Answer	Coding Efficiency	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
5	5	5	5	20	

Expected Date of Completion:..... Actual Date of Completion:.....

Group C

Assignment No: 3

Title of the Assignment: Write a smart contract on a test network, for Bank account of a customer for following operations: Deposit money Withdraw Money Show balance

Objective of the Assignment: Students should be able to learn about smart contract for banking application and able to perform some operation like Deposit money Withdraw Money Show balance

Prerequisite:

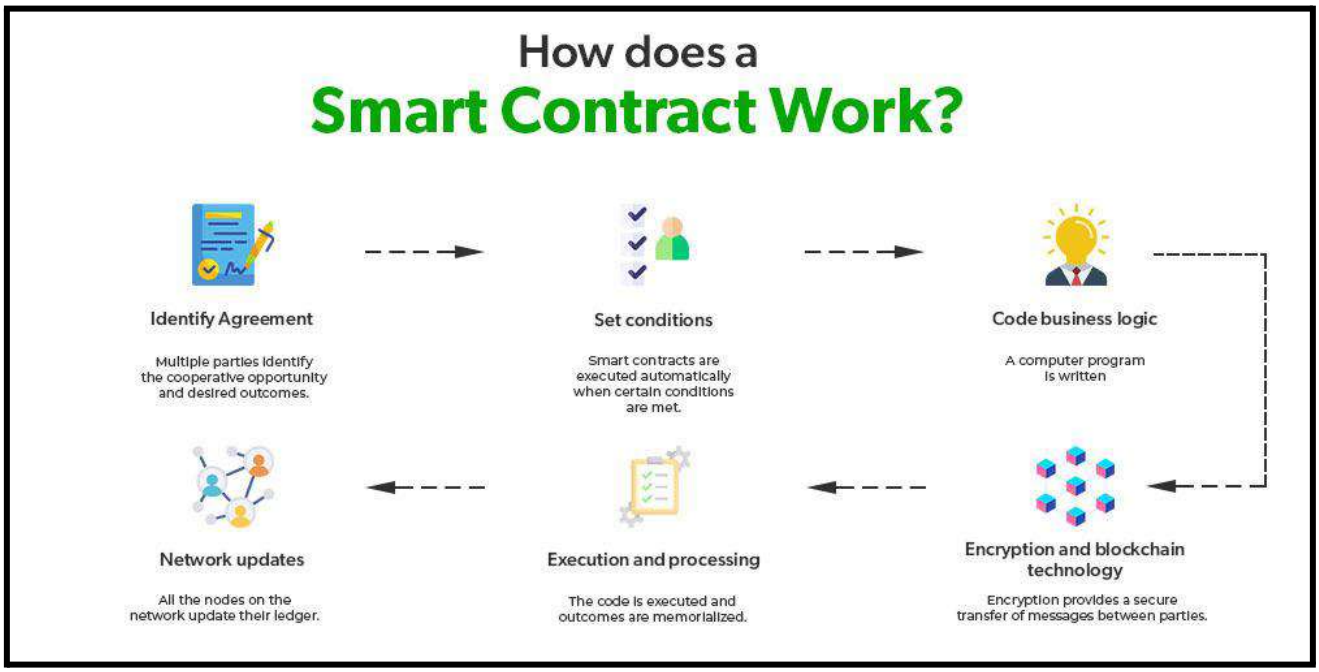
1. Basic knowledge of smart contract
2. Remix IDE
3. Basic knowledge of solidity

Contents for Theory:

1. **Smart contract**
2. **Remix IDE**
3. **Solidity Basics**
4. **Ether transaction**

Introduction to Smart Contract

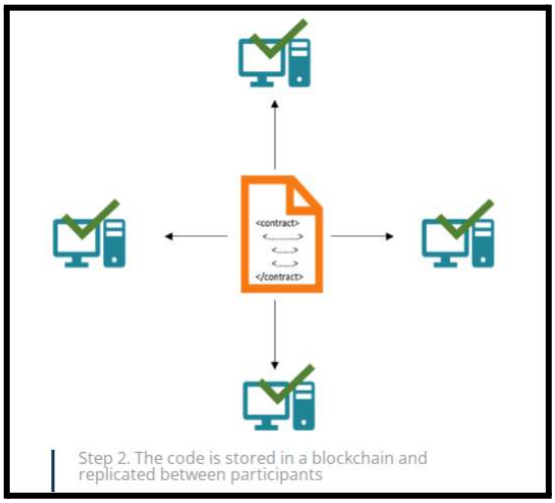
- Smart contracts are self-executing lines of code with the terms of an agreement between buyer and seller automatically verified and executed via a computer network.
- Nick Szabo, an American computer scientist who invented a virtual currency called "Bit Gold" in 1998,1 defined smart contracts as computerized transaction protocols that execute terms of a contract.2
- Smart contracts deployed to blockchains render transactions traceable, transparent, and irreversible.



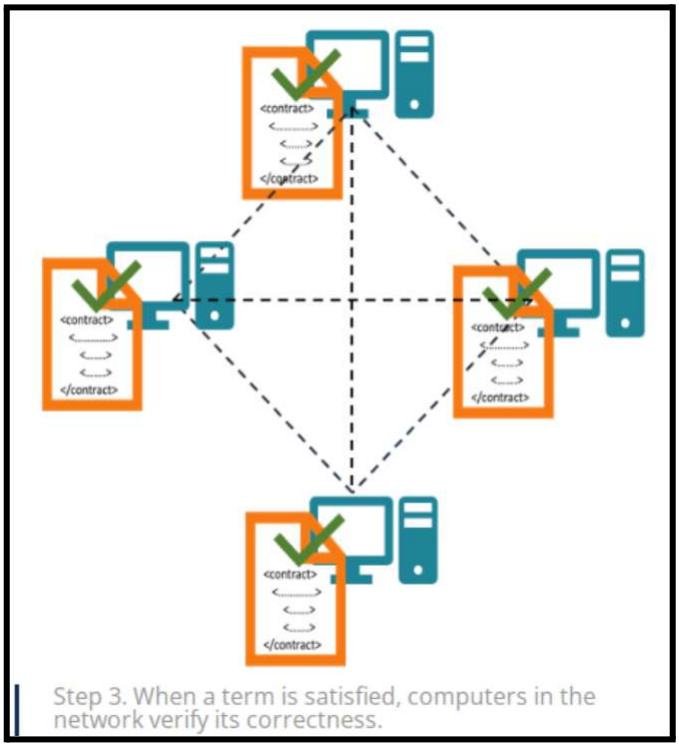
How does Smart Contract work?

- First, the contractual parties should determine the terms of the contract. After the contractual terms are finalized, they are translated into programming code. Basically, the code represents a number of different conditional statements that describe the possible scenarios of a future transaction.

- When the code is created, it is stored in the blockchain network and is replicated among the participants in the blockchain.



- Then, the code is run and executed by all computers in the network. If a term of the contract is satisfied and it is verified by all participants of the blockchain network, then the relevant transaction is executed.

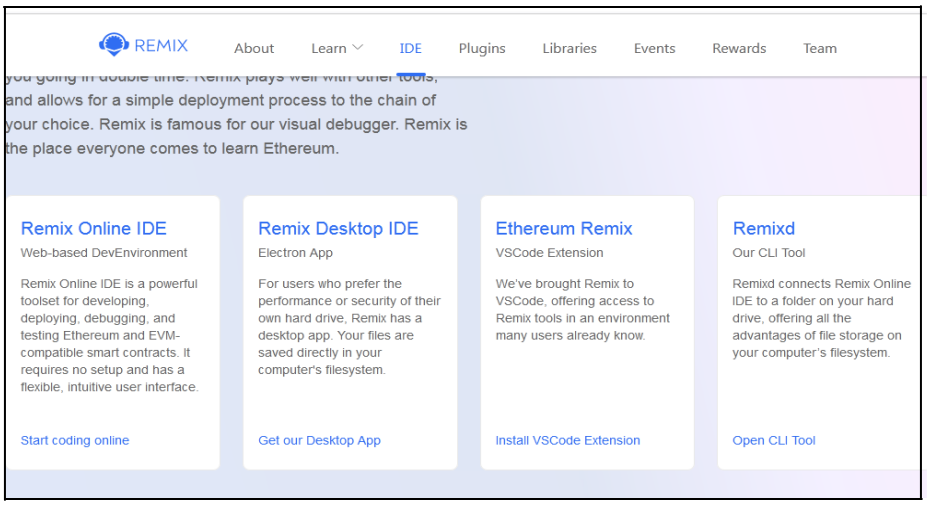


Remix IDE : Installation steps

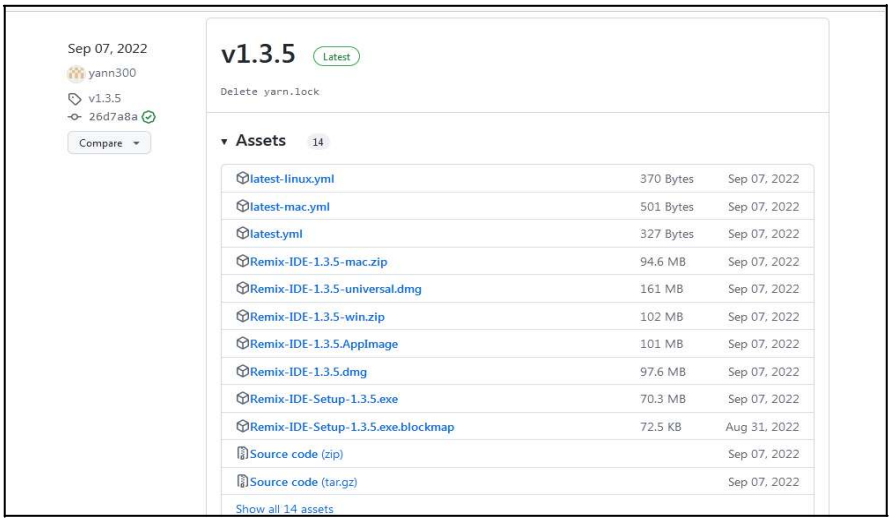
- First and foremost, head on over to the release page of the official Remix Desktop repository and grab the binary suited for your host system. This will be the .exe file for Windows users,

the .dmg for macOS and the .deb for Debian-derivative GNU/Linux systems. For Ubuntu and other AppImage setups, the .AppImage will be what you are looking for.

- Go to https://remix-project.org/
- Go to IDE option .Select Desktop IDE



- Select The file As per your choice



- Install Executable file,by double clicking on file(Applicable to .exe file Windows OS)

What is Solidity?

- Solidity is a rather simple language deliberately created for a simplistic approach to tackle real world solutions. Gavin Wood initially proposed it in August of 2014. Several developers of the Ethereum chain such as Christian Reitwiessner, Alex Beregszaszi, Liana Husikyan,

Yoichi Hirai and many more contributed to creating the language. The Solidity language can be executed on the Ethereum platform, that is a primary Virtual Machine implementing the blockchain network to develop decentralized public ledgers to create smart contract systems.

Solidity Basics

- To get started with the language and learn the basics let's dive into coding. We will begin by understanding the syntax and general data types, along with the variable data types. Solidity supports the generic value types, namely:
- Booleans: Returns value as either true or false. The logical operators returning Boolean data types are as follows:
- ! Logical negation
- && logical conjunction, "and"
- || logical disjunction, "or"
- == equality
- != inequality

Integers: Solidity supports int/unit for both signed and unsigned integers respectively. These storage allocations can be of various sizes. Keywords such as uint8 and uint256 can be used to allocate a storage size of 8 bits to 256 bits respectively. By default, the allocation is 256 bits. That is, uint and int can be used in place of uint256 and int256. The operators compatible with integer data types are:

- Comparisons: <=, <, ==, !=, >=, >. These are used to evaluate to bool.
- Bit operators: &, |, ^ bitwise exclusive 'or', ~ bitwise negation, "not".
- Arithmetic operators: +, -, unary -, unary +, *, /, % remainder, ** exponentiation, << left shift, >> right shift.

The EVM returns a Runtime Exception when the modulus operator is applied to the zero of a "divide by zero" operation.

Address: An address can hold a 20 byte value that is equivalent to the size of an Ethereum address. These address types are backed up with members that serve as the contract base.

String Literals: String literals can be represented using either single or double quotes (for example, "foo" or 'bar'). Unlike in the C language, string literals in Solidity do imply trailing value zeroes. For instance, “bar” will represent a three byte element instead of four. Similarly, in the case of integer literals, the literals are convertible inherently using the corresponding fit, that is, byte or string.

Modifier: In a smart contract, modifiers are used to ensure the coherence of the conditions defined before executing the code.

Solidity provides basic arrays, enums, operators, and hash values to create a data structure known as **“mappings**.” These mappings are used to return values associated with a given storage location. An Array is a contiguous memory allocation of a size defined by the programmer where if the size is initialized as K, and the type of element is instantiated as T, the array can be written as T[k].

Arrays can also be dynamically instantiated using the notation uint[][6]. Here the notation initializes a dynamic array with six contiguous memory allocations. Similarly, a two dimensional array can be initialized as arr[2][4], where the two indices point towards the dimensions of the matrix.

We will begin our programming venture with a simple structure of a contract. Consider the following code:

```
pragma solidity^0.4.0;
contract StorageBasic {
  uint storedValue;
  function set(uint var) {
     storedValue= var;
  }
  function get() constant returns (uint) {
   return storedValue;
  }
}
```

- The word “**Pragma**” refers to the instructions given to a compiler to sequentially execute the source code.
- Solidity is statically typed language. Therefore, each variable type irrespective of their scope can be instantiated at compile time. These elementary types can be further combined to create

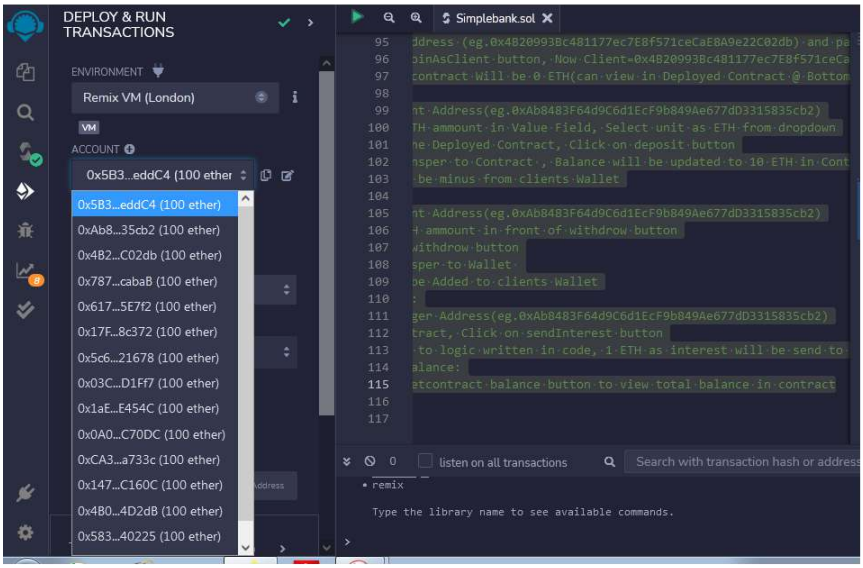
complex data types. These complex data types then synchronize with each other according to their respective preferences.

Steps to Run Banking Application

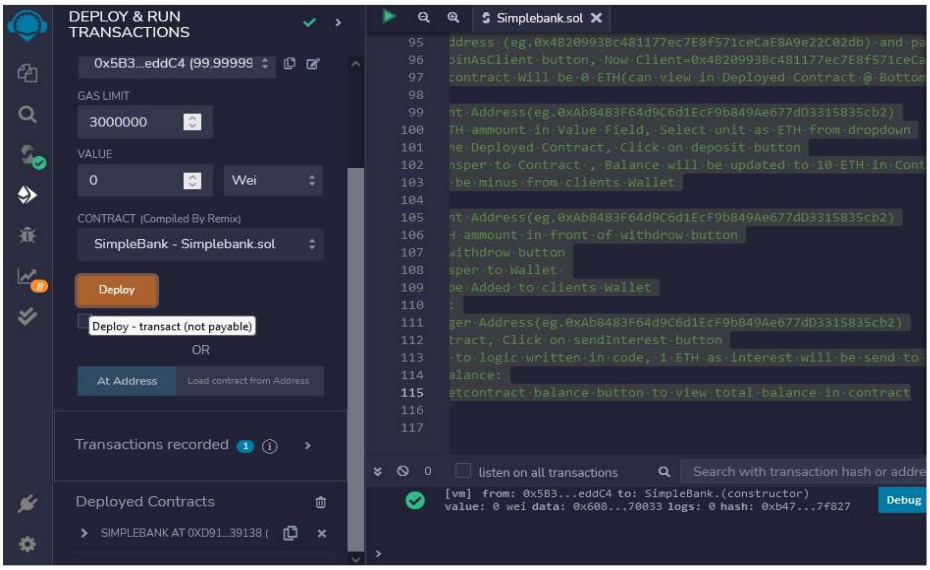
1.//Output steps

//Initially All Account has 100 fake ether

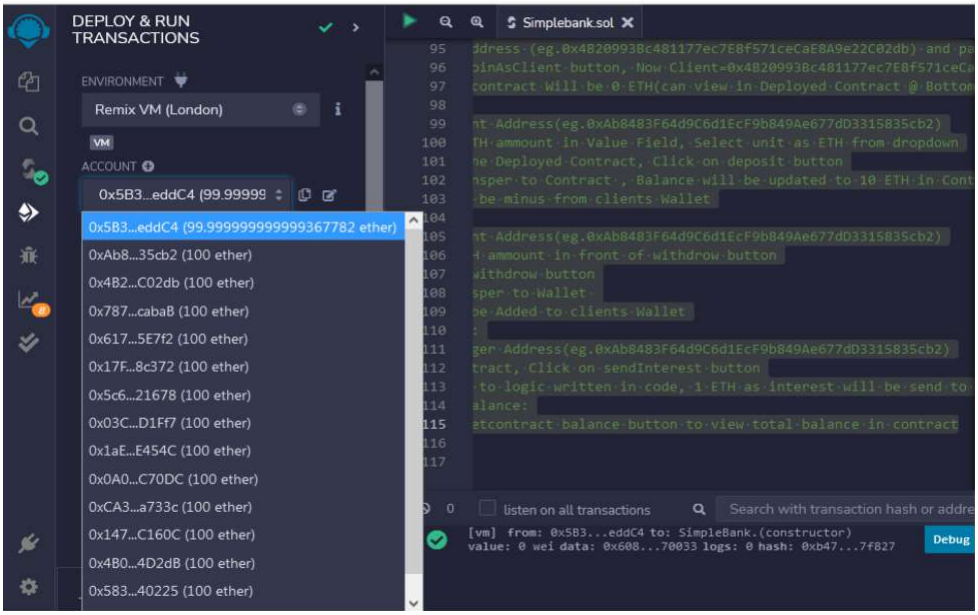
//Step 1: Select first Address (eg.0x5B38Da6a701c568545dCfcB03FcB875f56beddC4)



//Step 2: Click on Deploy button(Contract Created,Can view under Deployed Contract)



//After deploying contract 100 ETH turns to 99.99999.... ETH

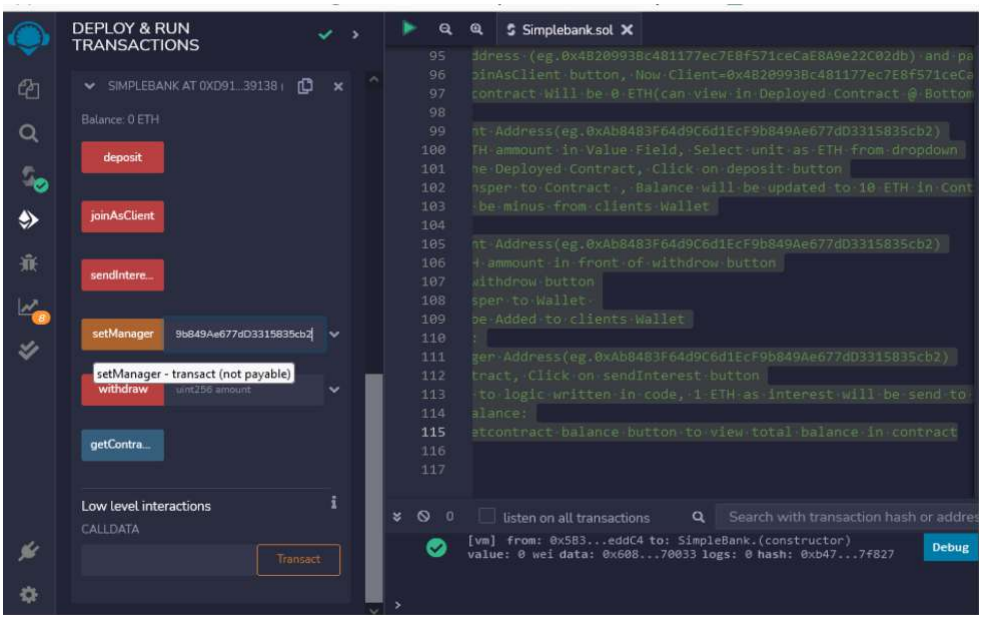


//Step 3: Set Manager: Follow Following instructions

// i.Select Onother Address(eg.0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2)

// ii.Copy this address (eg.0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2) and paste it in contract, infront of set Manager button

// iii. click on set manager button, Now Manager=0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2



//Step 4: join as Client: Follow Following instructions

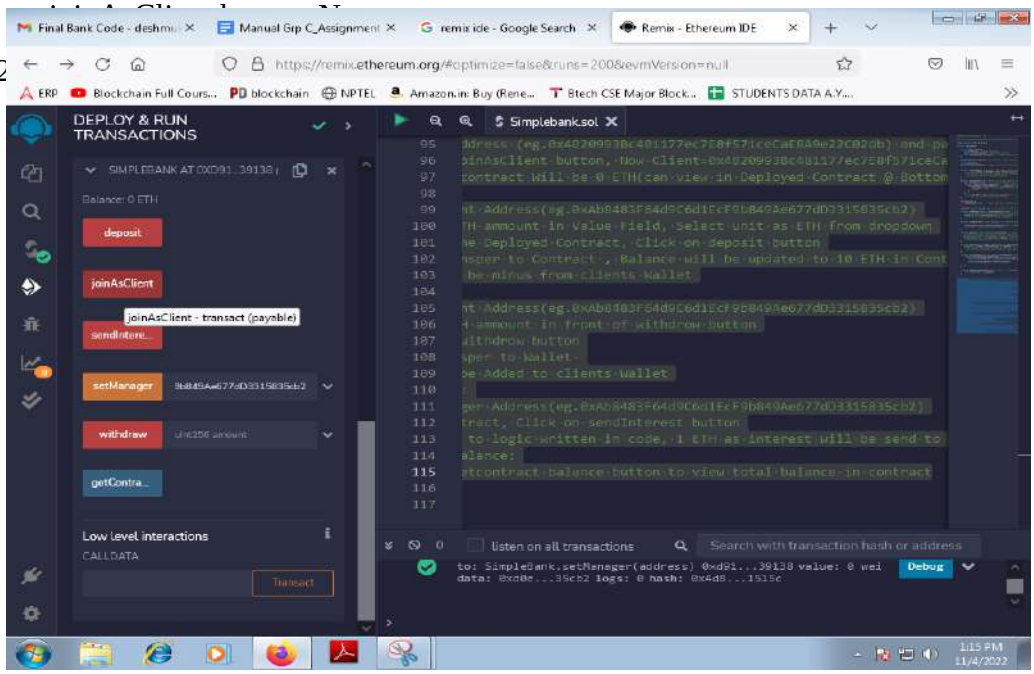
// i.Select Onother Address(eg.0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db)

// ii.Copy this address (eg.0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db) and paste it in

contract, infront of joinAsClient button

// iii.click

Client=0x4B2



//Initially Balance in contract Will be 0 ETH(can view in Deployed Contract @ Bottom)

//Step 5: Deposit:

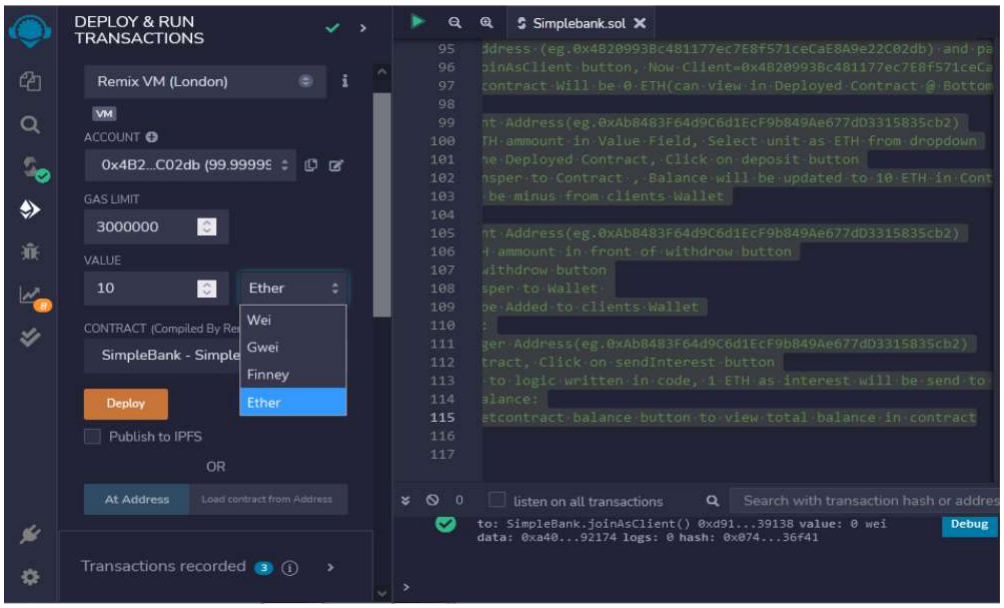
// i.Select Client Address(eg.0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2)

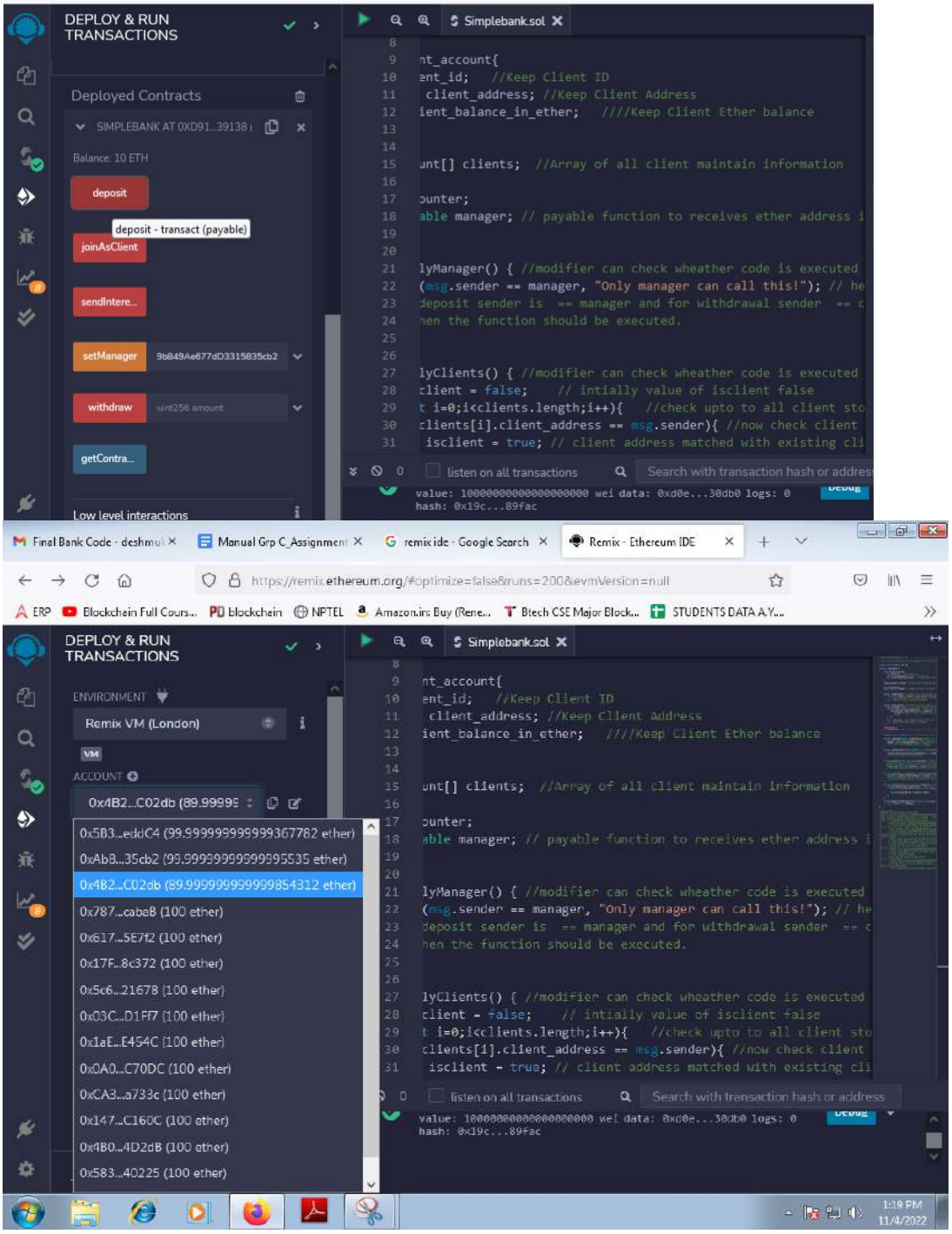
// ii. Enter 10 ETH ammount in Value Field, Select unit as ETH from dropdown

// iii. Come to the Deployed Contract, Click on deposit button

// iv. 10 ETH transper to Contract , Balance will be updated to 10 ETH in Contract

// V. 10 ETH will be minus from clients Wallet





//Step 6: Withdraw:

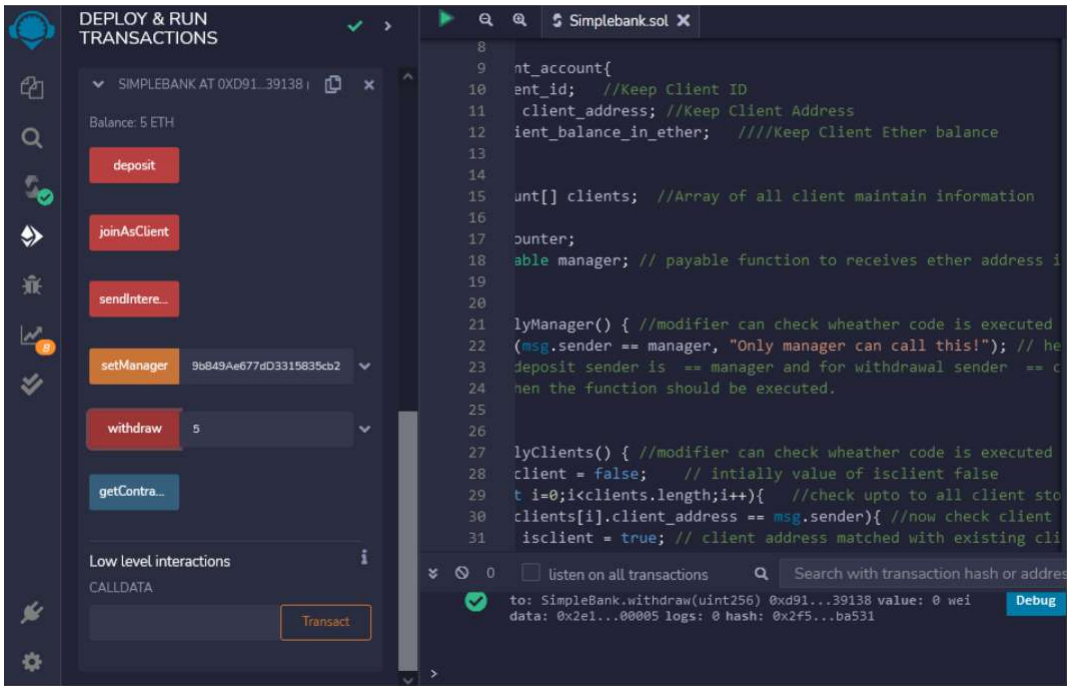
// i.Select Client Address(eg.0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2)

// ii. Enter 5 ETH ammount in front of withdrow button

// iii. Click on withdrow button

// iv. 5 ETH transper to Wallet

// V. 5 ETH will be Added to clients Wallet

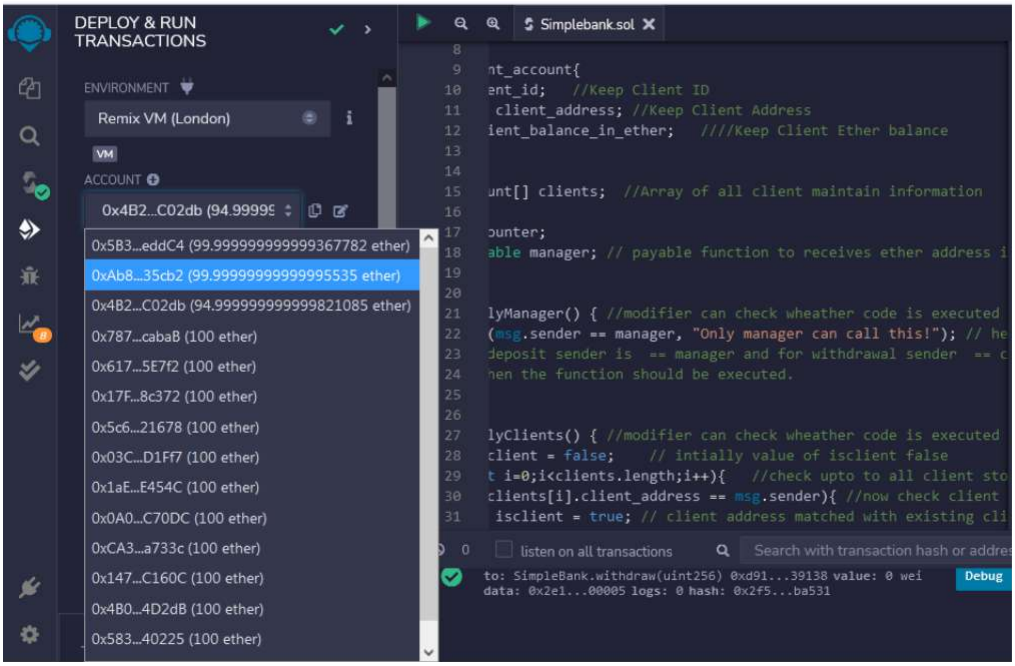


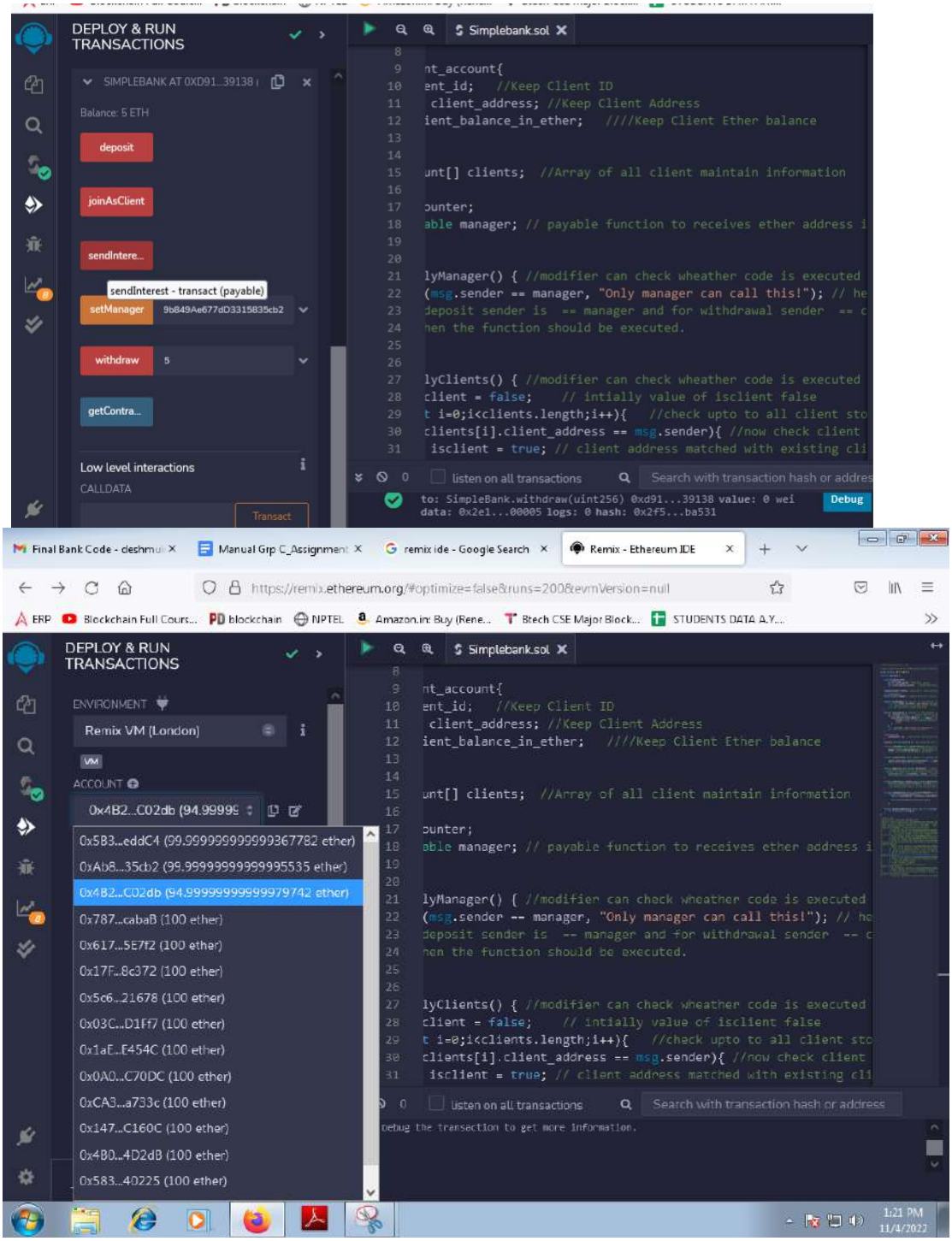
//Step 7: Send Interest:

// i.Select Manager Address(eg.0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2)

// ii.Come to contract, Click on sendInterest button

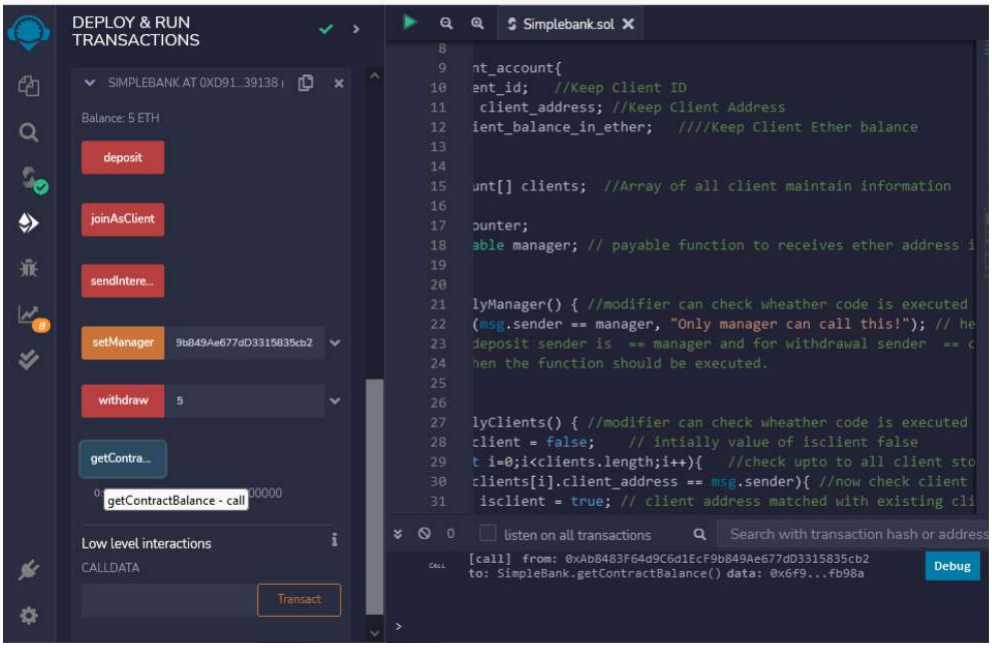
// iii. According to logic written in code, 1 ETH as interest will be send to Client Wallet





//Step 8: getContract Balance:

// i.Click on get contract balance button to view total balance in contract



Assignment Question

1. **What is Solidity?**
2. **What are the main differences between Solidity and other programming languages like Python, Java, or C++?**
3. **What is EVM bytecode?**
4. **What are the differences between Ethereum and blockchain and bitcoin?**

Reference link

- https://www.simplilearn.com/solidity-interview-questions-article

Code :

//SPDX-License-Identifier: MIT

```
//https://betterprogramming.pub/developing-a-smart-contract-by-using-remix-ide-81ff6f44ba2f

pragma solidity >=0.7.0 <0.9.0;

contract SimpleBank {

    struct client_account{
        int client_id;   //Keep Client ID
        address client_address; //Keep Client Address
        uint client_balance_in_ether;   ////Keep Client Ether balance
    }

    client_account[] clients;  //Array of all client maintain information

    int clientCounter;
    address payable manager; // payable function to receives ether address is datatype it is 20 byte hash address public key


    modifier onlyManager() { //modifier can check wheather code is executed accouding to condition for manager side
        require(msg.sender == manager, "Only manager can call this!"); // here sender is manager in this case
        // for deposit sender is  == manager and for withdrawal sender == client
        _; // when the function should be executed.
    }

    modifier onlyClients() { //modifier can check wheather code is executed accouding to condition for client side
        bool isclient = false;    // intially value of isclient false
        for(uint i=0;i<clients.length;i++){   //check upto to all client store in array
            if(clients[i].client_address == msg.sender){ //now check client address matched with sender only that client intiate transaction
                isclient = true; // client address matched with existing client address in bank database isclient value updated true.
                break;
            }
```

```
        }
        require(isclient, "Only clients can call this!"); // isclient
true here so allowed call the transaction.
        _; // when the function should be executed.
    }

    constructor() {
        clientCounter = 0;  // those client join contract assign there
ID intially it set 0
    }

    receive() external payable { } // this allows the smart contract to
receive ether

    function setManager(address managerAddress) public returns(string
memory){ //setManager method will be used to set the manager address to
variables
        // string memory store address of manager account instead of
store data
        manager = payable(managerAddress);// managerAddress is consumed
as a parameter and cast as payable to provide sending ether.
        return ""; // return payable address of manager
    }

    function joinAsClient() public payable returns(string memory){
//joinAsClient method will be used to make sure the client joins the
contract.

        clients.push(client_account(clientCounter++, msg.sender,
address(msg.sender).balance)); // push() array method to add items into
a storage array.
        return ""; // return all client details
    }

    function deposit() public payable onlyClients{  // deposit ==
client to contract by onlyclient
        //deposit method will be used to send ETH from the client
account to the contract.
    // We want this method to be callable only by clients who've joined
the contract, so the onlyClient modifier is used for this restriction.
        payable(address(this)).transfer(msg.value); //transfer methods
belongs to the contract, and it's dedicated to sending an indicated
amount of ETH between addresses.
```

```
        // The payable keyword makes receipt of the ETH transfer
possible so the amount of ETH indicated in the msg.value will be
transferred to the contract address.
    }

    function withdraw(uint amount) public payable onlyClients{ //
withdraw == contract to client by onlyclient
        payable(msg.sender).transfer(amount * 1 ether); // The address
of the sender( ie contract ) is held in the msg.sender variable.
        //The withdraw method will be used to send ETH from the
contract to the client account. It sends the unit of ETH indicated in
the amount parameter, from the contract to the client who sent the
transaction. We want this method to be callable only by clients who've
joined the contract either,
        // so the onlyClient modifier is used for this restriction.
    }

    function sendInterest() public payable onlyManager{ //The
sendInterest method will be used to send ETH as interest from the
contract to all clients. can called by only manager
        for(uint i=0;i<clients.length;i++){ // check client in database
            address initialAddress = clients[i].client_address; //
check client address

            payable(initialAddress).transfer(1 ether);

        }
    }

    function getContractBalance() public view returns(uint){
//getContractBalance method will be used to get the balance of the
contract we deployed.
        return address(this).balance;
    }
}

//Output steps
//Initially All Acount has 100 fake ether
//Step 1: Select first Address
(eg.0x5B38Da6a701c568545dCfcB03FcB875f56beddC4)
//Step 2: Click on Deploy button(Contract Created,Can view under
Deployed Contract)
//After deploying contract 100 ETH turns to 99.99999.... ETH
```

```
//Step 3: Set Manager: Follow Following instructions
//        i.Select Onother
Address(eg.0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2)
//       ii.Copy this address
(eg.0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2) and paste it in
contract, infront of set Manager button
//       iii. click on set manager button, Now
Manager=0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2
//Step 4: join as Client: Follow Following instructions
//        i.Select Onother
Address(eg.0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db)
//       ii.Copy this address
(eg.0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db) and paste it in
contract, infront of joinAsClient button
//       iii.click on joinAsClient button, Now
Client=0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db
//Initially Balanve in contract Will be 0 ETH(can view in Deployed
Contract @ Bottom)
//Step 5: Deposit:
//        i.Select Client
Address(eg.0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2)
//       ii. Enter 10 ETH ammount in Value Field, Select unit as ETH
from dropdown
//       iii. Come to the Deployed Contract, Click on deposit button
//       iv. 10 ETH transper to Contract , Balance will be updated to
10 ETH in Contract
//       V. 10 ETH will be minus from clients Wallet
//Step 6: Withdraw:
//        i.Select Client
Address(eg.0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2)
//       ii. Enter 5 ETH ammount in front of withdrow button
//       iii. Click on withdrow button
//       iv. 5 ETH transper to Wallet
//       V. 5 ETH will be Added to clients Wallet
//Step 7: Send Interest:
//        i.Select Manager
Address(eg.0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2)
//       ii.Come to contract, Click on sendInterest button
//       iii. According to logic written in code, 1 ETH as interest
will be send to Client Wallet
//Step 8: getContract Balance:
//         i.Click on getcontract balance button to view total balance
in contract
```