PRACTICAL NO. 4

Ethereum

Aim: To develop and deploy Dapp in Ethereum.

Ethereum: [1]

- Ethereum is a decentralized, open-source blockchain with smart contract functionality.
- Ether (ETH or Ξ) is the native cryptocurrency of the platform.
- Amongst cryptocurrencies, Ether is second only to Bitcoin in market capitalization.

Ethereum Virtual Machine (EVM): [1]

- The Ethereum Virtual Machine (EVM) is the runtime environment for transaction execution in Ethereum.

Ether: [1]

- Ether (ETH): Ether (ETH) is the cryptocurrency generated by the Ethereum protocol as a reward to miners in a proof-of-work system for adding blocks to the blockchain.
- It is the only currency accepted in the payment of transaction fees, which also go to miners.

Ether: [1]

- The block reward together with the transaction fees provide the incentive to miners to keep the blockchain growing (i.e. to keep processing new transactions).
- Therefore, ETH is fundamental to the operation of the network. Each Ethereum account has an ETH balance and may send ETH to any other account. The smallest subunit of ETH is known as a Wei and is equal to 10^{-18} ETH.

Accounts: [1]

- There are two types of accounts on Ethereum: user accounts (also known as externally-owned accounts) and contracts.
- Both types have an ETH balance, may send ETH to any account, may call any public function of a contract or create a new contract, and are identified on the blockchain and in the state by their address.

Gas: What is Gas in Ethereum? :

- Gas is a unit of account within the EVM used in the calculation of a transaction fee, which is the amount of ETH a transaction's sender must pay to the miner who includes the transaction in the blockchain. [1]
- The transaction fee is calculated in Gas, and paid for in Ether.
- The gas is the "fuel" of the Ethereum network, which is used to:
  - Conduct transactions
  - Execute smart contracts
  - Launch Dapps.
- Gas indicates the fee for a particular action or transaction.
- **Gas Limit:** is the maximum amount of Gas that a user is willing to pay for performing this action or confirming a transaction.
- **Gas Price:** is the amount that the user is willing to spend on each unit of Gas.

MetaMask: [2] 

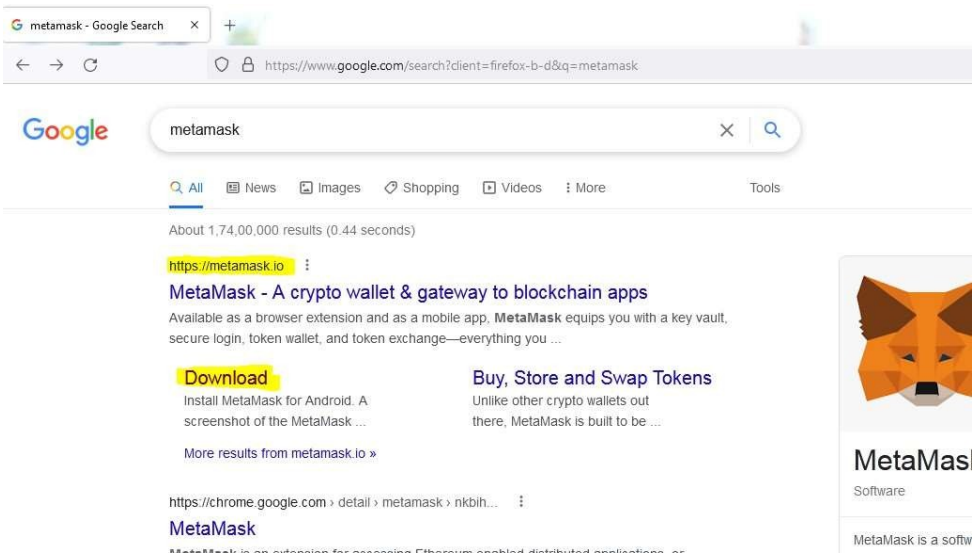
- MetaMask is a software cryptocurrency wallet used to interact with the Ethereum blockchain.
- It allows users to access their Ethereum wallet through a browser extension or mobile app, which can then be used to interact with decentralized applications(Dapps).
- MetaMask is developed by ConsenSys Software Inc., a blockchain software company focusing on Ethereum-based tools and infrastructure.
- MetaMask allows users to store and manage account keys, broadcast transactions, send and receive Ethereum-based cryptocurrencies and tokens, and securely connect to decentralized applications through a compatible web browser or the mobile app's built-in browser.
- The application includes an integrated service for exchanging Ethereum tokens by aggregating several decentralized exchanges (DEXs) to find the best exchange rate. This feature, branded as MetaMask Swaps, charges a service fee of 0.875% of the transaction amount.
- As of April 2021, MetaMask's browser extension had approximately 10 million monthly active users, according to The Financial Times.

Setup a MetaMask Ethereum Wallet:

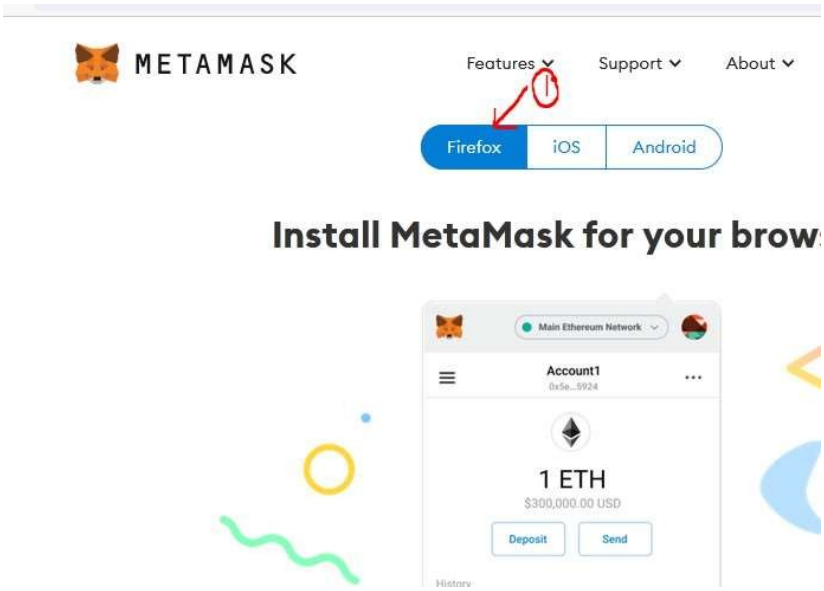
- MetaMask is just an Ethereum Browser and Ether wallet.
- It interacts with Ethereum Dapps and Smart Contracts without running a full Ethereum node.
- MetaMask add-on can be installed on Chrome, Firefox, Opera, and the new Brave browser.
- URL: https://metamask.io/

MetaMask Installation:

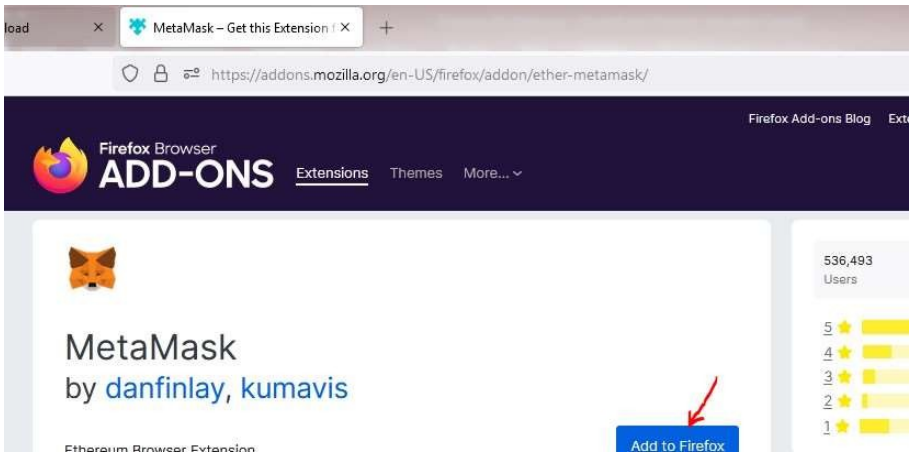
- Install MetaMask
- Add MetaMask extension to the Browser
- MetaMask will show up 12 words recovery key (Seed).
- These 12 words are the only way to restore MetaMask accounts.
- **Use URL: https://metamask.io/ or type metamask in browser. Click on download.**



- **Select option firefox (According to browser you have open, it shows you appropriate browser extension) and click on installed metamask for firefox.**

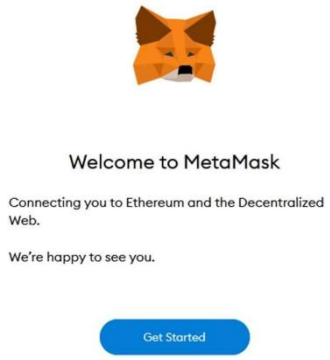


- **Click on add to firefox button.**

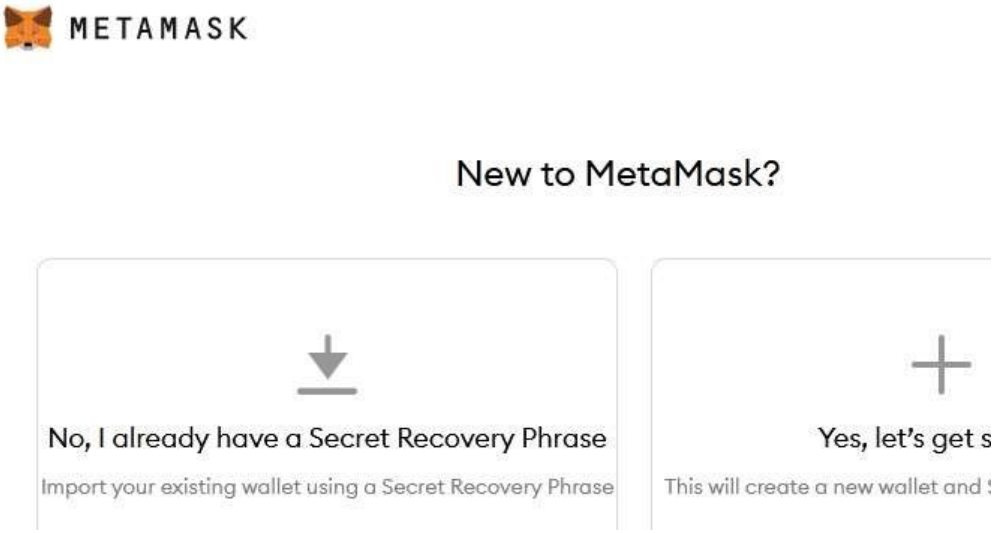


- **Click on logo of metamask.**

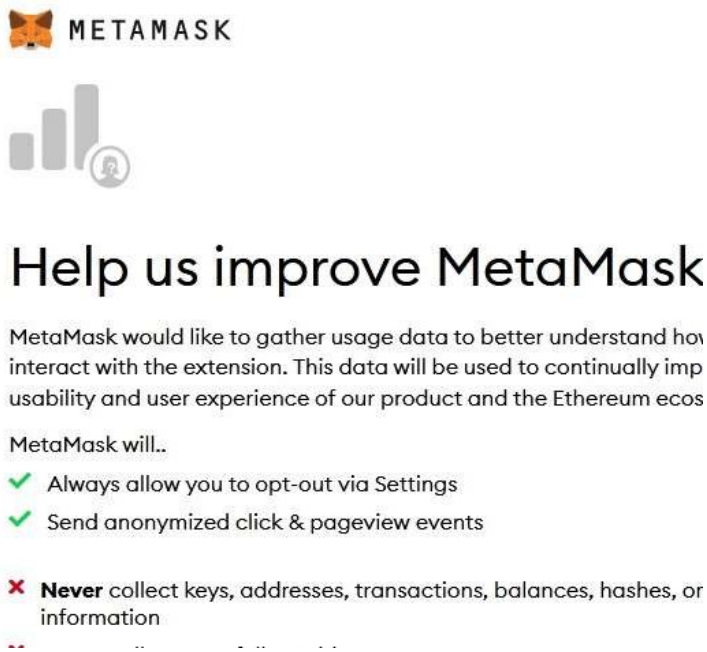
- **Click on Get Started button.**



- **Click on Create a Wallet button to create new wallet.**
- **If you have already created wallet we can import here using Import Wallet option. Here you need to pass 12 words recovery key (Seed) which you receive.**

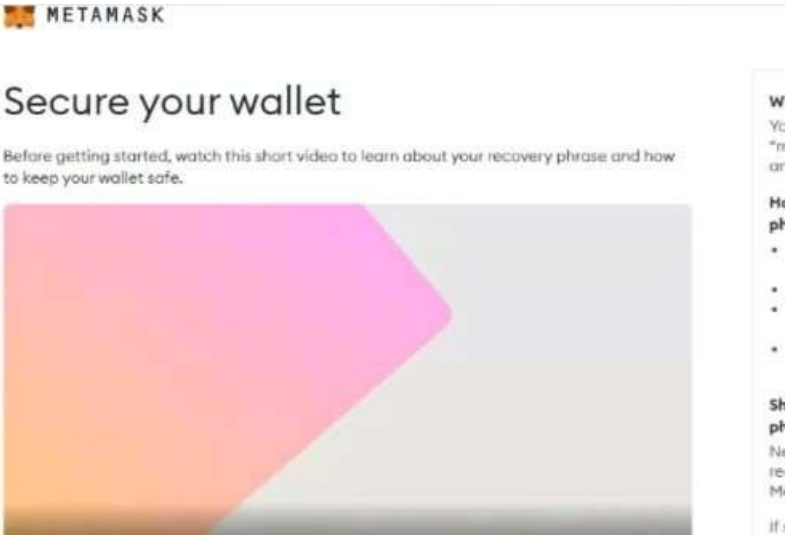


- **Click on I Agree button to accept the terms.**

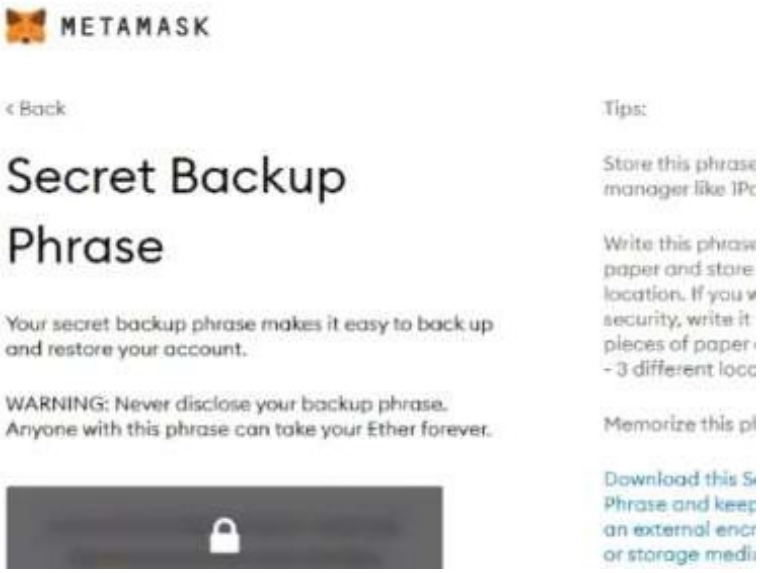


- **Enter new password and confirm password, select check box to accept Terms of Use and then click on create button.**

- **Click on next.**

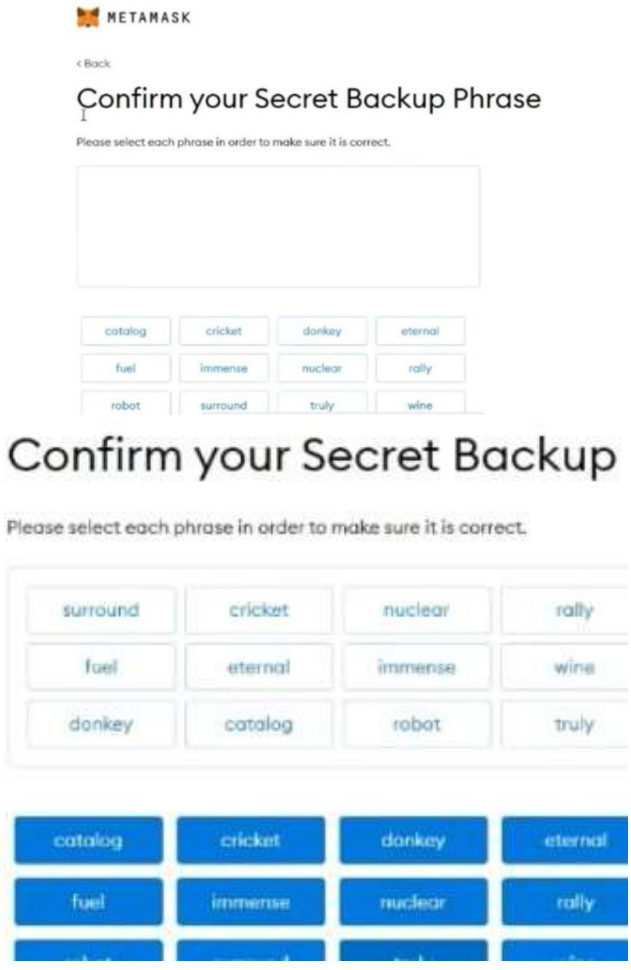


- **Click on Click Here to reveal secret words. This includes 12 secrete words in sequence which can be used to recover the account.**



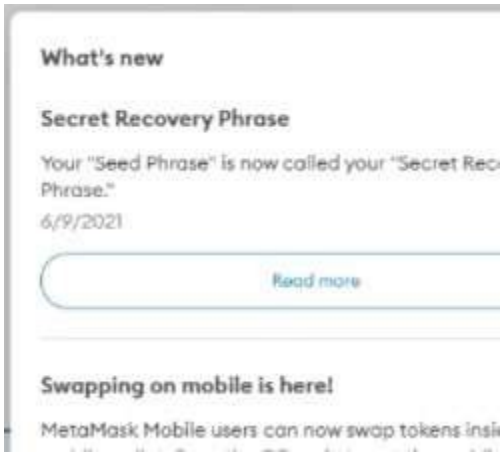
- **Copy these secrete 12 words in same sequence in other files or take screenshot.**
- **Then click on Next button.**

- **Select word of secret phrase in same order which we have seen in earlier step. Then Click on Confirm.**

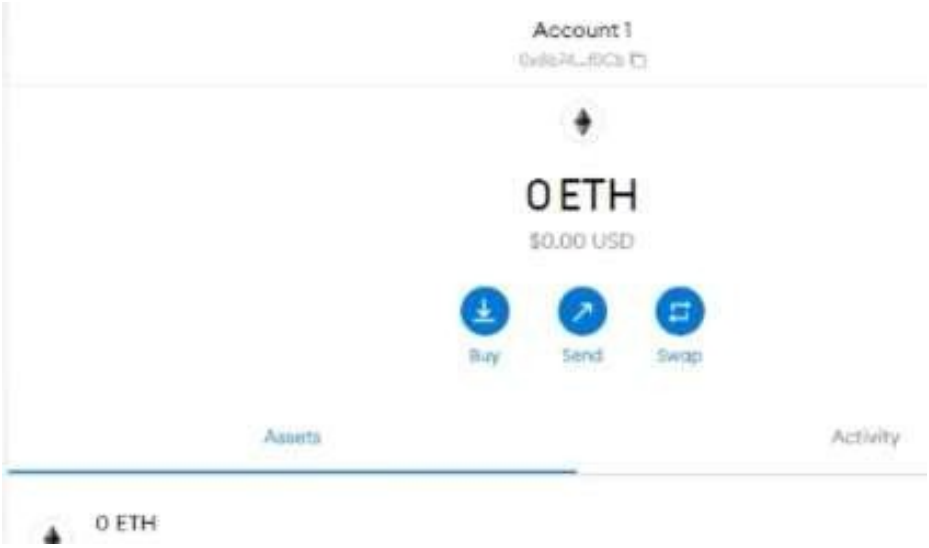


- **After successfully following all steps for new account creation Congratulations message appears. Finally click on All Done.**

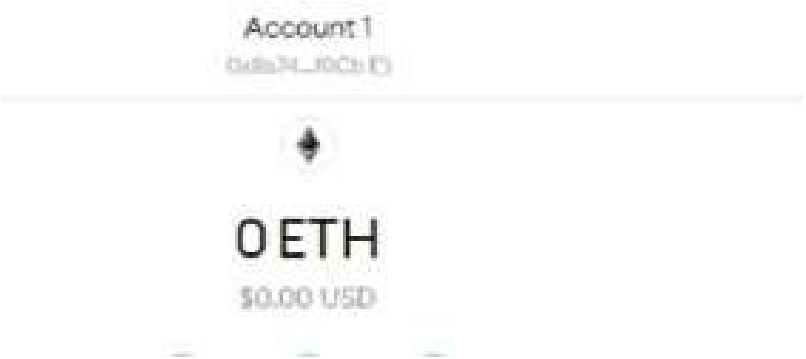
- **A pop up message will appears. Close it.**



- **Finally, it shows you the MetaMask wallet interface.**

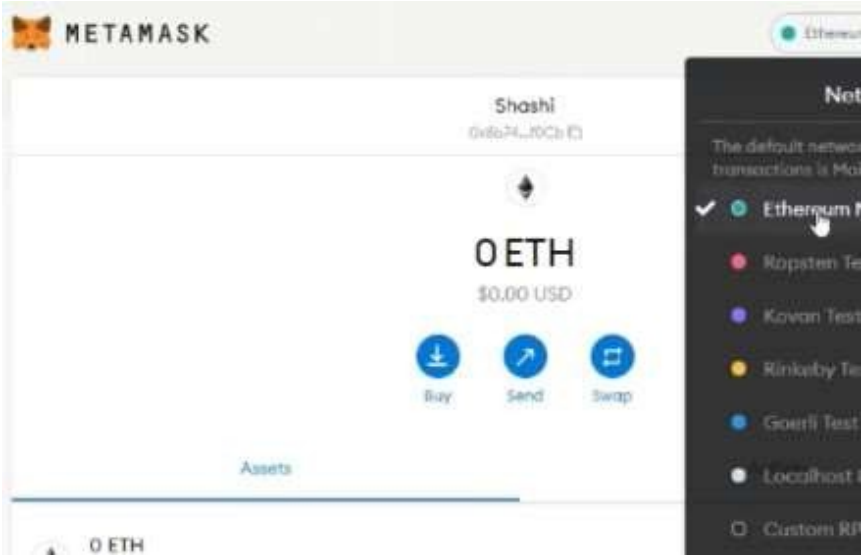


- **Click on three vertical dot i.e. Account option. Then select Account Details.**



- Here you can get option to change account name to any name. And also shows:
  - QR code of account
  - Wallet address i.e. public key
- You can also get private key
- For that click on Export Private Key
- Here it asks to enter password.

- **You can select any networks from the listed networks.**



Perform the following task:

- Connect to any test networks and request for ether using buy option.
- Create another account for e.g. Account 2
- Transfer some ether to another account i.e. Account 2. Here gas fee will be applicable which is transaction fee of transaction.
- Check the transaction details.

What are Smart Contracts?

- Smart contracts are lines of code that are stored on blockchain and automatically execute when predetermined terms and conditions are met.
- It's a computer protocol. It is called smart because of its ability to verify and execute a contract without any help from third parties. The contract exists in the decentralized Blockchain network and contains all the terms of a particular agreement.
- The blockchain is then updated when the smart contract is completed.

Benefits of Smart Contracts:

- Security: Blockchain transactions records are encrypted, and that makes them very hard to hack. Because each individual record is connected to previous and subsequent records on a distributed ledger.
- Savings: Smart contracts remove the need for intermediaries because participants can trust the visible data and the technology to properly execute the transaction.
- Trust: Smart contracts automatically execute transactions following predetermined rules, and the encrypted records of those transactions are shared across participants.
- Speed and accuracy: Smart contracts are digital and automated, so you won't have to spend time processing paperwork and correcting the errors.

How smart contracts works?

1. Two users create smart contract.
2. All contract terms are written as a code.
3. Smart contract is stored in blockchain.
4. Smart contract executes itself when events are triggered.

Task to be performed:

1. Develop the smart contract.
2. Compiling the smart contract.
3. Deploying the smart contract.
4. Interacting with smart contract.
5. Adding more functions to our code to make it more practical.

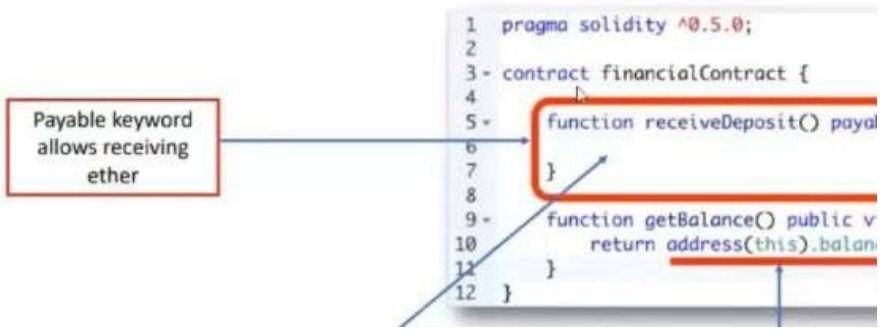
Remix IDE: Deployment Parameters:

1. **JavaScriptVM:** All the transactions will be executed in a sandbox blockchain the browser.
2. **Injected Provider:** Remix will connect to an injectedweb3 provider. Mist and Metamask are example of providers that inject web3, thus they can be used with this option.
3. **Web3Provider:** Remix will connect to a remote node. You will need to provide the URL address to the selected provider: geth, parity or any Ethereum client.

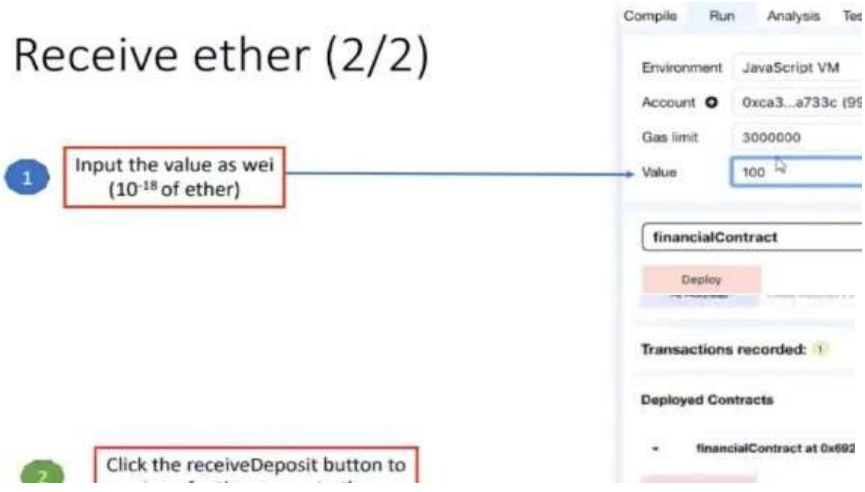
Example of Contract - Financial Contract in JavaScriptVM environment:

```
pragma solidity ^0.5.0;
contract FC
{
    uint balance=10500;
    function getBalance() public view returns (uint)
    {
        return balance;
    }
    function deposit(uint newDeposit) public
    {
        balance = balance + newDeposit;
    }
}
```

Transfer Funds (money) from account to the Contract:



Transfer Funds (money) from account to the Contract:



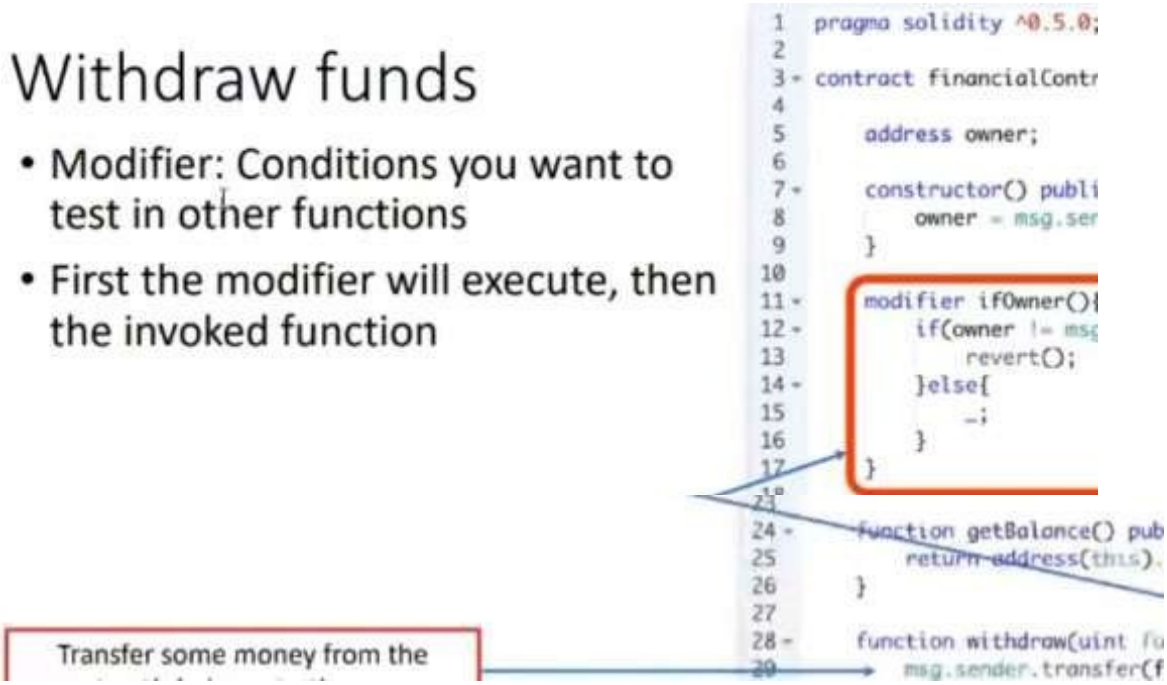
Save the address of the Contract Creator:

Here we can write constructor which will be called at the creation of the instance of the contract.

```
1  pragma solidity ^0.5.0;
2
3  contract financialContract {
4
5      address owner;
6
7      constructor() public{
8          owner = msg.sender;
9      }
10
11     function receiveDeposit() payable public{
12
13     }
```

Withdraw the funds (transferring fund from contract to account back):

Here we use modifier to apply and check condition before executing function.

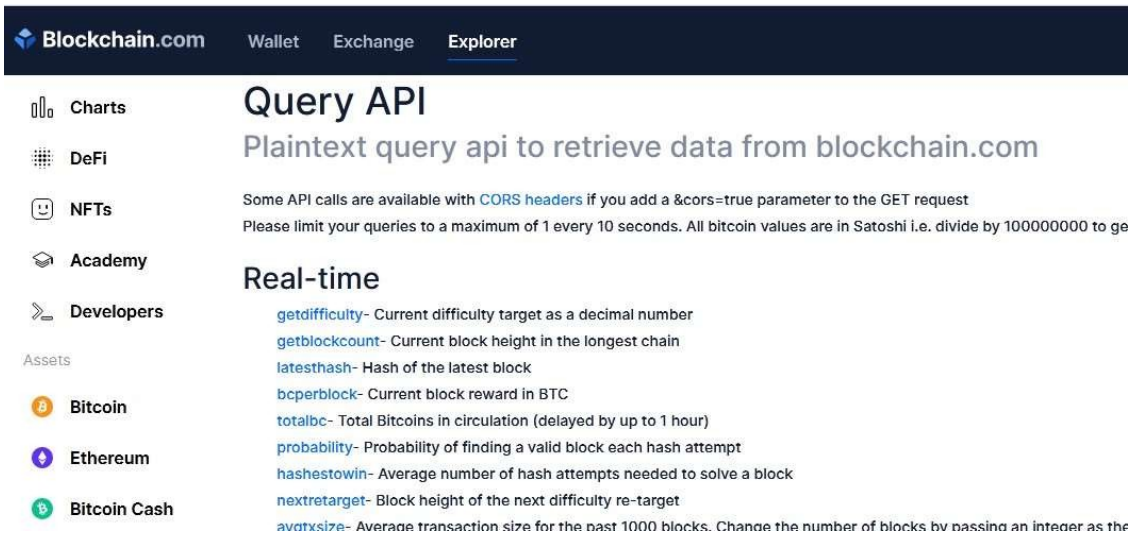


Use of special variables:

Use special variables in solidity contract to get information about blockchain and transaction details.

Name	Returns
blockhash(uint blockNumber) returns (bytes32)	Hash of the given block - or most recent, excluding c
block.coinbase (address payable)	Current block miner's
block.difficulty (uint)	Current block dif
block.gaslimit (uint)	Current block ga
block.number (uint)	Current block nu
block.timestamp (uint)	Current block timestamp a unix epoch
gasleft() returns (uint256)	Remaining g
msg.data (bytes calldata)	Complete call
msg.value (uint)	Number of wei sent with
now (uint)	Current block time
tx.gasprice (uint)	Gas price of the tra

Use of Query API: Use Query API to get information about blockchain and transaction details. Use following URL https://www.blockchain.com/api/q



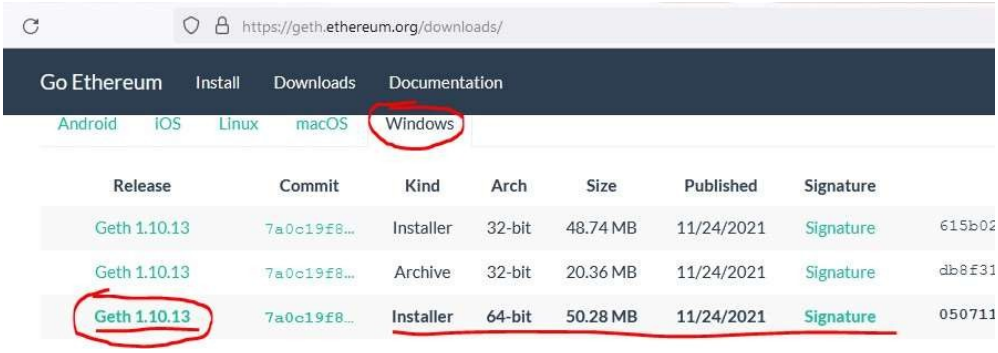
Setting up an Ethereum Node:[3]

- Ethereum node is any device that is running the Ethereum protocol (blockchain).
- When we connect to the Ethereum protocol we are on the Ethereum blockchain network.
- By running an Ethereum node we can connect to other nodes in the network, have direct access to the blockchain, and even do things like mine blocks, send transactions, and deploy smart contracts.

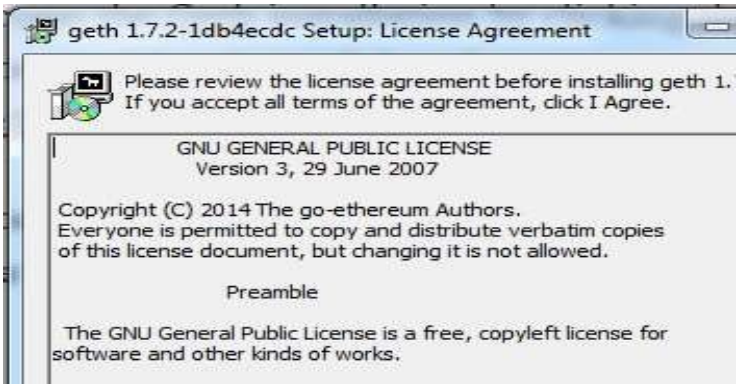
Creating private Ethereum blockchain using Geth (Go-Ethereum):[3]

Steps to install Geth:

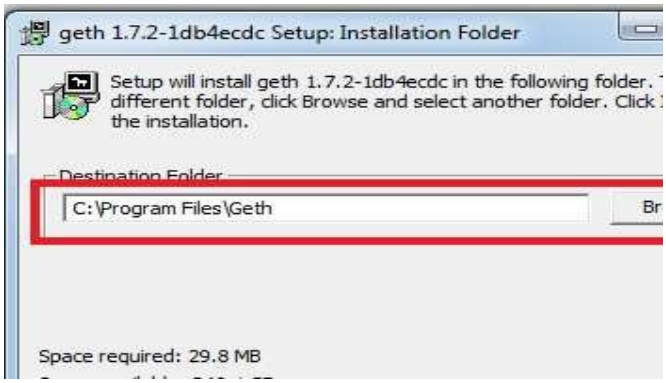
- Visit the Go Ethereum website and install Geth
- Visit here http://geth.Ethereum.org/downloads/
- Download the latest release of Geth for Windows, make sure you download the 64-bit version.



- Once your download is complete, open the installer and click "I Agree"



- You'll be prompted to select a destination folder for your download. By default, Geth will install under C:\Program Files\Geth



- **Close installation once complete**

Creating private Ethereum blockchain using Geth (Go-Ethereum):[3]

Establishing Our Own Private Ethereum Network:

- Create a new folder on your desktop called "private-chain".
- Open command prompt in this folder and create a data directory folder for our chaindata by typing "mkdir chaindata".
- Next, we need to create and save our genesis.json block in our Private Chain folder, as the genesis block will be used to initialize our private network and store data in the data directory folder "chaindata".

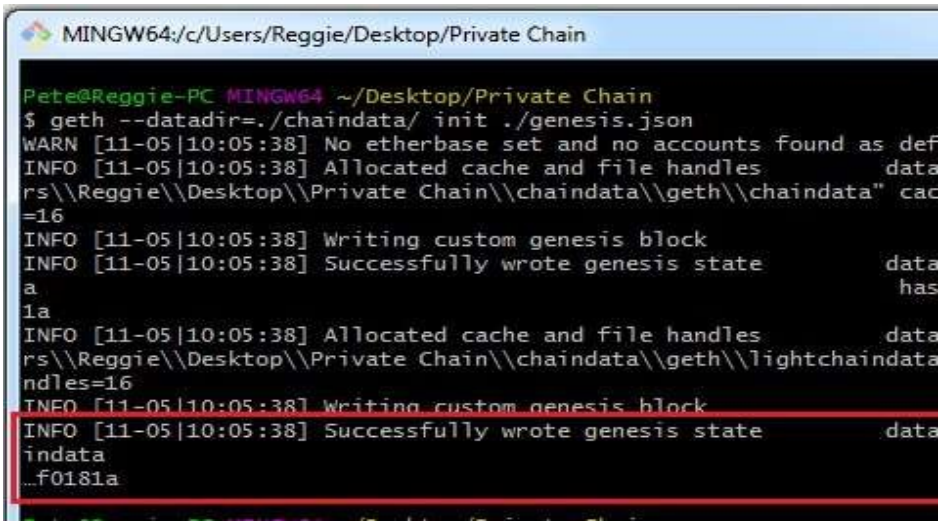
Open up notepad, copy & paste the code below into a new file called "genesis.json" and save this file in our Private-Chain folder.

```
{ "coinbase"   : "0x0000000000000000000000000000000000000001",
  "difficulty" : "0x20000",
  "extraData"  : "",
  "gasLimit"   : "0x2fefd8",
  "nonce"      : "0x0000000000000042",
  "mixhash"    : "0x0000000000000000000000000000000000000000000000000000000000000000",
  "parentHash" : "0x0000000000000000000000000000000000000000000000000000000000000000",
  "timestamp"  : "0x00",
  "alloc": {},
  "config": {
      "chainId": 15,
      "homesteadBlock": 0,
      "eip155Block": 0,
      "eip158Block": 0,
      "eip150Block": 0  }}
```

Start the Ethereum peer node (Start the Blockchain)

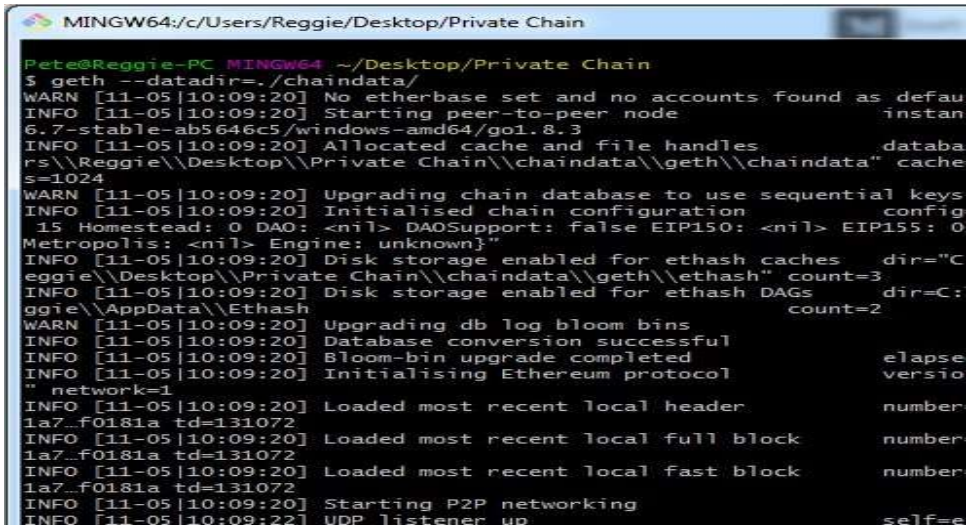
Run the following command:

geth --datadir chaindata init genesis.json



Now we can start Geth and connect to our own private chain.

geth --datadir=./chaindata/



- Minimize the terminal and Open new terminal.
- Type the command: Here is IPC is used to interact with Geth
  geth attach ipc:\\.\pipe\geth.ipc
- Command to create new account (here you need to account password) personal.newAccount()
- To know the all accounts eth.accounts
- To know the main account eth.coinbase
- To get balance of account eth.getBalance(eth.accounts[0])
- Start mining miner.start()
- Stop mining miner.stop()
- Again check the balance of same account eth.getBalance(eth.accounts[0])
- Create another account personal.newAccount()

Transaction:

- Unlock the account 0 to send transaction - personal.unlockAccount(eth.accounts[0])
- Command to send transaction – eth.sendTransaction({from: eth.coinbase, to: eth.accounts[1], value: web3.toWei(10, "ether")})

start mining and stop mining:

- Start mining miner.start() Stop mining miner.stop()
- Check the account balance after successfully mining the block, some ethers will be added to accounts 0 in Wei unit eth.getBalance(eth.accounts[0])
- Also check the balance of account 1 in Wei unit eth.getBalance(eth.accounts[1])
- Balance of second account in ether web3.fromWei(eth.getBalance(eth.accounts[1]), "ether")
- To get details of latest block eth.getBlock("latest")
- To get details of specific block eth.getBlock(35)

The Complete Blockchain Developer Toolkit :[3]

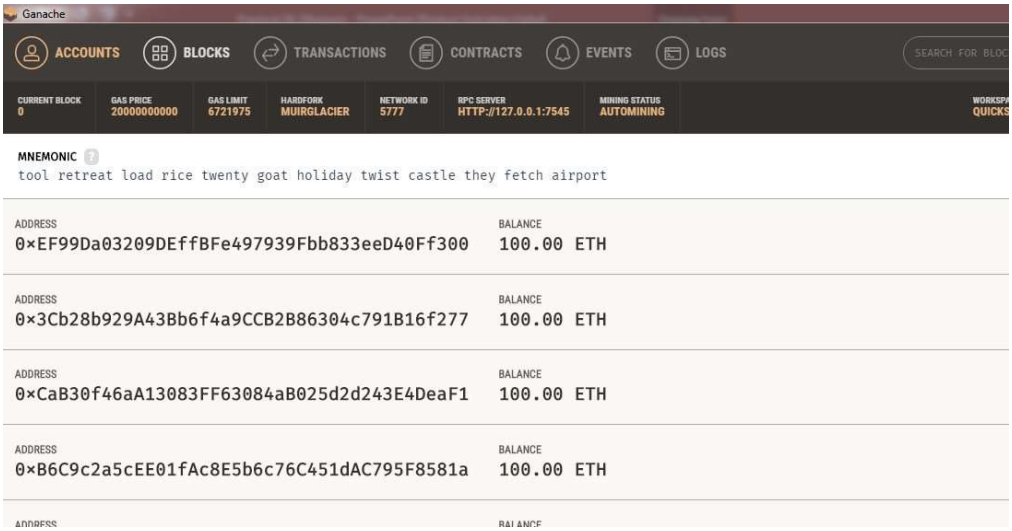
Introduction to Ganache:

- Ganache as your personal blockchain for Ethereum development.
- It will allow you to deploy smart contracts, develop applications, and run tests.
- From the above URL link downloaded the archived package, extract the installer and run through the setup steps.

Ganache Personal Blockchain Interface consist of:

- Accounts Page - this shows you all of the accounts that are automatically generated, along with their balances.
- Blocks Page - this shows you each block that has been mined on the personal blockchain network, along with the gas cost and transactions.
- Transactions Page - this list all the transactions that have taken place on the personal blockchain.
- Logs Page - this shows you all the server logs that you might need when debugging your application.
- Link to download Ganache https://www.trufflesuite.com/ganache
- Open the Ganache and then select the workspace QUICKSTART.

Ganache Personal Blockchain Interface :



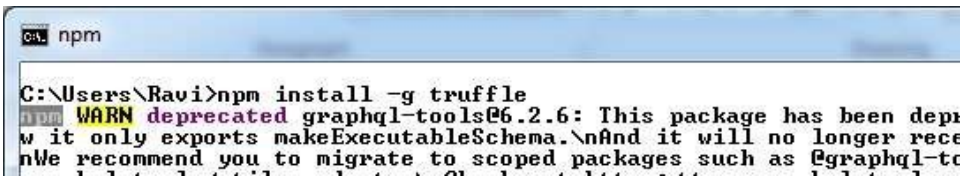
Developing smart contracts using Truffle:[4]

- Install **Node JS** and **Truffle Suite** to develop and migrate the smart contracts into the Private Blockchain network.
- Make use of Truffle tools like compile, migrate and test for compilation, migration and testing the smart contracts through Blockchain.
- Note that you have a private blockchain running, you need to configure your environment for developing smart contracts.
- The first dependency you'll need is Node Package Manager, or NPM, which comes with Node.js.
- You can see if you have node already installed by going to your terminal and typing: **node –v**
- Otherwise, Download from: URL: https://nodejs.org/en/

Truffle Framework:[4]

- Truffle Framework provides a suite of tools for developing Ethereum smart contacts with the Solidity programming language.
    - Smart Contract Management
    - Automated Testing
    - Deployment of smart contract
    - Migration of smart contract onto private blockchain network
    - Network Management
    - Script Runner
    - Client Side Development
- You can install Truffle with NPM in your command line like this:

  npm install -g truffle



Create a new truffle project:[4]

- Go to cmd prompt :

  mkdir blockchain-toolkit

cd blockchain-toolkit

truffle init

- Command to install touch for windows: **npm install -g touch-for-windows**
- Command to create file: **touch package.json**
- copy-and-pasting the below code into **package.json** file (It's configuration file required to run truffle project):

```
{
"name": "blockchain-toolkit",
"version": "1.0.0",
"description": "The Complete Blockchain Developer Toolkit for 2019 & Beyond",
"main": "truffle-config.js",
"directories": {
"test": "test"
},
"scripts": {
"dev": "lite-server",
"test": "echo \"Error: no test specified\" && sexit 1"
},
"author": "gregory@dappuniversity.com",
"license": "ISC",
"devDependencies": {
```

- copy-and-pasting the below code into **package.json** file

```
"bootstrap": "4.1.3",
"chai": "^4.1.2",
"chai-as-promised": "^7.1.1",
"chai-bignumber": "^2.0.2",
"dotenv": "^4.0.0",
"ganache-cli": "^6.1.8",
"lite-server": "^2.3.0",
"nodemon": "^1.17.3",
"solidity-coverage": "^0.4.15",
"truffle": "5.0.0-beta.0",
"truffle-contract": "3.0.6",
"truffle-hdwallet-provider": "^1.0.0-web3one.0"
}
}
```

- Save **package.json** file
- Run the following commad:

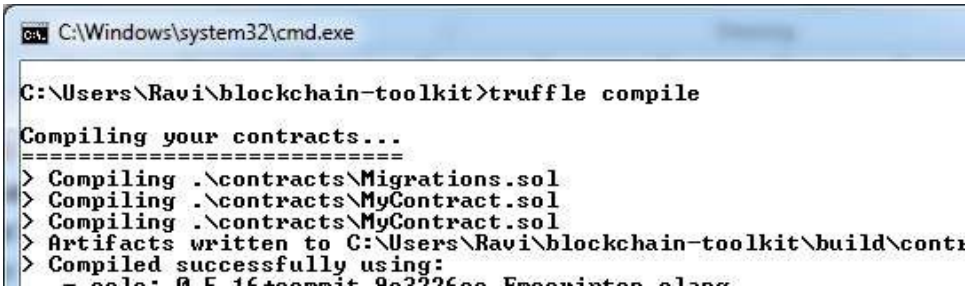
  npm install
- **Start developing a smart contract using solidity:**
- Run the touch command to create new contract:

  touch ./contracts/MyContract.sol
- Copy the below contract code and save in MyContract.sol

```
pragma solidity ^0.5.0;
contract MyContract {
string value;
constructor() public {
value = "myValue"; }
function get() public view returns(string memory) {
return value; }
function set(string memory _value) public {
value = _value; }
}
```

- Run the following command:

  truffle compile



- **Update the project configuration file to specify the personal blockchain network we want to connect to (Ganache):**
- Find the file **truffle-config.js** and paste the following code:

```
module.exports = {
networks: {
development: {
host: "127.0.0.1",
port: 7545,
network_id: "*" // Match any network id
}
},
solc: {
optimizer: {
enabled: true,
runs: 200
        }
   }
}
```

- **Create a migration script inside the migrations directory to deploy the smart contract to the personal blockchain network.**
- Run the following command:

  touch migrations/2_deploy_contracts.js
- Copy the below script into **2_depoy_contracts.js**

```
var MyContract = artifacts.require("./MyContract.sol");
```

```
module.exports = function(deployer)
{
deployer.deploy(MyContract);
};
```

- **Run the newly created migration script to deploy the smart contract to the personal blockchain network:**

  truffle migrate

- **Run the following commands:** truffle console

  MyContract.deployed().then((instance) => { app = instance })

- **Now we can read the storage value**

  app.get()

  Output as 'myValue'

- **Now we can set a new value like this:**

  app.set('New Value')

- **We can read that the value was updated like this:**

app.get()

Output as 'New Value'

- **You can exit the Truffle console by typing this command:**

  .exit

References:

1. https://en.wikipedia.org/wiki/Ethereum
2. https://en.wikipedia.org/wiki/MetaMask
3. https://codeburst.io/build-your-first-ethereum-smart-contract-with-solidity-tutorial-94171d6b1c4b
4. https://www.dappuniversity.com/articles/blockchain-developer-toolkit
5. https://www.tutorialspoint.com/solidity/solidity_quick_guide.htm

Exercise:

1. Install the metamask in browser. Setup the metamask digital cryptocurrency wallet. Create multiple accounts in metamask and connect with one of the etherum blockchain test network. Perform the task buy ethers and send ethers from one account to another. Take the screenshots of created accounts, account assets and account transactions which showing the details of transaction.
   (Use following url to get free ether for Sepolia Test Network: https://faucets.chain.link/sepolia)
2. Start **Ganache** (your personal private blockchain network). Connect Ganache with **MetaMask** and import the account from Ganache to MetaMask. Transfer funds from imported account to other account of MetaMask. Take the screenshots of created accounts, account assets and account transactions which showing the details of transaction from MetaMask and Ganache interface.

GoEthereum(Geth)

3. Create Ethereum node using **Geth (GoEthereum)** and create genesis block and create your personal private Ethereum blockchain. And use IPC to interact with Geth node to perform following task: create account, transfer funds using send transaction, mine the block, show the account balance before and after the mining the block, show the specific block details and access chain details.

RemixIDE – Injected Provider - Public Test Network (Goerli, Sepolia) or Ganache

4. Write a solidity smart contract for performing following task using **remixIDE** and deployed it on **public test network – Goerli / Sapolia** using **Injected provider** environment.
   a. To transfer funds (ethers) from user account to contract account.
   b. To withdraw funds (ethers) from contract account to user account.
   c. To apply restriction that only owner of the contract can withdraw funds (ethers) from contract account to his/her user account.

5. Write a smart contract to calculate the compound interest and deploy it on **Ganache** using **injected provider**.

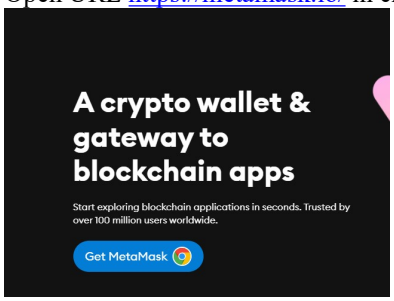
Truffle - Ganache

6. Build and test decentralized application **(Dapp) for Election Voting System** on the local Ethereum Blockchain Network **Ganache** using **truffle suite**.
7. Build and test decentralized application, **(Dapp) for Banking System** on the local Ethereum Blockchain Network **Ganache** using **truffle suite**.

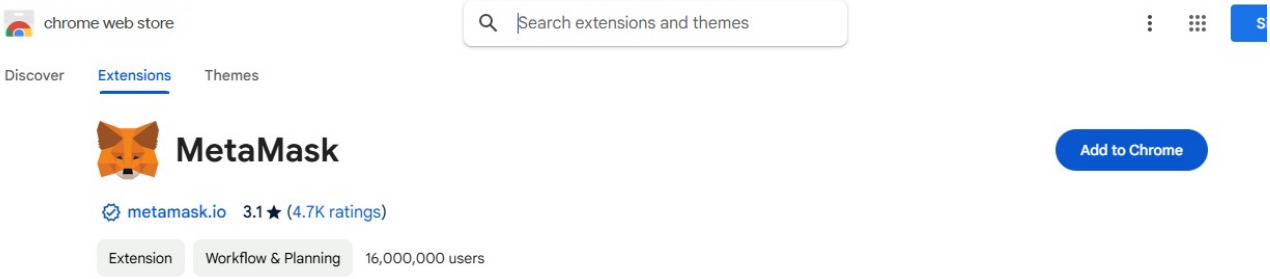
1.Install the metamask in browser. Setup the metamask digital cryptocurrency wallet. Create multiple accounts in metamask and connect with one of the etherum blockchain test network. Perform the task buy ethers and send ethers from one account to another. Take the screenshots of created accounts, account assets and account transactions which showing the details of transaction.
(Use following url to get free ether for Sepolia Test Network: https://faucets.chain.link/sepolia)

Installation of MetaMask:

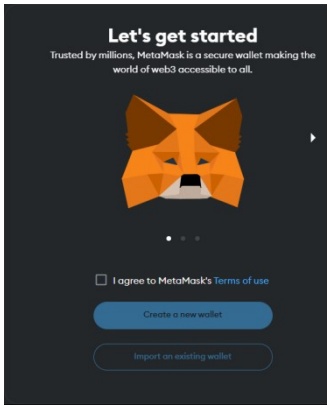
Open URL https://metamask.io/ in chrome browser



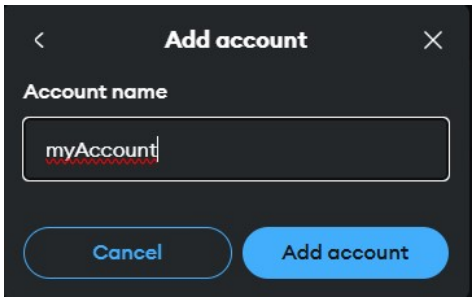
Add Extension to Chrome



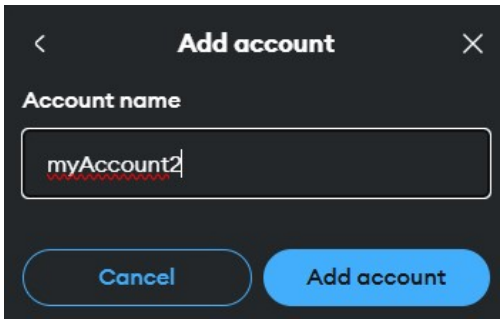
Create Wallet



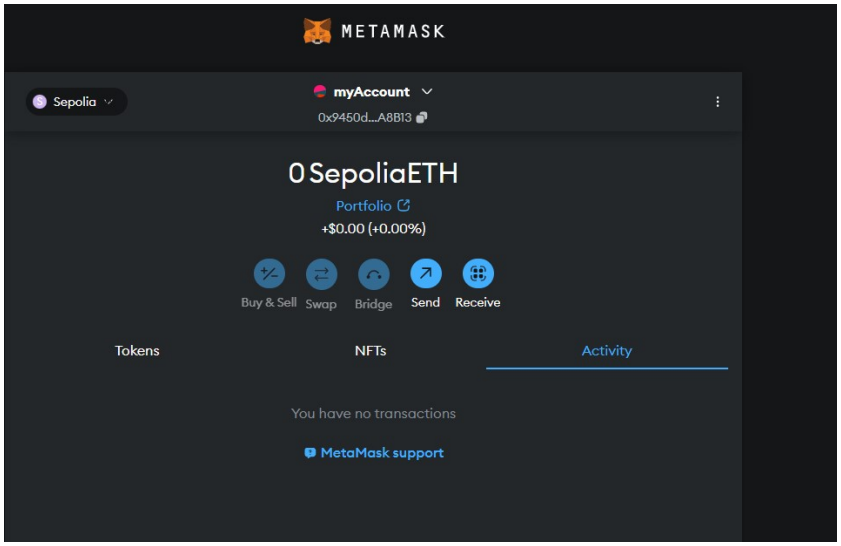
Create Account: myAccount



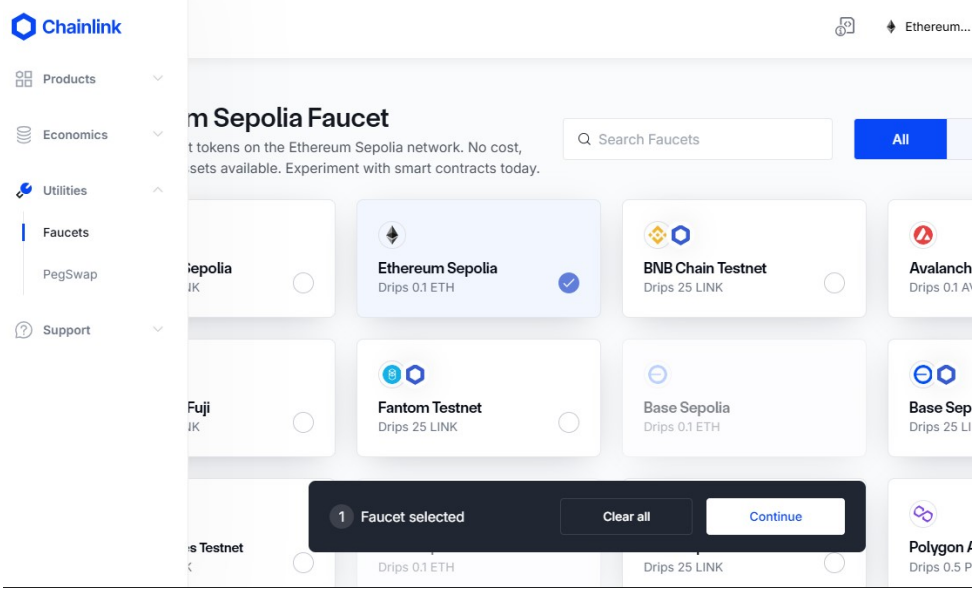
Create Account: myAccount2



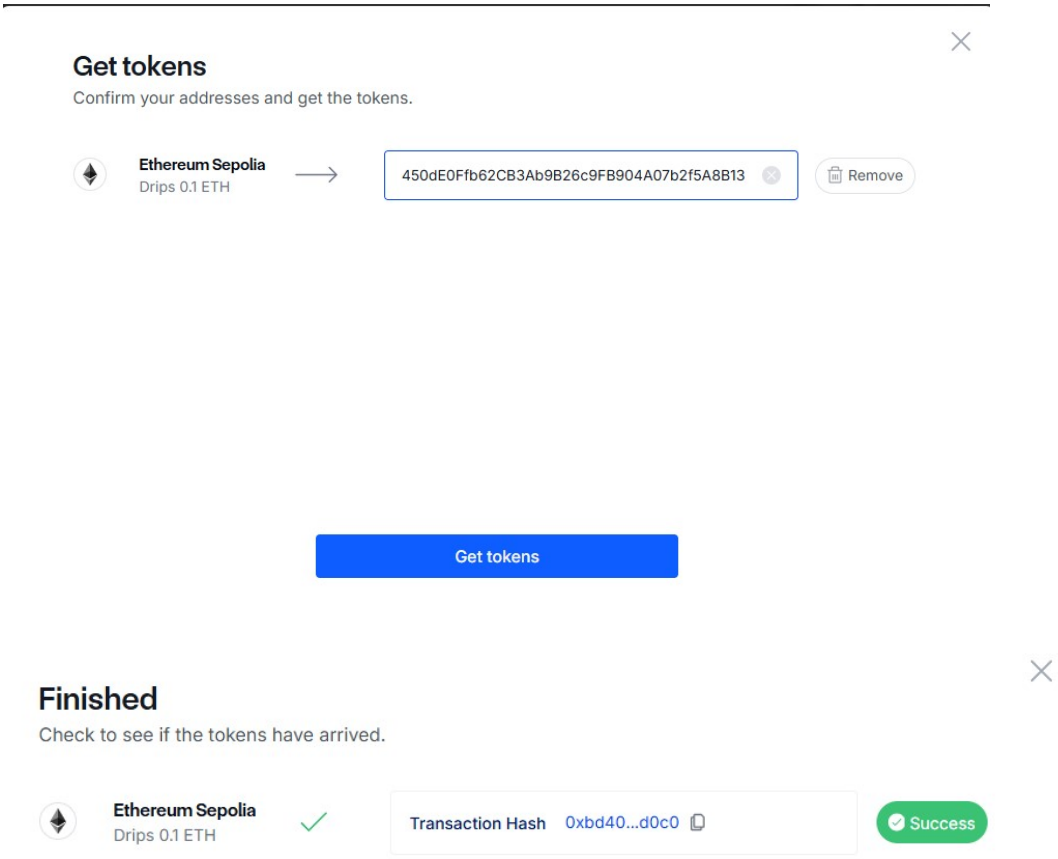
Interface:

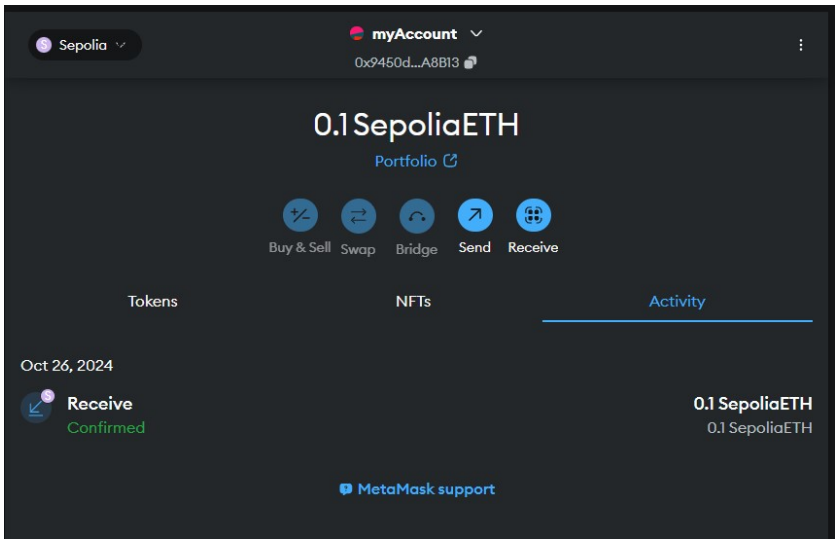


Open URL https://faucets.chain.link/sepolia in chrome browser for sepolia

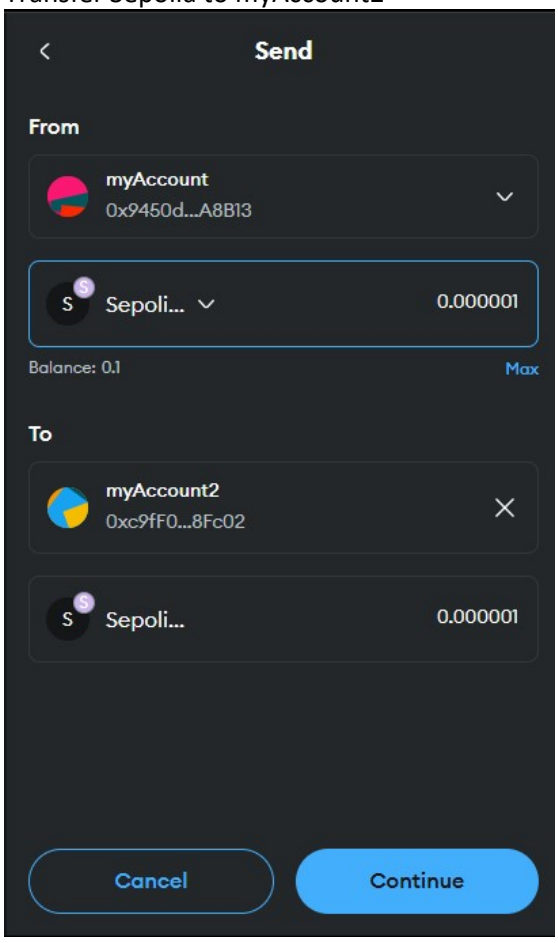


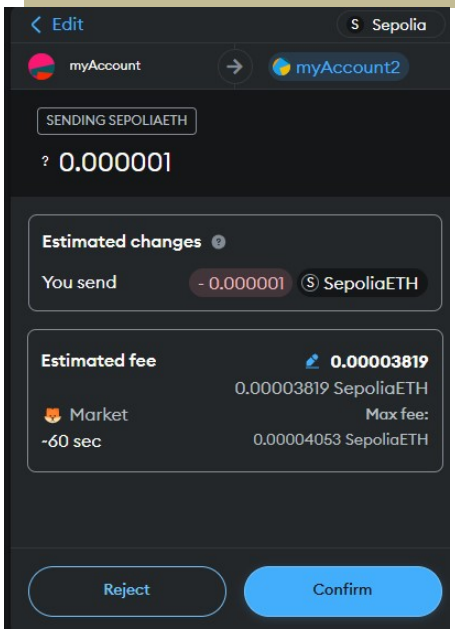
Claim Ethereum Sepolia



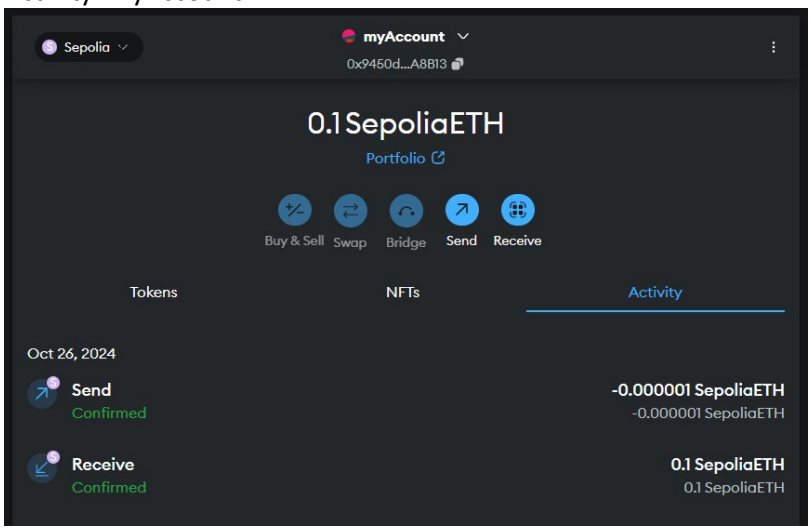


Transfer Sepolia to myAccount2

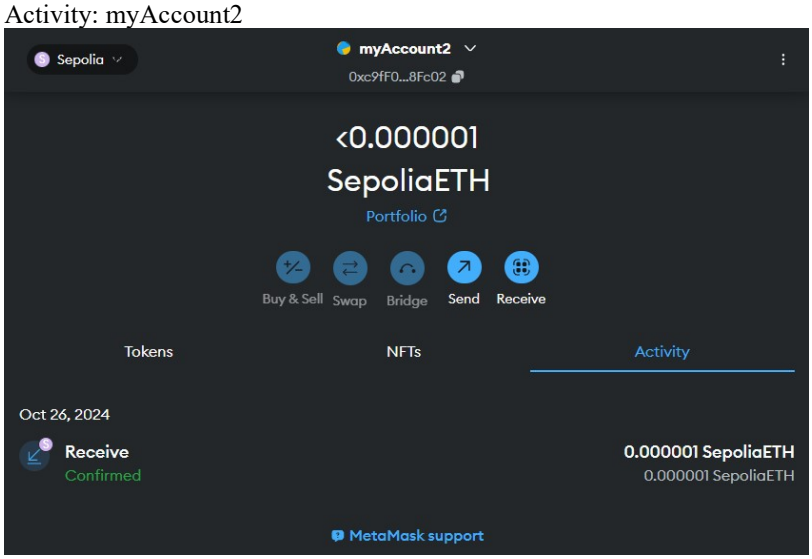




Activity: myAccount

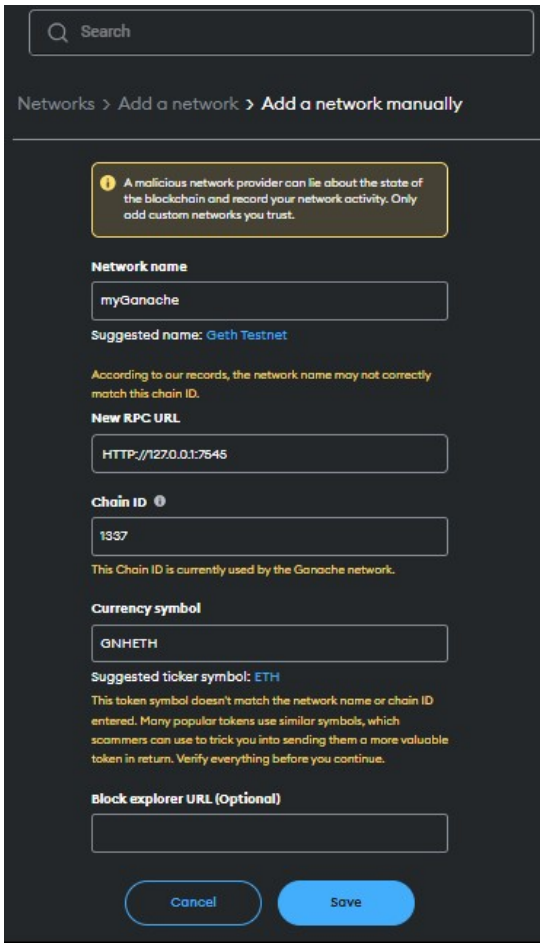


Activity: myAccount2



2. Start Ganache (your personal private blockchain network). Connect Ganache with **MetaMask** and import the account from Ganache to MetaMask. Transfer funds from imported account to other account of MetaMask. Take the screenshots of created accounts, account assets and account transactions which showing the details of transaction from MetaMask and Ganache interface.

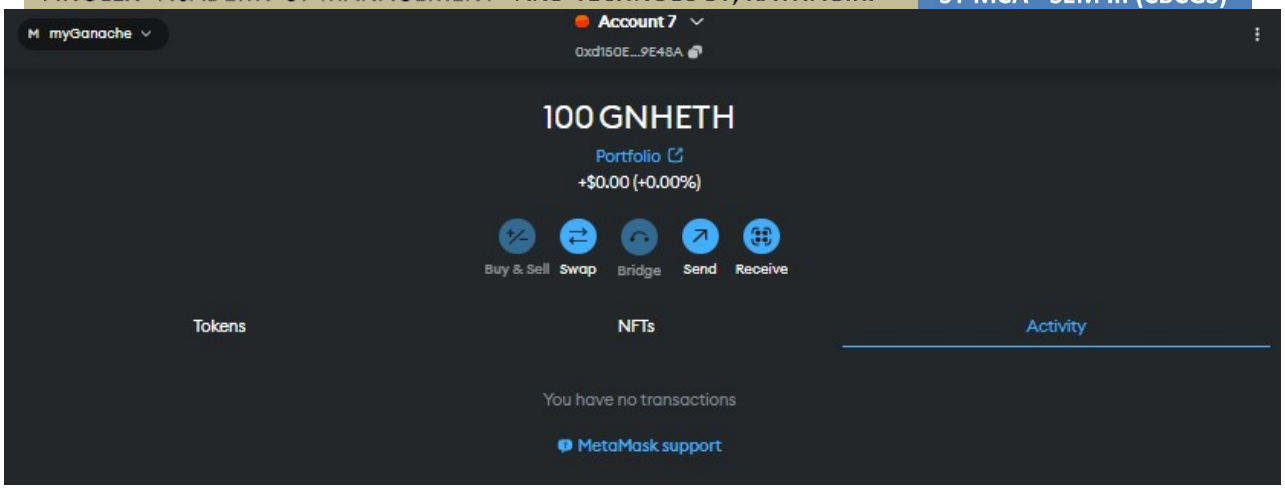
1. Save the Network



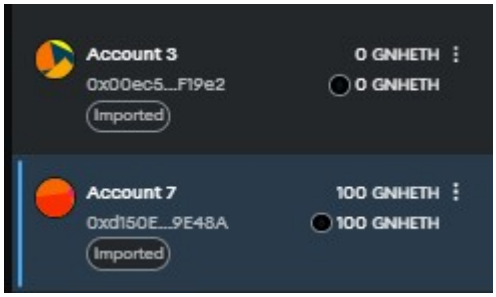
2. Copy the keys

ADDRESS	BALANCE	TX COUNT	INDEX	
0×d150E46b725682cf9Dc9Cf7De347ef7E4289E48A	100.00 ETH	0	0	
0×5080006f6c41f39Be30279f13C0cE0aF2E4979a1	100.00 ETH	0	1	
0×8aE2853a23913a50D817C3DCbeD1bc414EcF1F48	100.00 ETH	0	2	
0×2909b9FE823d5B3C98d0A6CC79e8c0B1BB7a245d	100.00 ETH	0	3	
0×0E237936b3B9Dd0644F9dB8b95955c51a2eb2f29	100.00 ETH	0	4	

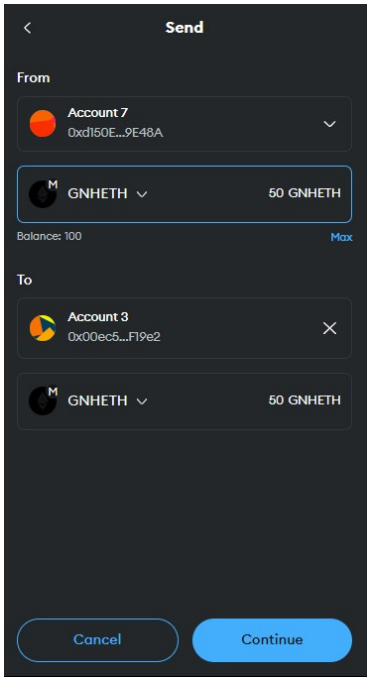
3. Claim The Ether



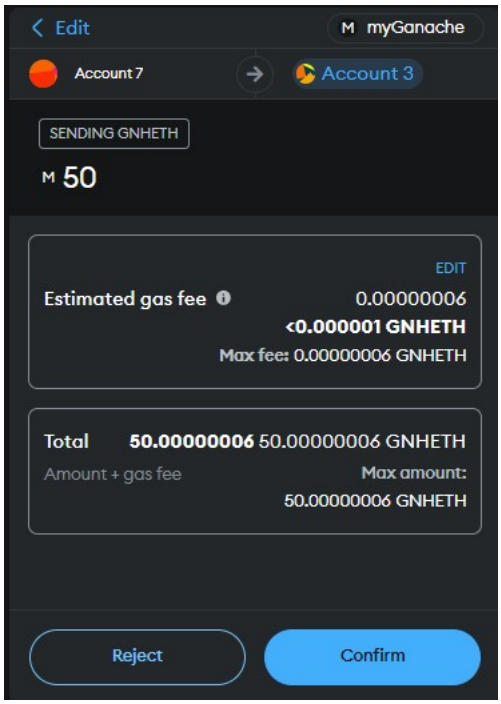
4. Account 7 and Account 3



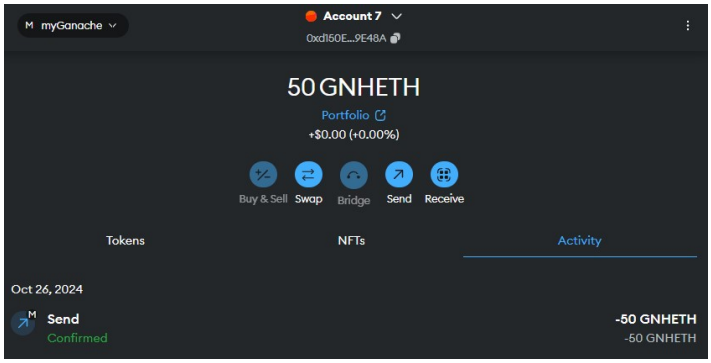
5. Transfer GNEtH



6. Confirm the Transaction



7. Interface: Account 7



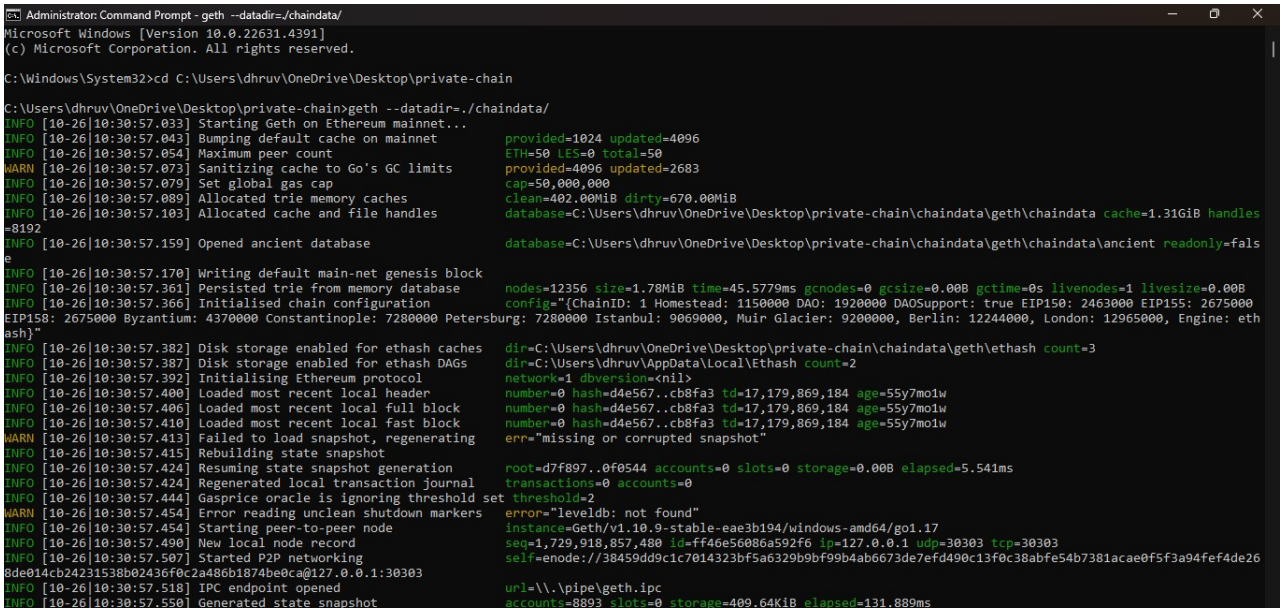
8. Ganache:

9. Transaction:

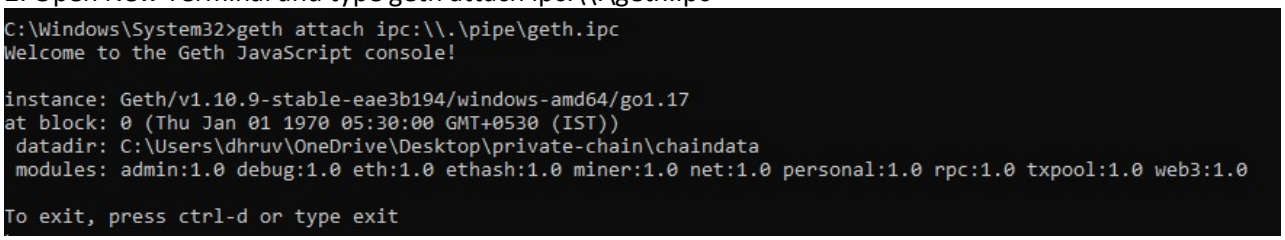
GOEthereum(Geth)

3. Create Ethereum node using Geth (**GoEthereum**) and create genesis block and create your personal private Ethereum blockchain. And use IPC to interact with Geth node to perform following task: create account, transfer funds using send transaction, mine the block, show the account balance before and after the mining the block, show the specific block details and access chain details.

1. Start Geth:



2. Open New Terminal and type geth attach ipc:\\.\geth.ipc



3. eth.accounts

4. personal.newAccount()

5. miner.start()

3. miner.stop()

```
> miner.stop()
null
```

4.eth.getBalance(eth.accounts[0])

```
> eth.getBalance(eth.accounts[0])
38500000000000000000
>
```

5.eth.sendTransaction({from: eth.coinbase, to:eth.accounts[1], value: web3.toWei(10, "ether")})

```
> eth.sendTransaction({from: eth.coinbase, to: eth.accounts[1], value: web3.toWei(10,"ether")})
"0x2781963205d2f96cdc120c522bbfa9ac1564bdc451faf014c336f033520a390e"
```

6. eth.getBalance(eth.coinbase)
   eth.getBalance(eth.accounts[1])

```
>
> eth.getBalance(eth.coinbase)
38500000000000000000
> eth.getBalance(eth.accounts[1])
10000000000000000000
>
```

7. eth.getBlock("latest")

```
> eth.getBlock("latest")
{
  difficulty: 134355,
  extraData: "0xd883010a09846765746886676f312e31378777696e646f7773",
  gasLimit: 3393305,
  gasUsed: 0,
  hash: "0x3912936d5557a3bf9aee80fa2788226c3fd583762452eb38cd5ac9e0c08f3061",
  logsBloom: "0x00000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000",
  miner: "0x392c7e5c09a9a279fddd81b55d8b755e9ad7bd16",
  mixHash: "0xc9aa4e14323da6ac89730d668dd1c6b70c592688c363ebc6d0b0440eea0ee389",
  nonce: "0x6d460822f4af84b2",
  number: 79,
  parentHash: "0x89f6f54e550f81bdb6a42c839df6410f47db70e3e501bf83ace443de26e9f62d",

  receiptsRoot: "0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421
",
  sha3Uncles: "0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347",

  size: 536,
  stateRoot: "0x04e0e88efb38e27c801d549843d8627a967d1cb69ed01e6c585fbc28ae578724",
  timestamp: 1729495612,
  totalDifficulty: 10591614,
  transactions: [],
  transactionsRoot: "0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363
b421",
  uncles: []
}
```

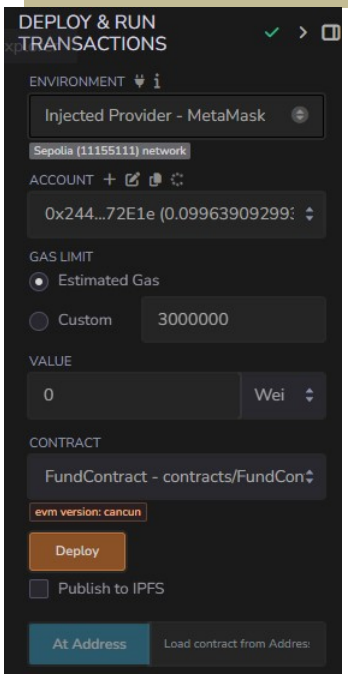
RemixIDE — Injected Provider- public Test Network (Goerli, Sepolia) or Ganache

4. Write a solidity smart contract for performing following task using **remixIDE** and deployed it on public **test network — Goerli / Sapolia** using **Injected provider** environment.
a. To transfer funds (ethers) from user account to contract account.
b. To withdraw funds (ethers) from contract account to user account.
c. To apply restriction that only owner of the contract can withdraw funds (ethers) from contract account to his/her user account.

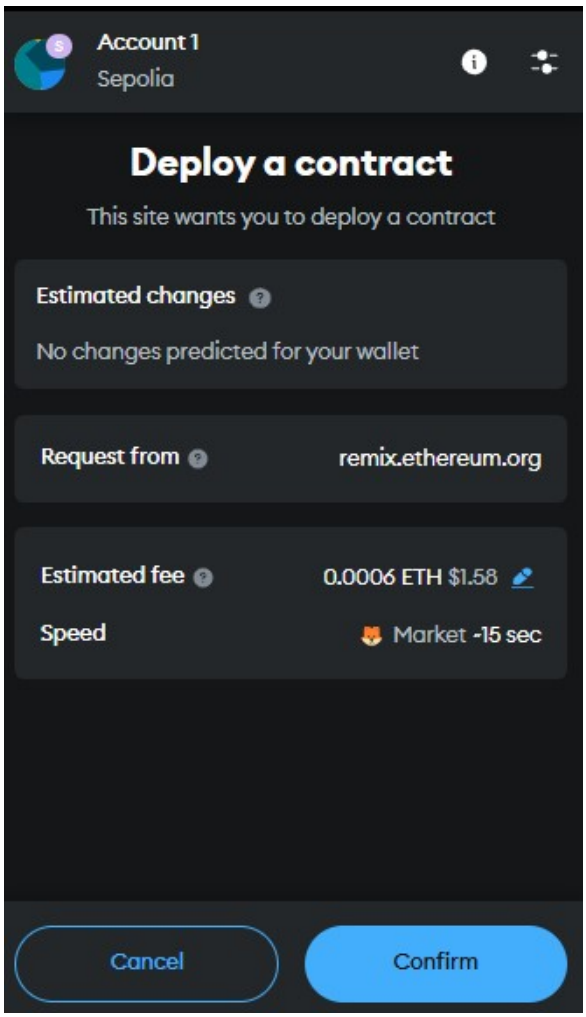
1. FundContract.sol:

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

contract FundContract {
    address public owner;
    constructor() {
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Only the owner can withdraw funds");
        _;
    }

    function deposit() external payable {
        require(msg.value > 0, "Must send some ether to deposit");
    }

    function withdraw(uint256 amount) external onlyOwner {
        require(amount <= address(this).balance, "Insufficient balance in contract");
        payable(owner).transfer(amount);
    }

    function getContractBalance() external view returns (uint256) {
        return address(this).balance;
    }
}
```

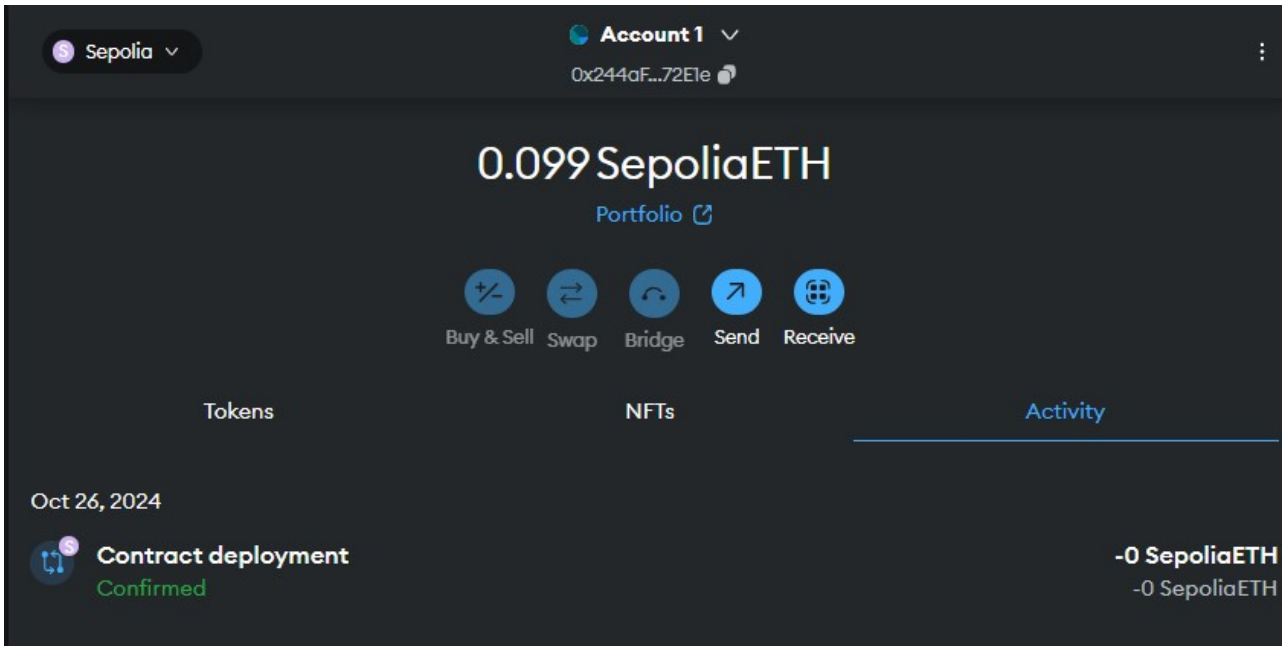
2. Deployment Window:



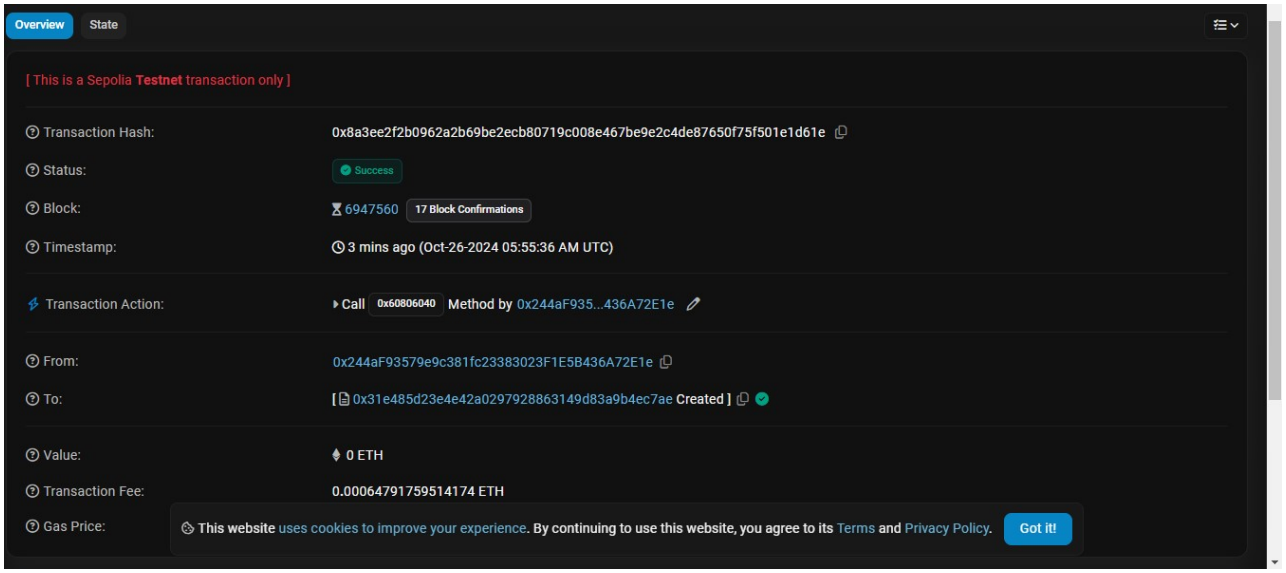
3. Confirm Transaction Window



4. MetaMask:



5. Transaction Details:

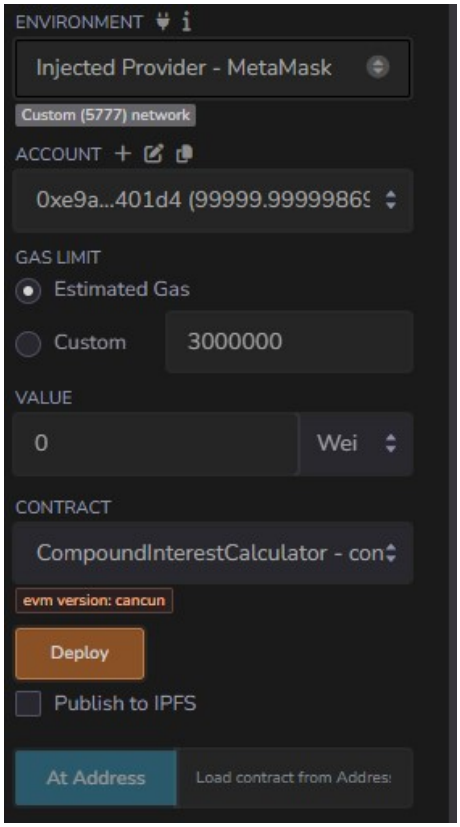


5 .Write a smart contract to calculate the compound interest and deploy it on **Ganache** using **injected provider.**

CompoundIntrest.sol:

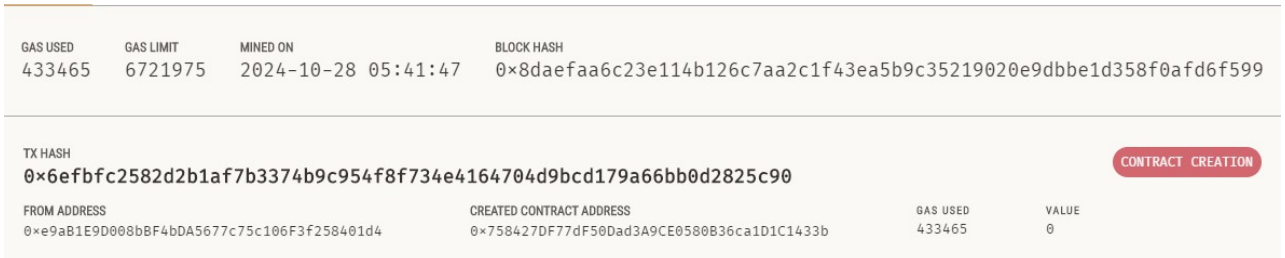
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract CompoundInterestCalculator {

    function calculateCompoundInterest(
        uint256 _principal,
        uint256 _rate,
        uint256 _time,
        uint256 _n
    ) public pure returns (uint256) {
        require(_principal > 0, "Principal must be greater than 0");
        require(_rate > 0, "Rate must be greater than 0");
        require(_time > 0, "Time must be greater than 0");
        require(_n > 0, "Number of times interest applied must be greater than 0");

        uint256 base = (1e18 + (_rate * 1e18 / (100 * _n)));
        uint256 exponent = _n * _time;

        uint256 amount = (_principal * power(base, exponent)) / 1e18;
        uint256 compoundInterest = amount - _principal;

        return compoundInterest;
    }

    function power(uint256 base, uint256 exponent) public pure returns (uint256) {
        uint256 result = 1e18; // Start with 1 (in 18 decimal precision)

        while (exponent > 0) {
            if (exponent % 2 == 1) {
                result = (result * base) / 1e18; // Multiply and maintain precision
            }
            base = (base * base) / 1e18; // Square the base and maintain precision
            exponent /= 2; // Divide exponent by 2
        }

        return result;
    }
}
```

Deploy the contract



Transactions on Ganache

Transaction



Truffle - Ganache

6. Build and test decentralized application (Dapp) for Election Voting System on the local Ethereum Blockchain Network Ganache using truffle suite.

VotingSystem.sol

```
//SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.8.27;

contract VotingSystem {
   struct Candidate {
      uint256 id;
      string name;
      uint256 voteCount;
   }

   mapping(address => bool) public voters;
   mapping(uint256 => Candidate) public candidates;

   uint256 public candidateCount;

   event votedEvent(uint256 indexed _candidateId);

   constructor() public {
      addCandidate("Candidate 1");
      addCandidate("Candidate 2");
      addCandidate("Candidate 3");
   }

   function addCandidate(string memory _name) public {
      candidateCount++;
      candidates[candidateCount] = Candidate(candidateCount, _name, 0);
   }

   function vote(uint256 _candidateId) public {
      require(!voters[msg.sender], "You already voted");
      require(_candidateId > 0 && _candidateId <= candidateCount);
      voters[msg.sender] = true;
      candidates[_candidateId].voteCount++;
      emit votedEvent(_candidateId);
   }

   function getCandidateDetails(uint _candidateId) public view returns (uint,string memory,uint) {
     return (candidates[_candidateId].id, candidates[_candidateId].name, candidates[_candidateId].voteCount);
   }


}
```

2_delpoy_contracts.js

```
var VotingSystem = artifacts.require("./VotingSystem.sol");
module.exports = function(deployer)
{
deployer.deploy(VotingSystem);
};
```

3. Open Terminal and type truffle migrate

```
C:\Users\dhruv\OneDrive\Desktop\blockchain-toolkit>truffle migrate

Compiling your contracts...
===========================
> Compiling .\contracts\VotingSystem.sol
> Artifacts written to C:\Users\dhruv\OneDrive\Desktop\blockchain-toolkit\build\contracts
> Compiled successfully using:
   - solc: 0.5.16+commit.9c3226ce.Emscripten.clang


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


2_deploy_contracts.js
=====================

   Deploying 'VotingSystem'
   ------------------------
   > transaction hash:    0x38112f2c9c024b2d8124f7d1726d477436b2415caa5f41918859493 3ec6327e2
   > Blocks: 0            Seconds: 0
   > contract address:    0x2Ae052232B70e9E5Bf8b9f97BFcd550b6F5E6776
   > block number:        1
   > block timestamp:     1729942291
   > account:             0x3C0BF12BDf9B792e3E309152B47427f5C6E33226
   > balance:             99.999998040997
   > gas used:            653001 (0x9f6c9)
   > gas price:           0.003 gwei
   > value sent:          0 ETH
   > total cost:          0.000001959003 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:      0.000001959003 ETH

Summary
=======
> Total deployments:   1
> Final cost:          0.000001959003 ETH
```

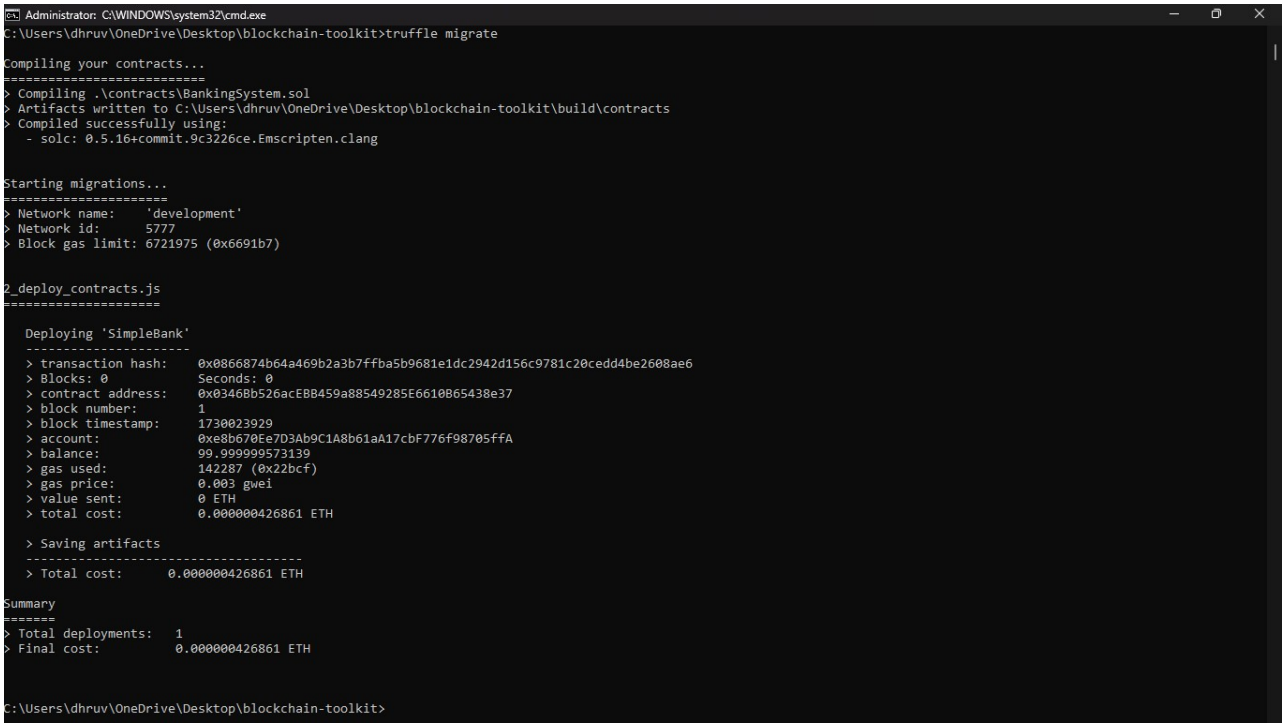
7. Build and test decentralized application, **(Dapp) for Banking System** on the local Ethereum Blockchain Network **Ganache** using **truffle suite.**

BankingSystem.sol:

```
pragma solidity >=0.5.16;
contract SimpleBank {
   // State variable to store the balance
   uint256 private balance;
   // Constructor to ini alize balance
   constructor () public {
      balance = 0;
   }
   // Func on to add (deposit) amount to the balance
   function addAmount(uint256 amount) public {
      balance += amount;
   }
   // Func on to withdraw amount from the balance
   function withdrawAmount(uint256 amount) public {
      require(amount <= balance, "Insufficient balance");
      balance -= amount;
   }
   // Func on to check the remaining balance
   function checkBalance() public view returns (uint256) {
      return balance;
   }
}
```

2_deploy_contracts.js

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

module.exports = function (deployer) {
   deployer.deploy(SimpleBank);
};
```



Conclusion: Developed and deployed Dapp in Ethereum.