Week - 1

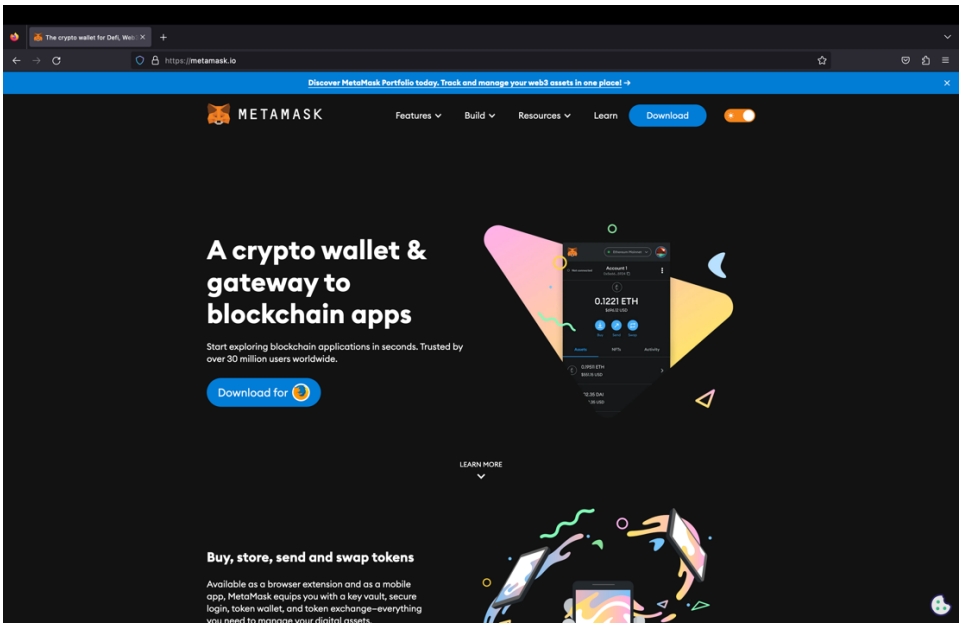
(a), (b)

Aim: To demonstrate

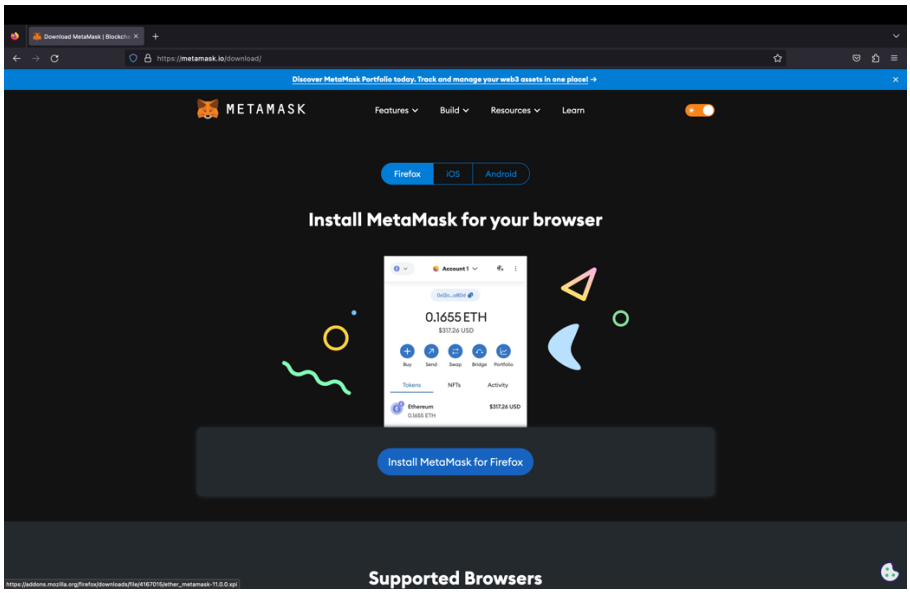
- the generation of Public private key pairs for Bitcoin and Ethereum addresses
- Connect to the Public/Testnet Ethereum Blockchain network using popular wallets (Metamask, Brave browser) and understand various terminologies like gas, gas fee, gas price, priority fee.

Installation of MetaMask:

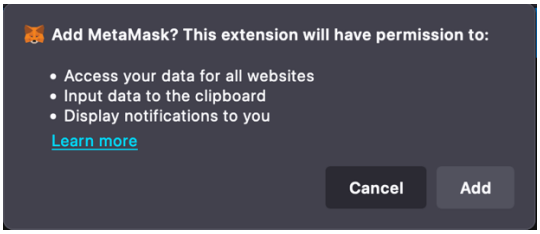
- Open any browser preferably Brave or Firefox and search for Metamask
- Open the link for metamask or go to https://metamask.io/



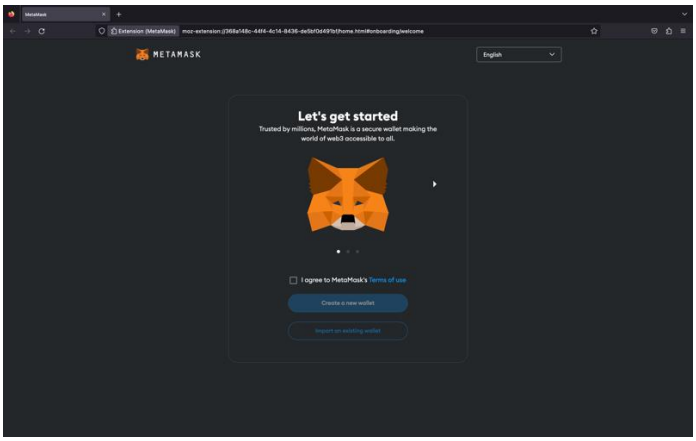
- Click on Install for Brave or Firefox



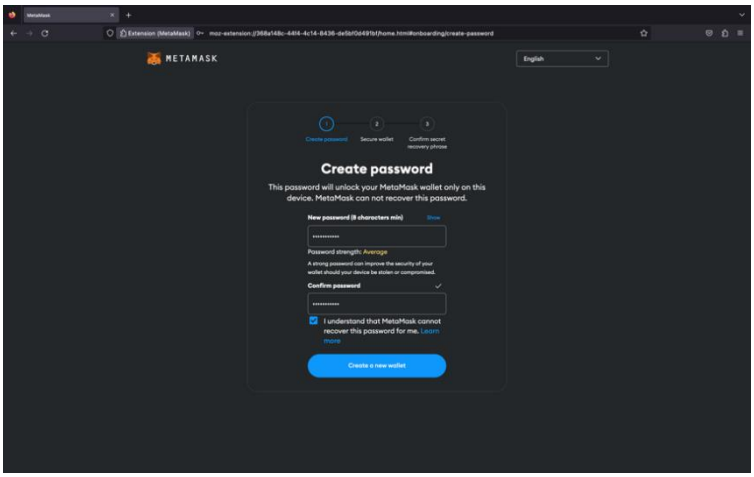
- After the extension is installed, allow the extension to be given the required permissions



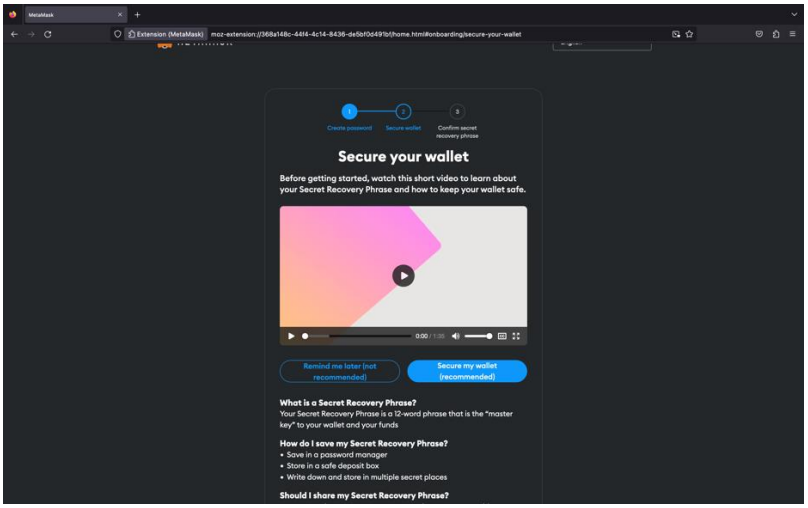
- Once the extension is opened, follow the process for account creation



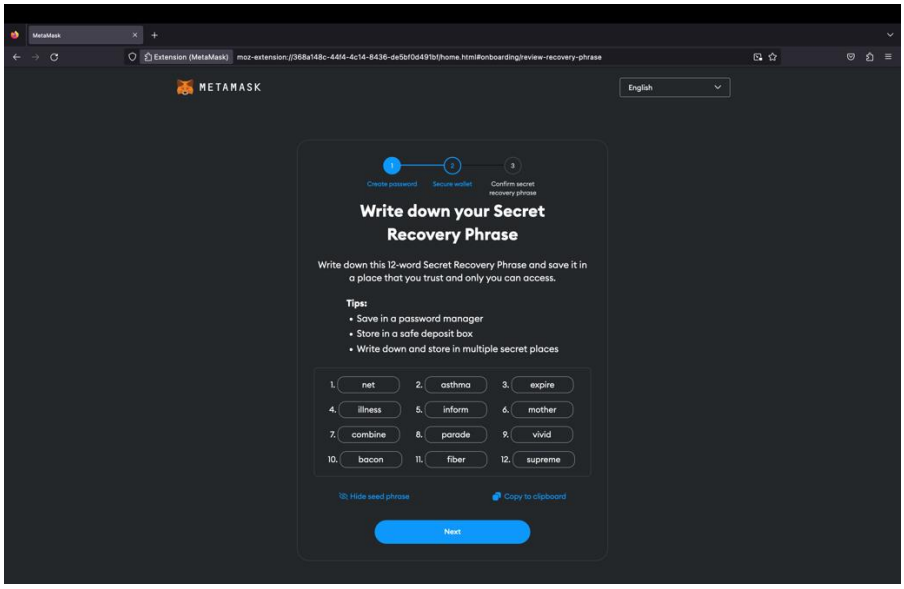
- Create a password following the required procedure



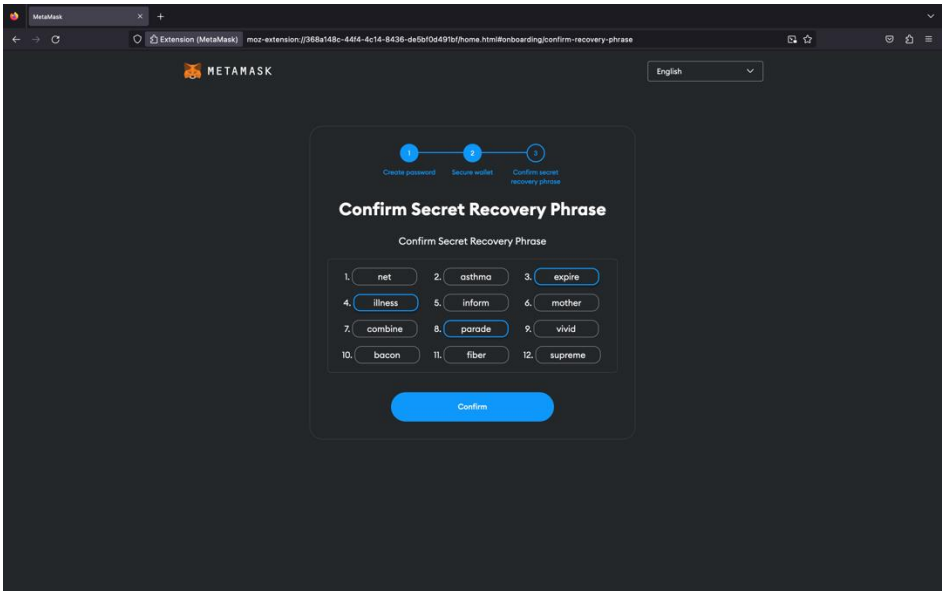
- Now click on secure the wallet



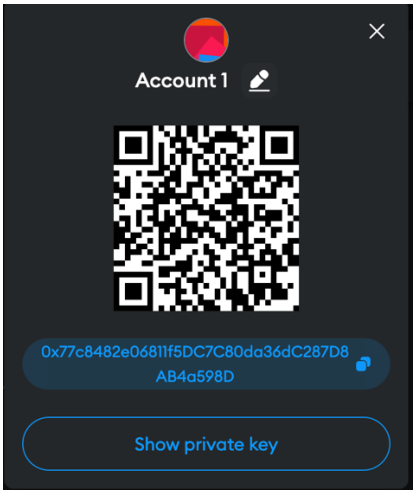
- Now make a note of the 12 secret words as they are important in retrieving the account if account is lost and also used in the next step



- Fill the words in the blanks using the previous saved words



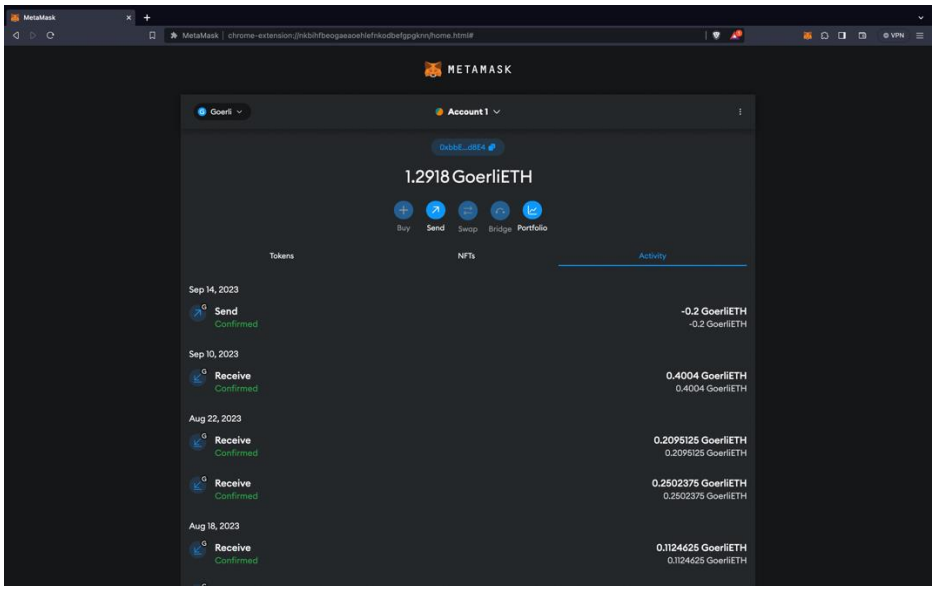
- Click on the account and open the options and click on show account details - Here we can see the QR code for account, Public Key & Private Key



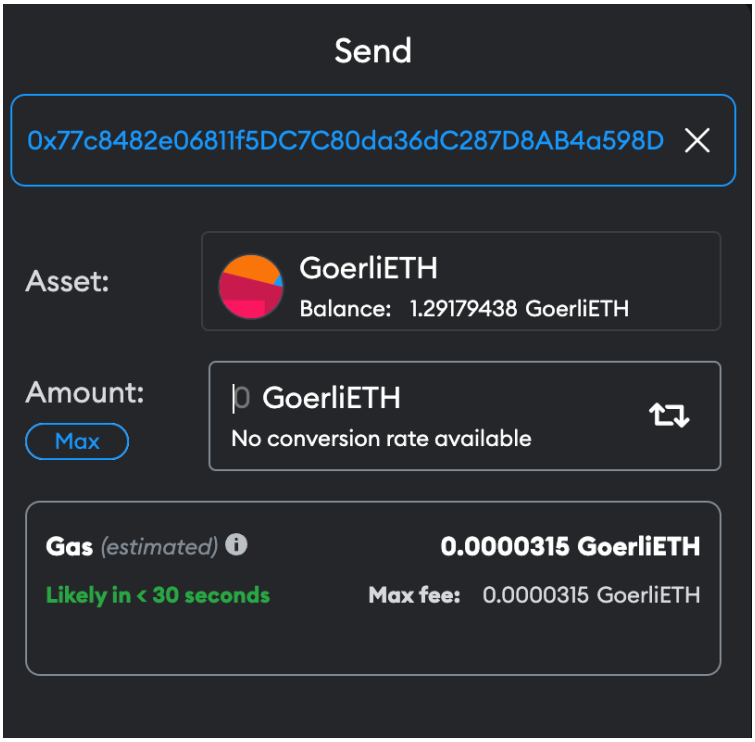
(c) Aim: To demonstrate the transfer of Test Ether from one account to another

Steps of Transfer in MetaMask:

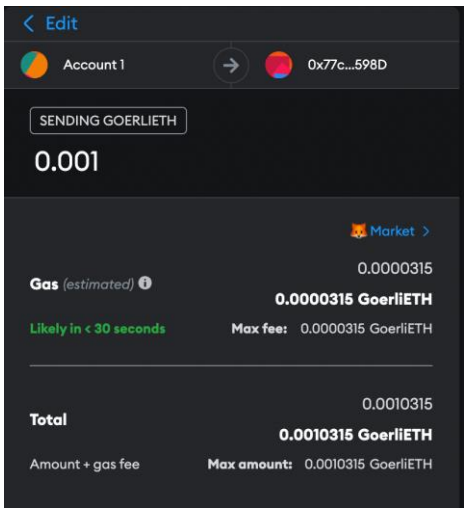
- Open the Metamask extension.
- Click on Send option.



- Enter the recipient's address/recipient's public key.



- Enter the details of the transactions including amount, gas price etc.
- Check the Transaction Details and click confirm



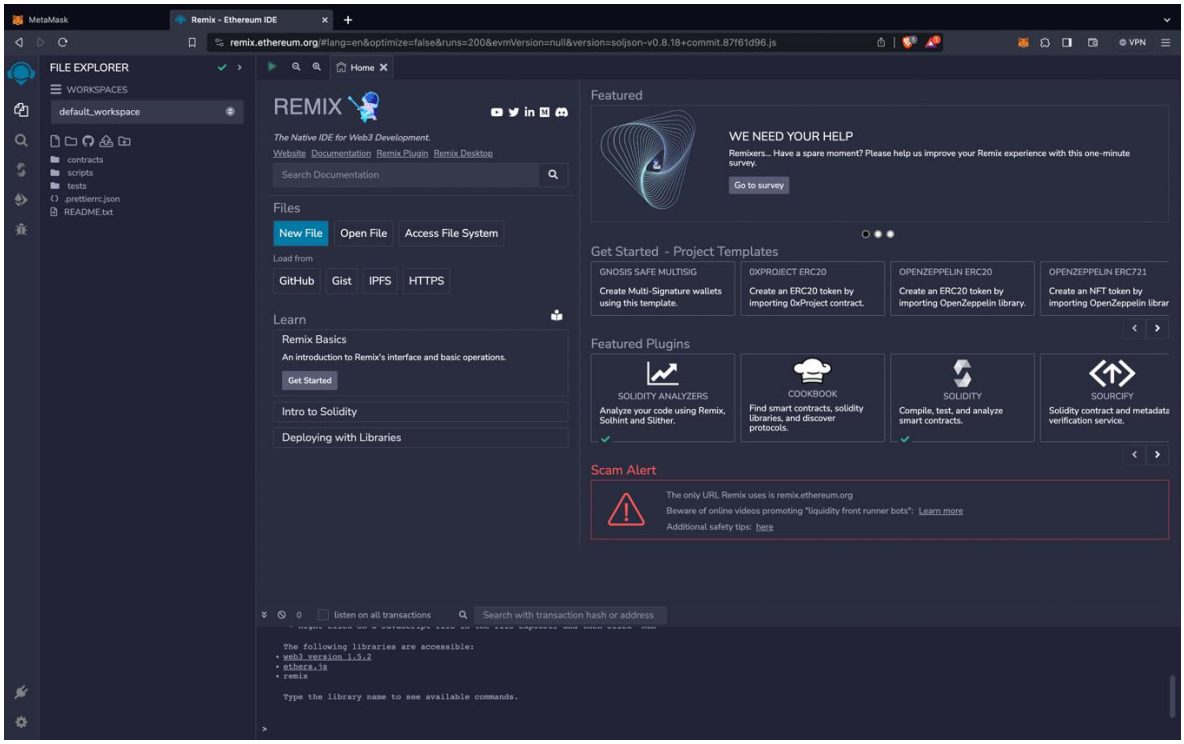
- The Test Ether is transferred to the recipient account

(d)

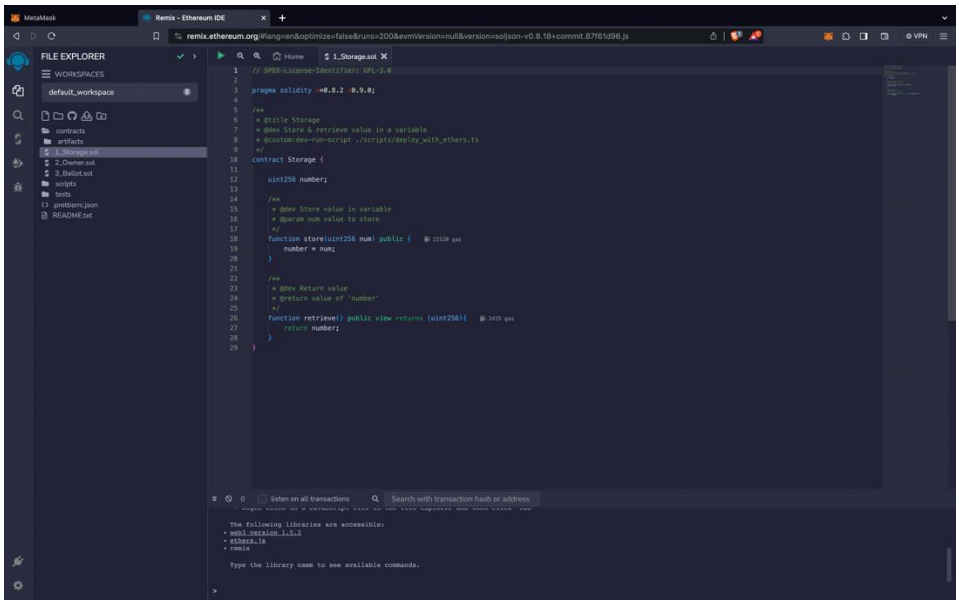
Aim: To demonstrate the transfer of Test Ether from one account to Smart Contract

Steps of Transfer in MetaMask:

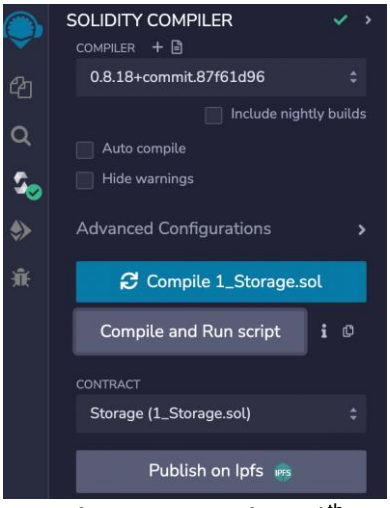
- Open any browser and search for Remix IDE
- Now click on the link or open https://remix.ethereum.org



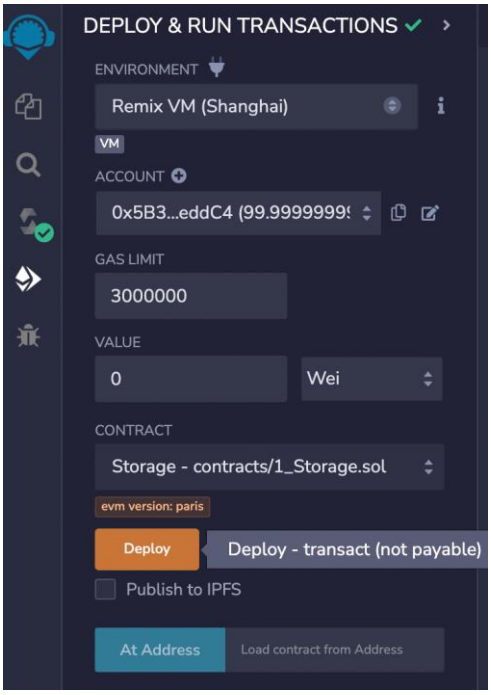
- Now in the home page, click on the contracts folder and open the file named Storage.sol



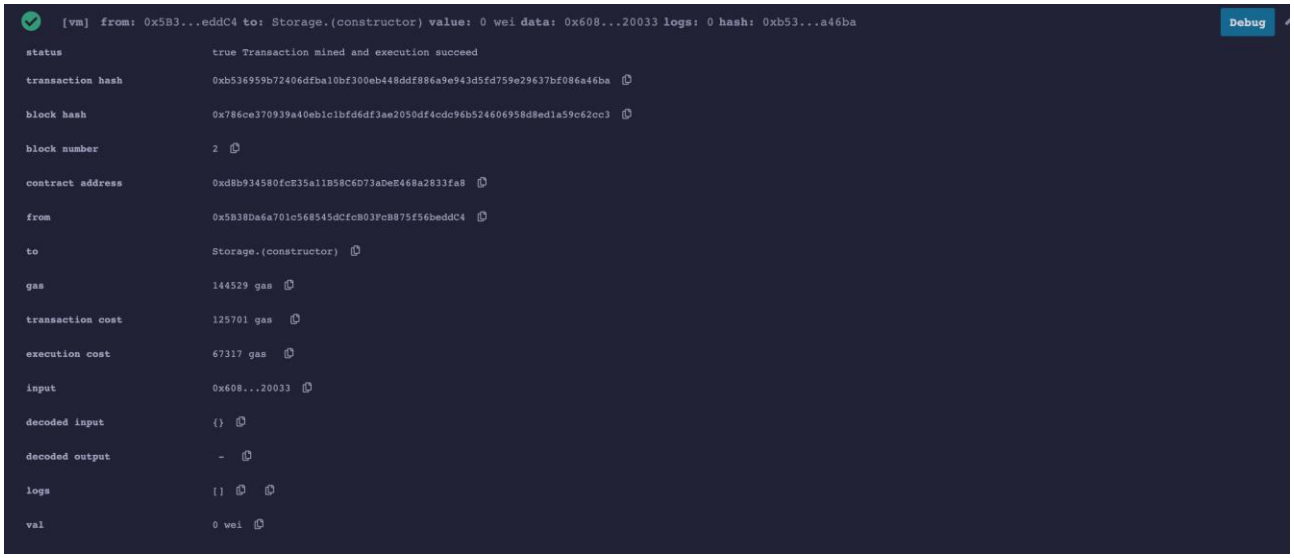
- Go to the 3rd option in the left-hand panel and click on compile & run



- Once you have a tick mark, go to the 4th option in the left panel and deploy the code



- Now in the output panel, you can observe all the details that are obtained in the execution of a Smart Contract

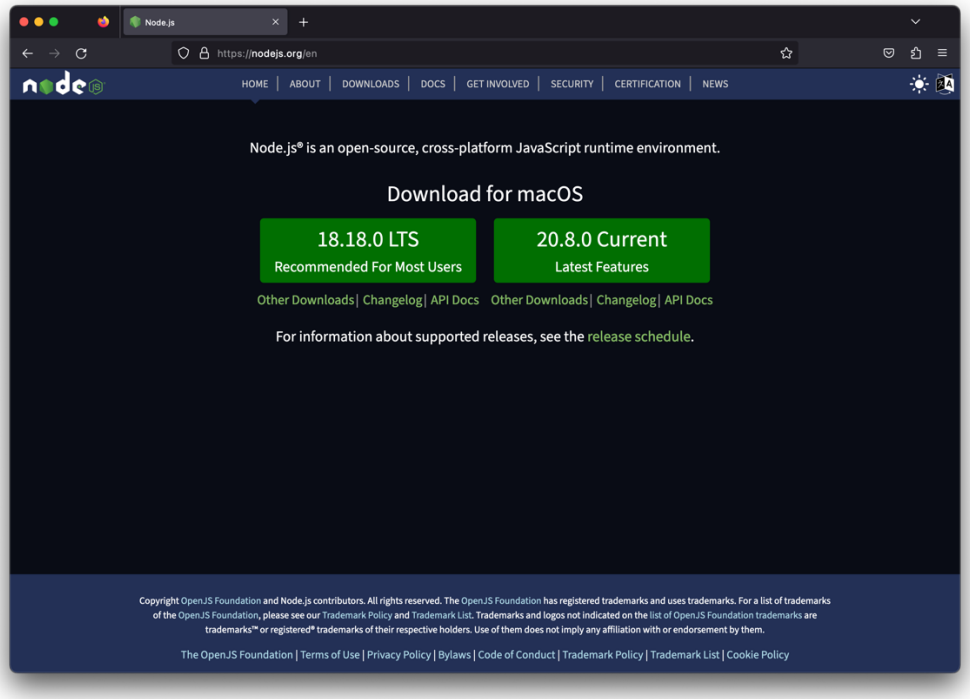

[vm] from: 0x5B3...eddC4 to: Storage.(constructor) value: 0 wei data: 0x608...20033 logs: 0 hash: 0xb53...a46ba
Debug
status
true Transaction mined and execution succeed
transaction hash
0xb536959b72406dfba10bf300eb448ddf886a9e943d5fd759e29637bf086a46ba
block hash
0x786ce370939a40eb1c1bfd6df3ae2050df4cdc96b524606958d8ed1a59c62cc3
block number
2
contract address
0xd8b934580fcE35a11B58C6D73aDeE468a2833fa8
from
0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to
Storage.(constructor)
gas
144529 gas
transaction cost
125701 gas
execution cost
67317 gas
input
0x608...20033
decoded input
{}
decoded output
-
logs
[]
val
0 wei

Week - 2

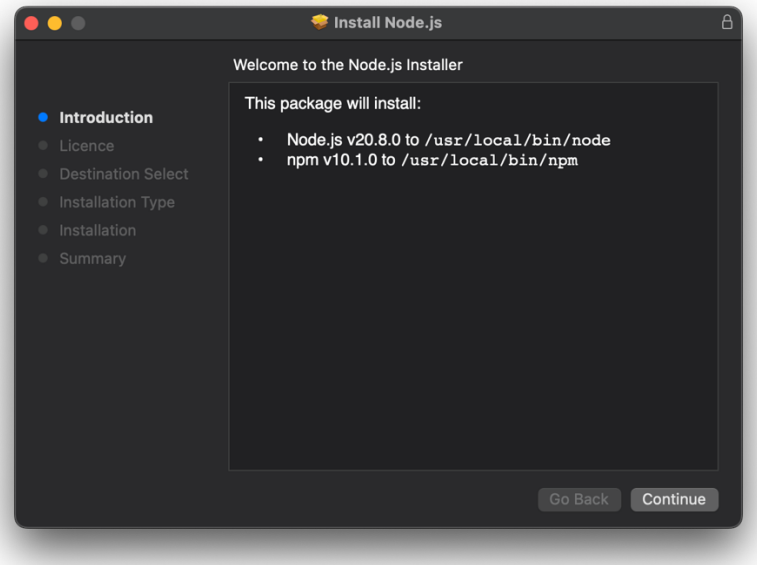
Aim: To demonstrate the installation of Node.JS and Web3.JS

Installation of Node.js:

- Open any browser and search for Node JS
- Click on the link of official website for Node JS or website - https://nodejs.org/en
- Now click on the version to be installed



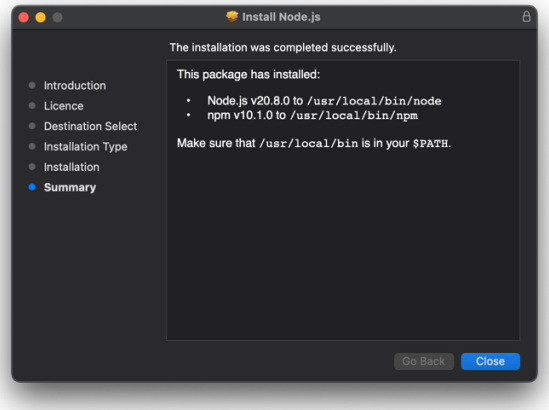
- Now open the downloaded package and unzip the package
- Now complete the installation by following the steps

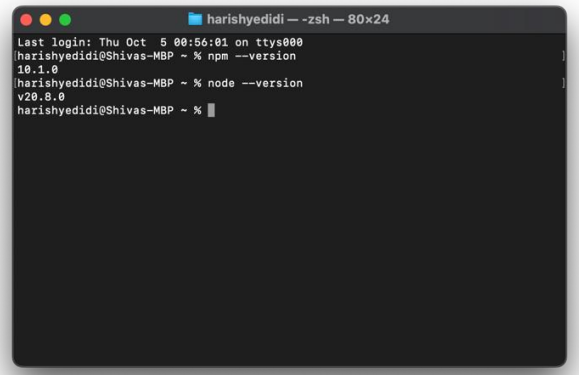


- Once installation is complete, to verify the installation of NodeJS, open Terminal and run following commands for verification

node -version

npm – version

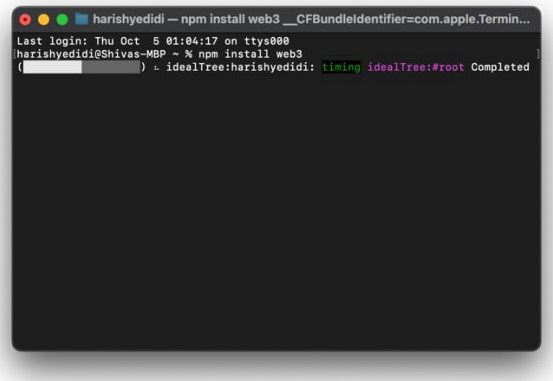


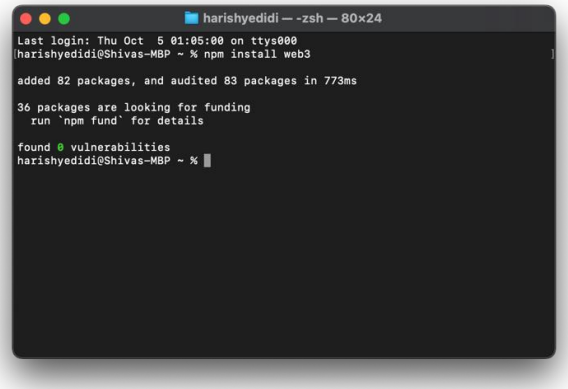


Installation of Web3.js:

- After the installation of NodeJS, open the terminal and run the following command

npm install web3





- The verification of the installation can be made by running a JS code with the module Web3 imported into it and check for errors

Week – 3

Aim: To demonstrate the transfer of Ether from one account to another using Web3.js

Program Code:

```
const { Web3 } = require('web3');
const web3 = new Web3
('https://mainnet.infura.io/v3/ef08dd1641894d488d8327d4f9f6552c');
// Sender account information
const senderAddress = '0xbbEEeD7a7d9513989a3d70d2340d3096CBf4d8E4';
const senderPrivateKey =
'786fa44e991ec4c46dbd7e52390c534e38fe4d98a7e67e049e217329bcc423c3';
// Recipient account information
const recipientAddress = '0x4fa2714c3EEDDd6E31a8F71e196441B3C7c899ed';
// Transfer Ether function
async function transferEther() {
        // Get the current gas price
        const gasPrice = await web3.eth.getGasPrice();
        // Estimate the gas required for the transaction
        const gasEstimate = await web3.eth.estimateGas({
                from: senderAddress,
                to: recipientAddress,
                value: web3.utils.toWei('0.001', 'ether'),
        });
        // Create the raw transaction object
        const rawTx = {
        from: senderAddress,
        to: recipientAddress,
        value: web3.utils.toHex(web3.utils.toWei('0.001', 'ether')),
        gasPrice: web3.utils.toHex(gasPrice),
        gasLimit: web3.utils.toHex(gasEstimate), };
        // Sign the transaction with the sender's private key
        const signedTx = await web3.eth.accounts.signTransaction(rawTx,
        senderPrivateKey);
        // Send the signed transaction to the Ethereum network
        const txReceipt = await
        web3.eth.sendSignedTransaction(signedTx.rawTransaction);
        console.log('Transaction successful.Transaction hash:',
        txReceipt.transactionHash);
}
// Call the transferEther function to initiate the transaction
```

transferEther();

Output:

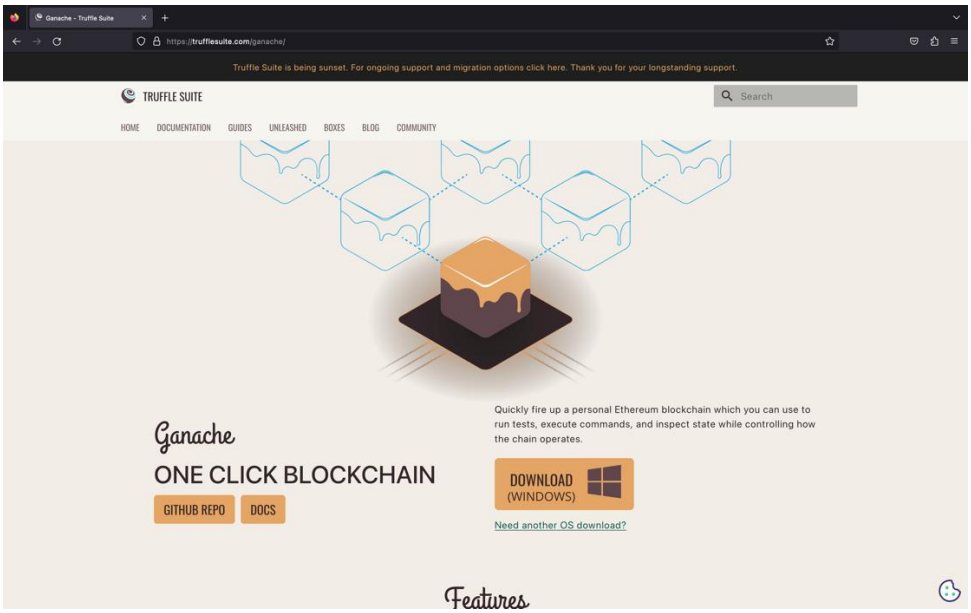
```
innerError: {
        code: -32000,
        message: 'err: insufficient funds for gas * price + value: address 0xbbEEeD7a7d9513989a3d70d2340d3096CBf4d8E4 have 0 want 1000000000000000 (supplied gas 30000000)'
    },
    code: 101,
    data: undefined,
    request: {
        jsonrpc: '2.0',
        id: '271186ca-1751-4408-8978-0dbb442dd068',
        method: 'eth_estimateGas',
        params: [
            {
                from: '0xbbEEeD7a7d9513989a3d70d2340d3096CBf4d8E4',
                to: '0x4fa2714c3EEDDd6E31a8F71e196441B3C7c899ed',
                value: '0x38d7ea4c68000'
            },
            'latest'
        ]
    }
}

Node.js v20.5.0
```

Week – 4

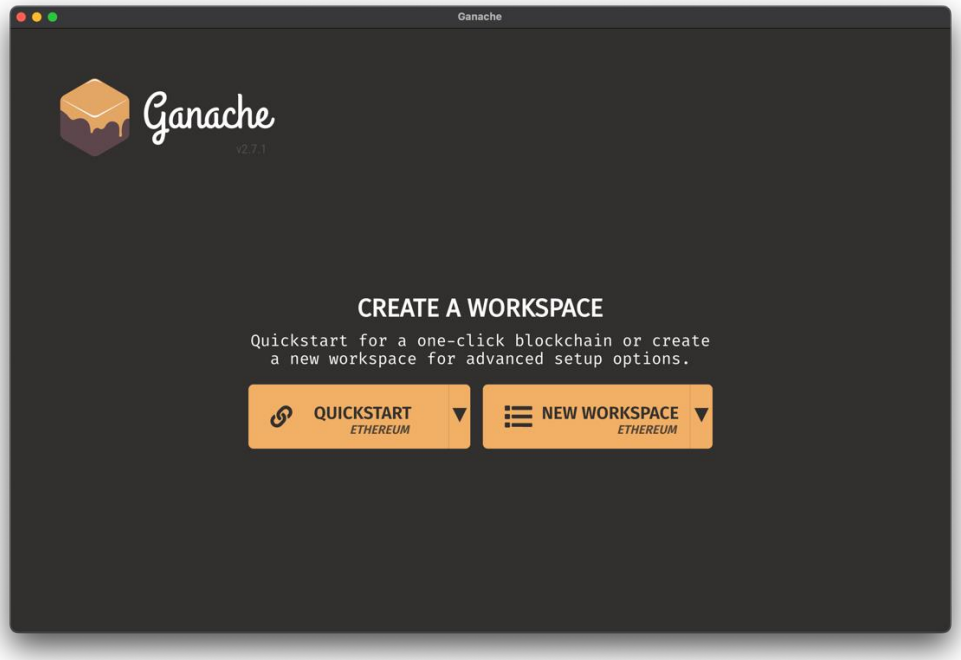
Aim: To demonstrate the creation of a Private Ethereum Blockchain Network

Installation of Ganache:

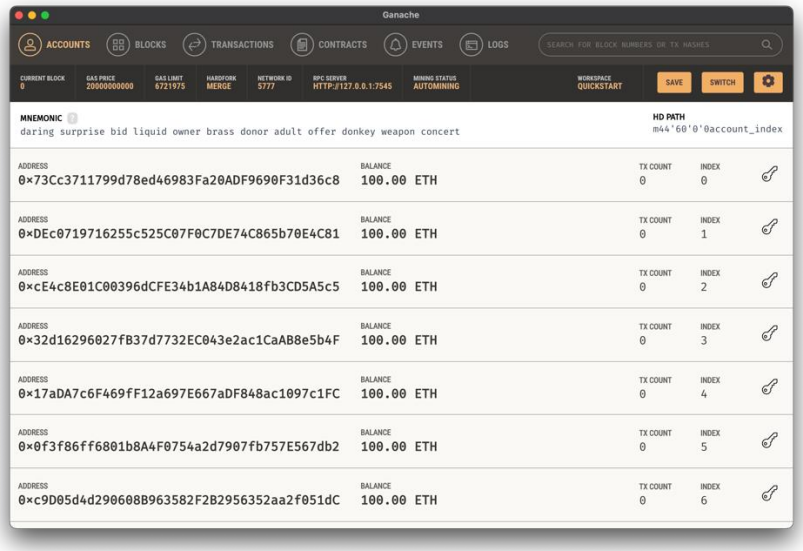
- To use a private Blockchain Network, we make use of an application named Ganache Truffle Suite
- To install, go to Google - search Ganache Truffle Suite and click on the first link
- Click on the install for Windows option, and complete the installation process



- After the installation process is complete, open the Ganche App



- Now click on the Quick Start option and then we land on the home page and we make use of the HTTP address present in the homepage



Program Code:

```
const { Web3 } = require('web3');
const web3 = new Web3
('https://goerli.infura.io/v3/ef08dd1641894d488d8327d4f9f6552c');
const ganacheUrl = 'HTTP://127.0.0.1:7545';
web3.eth.net.getId()
        .then((networkId) => { console.log('Connected to network ID:', networkId);
})
        .catch((error) => { console.log('Connected to network ID:', networkId); })
        .catch((error) => { console.error('Error connecting to Ganache:', error); });
const accountAddress = '0xbbEEeD7a7d9513989a3d70d2340d3096CBf4d8E4';
web3.eth.getBalance(accountAddress)
        .then((balance) => {
                console.log('Account balance:', web3.utils.fromWei(balance, 'ether'),
        'ETH');
        })
        .catch((error) => {
                console.error('Error fetching balance:', error);
        });
```

Output:

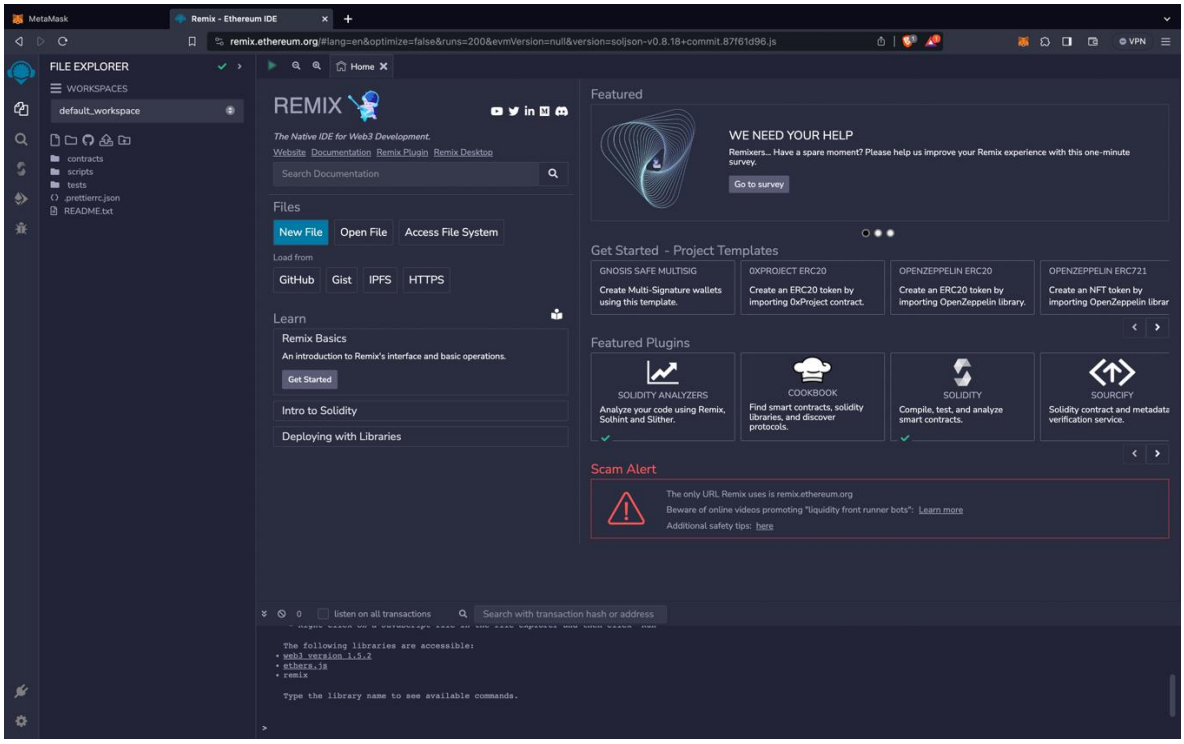
```
Connected to network ID: 5n
Account balance: 1.291794384975127259 ETH
```

Week - 5

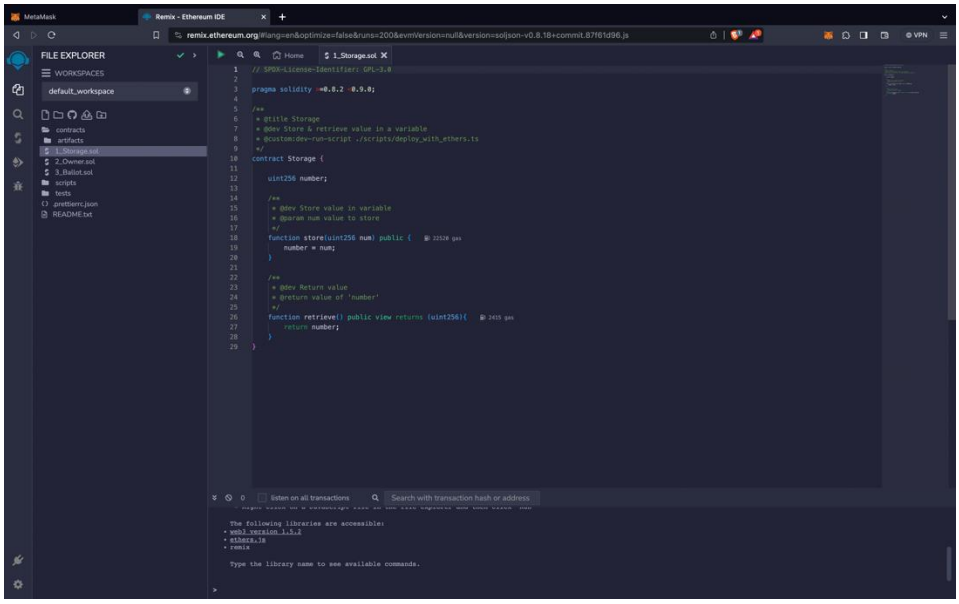
Aim: To demonstrate the interaction with Smart Contracts using Web3.js

Program Code:

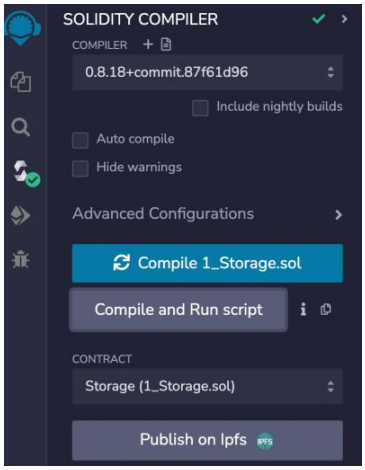
- Open any browser and search for Remix IDE
- Now click on the link or open https://remix.ethereum.org



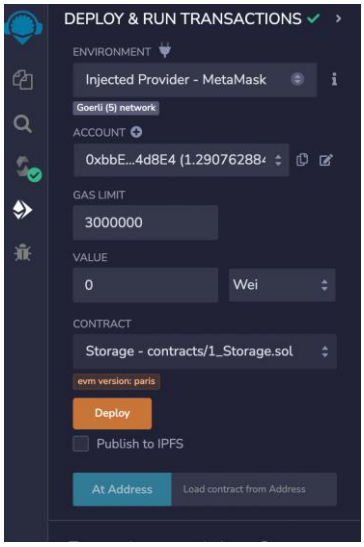
- Now in the home page, click on the contracts folder and open the file named Storage.sol



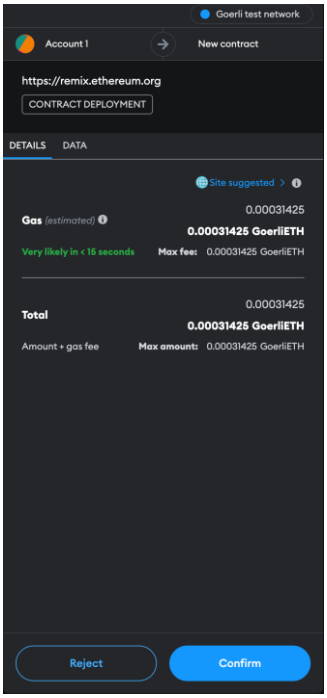
- Go to the 3rd option in the left-hand panel and click on compile & run script



- Once you have a tick mark, go to the 4th option in the left panel, change the environment to "Injected Provider – MetaMask" and deploy the code

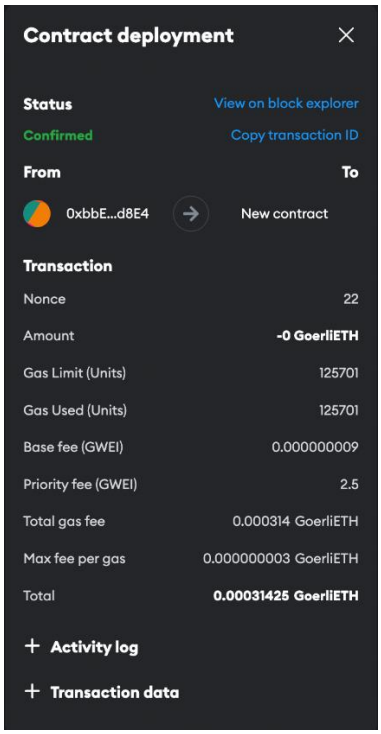


- Now you have a pop-up which is from the Metamask asking you whether you authorize the smart contract, click confirm

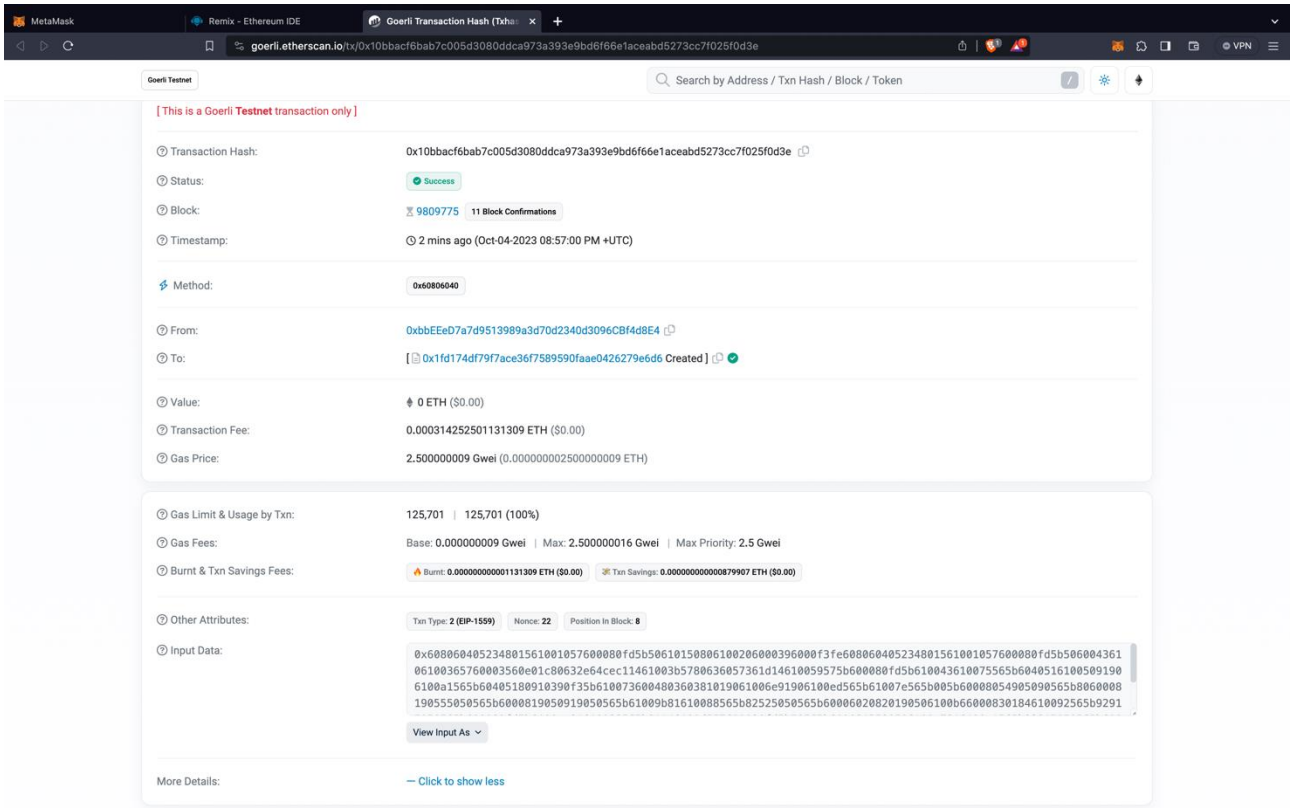


- Now go to the metamask extension and check for status of the contract

- Click on the contract transaction for the details of Contract



- For a detailed report on the transaction, click on “View on Block Explorer”


MetaMask
Remix - Ethereum IDE
Goerli Transaction Hash (Txha
goerli.etherscan.io/tx/0x10bbacf6bab7c005d3080ddca973a393e9bd6f66e1aceabd5273cc7f025f0d3e
Goerli Testnet
Search by Address / Txn Hash / Block / Token
[This is a Goerli Testnet transaction only]
Transaction Hash:
0x10bbacf6bab7c005d3080ddca973a393e9bd6f66e1aceabd5273cc7f025f0d3e
Status:
Success
Block:
9809775
11 Block Confirmations
Timestamp:
2 mins ago (Oct-04-2023 08:57:00 PM +UTC)
Method:
0x60806040
From:
0xbbEEeD7a7d9513989a3d70d2340d3096CBf4d8E4
To:
[0x1fd174df79f7ace36f7589590faae0426279e6d6 Created]
Value:
0 ETH ($0.00)
Transaction Fee:
0.000314252501131309 ETH ($0.00)
Gas Price:
2.500000009 Gwei (0.000000002500000009 ETH)
Gas Limit & Usage by Txn:
125,701 | 125,701 (100%)
Gas Fees:
Base: 0.000000009 Gwei | Max: 2.500000016 Gwei | Max Priority: 2.5 Gwei
Burnt & Txn Savings Fees:
Burnt: 0.000000000001131309 ETH ($0.00)
Txn Savings: 0.000000000000879907 ETH ($0.00)
Other Attributes:
Txn Type: 2 (EIP-1559)
Nonce: 22
Position In Block: 8
Input Data:
0x6080604052348015610010576000 80fd5b5061015080610020600039 6000f3fe6080604052348015610010576000 80fd5b506004361
0610036576000 3560e01c80632e64cec11461003b5780636057361d14610059575b600080fd5b610043610075565b6040516100509190
6100a1565b60405180910390f35b610073600480360381019061006e91906100ed565b61007e565b005b6000805490509056 5b8060008
190555050565b600081905091905056 5b61009b81610088565b82525050565b6000602082019050610 0b6000083018461009256 5b9291
View Input As
More Details:
— Click to show less

Week – 6

Aim: To demonstrate the basics of Rust with the help of several functions which constitute of the following:

- Displaying the statements
- Demonstration of data types
- Formatting of strings & numbers
- Computation of arithmetic operations with user inputs
- Bitwise & logical operators
- Swap 2 numbers without using temporary variables.

Program Code:

```
use std::io;

fn main() {
        println!("Week-6 (A)");
        println!("Hello, World!\n");
        println!("Week-6 (B)");
        week6b();
        println!("\nWeek-6 (C)");
        week6c();
        println!("\nWeek-6 (D)");
        println!("Enter any 2 numbers: ");
        let mut input = String::new();
        io::stdin() .read_line(&mut input) .expect("Failed to read line");
        let a: i32 = input.trim().parse().expect("Invalid input");
        let mut input = String::new();
        io::stdin() .read_line(&mut input) .expect("Failed to read line");
        let b: i32 = input.trim().parse().expect("Invalid input");
        week6d(a, b);
        println!("\nWeek-6 (E)");
        week6e(a, b);
        println!("\nWeek-6 (F)");
        week6f(a, b);
}
```

```
fn week6b() {
    let i: i32 = 42;
    let f: f64 = 3.14;
    let c: char = '😍';
    let b: bool = true;
    println!("Integer: {}", i);
    println!("Floating Point: {}", f);
    println!("Character: {}", c);
    println!("Boolean: {}", b);
}
fn week6c() {
    let name = "abcde";
    let age = 20; println!("My name is {} and I am {} years old.", name, age);
    println!("Formatted Age: {:05}", age);
    println!("Formatted PI: {:.2}", 3.14159);
}
fn week6d(x: i32, y: i32) {
    println!("Sum: {}", x + y);
    println!("Difference: {}", x - y);
    println!("Product: {}", x * y);
    println!("Quotient: {}", x / y);
    println!("Remainder: {}", x % y);
}
fn week6e(a: i32, b: i32) {
    println!("Bitwise AND: {}", a & b);
    println!("Bitwise OR: {}", a | b);
    println!("Bitwise XOR: {}", a ^ b);
    println!("Logical AND: {}", a > 0 && b > 0);
    println!("Logical OR: {}", a > 0 || b > 0);
```

```
        println!("Logical NOT: {}", !(a > 0));
}
fn week6f(mut x: i32, mut y: i32) {
        println!("Before Swapping, x = {}, y = {}", x, y);
        x = x + y;
        y = x - y;
        x = x - y;
        println!("After Swapping, x = {}, y = {}", x, y);
}
```

Output:

```
Week-6 (A)
Hello, World!
Week-6 (B)
Integer: 42
Floating Point: 3.14
Character: 😍
Boolean: true
Week-6 (C)
My name is abcde and I am 20 years old.
Formatted Age: 00020
Formatted PI: 3.14
Week-6 (D)
Enter any 2 numbers:
20
30
Sum: 50
Difference: -10
Product: 600
Quotient: 0
Remainder: 20
```

Week-6 (E)

Bitwise AND: 20

Bitwise OR: 30

Bitwise XOR: 10

Logical AND: true

Logical OR: true

Logical NOT: false

Week-6 (F)

Before Swapping, x = 20, y = 30

After Swapping, x = 30, y = 20

Week - 7

Aim: To write a program that demonstrates the Compound Data Types (Arrays & Tuples)

Program Code:

```
fn main() {
    arr();
    tup();
}

fn arr() {
    println!("Demonstration of Arrays: ");
    let a: [i32; 6] = [42, 57, 95, 21, 32, 85];
    // printing all the elements of the array
    println!("Array Elements: {:?}", a);
    println!("Accessing 5th element: ");
    println!("5th element = {}", a[4]);
}

fn tup() {
    println!("\nDemonstration of Tuples: ");
    let b: (&str, i32, bool) = ("abcde", 20, true);
    // printing all the elements of the tuple
    println!("Tuple Elements: {:?}", b);
    println!("Accessing 2nd Element: ");
    println!("2nd Element = {}", b.1);
}
```

Output:

Demonstration of Arrays:

Array Elements: [42, 57, 95, 21, 32, 85]

Accessing 5th element:

5th element = 32

Demonstration of Tuples:

Tuple Elements: ("abcde", 20, true)

Accessing 2nd Element:

2nd Element = 20

Week - 8

Aim: To write a program that demonstrates the working of loops and Conditional Loops

Program Code:

```
fn main() {
    // While loop
    let mut count = 0;
    while count < 5 {
        println!("While loop count: {}", count);
        count += 1;
    }
    // For loop
    for i in 1..=5 {
        println!("For loop count: {}", i);
    }
    // Loop with break
    let mut i = 0;
    loop {
        println!("Loop count: {}", i);
        i += 1;
        if i >= 5 { break; }
    }
    // Conditional loop - while let
    let mut optional_number = Some(5);
    while let Some(number) = optional_number {
        println!("Conditional loop: {}", number);
        optional_number = None;
    }
}
```

Output:

```
While loop count: 0
```

```
While loop count: 1
While loop count: 2
While loop count: 3
While loop count: 4
For loop count: 1
For loop count: 2
For loop count: 3
For loop count: 4
For loop count: 5
Loop count: 0
Loop count: 1
Loop count: 2
Loop count: 3
Loop count: 4
```

Week - 9

Aim: To write a program that demonstrates the working of loops and Conditional Loops

Program Code:

```
fn fun1(s: String){
        println!("This '{}'value is passed from Main Function\n",s);
}
fn fun2(t: i64) -> i64{
        t*t*t
}
fn main() {
        println!("Assigning Values to Variables");
        let x = 5;
        let y = x;
        println!("Assigned Values are: x = {}, y = {}\n",x,y);
        println!("Passing Values to Functions");
        let i = String::from("abcde");
        fun1(i);
        println!("Returning Values from Functions");
        let z = fun2(x);
        println!("The returned Value from Function is = {}",z);
}
```

Output:

```
Assigning Values to Variables
Assigned Values are: x = 5, y = 5
Passing Values to Functions
This 'abcde' value is passed from Main Function
Returning Values from Functions
The returned Value from Function is = 125
```

Week - 10

Aim: To write a program that demonstrates the generation of a random number

Program Code:

```
use std::io;
use rand::Rng;
fn main(){
    let mut input = String::new();
    io::stdin().read_line(&mut input).expect("Failed to read line");
    let x: i32 = input.trim().parse().expect("Invalid input");
    let mut input = String::new();
    io::stdin().read_line(&mut input).expect("Failed to read line");
    let y: i32 = input.trim().parse().expect("Invalid input");
    let r = rand::thread_rng().gen_range(x..=y);
    println!("Random Number generated between {} and {} is = {}",x,y,r);
}
```

Output:

```
20
30
Random Number generated between 20 and 30 is = 27
```

Week - 11

Aim: To write a program that compares the guessed number with a Secret generated number

Program Code:

```
use rand::Rng;
use std::cmp::Ordering;
use std::io;
fn main() {
    let secret_number = rand::thread_rng().gen_range(1..101);
    println!("Guess the secret number between 1 and 100");
    loop {
        let mut guess = String::new();
        io::stdin() .read_line(&mut guess) .expect("Failed to read line");
        let guess: i32 = guess.trim().parse().expect("Enter your Guess =
");
        match guess.cmp(&secret_number) {
            Ordering::Less => println!("Too Small!"),
            Ordering::Greater => println!("Too High!"),
            Ordering::Equal => {
                println!("You guessed the number");
                break;
            }
        }
    }
}
```

Output:

Guess the secret number between 1 and 100

40

Too Small!

80

Too Small!

90

Too High!

87

Too High!

85

Too Small!

86

You guessed the number

Week - 12

Aim: To write a program that demonstrates the concept of Borrowing in Rust.

Program Code:

```
fn main() {
let mut x = 10;
let y = &x;
println!("Original Value x = {}", x);
println!("Borrowed Value of x in variable y = {}", y);
}
```

Output:

```
Original Value x = 10
Borrowed Value of x in variable y = 10
```