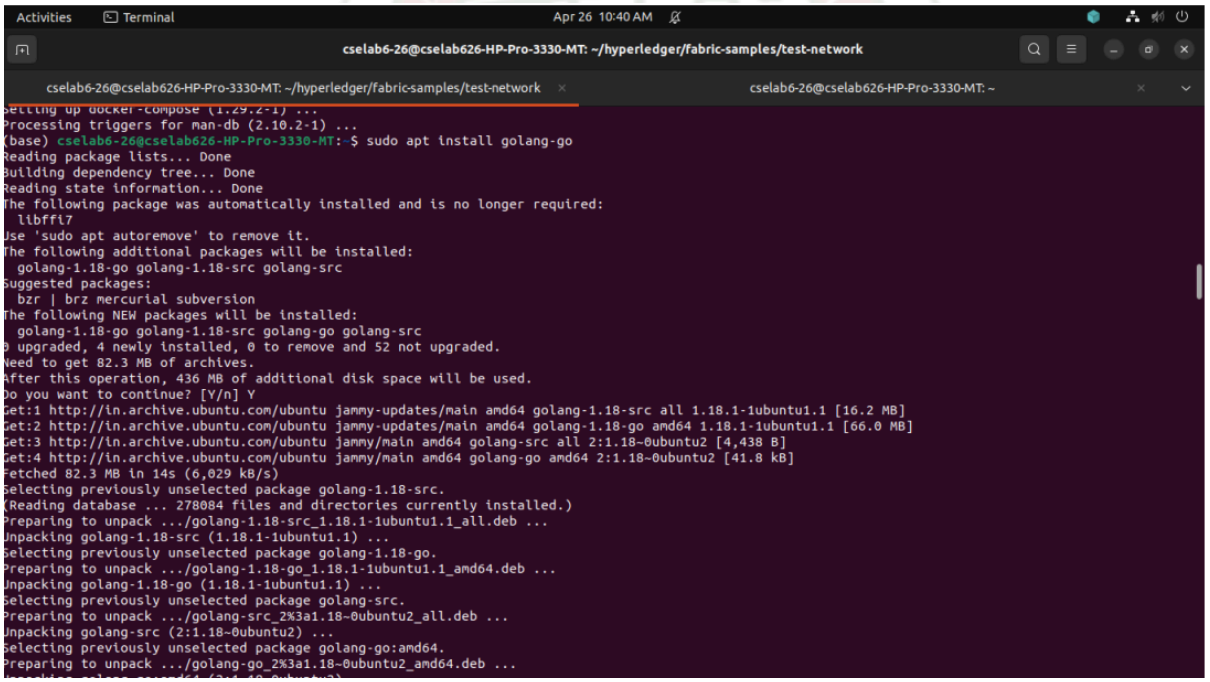
- Ledgers (one per channel – comprised of the blockchain and the state database)
- Smart contract(s) (aka chaincode)
- Peer nodes
- Ordering service(s)
- Channel(s)
- Fabric Certificate Authorities

COMMANDS USED IN SETTING UP NETWORK:

step 1: sudo apt install curl step2 : sudo apt-get update step3 : sudo apt install docker.io
step4 : Dependency packages: sudo snap install docker Version: docker --version
Doc comp: sudo apt install docker-compose step5 : sudo apt install golang-go
Chk version.
step6 : sudo apt install nodejs step7 : sudo apt install npm
step8 : sudo apt install python2 -y (or)
step9 : git config --global core.autocrlf false (from rep to folders to automatic new lines disabling) git config --global core.longpaths true (file name lenght enabling true for sample files while installing by default)
step10 : Create a folder. Mkdir hyperledger step11 : cd folder
step12 :curl -sSL http://bit.ly/2ysbOFE | bash -s We get fabric samples
step13 :Go to test network...
step14 :Network. Sh file ---Execute it
Can be excuted in any language (go/java/node)
Go
step15 : . /network.sh generate Generates a file showing organisations step16 : . /network.sh up
Creates peers on multiple organisations i.e consortium... Channels will be created among peers Also can be created manually... (./network.sh up createChannel -ca -c my channel -s couchdb

EXECUTION:





OUTPUT:

A Channel in the test network is created which is displayed.

2) AIM: To Set up IPFS Network.

DESCRIPTION:

The InterPlanetary File System (IPFS) is a protocol, hypermedia and file sharing peer- to-peer network for storing and sharing data in a distributed file system. IPFS uses content- addressing to uniquely identify each file in a global namespace connecting IPFS hosts.

IPFS allows users to host and receive content in a manner similar to BitTorrent. As opposed to a centrally located server, IPFS is built around a decentralized system of user-operators who hold a portion of the overall data, creating a resilient system of file storage and sharing. Any user in the network can serve a file by its content address, and other peers in the network can find and request that content from any node who has it using a distributed hash table (DHT).

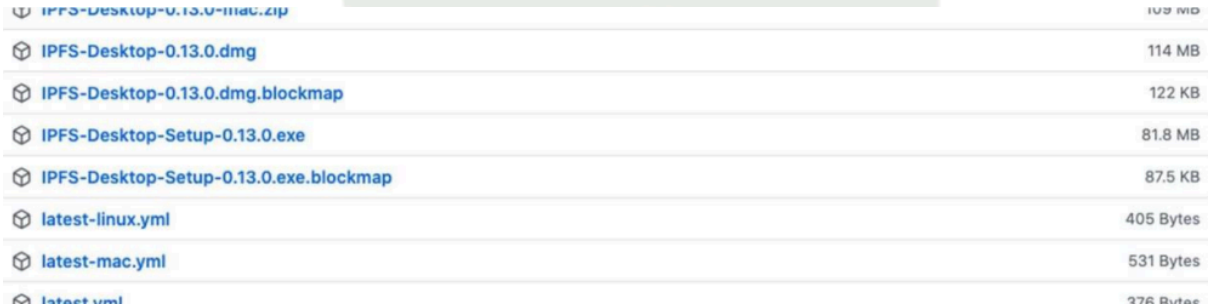
A distributed hash table (DHT) is a distributed system that provides a lookup service similar to a hash table. key–value pairs are stored in a DHT, and any participating node can efficiently retrieve the value associated with a given key. The main advantage of a DHT is that nodes can be added or removed with minimum work around re-distributing keys.

In contrast to BitTorrent, IPFS aims to create a single global network. This means that if two users publish a block of data with the same hash, the peers downloading the content from "user 1" will also exchange data with the ones downloading it from "user 2". IPFS aims to replace protocols used for static webpage delivery by using gateways which are accessible with HTTP. Users may choose not to install an IPFS client on their device and instead use a public gateway. A list of these gateways is maintained on the IPFS GitHub page.

IPFS EXECUTION IN DESKTOP APP:

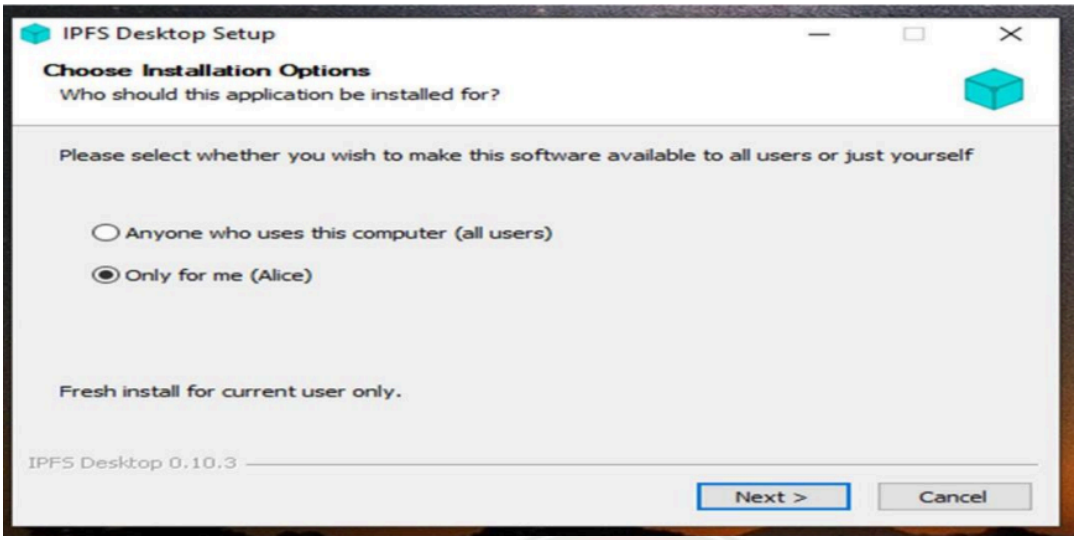
Installation steps:

1. Go to https://github.com/ipfs/ipfs-desktop/releases
2. Find the link ending in .exe for the latest version of IPFS Desktop:

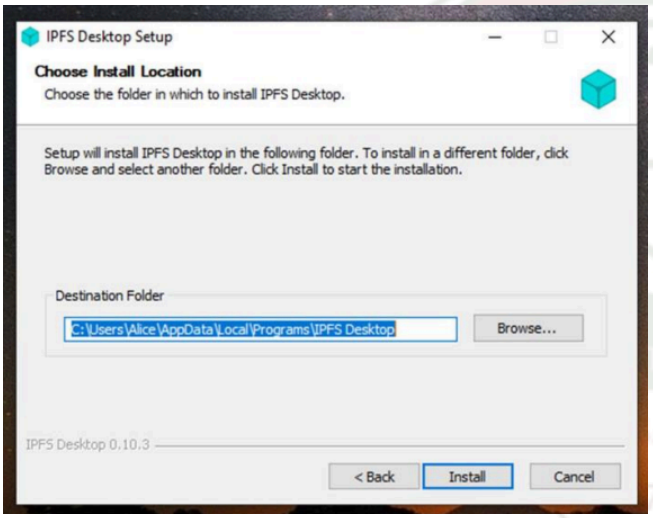


3. Run the .exe file to start the installation.
4. Select whether you want to install the application for just yourself or all users on the computer.

Click Next:

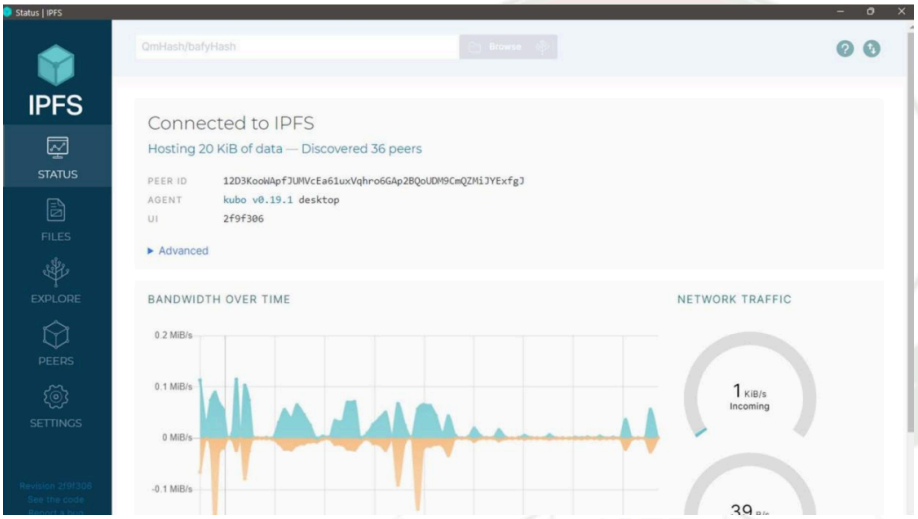


5. Select the install location for the application. The default location is usually fine. Click Next:



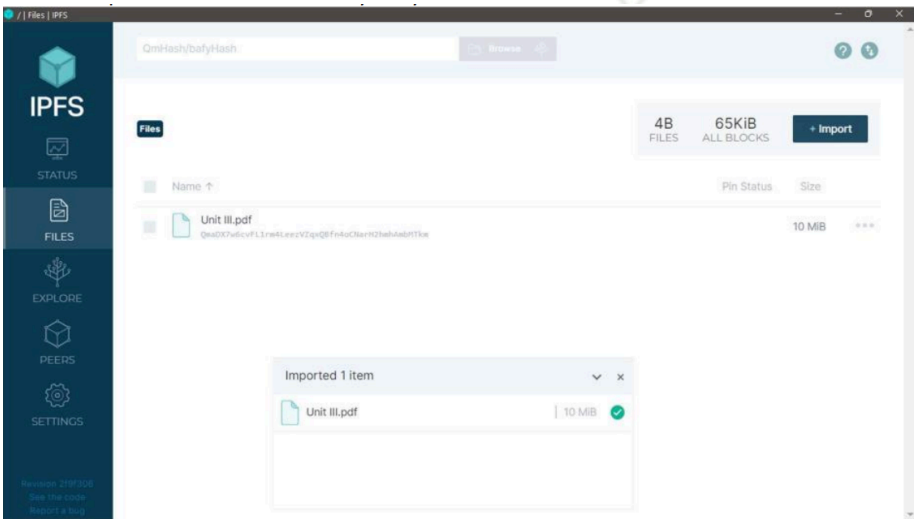
6. Wait for the installation to finish and click Finish:

Now, open IPFS and the screen looks like this

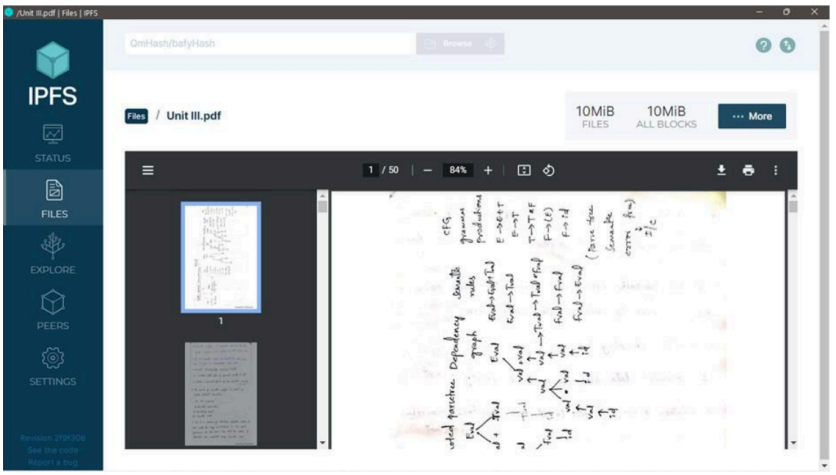


We can add and download files to the network…

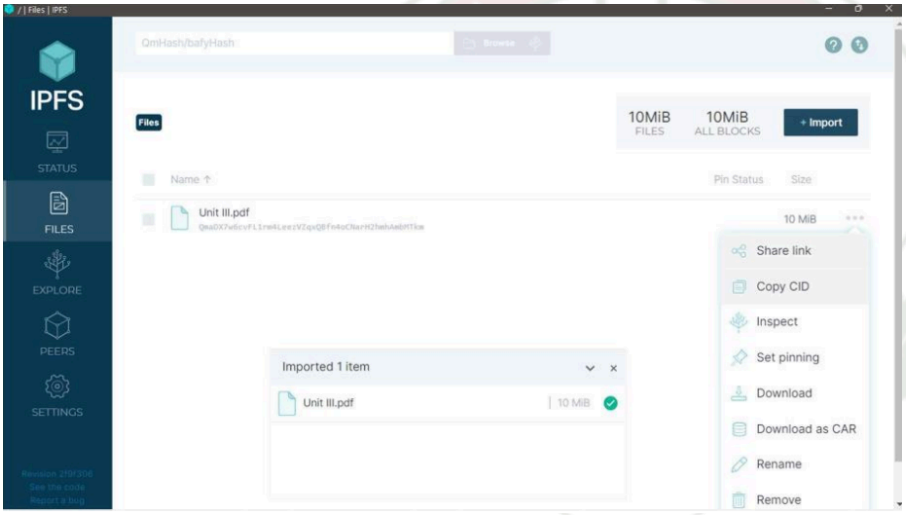
Click on import button and select any file you want to add



We can open the file by clicking on it, each file has its hash value



The pdf got uploaded.Now we can also download the files using their CID,A content identifier, or CID, is a label used to point to material in IPFS. It doesn't indicate where the content is stored, but it forms a kind of address based on the content itself. CIDs are short, regardless of the size of their underlying content.



Copy CID and search for http://localhost:8080/ipfs/<CID> Here the hash is QmaDX7w6cvFL1rm4LeezVZqxQBfn4oCNarH2hmhAmbMTkm So we download the pdf from

http://localhost:8080/ipfs/QmaDX7w6cvFL1rm4LeezVZqxQBfn4oCNarH2hm hAmbMTkm

Applications

- Filecoin is an IPFS-based cooperative storage cloud, also authored by Protocol Labs.
- Cloudflare runs a distributed web gateway to simplify, speed up, and secure access to IPFS without needing a local node.
- Microsoft's self-sovereign identity system, Microsoft ION, builds on the Bitcoin blockchain and IPFS through a Sidetree-based DID network

3) AIM: TO WRITE A SMART CONTRACT FOR THE GIVEN CASE STUDY
DESCRIPTION:
CASE STUDY: EDUCATION – FEE PAYMENT
Smart contracts are simply programs stored on a blockchain that run when predetermined conditions are met. They typically are used to automate the execution of an agreement so that all participants can be immediately certain of the outcome, without any intermediary's involvement or time loss. They can also automate a workflow, triggering the next action when conditions are met.
Smart contracts are digital contracts stored on a blockchain that are automatically executed when predetermined terms and conditions are met

In this case study we have implement fee transaction such that we can see the four main aspects of Fee transaction are :

- Amount of fees paid
- Amount of Total fees
- The fees status
- Due Amount

In a fee payment system for college, there are several entities and their roles involved:

Students: They are the ones who pay the fees for their courses. They need to have access to the payment system to make payments and receive confirmation of their transactions.
College: The college is responsible for setting up the fee payment system and ensuring that it is secure and efficient. They may also provide support to students who have difficulties with the payment system.
Payment Gateway: A payment gateway is a third-party service that processes payments. It enables the college to accept payments from students using various payment methods, such as credit cards, debit cards, and online bank transfers.
Banks: Banks play a vital role in the fee payment system. They provide the infrastructure for processing payments and transferring funds between accounts. They also provide security measures to protect the payment system from fraud.
Administrators: Administrators manage the fee payment system on behalf of the college. They may be responsible for maintaining the system, resolving issues, and ensuring that transactions are processed accurately and in a timely manner.
Fee: Fee refers to the amount of money that students are required to pay for their courses or other expenses related to their education. The fee payment system may involve a variety of payment methods, such as credit cards, debit cards, online bank transfers, and other electronic payment options.
Fee status: It is important for the fee payment system to accurately determine a student's fee status to ensure that they are charged the correct fees and to avoid any errors or discrepancies in the payment process. Fee due to be paid has to be provided appropriately in order to overcome any kind of errors.
Fee payment applications built on blockchain technology have both advantages and limitations. Some of them are:

Advantages:

- Decentralized: Blockchain technology is decentralized, which means there is no central authority or intermediary involved in the transactions. This makes the process more secure, transparent, and efficient.
- Faster transaction speed: Blockchain technology enables faster transaction speeds as it eliminates the need for intermediaries, such as banks or payment processors. This also reduces the associated fees and makes transactions more affordable.
- Lower transaction fees: The use of blockchain technology eliminates the need for intermediaries, which reduces the cost of transactions. This makes it possible to charge lower transaction fees compared to traditional payment methods.
- Transparency: The use of blockchain technology provides a high level of transparency as all transactions are recorded on a public ledger that can be viewed by anyone. This ensures that there is no fraudulent activity, and the system is fair to all parties involved.

Limitations:

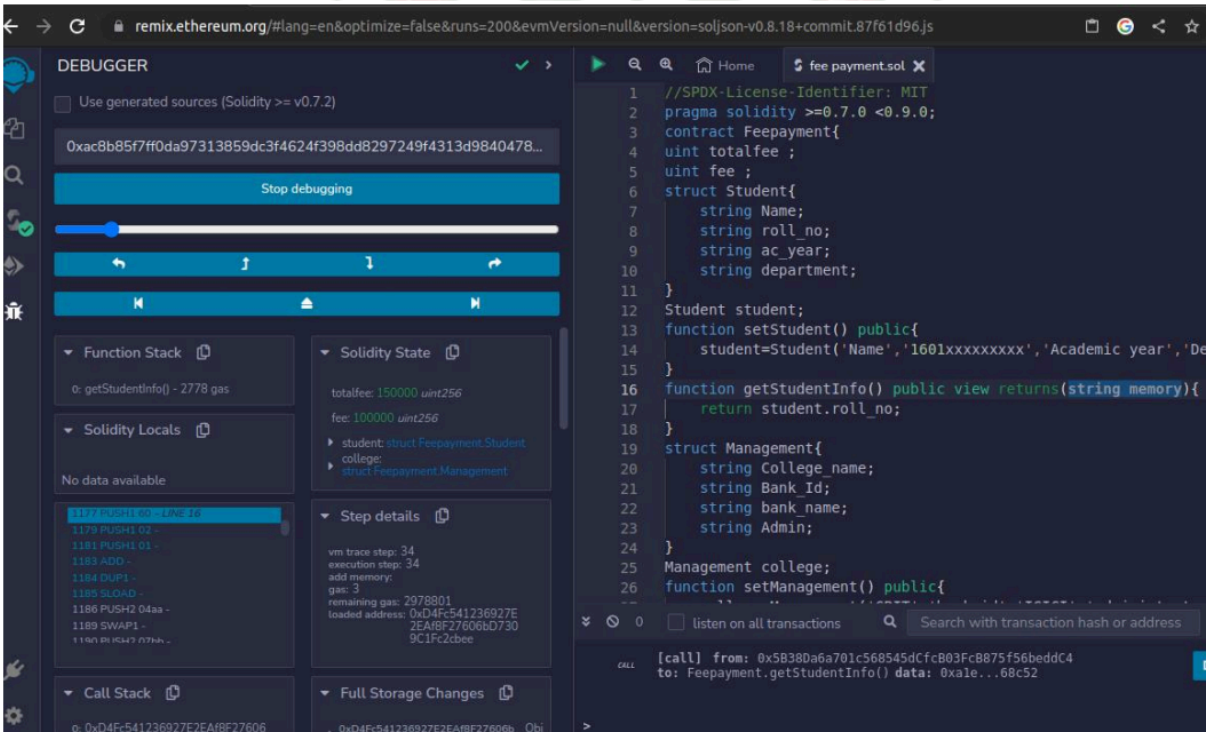
- Complexity: Blockchain technology can be complex to understand and use for the average person. This may limit the adoption of fee payment applications built on blockchain technology.
- Security: While blockchain technology is generally considered to be secure, there have been cases of security breaches and hacking attempts. This can lead to the loss of funds and personal information, which can be detrimental to users.
- Scalability: The current blockchain technology may not be able to handle a large number of transactions at once, which can lead to slow transaction speeds and higher fees.
- Regulations: The use of blockchain technology for financial transactions is a relatively new concept, and there may be regulatory challenges that need to be addressed before wider adoption can occur.

CODE:

```
//SPDX-License-Identifier: MIT pragma solidity >=0.7.0 <0.9.0; contract Feepayment{
uint totalfee ; uint fee ; struct Student{
string Name; string roll_no; string ac_year; string department;
}
Student student;
function setStudent() public{
student=Student('Name','1601xxxxxxxxx','Academic year','Department-Branch');
}
function getStudentInfo() public view returns(string memory){ return student.roll_no;
}
struct Management{ string College_name; string Bank_Id; string bank_name; string Admin;
}
Management college;
function setManagement() public{ college=Management('CBIT','bank_id','ICICI','administrator');
}
function getManagementInfo()public view returns(string memory){ return college.College_name;
}
function totalfeeset(uint) public{ totalfee = 150000;
}
```

```
function feeset(uint _paidfee) public{ fee = _paidfee;
}
function DUE() view public returns(uint){ uint due = totalfee - fee;
return due;
}
function getResult() public view returns(string memory){ if (fee ==150000){
return "The fee is paid" ;
}
else if (fee <= 149999 && fee >0){
return "There is a due in the fees to be paid";
}
else{
return "Fees is Not Paid Yet!";
}
}
}
```

OUTPUT:



4) AIM: To explore Dapps and launch using Ethereum

DESCRIPTION: A dApp (decentralized application) is a type of software application that runs on a decentralized network, such as a blockchain. Unlike traditional applications that run on centralized servers, dApps are designed to be decentralized, meaning they operate on a distributed network of computers or nodes, and the data is stored on a public ledger that is tamper-proof.

One of the key features of dApps is that they are open-source, meaning their source code is available for anyone to view, modify, or enhance. This enables anyone to contribute to the development of the dApp and ensure its integrity.

Another important feature of dApps is that they often use smart contracts, which are self- executing contracts with the terms of the agreement directly written into the code. This allows dApps to automate complex processes, such as payments or supply chain management, without requiring intermediaries or third parties.

There are many different types of dApps, including decentralized finance (DeFi) applications, gaming and entertainment applications, social media platforms, and more. Some popular dApps include Uniswap, a DeFi platform for exchanging cryptocurrencies, Cryptokitties, a blockchain-based game, and Brave, a privacy-focused web browser.

One of the advantages of dApps is that they can be more transparent and secure than traditional applications, as the data is stored on a decentralized network that is difficult to hack or corrupt. Additionally, dApps can provide greater control and ownership to users, as they can interact with the application without needing to rely on a central authority.

However, there are also challenges to developing and using dApps, such as scalability issues and the complexity of programming on decentralized networks. Overall, dApps are an exciting development in the world of software applications, offering new opportunities for innovation and collaboration.

Installation Steps:

Installing a dApp (decentralized application) can vary depending on the specific dApp and the platform you are using. However, here are some general steps you can follow to install a dApp:

1. Choose your preferred platform: dApps can be installed on various platforms such as Ethereum, EOS, TRON, etc. Choose the platform that supports the dApp you want to install.
2. Install a compatible wallet: dApps require a wallet to store and manage cryptocurrencies. Choose a compatible wallet for the platform you have chosen. For example, for Ethereum, you can install Metamask, for EOS, you can install Scatter, and for TRON, you can install TronLink.
3. Fund your wallet: Most dApps require you to have some cryptocurrency to use them. Fund your wallet with the cryptocurrency required by the dApp you want to install.
4. Locate the dApp: Find the dApp you want to install. You can search for dApps on the platform's website or through a dApp store.
5. Install the dApp: Follow the instructions provided by the dApp to install it. In most cases, you will need to approve the installation using your wallet.
6. Use the dApp: Once the dApp is installed, you can use it as per its instructions. Some dApps may require you to connect your wallet to use them.

Note: Keep in mind that dApps are still in the early stages of development, and the installation process may vary depending on the specific dApp and platform. Always follow the instructions provided by the dApp to avoid any issues.

SCREENSHOTS: