INDEX

18CSE431J-Distributed Ledger Technology

Lab-1 Demo on basic Blockchain

Aim:- To implement a basic demo on blockchain using python.

Procedure:-

- We use the DateTime library to attach a timestamp to each block that is created or mined.
- The hashlib will be used to hash a block, JSON will be used to encode the block before we hash it.
- jsonify from the Flask library will be used to return messages.
- To start building our blockchain, we create a Blockchain class. The__init__method will consist of a variable called chain to store a list of all the blocks in the blockchain.
- The create_blockchain() method will allow us to create our Genesis block on instantiation of the class.
- The create_blockchain() method will take two default arguments which are proof with a value of one(1), and the previous_hash with a value of zero(0).
- This aspect of the code shows the importance of having a background on how blockchain works.
- In a blockchain, there is always a first block called the *Genesis block*, and this block does not have any previous_hash.
- Within the create_blockchain function, we include a block variable of type dictionary that will be used to define each block in a blockchain.

The dictionary will take the following key-value pairs:

- Index: An index key will store the blockchain's length. It is represented by the chain variable in the__init__method with an added value of one(1). We will use this variable to access each block in the chain.
- Timestamp: The timestamp key will take a value of the current Date and Time the block was created or mined.
- Proof: This key will receive a proof value that will be passed to the function when called. Note that this variable refers to the proof of work.

- Previous hash: Lastly, the previous hash key takes a value of previous_hash from the function which is equivalent to the hash of the previous block.
- By adding these key-value pairs, we then append this *block* to the *chain* and return the block itself.
- We create a new variable and name it `last_block` and pass in a value of the last block in the list. Then we return the `last_block`.

The proof of work function

- In the create_blockchain() function, we had a variable called proof. This variable represents the proof of work done to mine a block.
- As the programmer of the blockchain, we need to create an algorithm that the miners will solve to mine a block successfully.
- We start by creating a new method called proof_of_work() and then we pass two parameters which are self and previous_proof.
- In the method, we create a variable to store the proof submitted by miners. We call it new_proof and set the value to one(1).
- Next, we create a control statement to check the status of the proof of work, which by default will be False.
- Therefore, we create a new variable called check_proof and assign it a False value.

Moving forward, we create a new variable called hash_operation, and we assign a value of hashlib.sha256(str(the algorithm).encode()).hexdigest().
This is how we encode the problem in a cryptographic hexadecimal digit with the use of the SHA256 hash library.

CODE:-

```
import datetime
import json
import hashlib
from flask import Flask, jsonify
class Blockchain:
  def __init__(self):
      self.chain = []
      self.create_blockchain(proof=1, previous_hash='0')

  def create_blockchain(self, proof, previous_hash):
      block = {
          'index': len(self.chain) + 1,
          'timestamp': str(datetime.datetime.now()),
          'proof': proof,
          'previous_hash': previous_hash
      }

      self.chain.append(block)
      return block

  def get_previous_block(self):
      last_block = self.chain[-1]
      return last_block

  def proof_of_work(self, previous_proof):
      # miners proof submitted
      new_proof = 1
      # status of proof of work
      check_proof = False
      while check_proof is False:
```

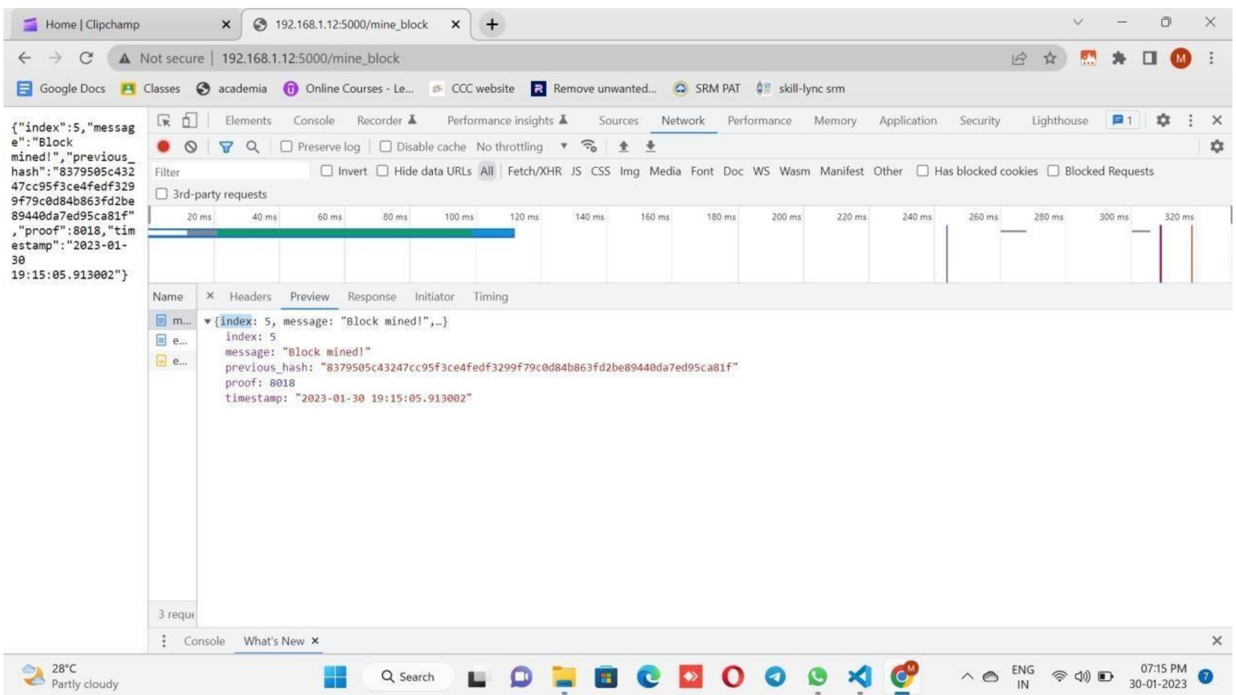
```
        # problem and algorithm based off the previous proof and new proof
                hash_operation = hashlib.sha256(str(new_proof ** 2 - previous_proof **
2).encode()).hexdigest()
        # check miners solution to problem, by using miners proof in cryptographic encryption
        # if miners proof results in 4 leading zero's in the hash operation, then:
        if hash_operation[:4] == '0000':
            check_proof = True
        else:
            # if miners solution is wrong, give mine another chance until correct
            new_proof += 1
    return new_proof

  # generate a hash of an entire block
  def hash(self, block):
    encoded_block = json.dumps(block, sort_keys=True).encode()
    return hashlib.sha256(encoded_block).hexdigest()

  # check if the blockchain is valid
  def is_chain_valid(self, chain):
    # get the first block in the chain and it serves as the previous block
    previous_block = chain[0]
    # an index of the blocks in the chain for iteration
    block_index = 1
    while block_index < len(chain):
        # get the current block
        block = chain[block_index]
            # check if the current block link to previous block has is the same as the hash of the
previous block
        if block["previous_hash"] != self.hash(previous_block):
            return False
```

```
            # get the previous proof from the previous block
            previous_proof = previous_block['proof']

            # get the current proof from the current block
            current_proof = block['proof']

            # run the proof data through the algorithm
                        hash_operation = hashlib.sha256(str(current_proof ** 2 - previous_proof **
2).encode()).hexdigest()
            # check if hash operation is invalid
            if hash_operation[:4] != '0000':
                return False
            # set the previous block to the current block after running validation on current block
            previous_block = block
            block_index += 1
        return True
app = Flask(_name_)
blockchain = Blockchain()
@app.route('/mine_block', methods=['GET'])
def mine_block():
    # get the data we need to create a block
    previous_block = blockchain.get_previous_block()
    previous_proof = previous_block['proof']
    proof = blockchain.proof_of_work(previous_proof)
    previous_hash = blockchain.hash(previous_block)
    block = blockchain.create_blockchain(proof, previous_hash)
    response = {'message': 'Block mined!',
                'index': block['index'],
                'timestamp': block['timestamp'],
                'proof': block['proof'],
                'previous_hash': block['previous_hash']}
```

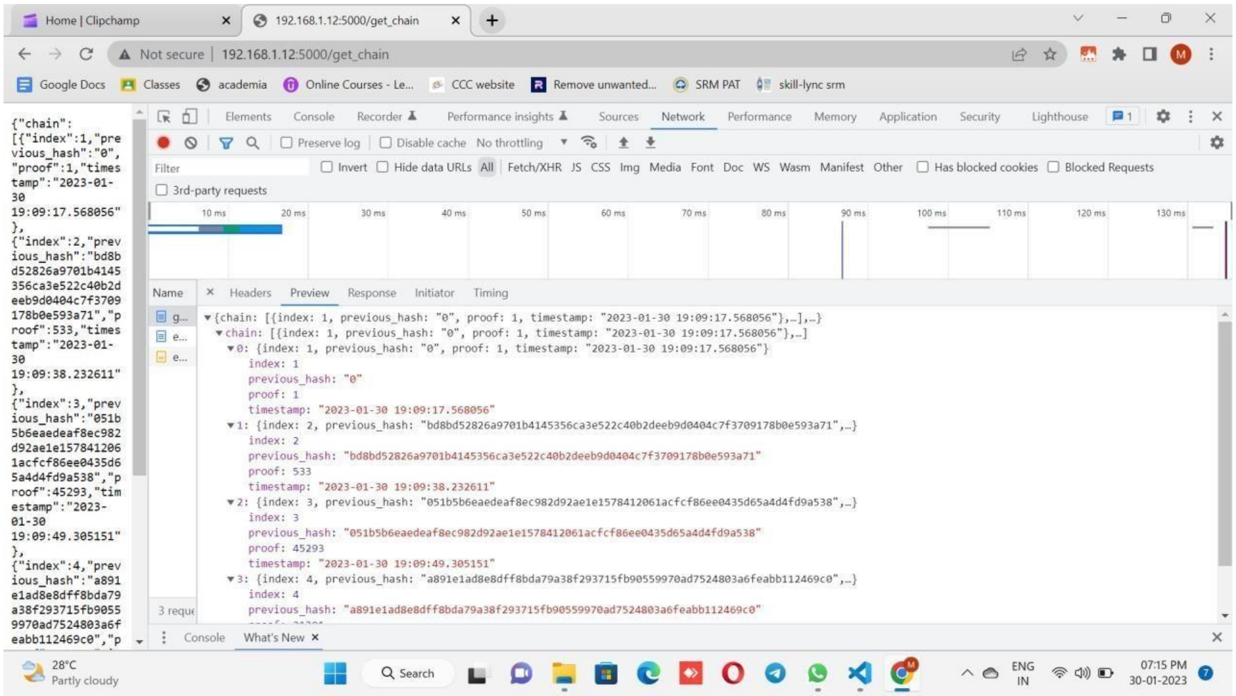
```
  return jsonify(response), 200
@app.route('/get_chain', methods=['GET'])
def get_chain():
  response = {'chain': blockchain.chain,
             'length': len(blockchain.chain)}
  return jsonify(response), 200
app.run(host='0.0.0.0', port=5000)
```

OUTPUT :-

Miner_block



Chain of blocks:-



RESULT:- The demo on blockchain is executed and implemented successfully.

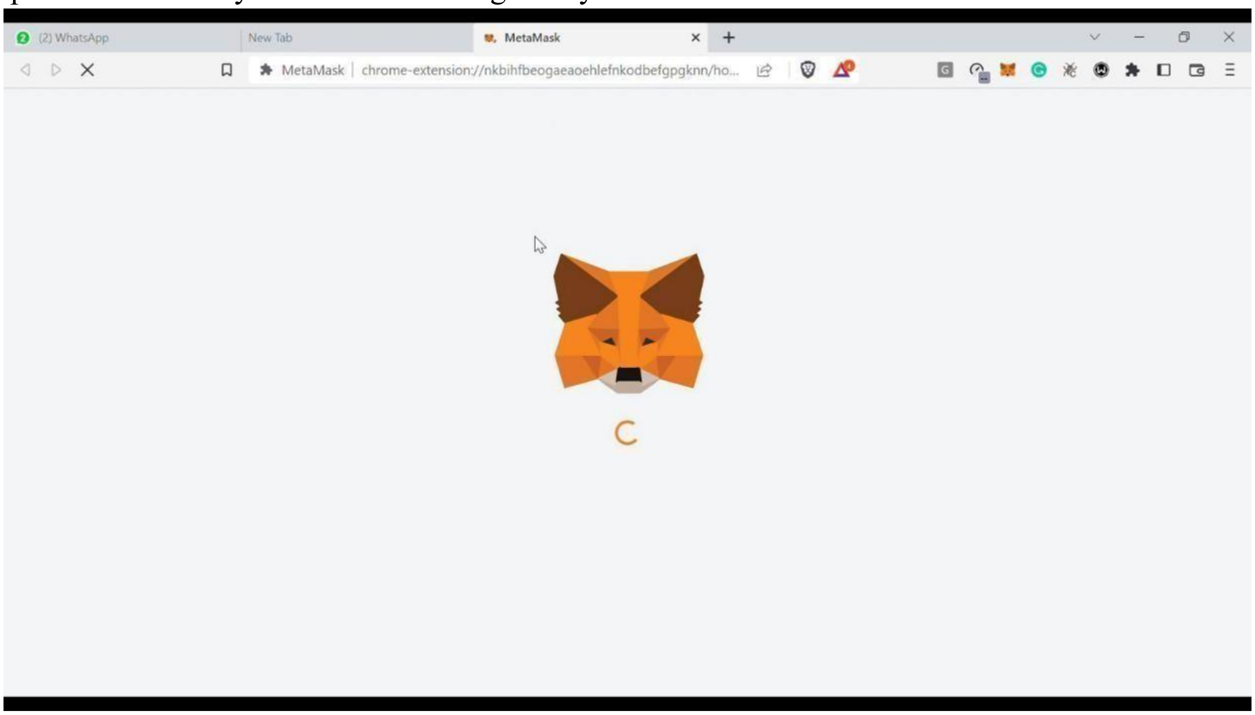
18CSE431J-Distributed Ledger Technology

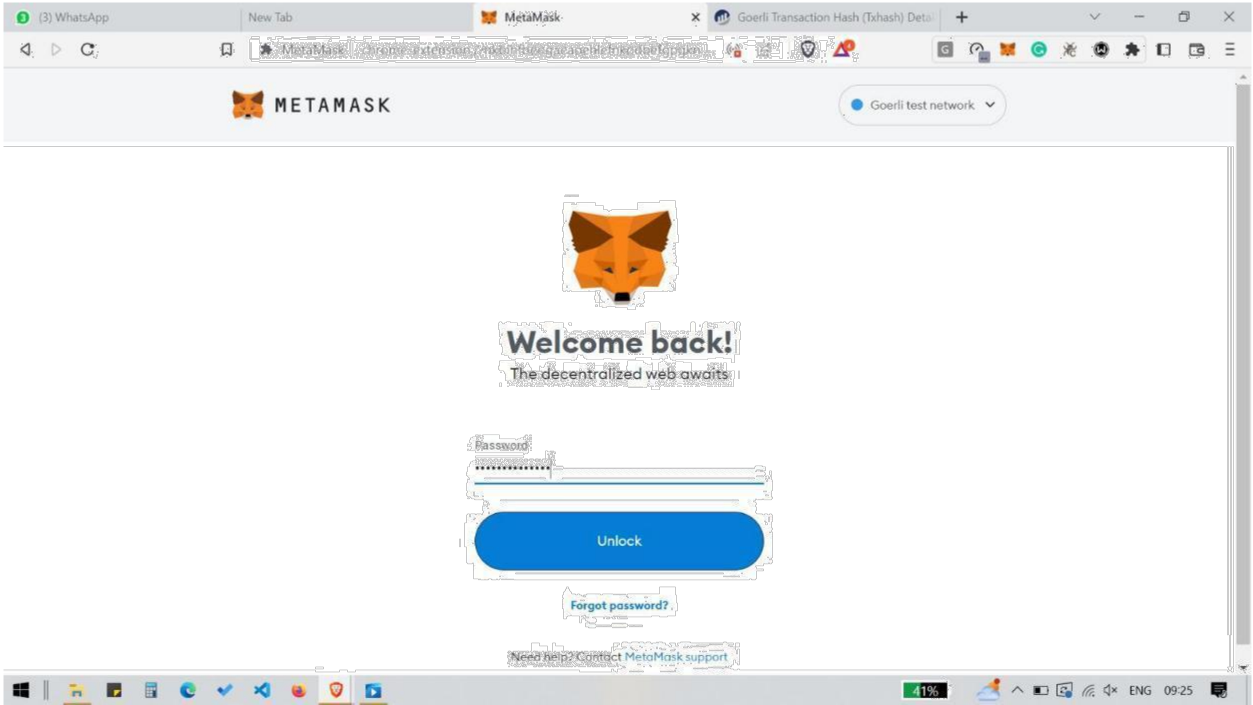
Lab-2 Demo on basic Blockchain

AIM:- To implement a basic demo on Ethereum transactions using Metamask.

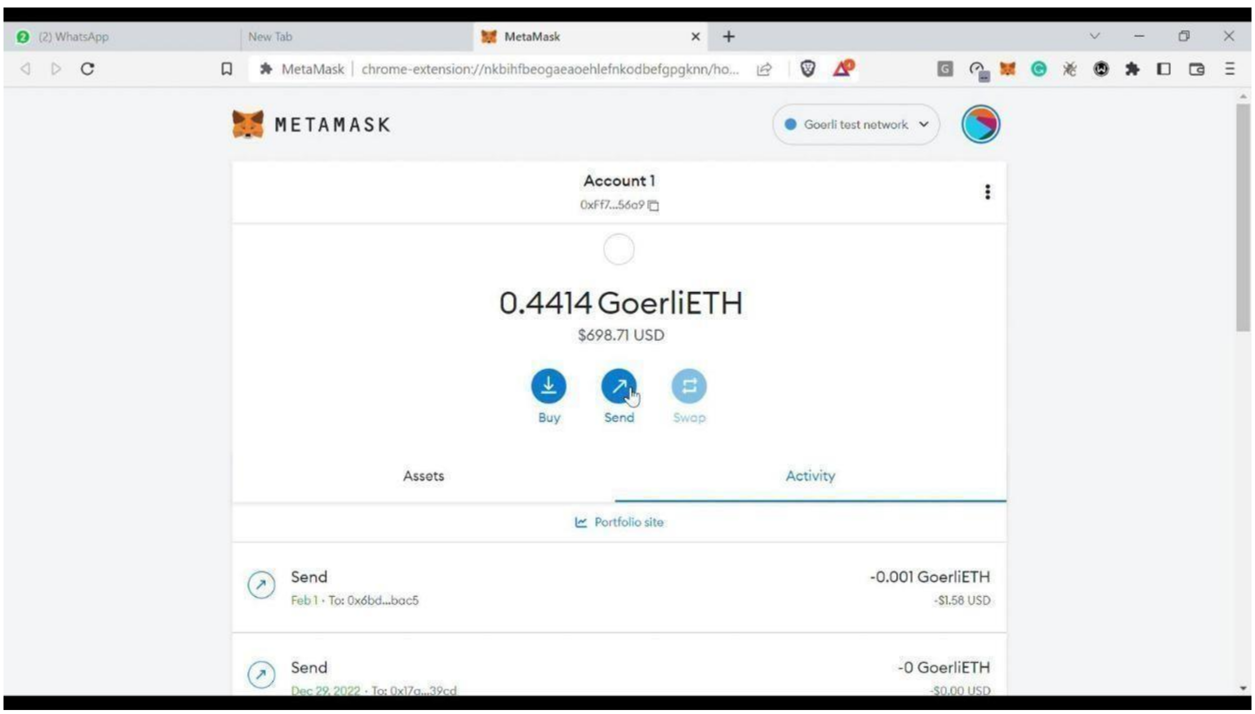
STEPS:-

1. Open metamask in your browser and log in to your account.

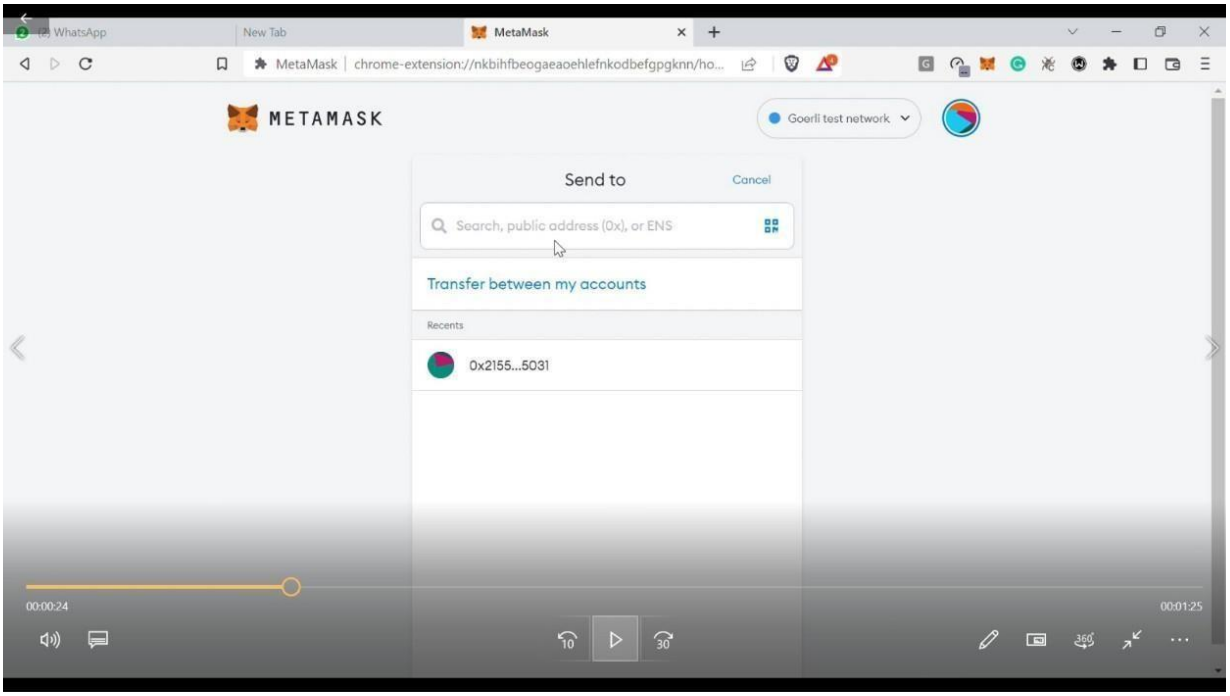


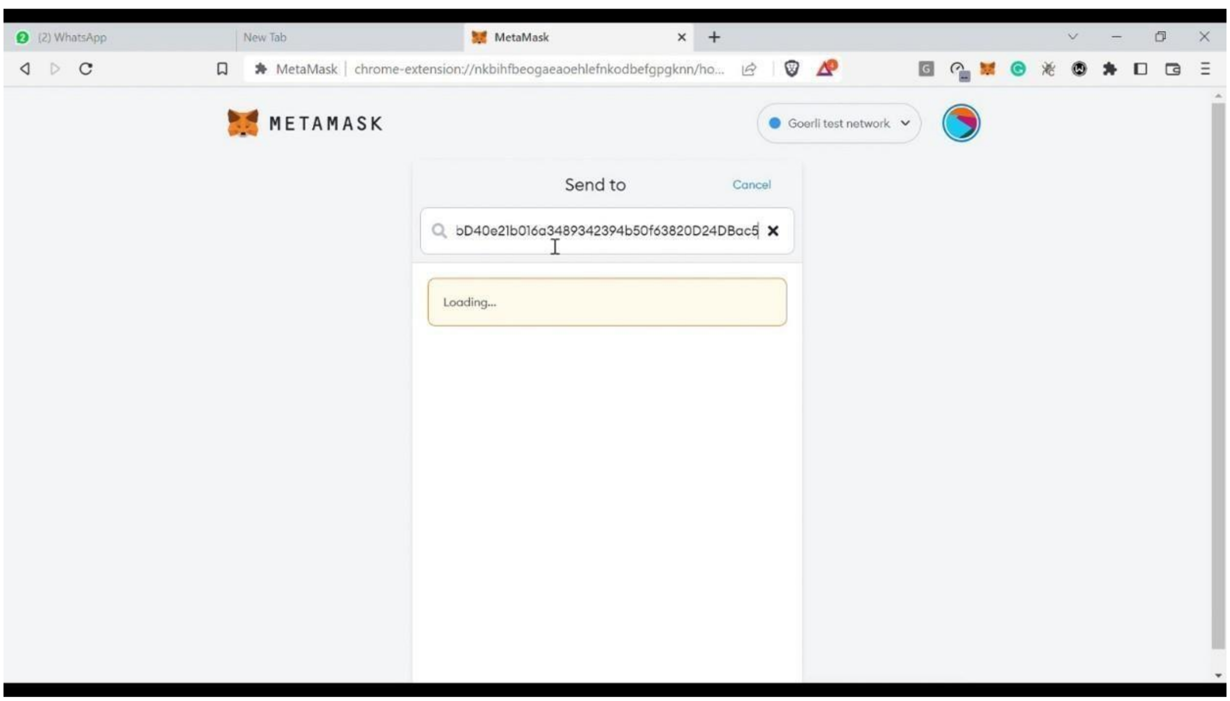


2. Click the Send button

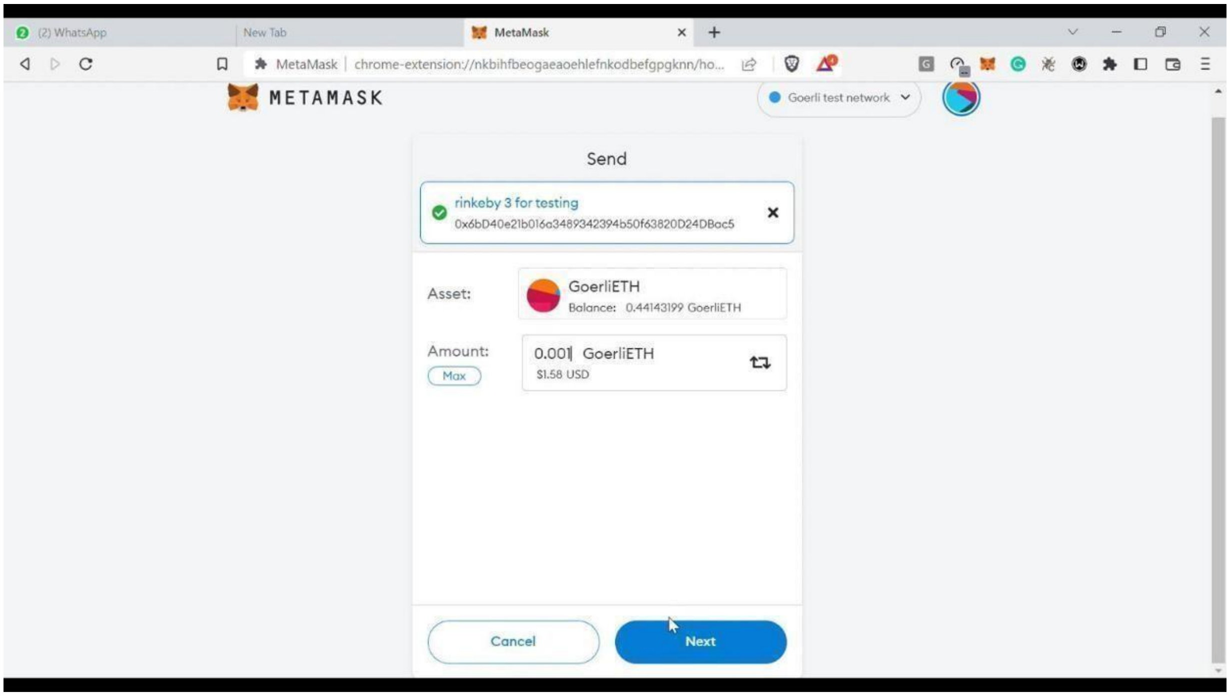


3. Enter the recipient's Ethereum address in the "To "field

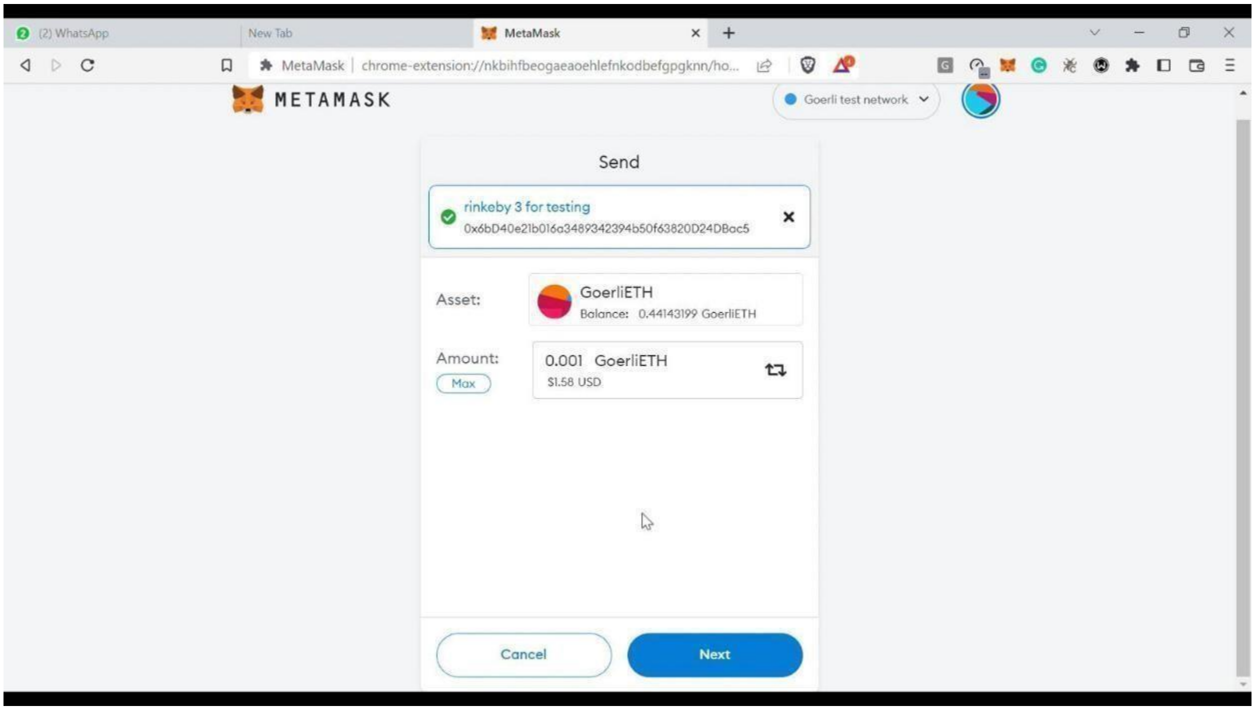




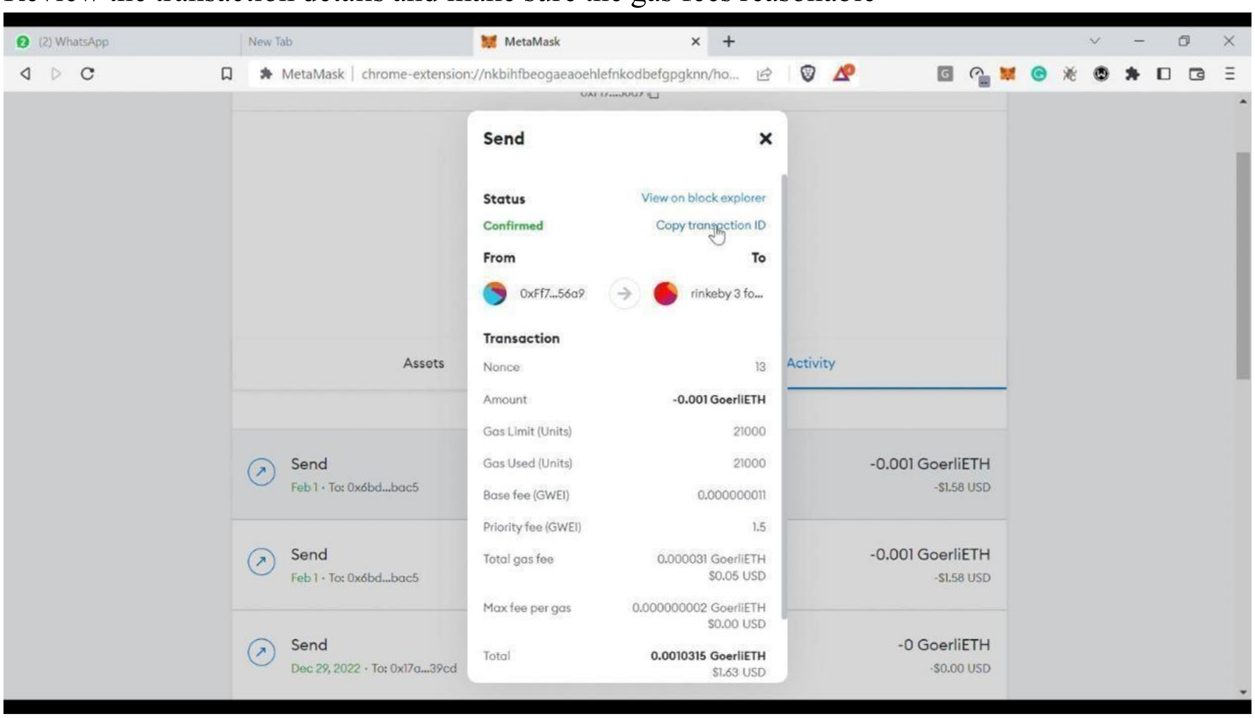
4. Enter the amount of Ethereum you want to send in the Amount field.



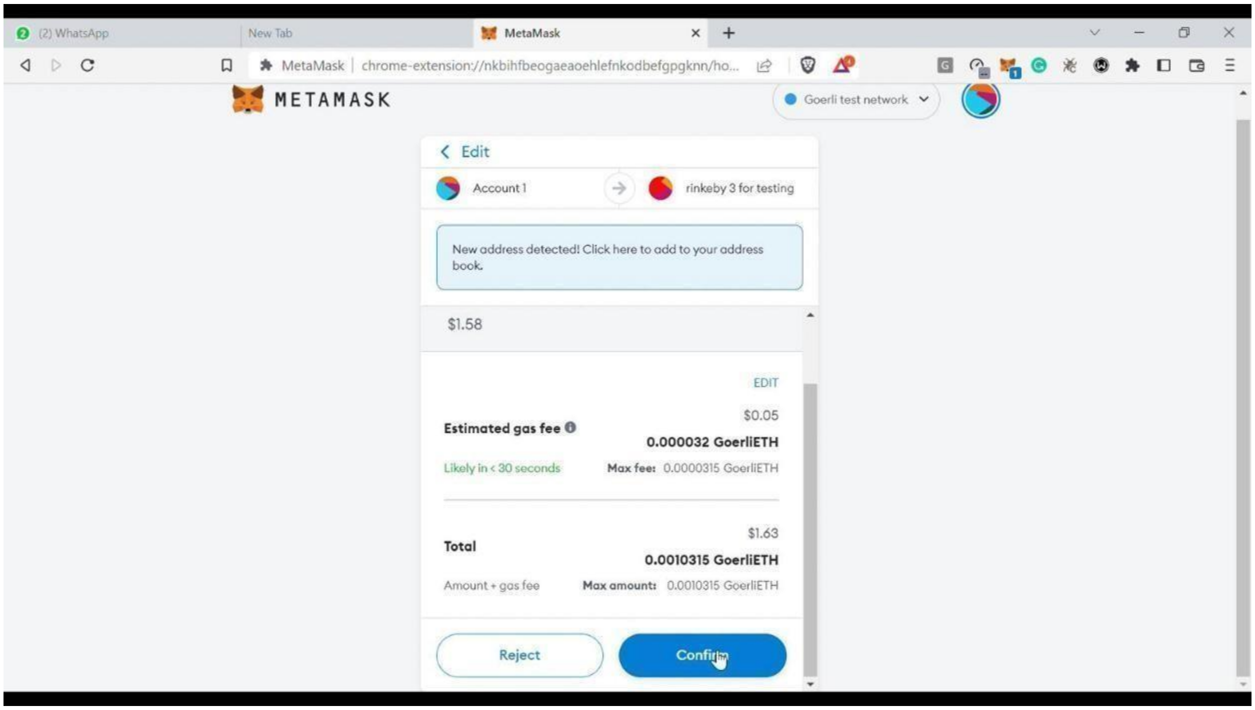
5. Verify that the details of the transaction are correct and click next.



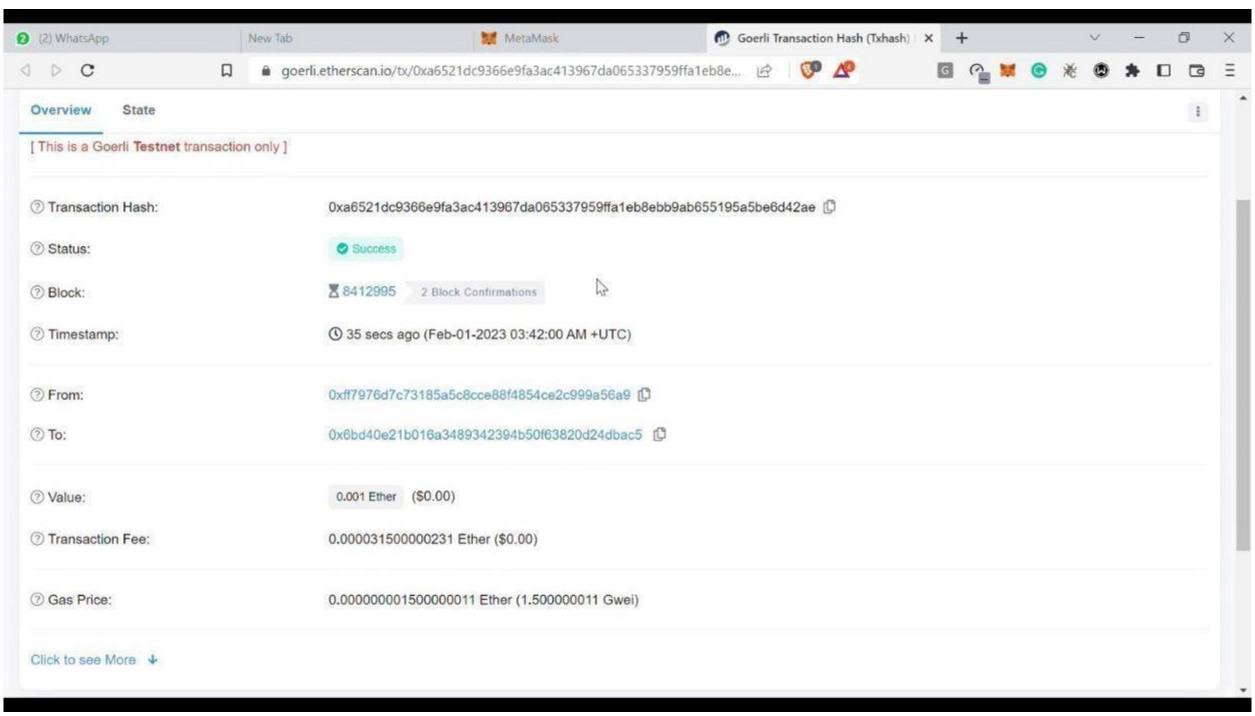
6. Review the transaction details and make sure the gas fees reasonable



7. Click confirm to initiate the transaction.



8. Wait for the transaction to be confirmed on the Ethereum Network, which can take a few minutes.



Result: Ethereum Transaction was implemented successfully.

18CSE431J-Distributed Ledger Technology

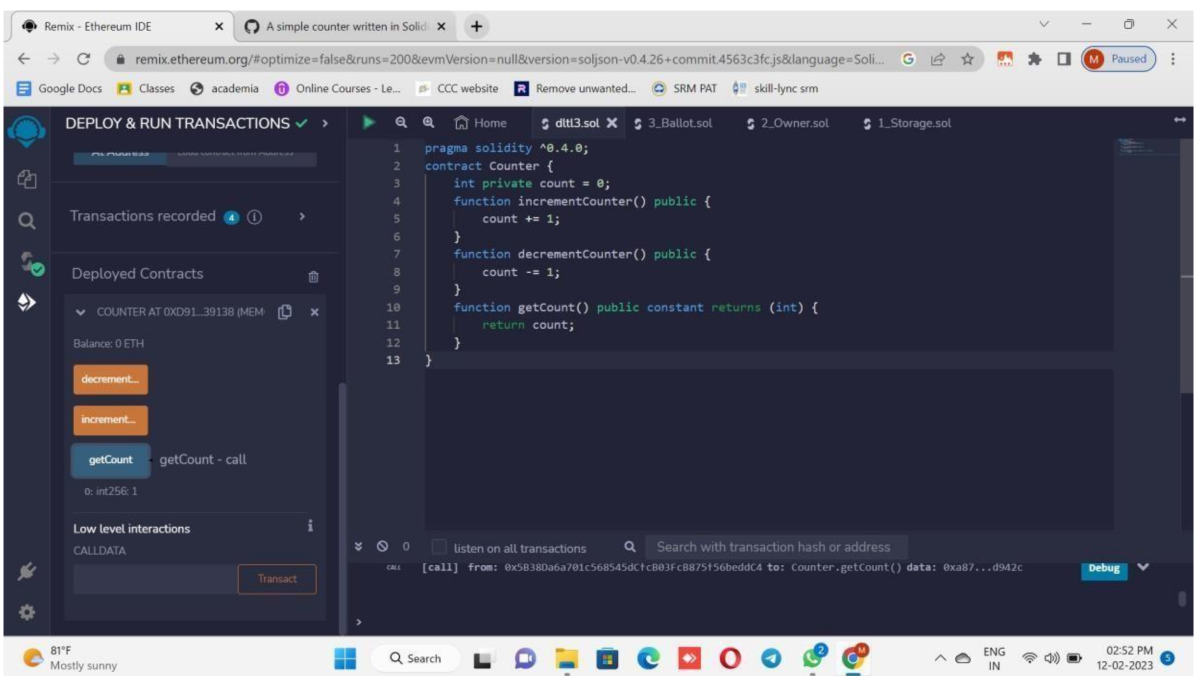
Lab-3 Solidity Smart Contract

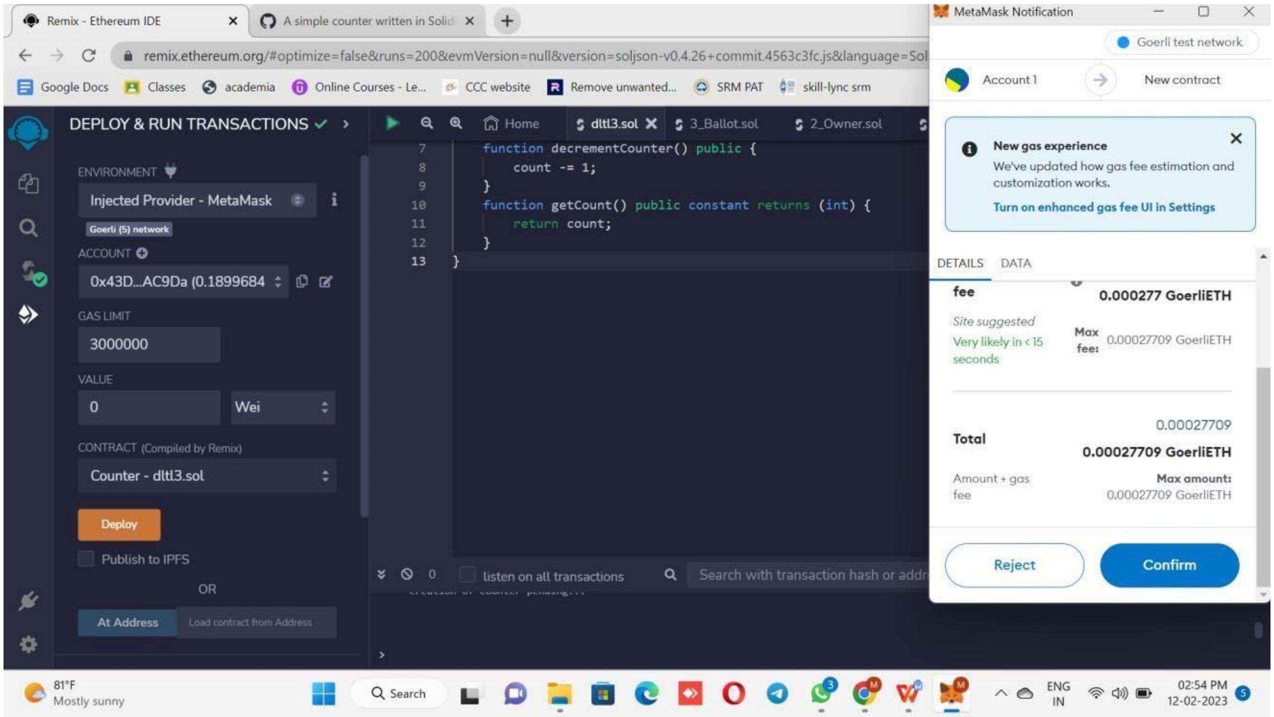
AIM:- To implement a basic demo on Solidity Smart Contract and deploy it on a test network.

STEPS:-

1. Choose the right version of Solidity and set up a development environment.
2. Define the contract structure and state variables, including data types and visibility.
3. Write the constructor and any additional functions for the contract's desired behavior.
4. Define the contract's modifiers and events, if any.
5. Add error handling and security measures, such as checking for overflow or reentrancy attacks.
6. Compile the contract using a Solidity compiler and test it on a local blockchain or a test network.
7. Deploy the contract to the desired blockchain network.
8. Monitor the contract's performance and make updates as necessary.

CODE IMPLEMENTATION:-





Result: Solidity Contract was created and deployed in the remix test network successfully.

18CSE431J-Distributed Ledger Technology

Lab-4 Implementation of various functions in Solidity

Aim: To implement and execute various functions in Solidity.

There are various types of Solidity functions we'll be covering in this section including view functions, pure functions, special functions, and fallback functions.

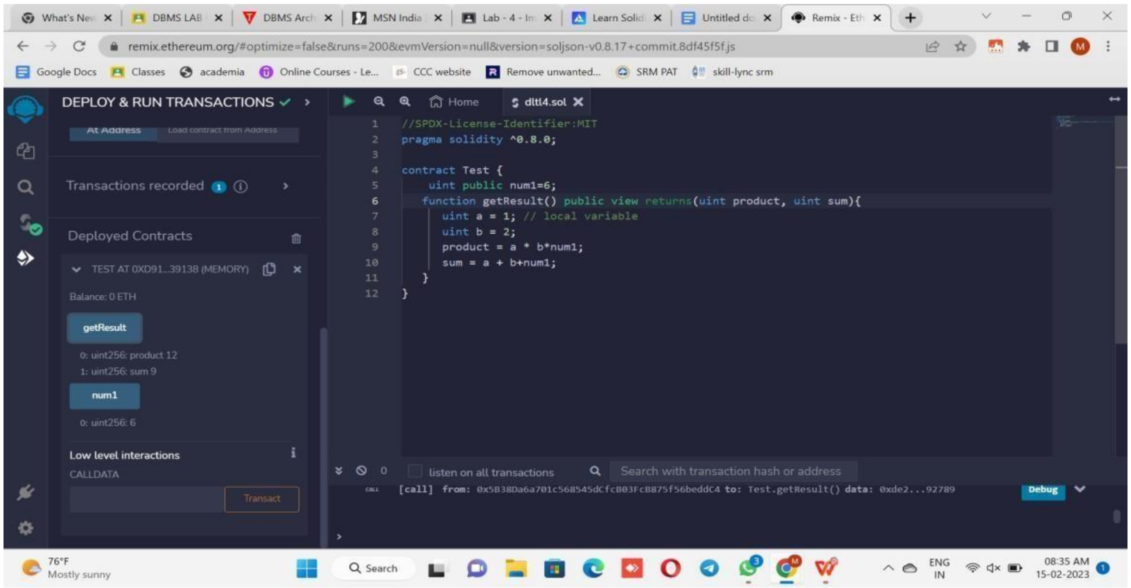
1. View Functions

In Solidity, view functions are read-only and cannot alter the state variables defined in a smart contract. The syntax for a view function is as follows:

```
function <function-name>() <access-modifier> view returns() {
        // function body
}
```

Code:-

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

contract Test {
   uint public num1=6;
  function getResult() public view returns(uint product, uint sum){
    uint a = 1; // local variable
    uint b = 2;
    product = a * b*num1;
    sum = a + b+num1;
  }
}
```



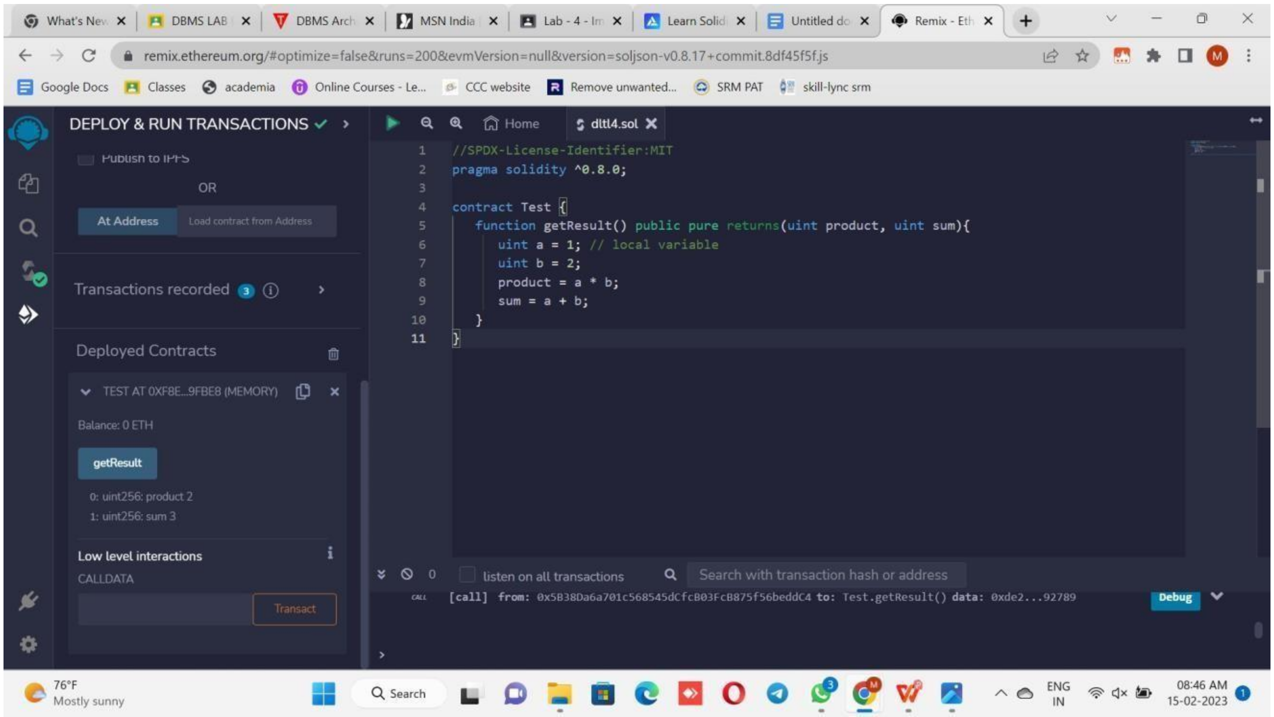
2. Pure functions

A pure function declares that no state variable will be changed or read. Typically pure functions serve some common utility or calculation. The syntax for a pure function is as follows:

function <function-name>() <access-modifier> pure returns() {
// function body
}

Code:-

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

contract Test {
  function getResult() public pure returns(uint product, uint sum){
    uint a = 1; // local variable
    uint b = 2;
    product = a * b;
    sum = a + b;
  }
}
```

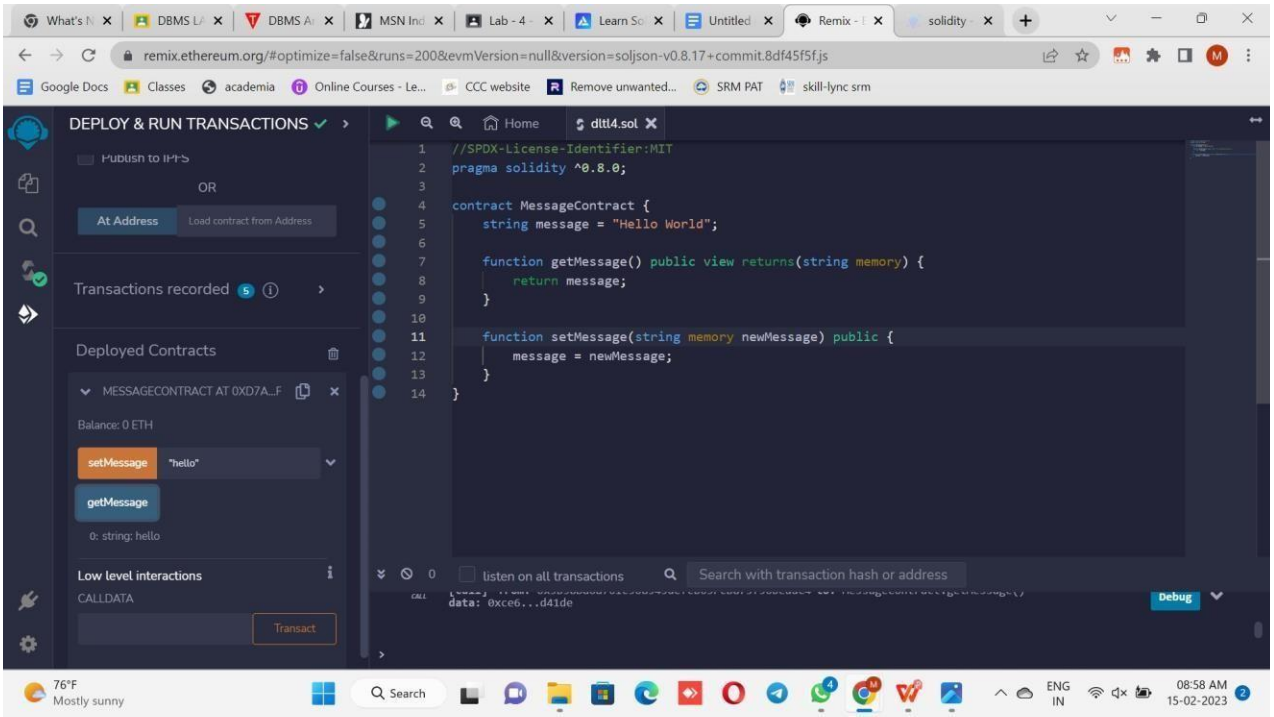


3. Getter and Setter Function

State variables defined as public have a getter function that is automatically created by the compiler. The function has the same name as the variable and has external visibility.

Code:-

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

contract MessageContract {
    string message = "Hello World";

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

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

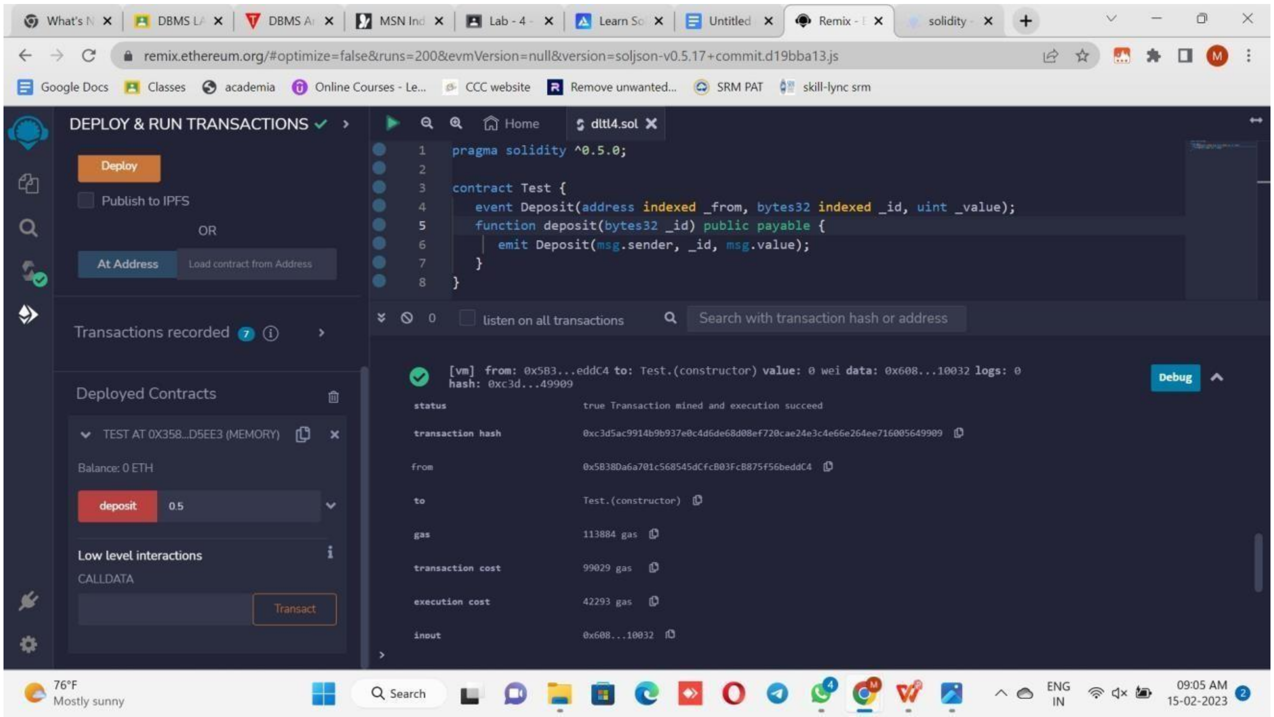


4. Event function

Event is an inheritable member of a contract. An event is emitted, it stores the arguments passed in transaction logs. These logs are stored on blockchain and are accessible using address of the contract till the contract is present on the blockchain. An event generated is not accessible from within contracts, not even the one which have created and emitted them.

Code:-

```
pragma solidity ^0.5.0;

contract Test {
  event Deposit(address indexed _from, bytes32 indexed _id, uint _value);
  function deposit(bytes32 _id) public payable {
    emit Deposit(msg.sender, _id, msg.value);
  }
}
```



RESULT:- The implementation of various functions in solidity is verified and executed successfully.

18CSE431J-Distributed Ledger Technology

Lab-5 Interacting with Ethereum Testnet

Aim:- Interacting with Ethereum Testnet.

Testnet:-

In blockchain technology, a **testnet** is an instance of a blockchain powered by the same or a newer version of the underlying software, to be used for testing and experimentation without risk to real funds or the main chain.Testnet coins are separate and distinct from the official (mainnet) coins, don't have value, and can be obtained freely from faucets.

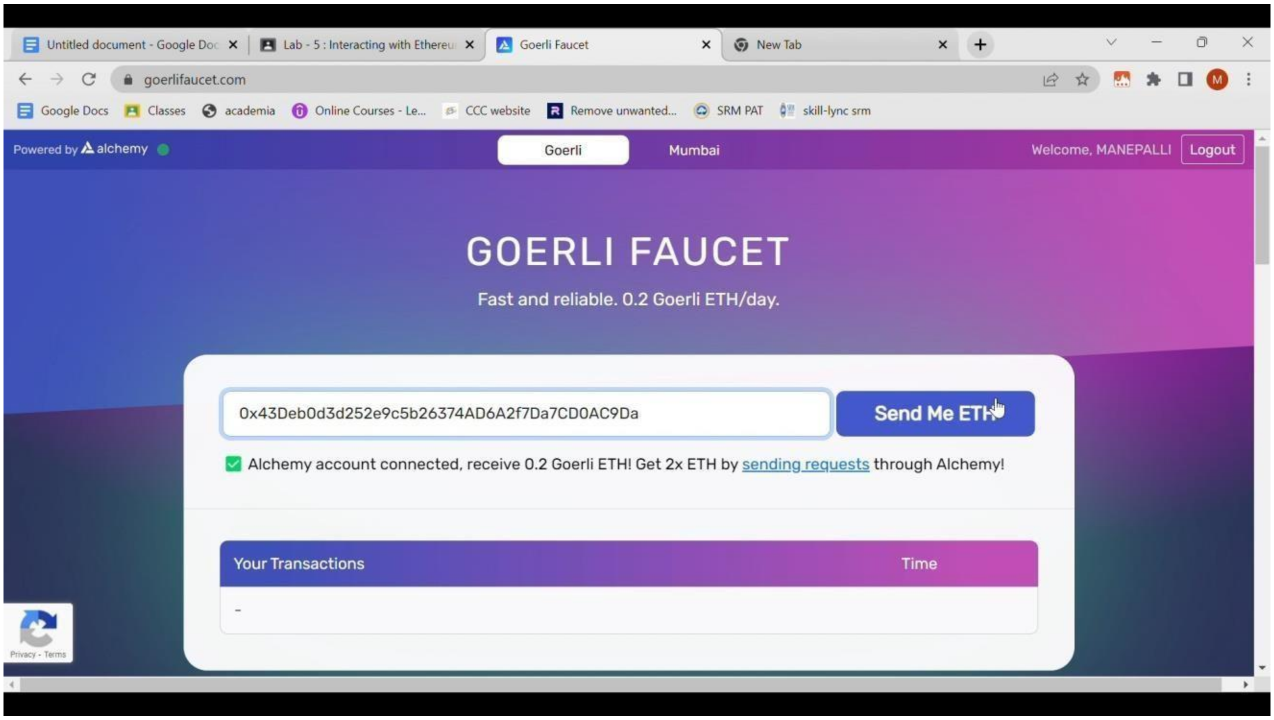
Testnets allow for the development of blockchain applications without the risk of losing funds.

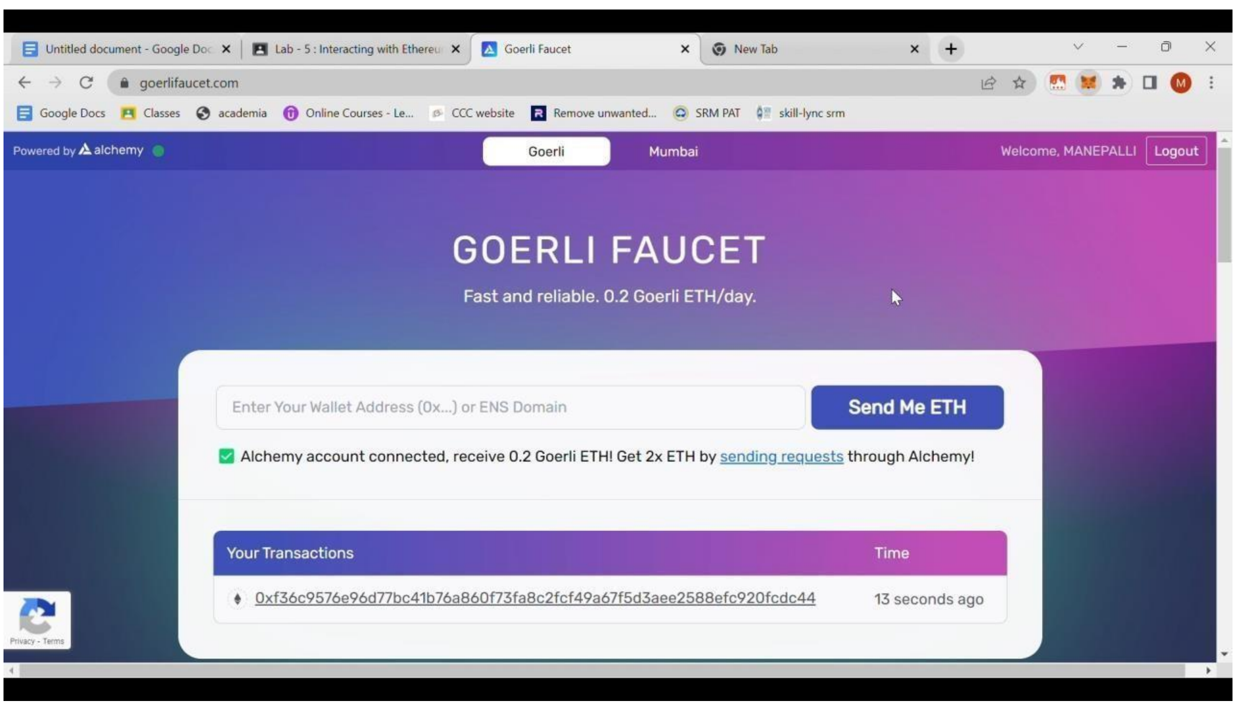
Faucet:-

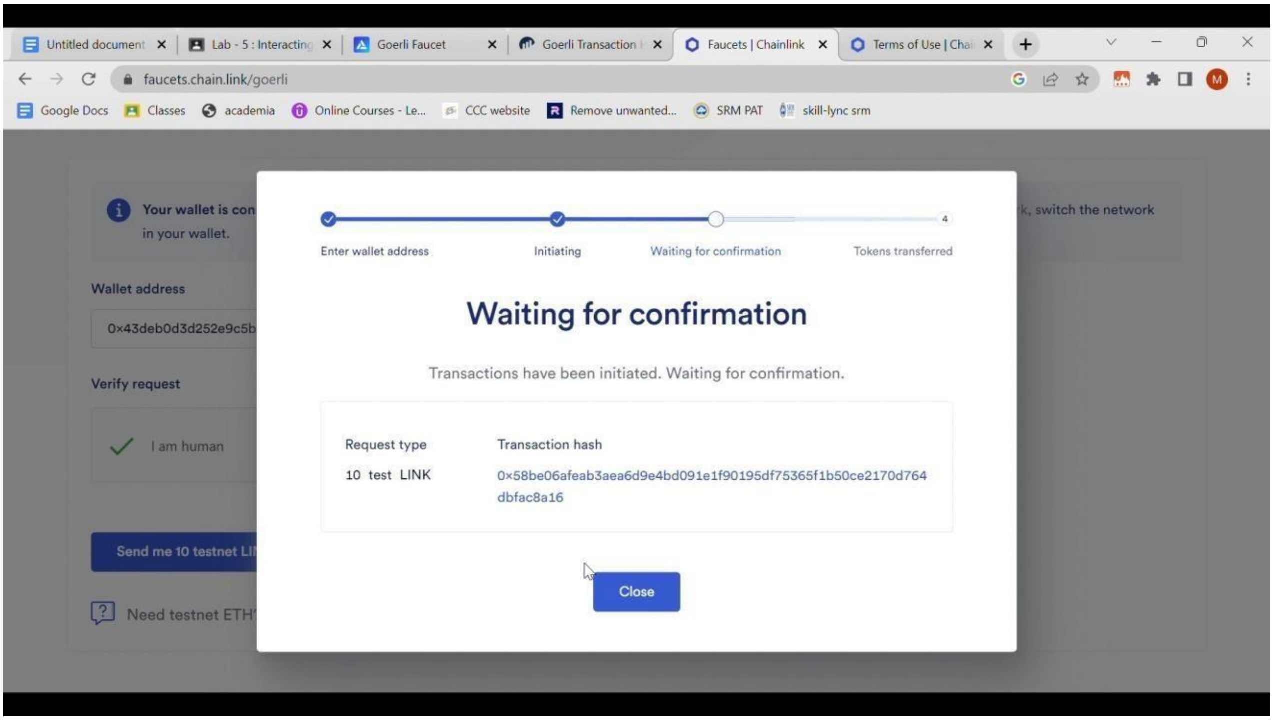
A crypto faucet lets users earn small crypto rewards by completing simple tasks. The metaphor is based on how even one drop of water from a leaky faucet could eventually fill up a cup. There are various kinds of crypto faucets, including bitcoin (BTC), Ethereum (ETH), and BNB faucets.

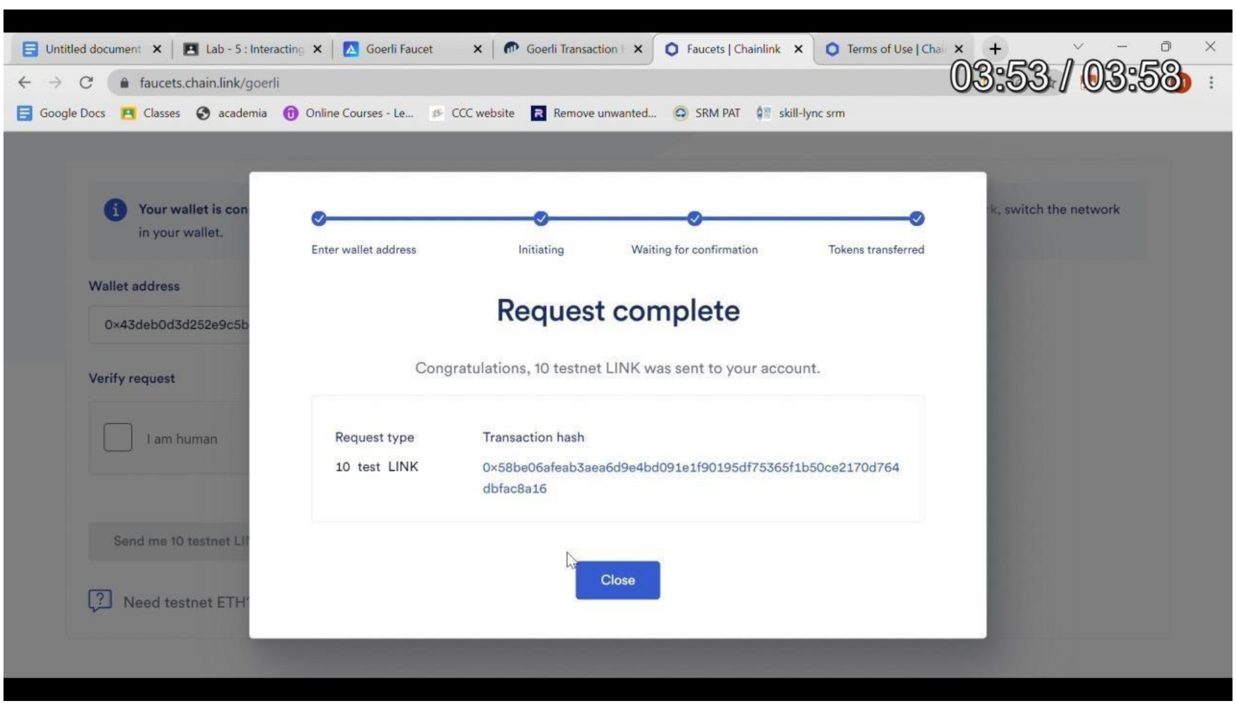
Steps:-

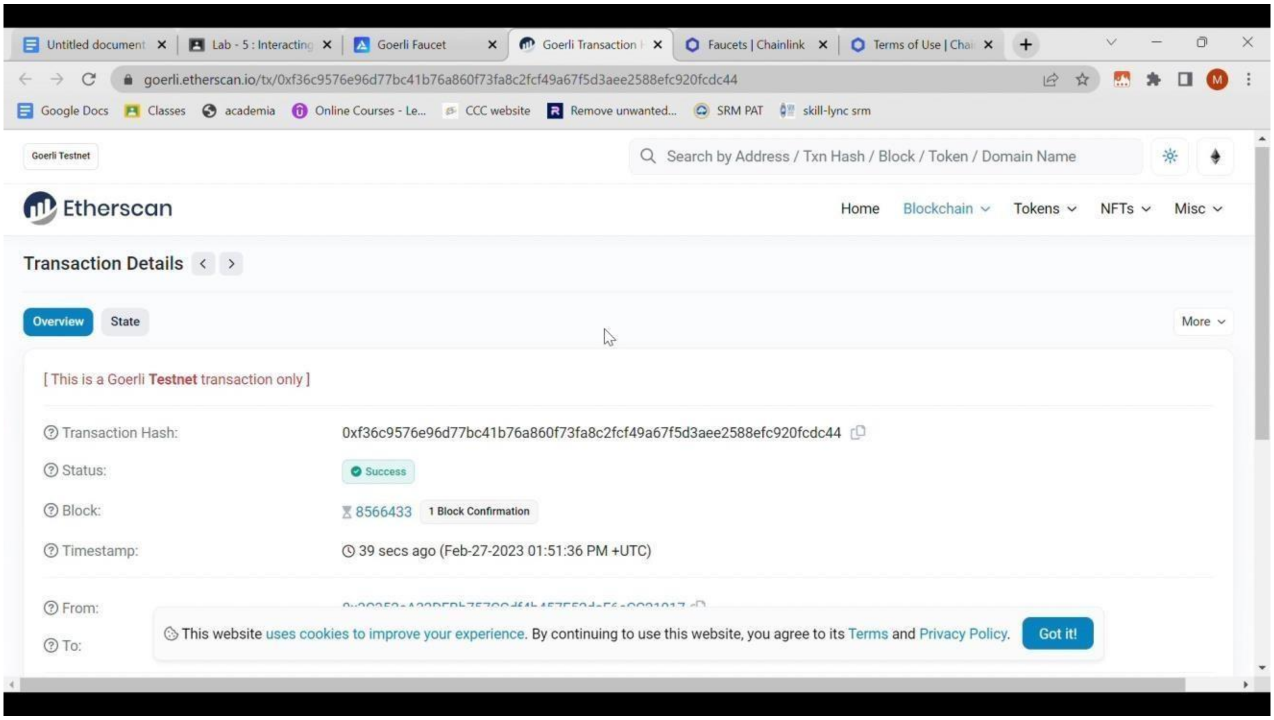
- Open goerli faucet and enter the metamask public address.
- Request eth from goerli faucet and wait for the confirmation.
- Verify transaction in blockchain explorer.
- Open chain link faucet and connect your metamask public address.
- Request eth and wait for the confirmation message.


Untitled document - Google Doc
Lab - 5 : Interacting with Ethereu
Goerli Faucet
New Tab
goerlifaucet.com
Google Docs
Classes
academia
Online Courses - Le...
CCC website
Remove unwanted...
SRM PAT
skill-lync srm
Powered by alchemy
Goerli
Mumbai
Welcome, MANEPALLI
Logout
GOERLI FAUCET
Fast and reliable. 0.2 Goerli ETH/day.
0x43Deb0d3d252e9c5b26374AD6A2f7Da7CD0AC9Da
Send Me ETH
Alchemy account connected, receive 0.2 Goerli ETH! Get 2x ETH by sending requests through Alchemy!
Your Transactions
Time
Privacy - Terms


Untitled document - Google Doc
Lab - 5 : Interacting with Ethereu
Goerli Faucet
New Tab
goerlifaucet.com
Google Docs
Classes
academia
Online Courses - Le...
CCC website
Remove unwanted...
SRM PAT
skill-lync srm
Powered by alchemy
Goerli
Mumbai
Welcome, MANEPALLI
Logout
GOERLI FAUCET
Fast and reliable. 0.2 Goerli ETH/day.
Enter Your Wallet Address (0x...) or ENS Domain
Send Me ETH
Alchemy account connected, receive 0.2 Goerli ETH! Get 2x ETH by sending requests through Alchemy!
Your Transactions
Time
0xf36c9576e96d77bc41b76a860f73fa8c2fcf49a67f5d3aee2588efc920fcdc44
13 seconds ago
Privacy - Terms


faucets.chain.link/goerli
Enter wallet address
Initiating
Waiting for confirmation
Tokens transferred
Waiting for confirmation
Transactions have been initiated. Waiting for confirmation.
Request type
Transaction hash
10 test LINK
0x58be06afeab3aea6d9e4bd091e1f90195df75365f1b50ce2170d764dbfac8a16
Close


faucets.chain.link/goerli
03:53 / 03:58
Enter wallet address
Initiating
Waiting for confirmation
Tokens transferred
Request complete
Congratulations, 10 testnet LINK was sent to your account.
Request type
Transaction hash
10 test LINK
0x58be06afeab3aea6d9e4bd091e1f90195df75365f1b50ce2170d764dbfac8a16
Close



Result:-The implementation of interacting with ethereum testnet is verified and executed successfully.

18CSE431J-Distributed Ledger Technology

Lab-6 Using Inheritance in Solidity

Aim:- Implementation of inheritance in **SOLIDITY**.

Inheritance is one of the most important features of the object-oriented programming language. It is a way of extending the functionality of a program, used to separate the code, reduces the dependency, and increases the re-usability of the existing code.

Solidity supports inheritance between smart contracts, where multiple contracts can be inherited into a single contract. The contract from which other contracts inherit features is known as a base contract, while the contract which inherits the features is called a derived contract. Simply, they are referred to as parent-child contracts.

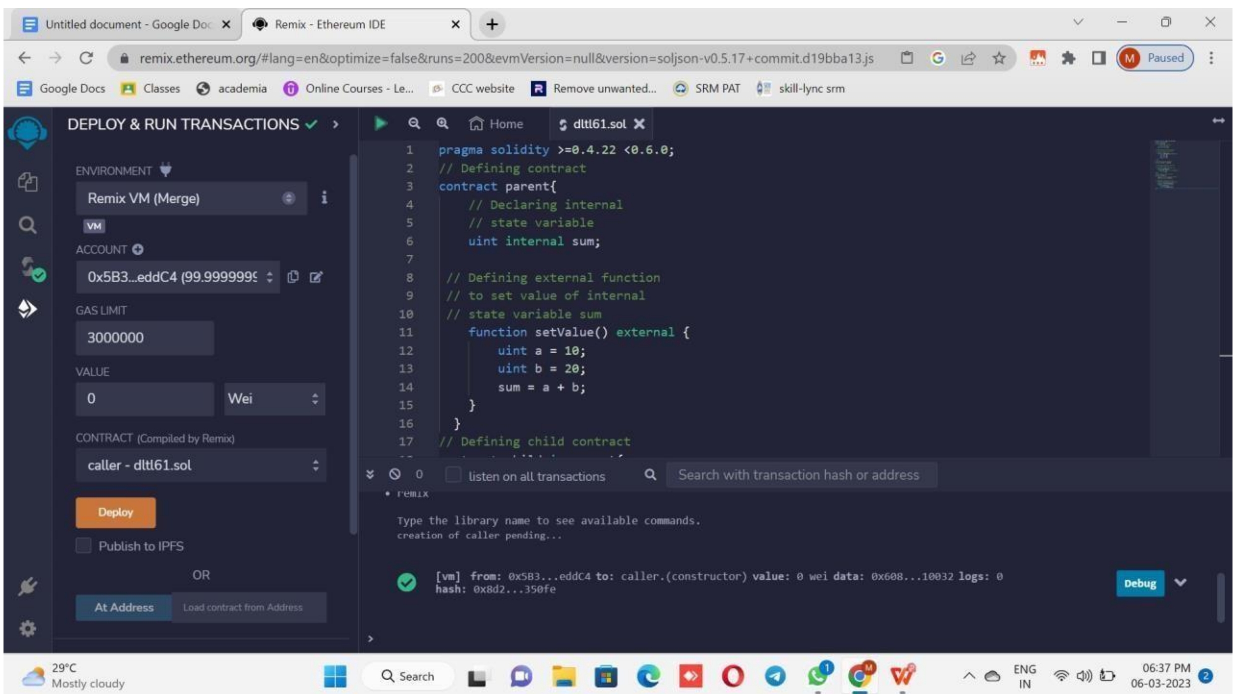
SINGLE LEVEL INHERITANCE:-

Code:-

```
pragma solidity >=0.4.22 <0.6.0;
// Defining contract
contract parent{
   // Declaring internal
   // state variable
   uint internal sum;

 // Defining external function
 // to set value of internal
 // state variable sum
   function setValue() external {
       uint a = 10;
       uint b = 20;
       sum = a + b;
   }
  }
// Defining child contract
contract child is parent{

 // Defining external function
 // to return value of
 // internal state variable sum
   function getValue(
```

```
    ) external view returns(uint) {
        return sum;
  } }
// Defining calling contract
contract caller {
 // Creating child contract object
    child cc = new child();

 // Defining function to call
 // setValue and getValue functions
    function testInheritance(
    ) public returns (uint) {
        cc.setValue();
        return cc.getValue();
  } }
```

OUTPUT:-



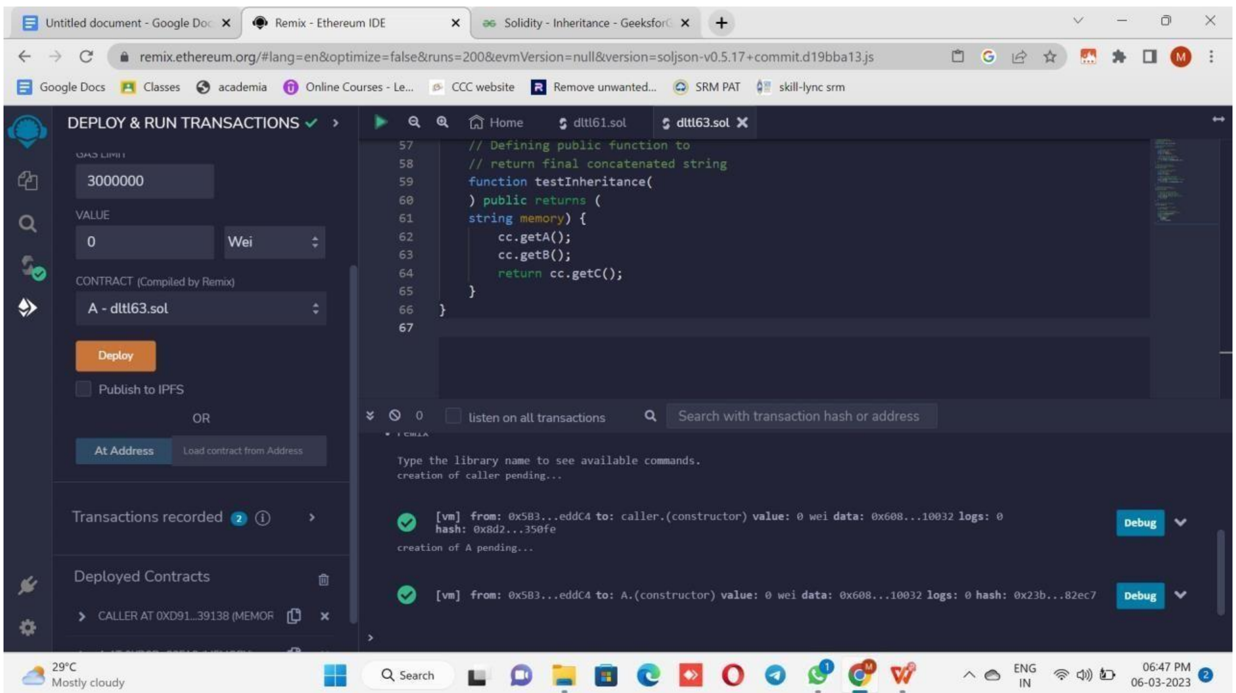
MULTILEVEL INHERITANCE:-

Code:-

```
pragma solidity >=0.4.22 <0.6.0;
// Defining parent contract A
contract A {
 // Declaring internal
 // state variable
 string internal x;

 // Defining external function
 // to set value of
 // internalstate variable
 function getA() external {
 x = "Vamsi Manepalli";
 }

 // Declaring internal
 // state variable
 uint internal sum;

 // Defining external function
 // to set the value of
 // internal state variable sum
 function setA() external {
 uint a = 10;
 uint b = 20;
 sum = a + b;
 } }
// Defining child contract B
// inheriting parent contract A
contract B is A {
 // Defining external function to
 // return state variable x
 function getAstr(
 ) external view returns(string memory){
 return x;
 }
}
contract C is A {
 function getAValue(
```

```
) external view returns(uint){
return sum;
} }
contract caller {
B contractB = new B();
C contractC = new C();
function testInheritance(
) public returns (
string memory, uint) {
return (
contractB.getAstr(), contractC.getAValue());
} }
```

OUTPUT:-

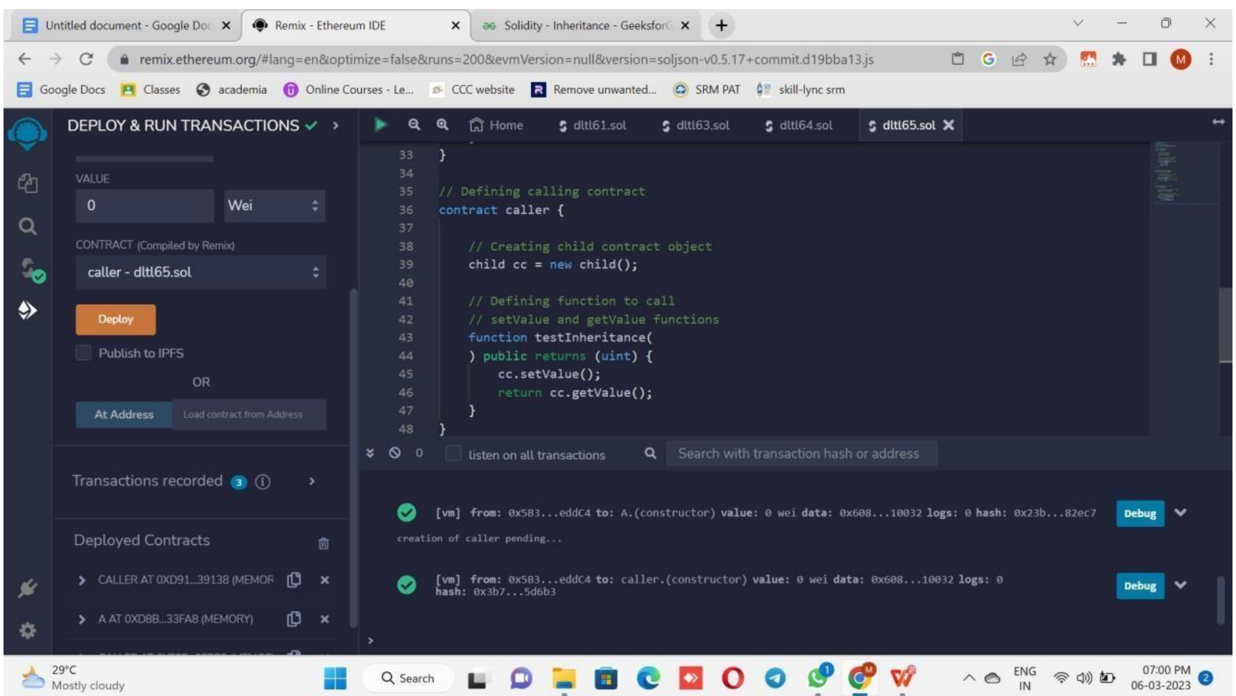


MULTIPLE INHERITANCE:-
Code:-

```
pragma solidity >=0.4.22 <0.6.0;
// Defining contract A
contract A {
 // Declaring internal
 // state variable
 string internal x;
 // Defining external function
 // to set value of
 // internal state variable x
 function setA() external {
 x = "GeeksForGeeks";
 } }
// Defining contract B
contract B {
 // Declaring internal
// state variable
 uint internal pow;
 // Defining external function
 // to set value of internal
 // state variable pow
 function setB() external {
 uint a = 2;
 uint b = 20;
 pow = a ** b;

 } }
// Defining child contract C
// inheriting parent contract
// A and B
contract C is A, B {
// Defining external function
// to return state variable x
function getStr( ) external returns(string memory) {
 return x;
 }
 // Defining external function
 // to return state variable pow
 function getPow(
```

```
) external returns(uint) {
return pow;
} }
// Defining calling contract
contract caller {
// Creating object of contract C
C contractC = new C();
// Defining public function to
// return values from functions
// getStr and getPow
function testInheritance(
) public returns(string memory, uint) {
contractC.setA();
contractC.setB();
return (
contractC.getStr(), contractC.getPow());
} }
```

OUTPUT:-



RESULT:- Inheritance using solidity is implemented and executed successfully.

18CSE431J-Distributed Ledger Technology

Lab-7 Implementation of Basic commands in Hyperledger

Aim:-
To understand the fundamentals of Hyperledger Fabric and learn basic commands for opening and closing a network in Hyperledger. This includes learning how to start and stop a network using simple commands.

Pre-requisites:-
- Git
- IDE
- Docker Desktop.

Introduction:-

- Hyperledger is an open source project created to support the development of blockchain-based distributed ledgers. Hyperledger consists of a collaborative effort to create the needed frameworks, standards, tools and libraries to build blockchain and related applications.
- Since Hyperledger's creation by the Linux Foundation in 2016, the project has had contributions from organizations such as IBM and Intel, Samsung, Microsoft, Visa, American Express and blockchain startups such as Blockforce. In all, the collaboration includes banking, supply chain management, internet of things (IOT), manufacturing and production-based fields.
- Hyperledger acts as a hub for different distributed ledger frameworks and libraries. With this, a business could use one of Hyperledger's frameworks, for example, to improve the efficiency, performance and transactions in their business processes.
- Hyperledger works by providing the needed infrastructure and standards for developing blockchain systems and applications. Developers use Hyperledger Greenhouse (the frameworks and tools that make up Hyperledger) to develop business blockchain projects. Network participants are all known to each other and can participate in consensus-making processes.

Implementation:

1. Install the required binaries, images and dockers of Hyperledger. Visit the official documentation for choosing your required version.https://hyperledger-fabric.readthedocs.io/en/release-2.3/install.html#installing-the-latest-release

2. Open your favorite IDE, most preferably VScode. Try to clone the official GitHub repository of Hyperledger Fabric. Type the below command in the terminal. command – git clone https://github.com/hyperledger/fabric-samples

3. Change the directory to test-network so as to get the access of switching the network on and off. Command – cd test-network

4. Create a repository called chaincode, which is used for the deployment of chaincode contracts which are primitive for Hyperledger. Command – mkdir chaincode

5. Open the new network in the chaincode directory for a new connection using the below command. Command - /.network.sh up . This command creates a Fabric network that consists of two peer nodes, one ordering node. No channel is created when you run ./network.sh up

6. To stop the network after the connection, use the following command: Command - /.network.sh down. This command tops the Fabric.

Output:

PS C:\Users\Asus\Desktop\Distributed Ledger\fabric-samples> cd .\test-network\

PS C:\Users\Asus\Desktop\Distributed Ledger\fabric-samples\test-network> ./network.sh up

PS C:\Users\Asus\Desktop\Distributed Ledger\fabric-samples\test-network>

[main 2023-03-16T04:00:57.747Z] update#setState idle [main 2023-03-16T04:01:00.392Z] [UtilityProcess id: 1, type: extensionHost, pid: <none>]: creating new...

[main 2023-03-16T04:01:00.458Z] [UtilityProcess id: 1, type: extensionHost, pid: 10616]: successfully created

18CSE431J-Distributed Ledger Technology

Lab-8 Implementation of Merkle Tree using Solidity

Aim:-
To implement Merkle tree using solidity.

Introduction:-
A Merkle tree is a hash-based data structure that is a generalization of the hash list. It is a tree structure in which each leaf node is a hash of a block of data, and each non-leaf node is a hash of its children. Typically, Merkle trees have a branching factor of 2, meaning that each node has up to 2 children.

Merkle trees are used in distributed systems for efficient data verification. They are efficient because they use hashes instead of full files. Hashes are ways of encoding files that are much smaller than the actual file itself. Currently, their main uses are in peer-to-peer networks such as Tor, Bitcoin, and Git.

In various distributed and peer-to-peer systems, data verification is very important. This is because the same data exists in multiple locations. So, if a piece of data is changed in one location, it's important that data is changed everywhere. Data verification is used to make sure data is the same everywhere.

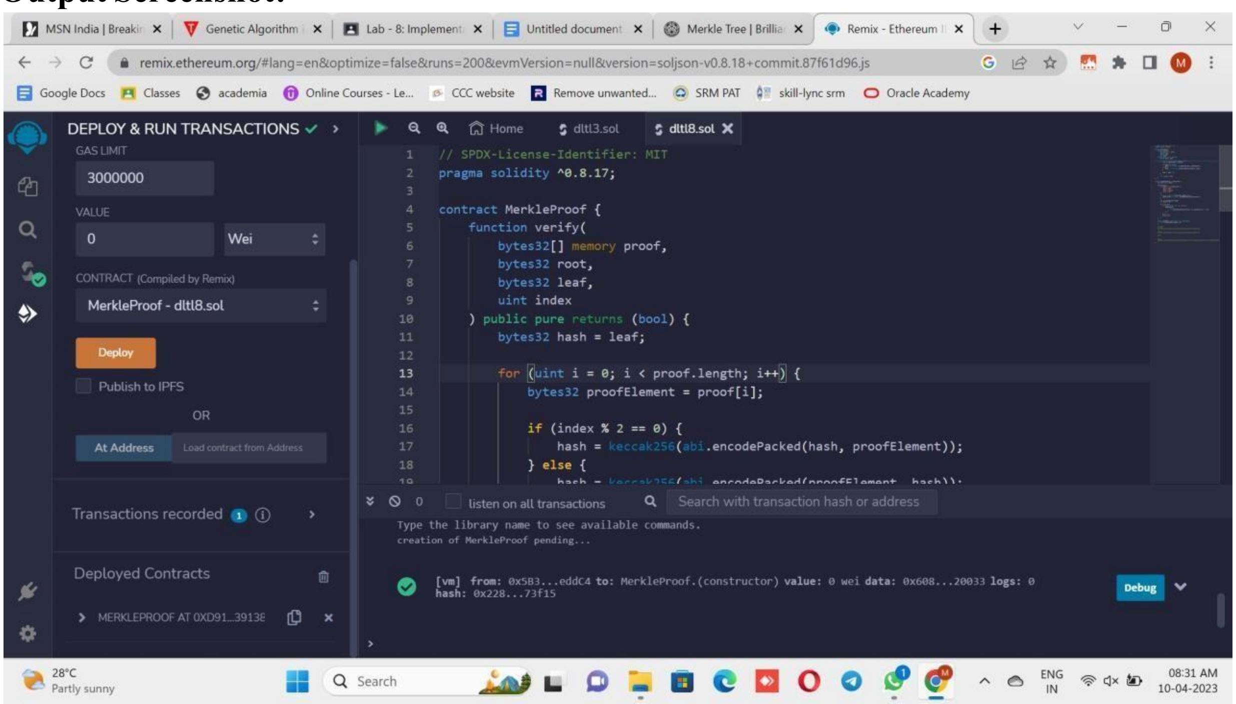
However, it is time-consuming and computationally expensive to check the entirety of each file whenever a system wants to verify data. So, this is why Merkle trees are used. Basically, we want to limit the amount of data being sent over a network (like the Internet) as much as possible. So, instead of sending an entire file over the network, we just send a hash of the file to see if it matches.

Code:-
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

contract MerkleProof {
    function verify(
        bytes32[] memory proof,
        bytes32 root,
        bytes32 leaf,
        uint index
    ) public pure returns (bool) {
        bytes32 hash = leaf;

        for (uint i = 0; i < proof.length; i++) {
            bytes32 proofElement = proof[i];
```

```
            if (index % 2 == 0) {
                hash = keccak256(abi.encodePacked(hash, proofElement));
            } else {
                hash = keccak256(abi.encodePacked(proofElement, hash));
            }

            index = index / 2;
        }

        return hash == root;
    }
}

contract TestMerkleProof is MerkleProof {
    bytes32[] public hashes;

    constructor() {
        string[4] memory transactions = [
            "alice -> bob",
            "bob -> dave",
            "carol -> alice",
            "dave -> bob"
        ];

        for (uint i = 0; i < transactions.length; i++) {
            hashes.push(keccak256(abi.encodePacked(transactions[i])));
        }

        uint n = transactions.length;
        uint offset = 0;

        while (n > 0) {
            for (uint i = 0; i < n - 1; i += 2) {
                hashes.push(
                    keccak256(
                        abi.encodePacked(hashes[offset + i], hashes[offset + i + 1])
                    )
                );
            }
            offset += n;
            n = n / 2;
        }
    }

    function getRoot() public view returns (bytes32) {
```

```
        return hashes[hashes.length - 1];
    }

    /* verify
    3rd leaf
    0xdca3326ad7e8121bf9cf9c12333e6b2271abe823ec9edfe42f813b1e768fa57b

    root
    0xcc086fcc038189b4641db2cc4f1de3bb132aefbd65d510d8175915509378l8c7

    index
    2

    proof

[0x8da9e1c820f9dbd1589fd6585872bc1063588625729e7ab0797cfc63a00bd950,0x995788ffc10
3b987ad50f5e5707fd094419eb12d9552cc423bd0cd86a3861433]
    */
}
```

Output Screenshot:-



Result:-The implementation of Merkle tree using solidity is implemented successfully.

18CSE431J-Distributed Ledger Technology

Lab-9 To explore different data locations in solidity

Aim: To explore different data locations in solidity.

Procedure:

In Solidity, data can be stored in different locations depending on how they are declared. The most common data locations in Solidity are:

1. Storage: Storage is a persistent data location that is written to the Ethereum blockchain. It is used to store data that needs to be stored permanently, such as contract state variables.
2. Memory: Memory is a temporary data location that is used to store data during the execution of a function. It is used to store data that is not needed outside the function, such as function parameters and local variables.
3. Stack: The stack is a temporary data location used to hold the values of operands during the execution of a function. It is used for intermediate calculations and is automatically managed by the Solidity compiler.

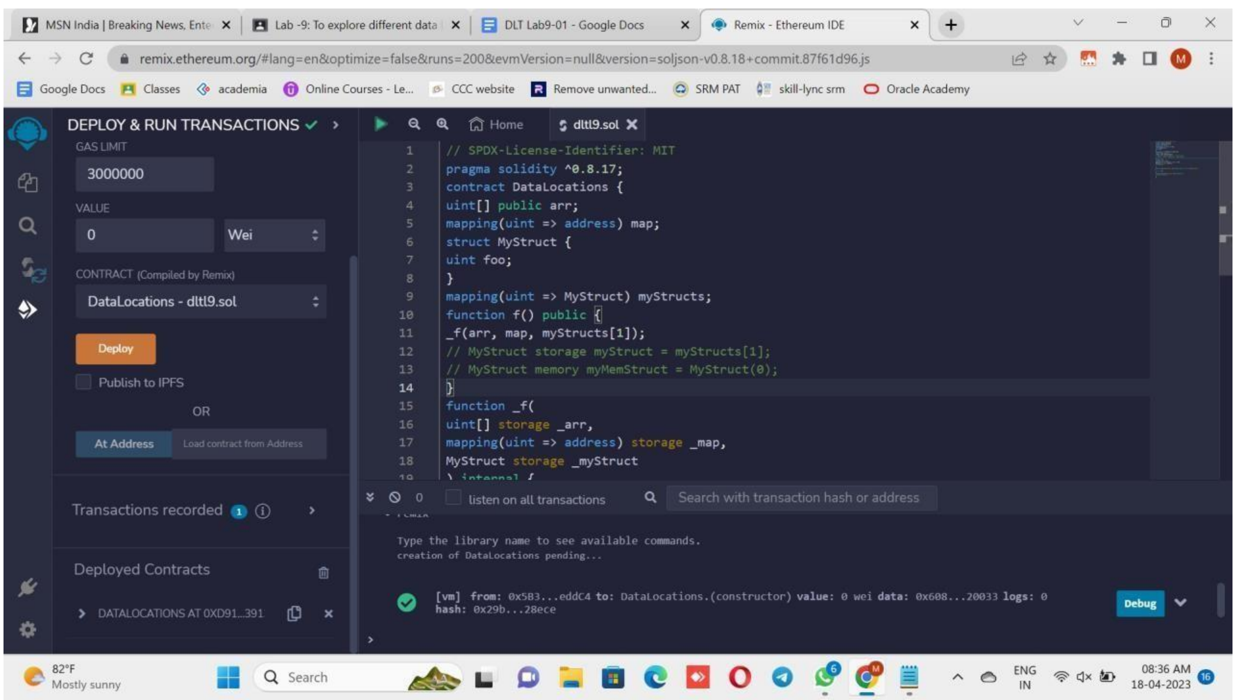
It is important to note that Solidity automatically determines the data location for most types of variables. However, you can explicitly specify the data location using the storage, memory, and calldata keywords.

Code:-

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;
contract  DataLocations {
uint[] public  arr;
mapping(uint => address) map;
struct MyStruct {
uint foo;
}
mapping(uint => MyStruct) myStructs;
function f() public {
_f(arr, map, myStructs[1]);
// MyStruct storage myStruct = myStructs[1];
// MyStruct memory myMemStruct = MyStruct(0);
}
```

```
function _f(
uint[] storage _arr,
mapping(uint => address) storage _map,
MyStruct storage _myStruct
) internal {

}
function g(uint[] memory _arr) public pure returns (uint[] memory)
{
return _arr;
}
function h(uint[] calldata _arr) external {
//calldata function
}
}
```

Output Screenshot:-



Result:-The implementation of different data locations in solidity is executed and verified successfully.

18CSE431J-Distributed Ledger Technology

Lab-10 Implementing Supply Management System using Solidity

Aim: To implement supply management system using solidity.

Procedure:

To implement a supply management system using Solidity, you can follow these general steps:

- Define the data structures needed for the system, such as the products and their properties, the suppliers, and the customers.
- Define the smart contracts that will be used to manage the data and the transactions between the parties involved.
- Implement the logic for creating, updating, and deleting products, suppliers, and customers.
- Define the logic for ordering and receiving products. This should involve verifying that the supplier has enough inventory and that the customer has enough funds to pay for the order.
- Implement the logic for tracking the inventory levels of the products.
- Define the logic for handling payments, including the ability to receive payments from customers and pay suppliers for their products.
- Implement the necessary security measures to ensure that only authorized parties can access the system and perform transactions.

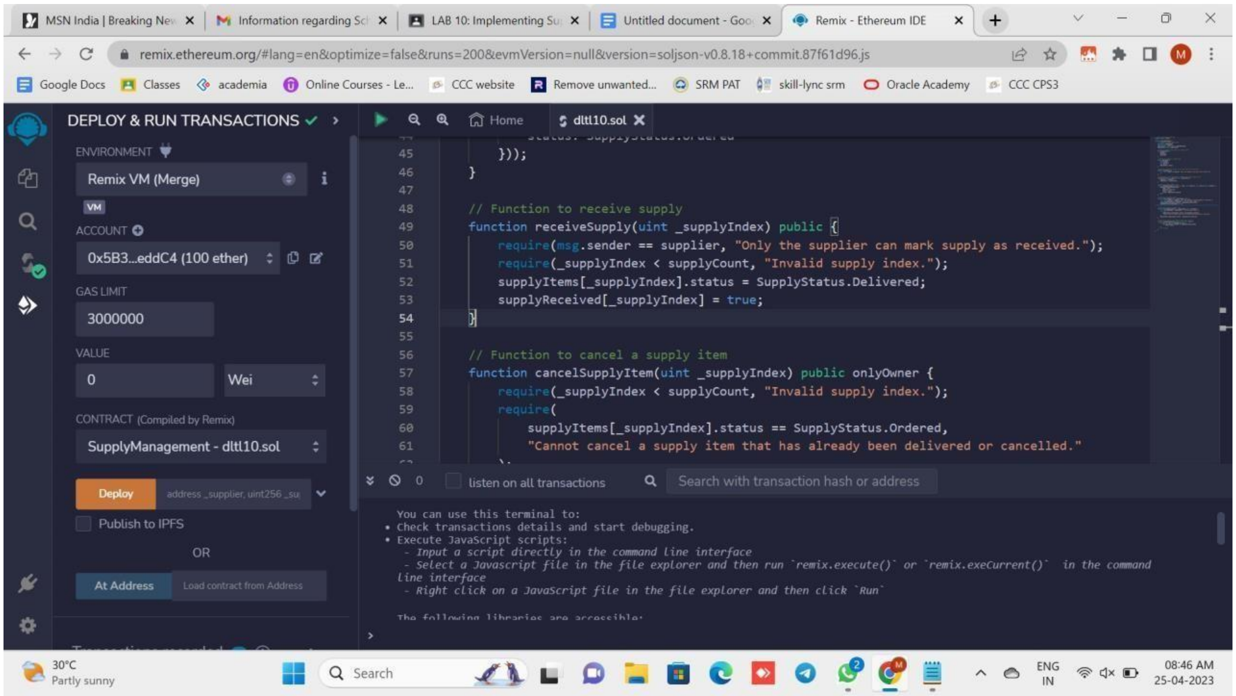
Code:-

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

contract SupplyManagement {
  // Variables to store supply data
  uint public supplyCount;
  address public supplier;
  mapping(uint => bool) public supplyReceived;
```

```
    SupplyItem[] public supplyItems;

    // Enum to represent the status of a supply item
    enum SupplyStatus {
        Ordered,
        Delivered,
        Cancelled
    }

    // Struct to represent a supply item
    struct SupplyItem {
        string name;
        uint quantity;
        uint price;
        SupplyStatus status;
    }

    // Modifier to restrict access to only the owner of the contract
    modifier onlyOwner() {
        require(msg.sender == supplier, "Only the supplier can access this function.");
        _;
    }

    // Constructor to set the supplier address and initial supply count
    constructor(address _supplier, uint _supplyCount) {
        supplier = _supplier;
        supplyCount = _supplyCount;
    }

    // Function to add a supply item
    function addSupplyItem(string memory _name, uint _quantity, uint _price) public onlyOwner
{
        supplyItems.push(SupplyItem({
            name: _name,
            quantity: _quantity,
            price: _price,
            status: SupplyStatus.Ordered
        }));
    }
```

```
    // Function to receive supply
    function receiveSupply(uint _supplyIndex) public {
        require(msg.sender == supplier, "Only the supplier can mark supply as received.");
        require(_supplyIndex < supplyCount, "Invalid supply index.");
        supplyItems[_supplyIndex].status = SupplyStatus.Delivered;
        supplyReceived[_supplyIndex] = true;
    }

    // Function to cancel a supply item
    function cancelSupplyItem(uint _supplyIndex) public onlyOwner {
        require(_supplyIndex < supplyCount, "Invalid supply index.");
        require(
            supplyItems[_supplyIndex].status == SupplyStatus.Ordered,
            "Cannot cancel a supply item that has already been delivered or cancelled."
        );
        supplyItems[_supplyIndex].status = SupplyStatus.Cancelled;
    }

    // Function to check if all supplies have been received
    function isSupplyComplete() public view returns (bool) {
        for (uint i = 0; i < supplyCount; i++) {
            if (supplyItems[i].status != SupplyStatus.Delivered)
                return false;
        }
        return true;
    }
}
```

Output:-



Result:- Supply management system using solidity is implemented and executed successfully.