Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA: 3.18)

JUNAID GIRKAR
60004190057
BE COMPS A2

BLOCKCHAIN TECHNOLOGIES

EXPERIMENT - 2

AIM: Create, Test and Deploy an Ethereum Smart Contract

THEORY:

A "smart contract" is simply a program that runs on the Ethereum blockchain. It's a collection of code (its functions) and data (its state) that resides at a specific address on the Ethereum blockchain.

Smart contracts are a type of Ethereum account. This means they have a balance and can be the target of transactions. However they're not controlled by a user, instead they are deployed to the network and run as programmed. User accounts can then interact with a smart contract by submitting transactions that execute a function defined on the smart contract. Smart contracts can define rules, like a regular contract, and automatically enforce them via the code. Smart contracts cannot be deleted by default, and interactions with them are irreversible.

With conventional contracts, a document outlines the terms of a relationship between two parties, which is enforceable by law. If one Party A violates the terms, Party B can take Party A to court for not complying with the agreement. A smart contract fortifies such agreements in code so the rules are automatically enforced without courts (or any third party) getting involved.

Some common ways of using smart contracts are:

- Multisignature accounts: Funds can only be spent when a required percentage of people agree.
- Encoding financial agreements: Manage agreements between users. Say, if one person buys insurance from an insurance company, the rules of when the insurance can be redeemed can be programmed into a smart contract.
- Agreements based on the outside world: Pull in data from the outside world (financial, political, or whatever) with the help of oracles.
- Provide third party: Similar to how a software library works, smart contracts can work with other smart contracts in a chain.

- Storage: Store information about an application, such as domain registration information or membership records. Storage in a blockchain like Ethereum is unique in that the data is immutable and can't be erased.

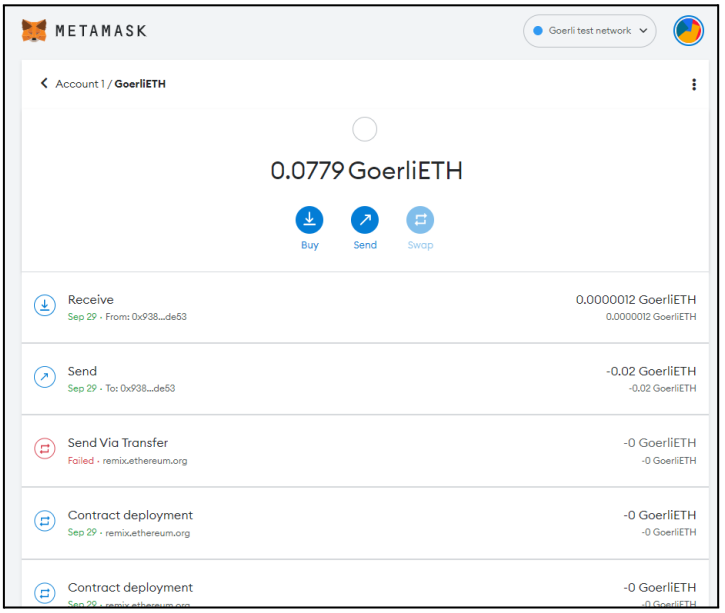
How is a smart contract set up?

- A developer can create a smart contract by writing a slab of code – spelling out the rules, such as that 10 ether can only be retrieved by Alice 10 years from now.
- The developer then pushes the smart contract to the Ethereum network, which is what enforces the contract – not allowing anyone to take the money unless they follow the exact rules in the code. Thousands of computers from around the world then all have a copy of this smart contract.

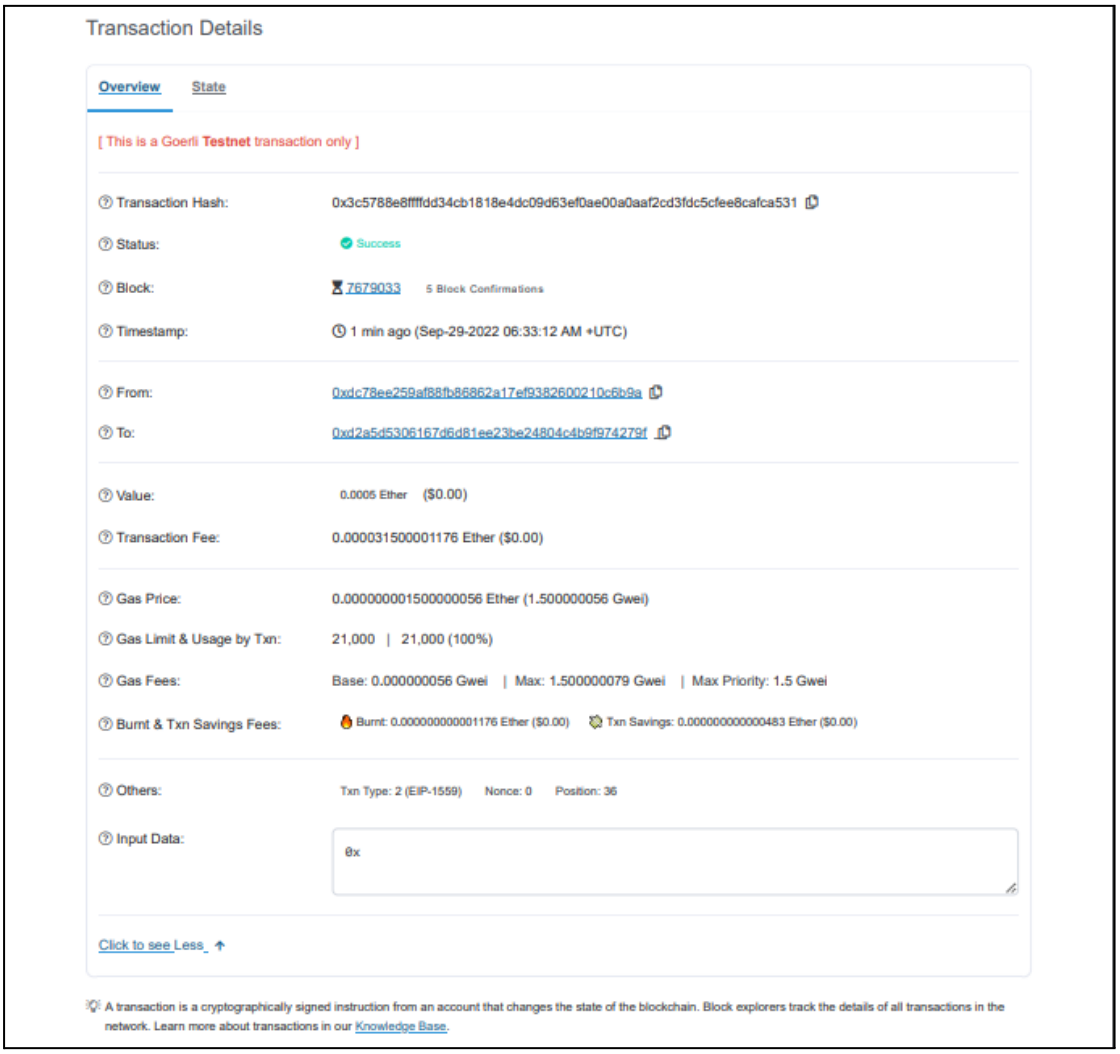
How do I use a smart contract?

- Anyone can use smart contracts if they have Ethereum's native token ether, which can be bought on cryptocurrency exchanges.
- Ethereum apps will usually provide instructions for how to use their specific app and underlying smart contracts. A common method is to use an Ethereum wallet tool, such as Metamask, to send the ether.
- Users can use smart contracts for a range of use cases. Users can publish uncensorable posts to microblogging apps or lend out money without an intermediary, using a variety of Ethereum apps.

Creation of a Metamask Wallet and using free Goerli Ethers from the faucet:



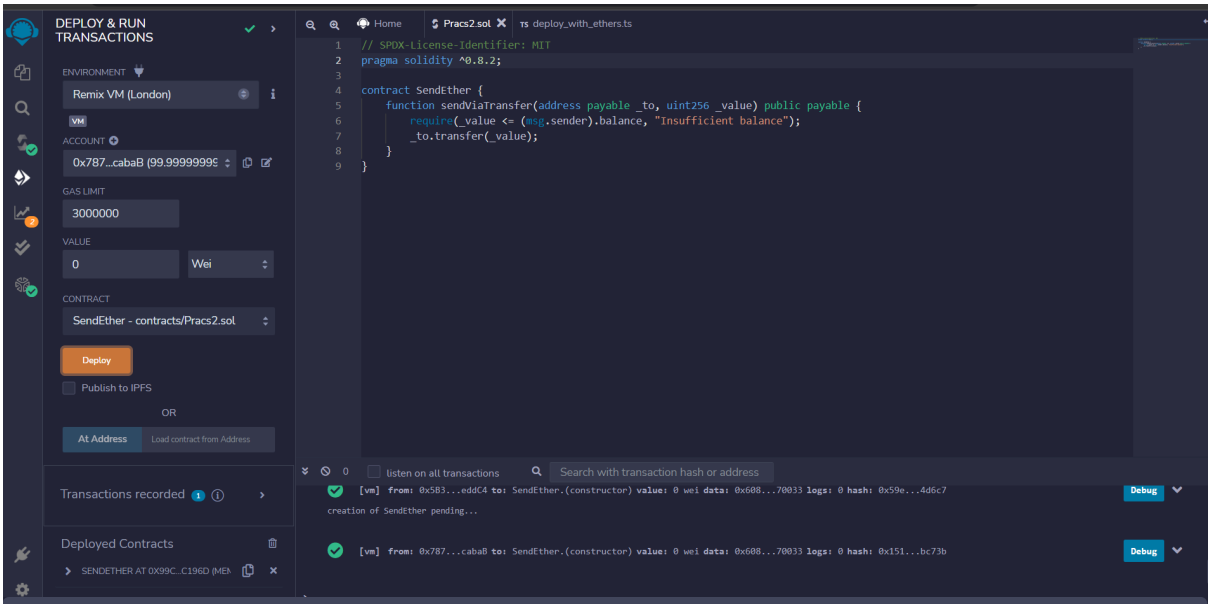
Doing a transaction of Goerli Ethers to another account id:



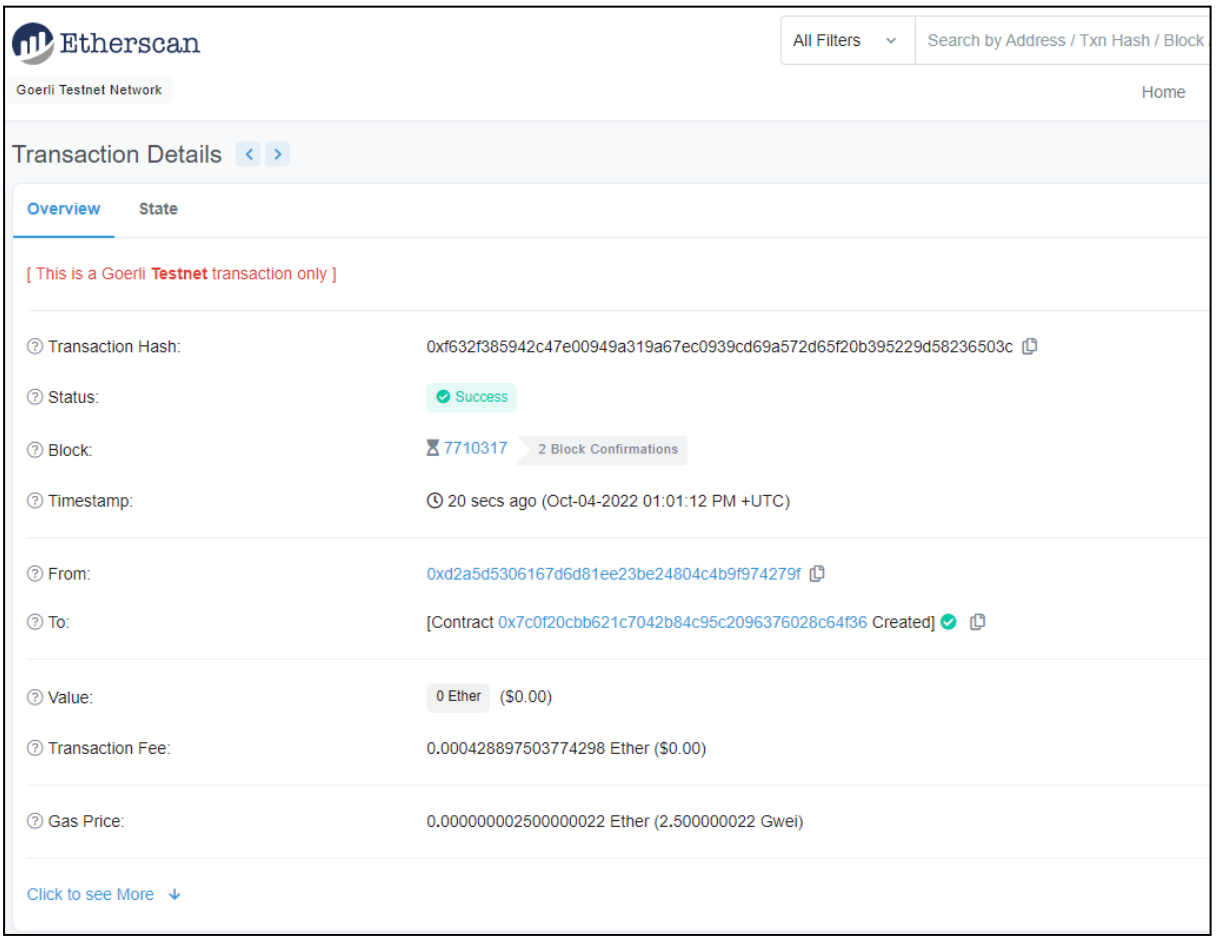
Creating a Smart Contract:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.2;

contract SendEther {
    function sendViaTransfer(address payable _to) public payable {
        require(msg.value <= (msg.sender).balance, "Insufficient
balance");
        _to.transfer(msg.value);
    }
}
```

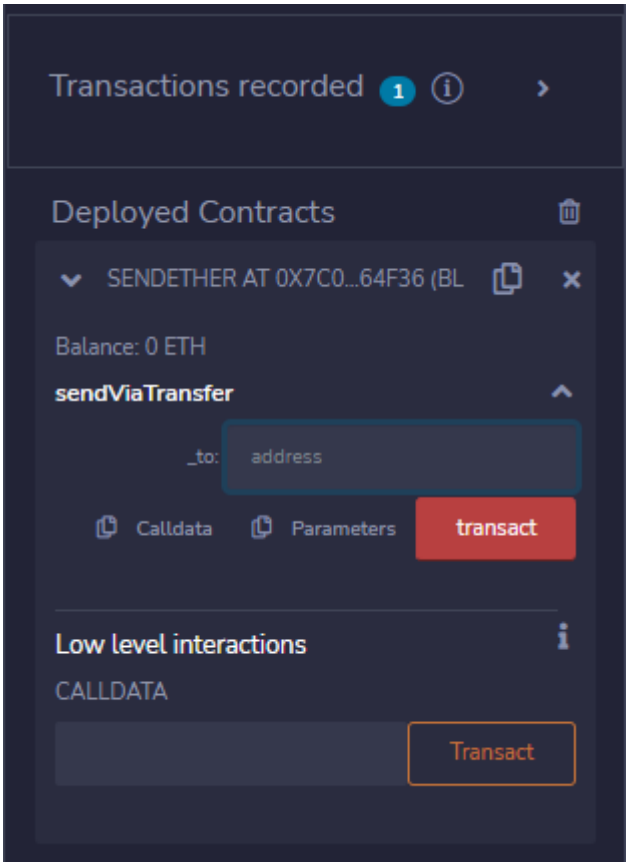
Deploying the Smart Contract:



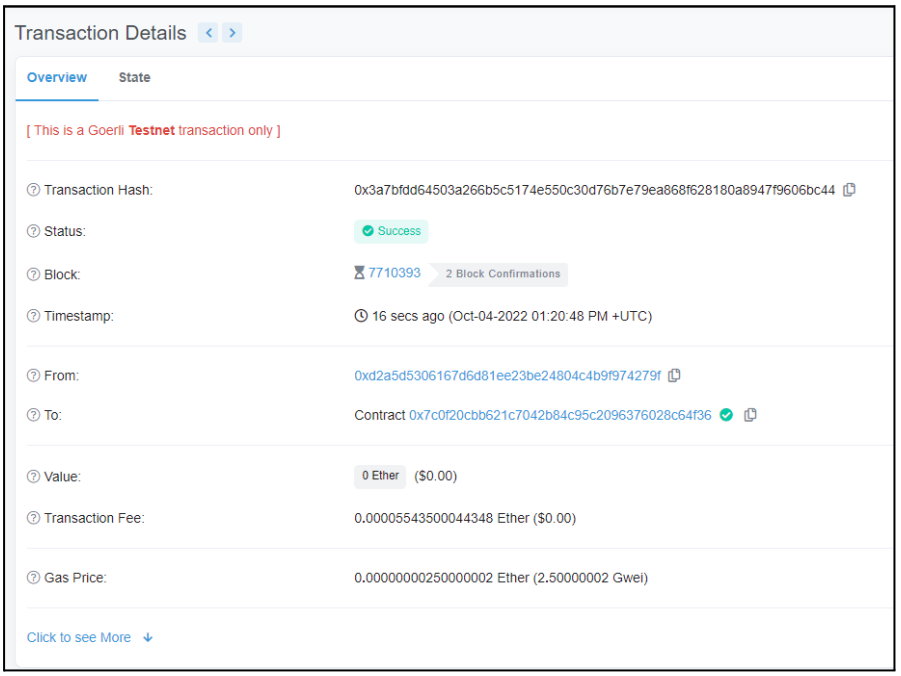
Transaction Fees for deploying the Smart Contract:



Using the Smart Contract:



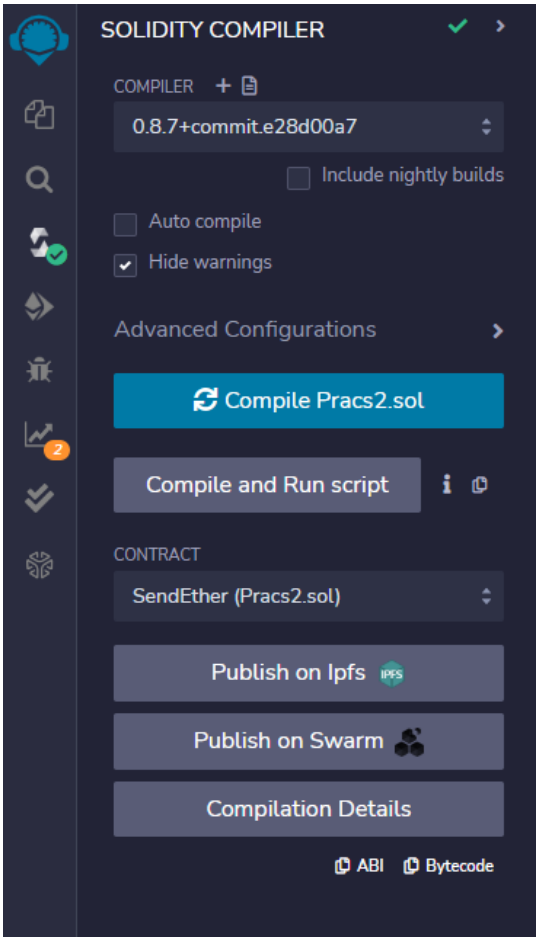
Ethers sent via Smart Contract:



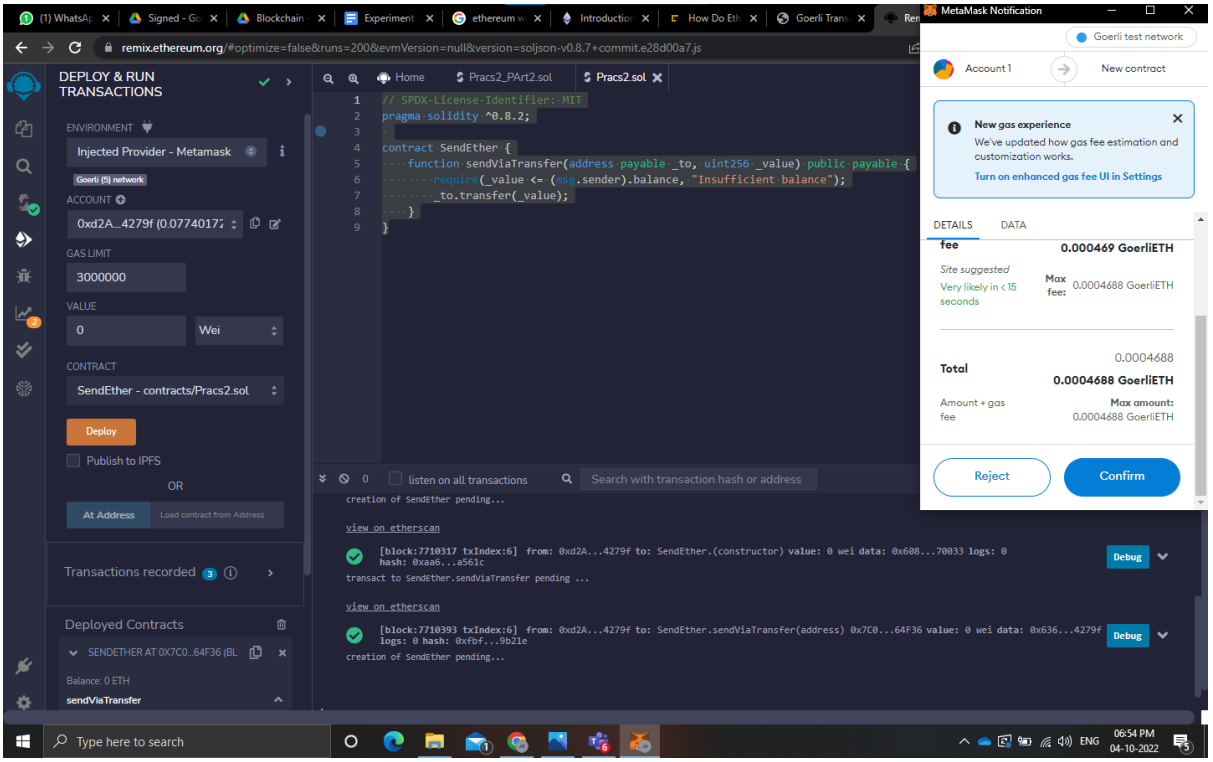
Creating a Second Smart Contract that can take value input:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.2;

contract SendEther {
    function sendViaTransfer(address payable _to, uint256 _value) public
payable {
        require(_value <= (msg.sender).balance, "Insufficient balance");
        _to.transfer(_value);
    }
}
```

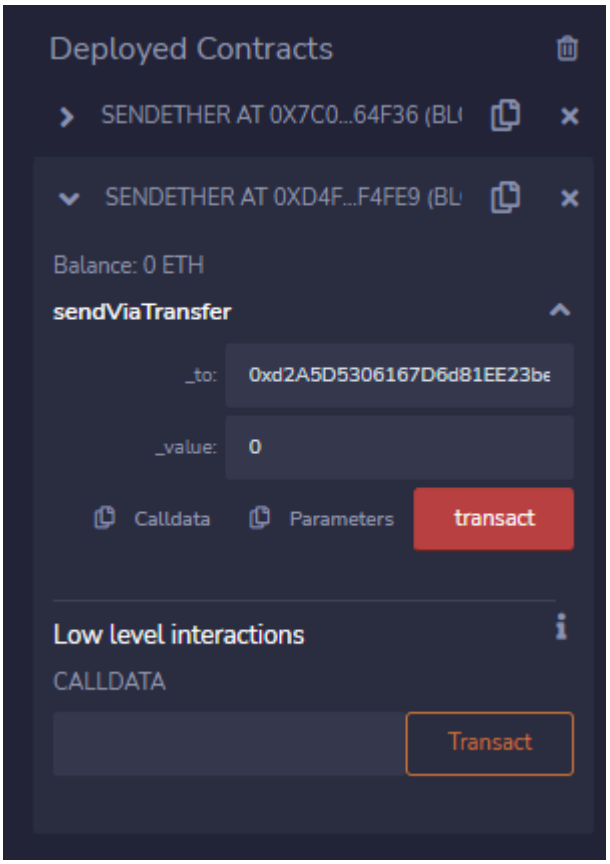
Compiling the Smart Contract on EVM (Ethereum Virtual Machine)



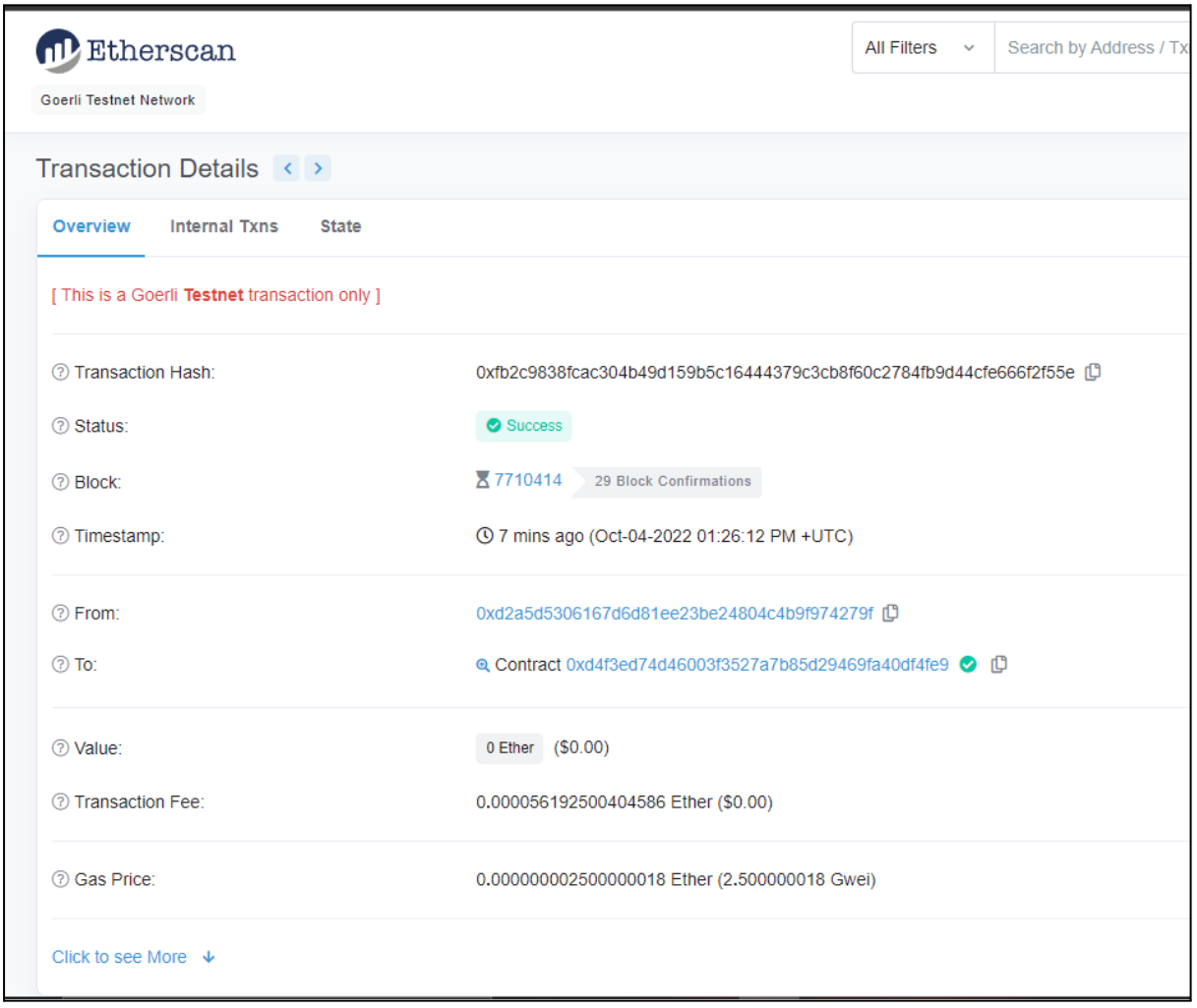
Deploying the Smart Contract:



Taking Transaction Value as input:



Transaction:



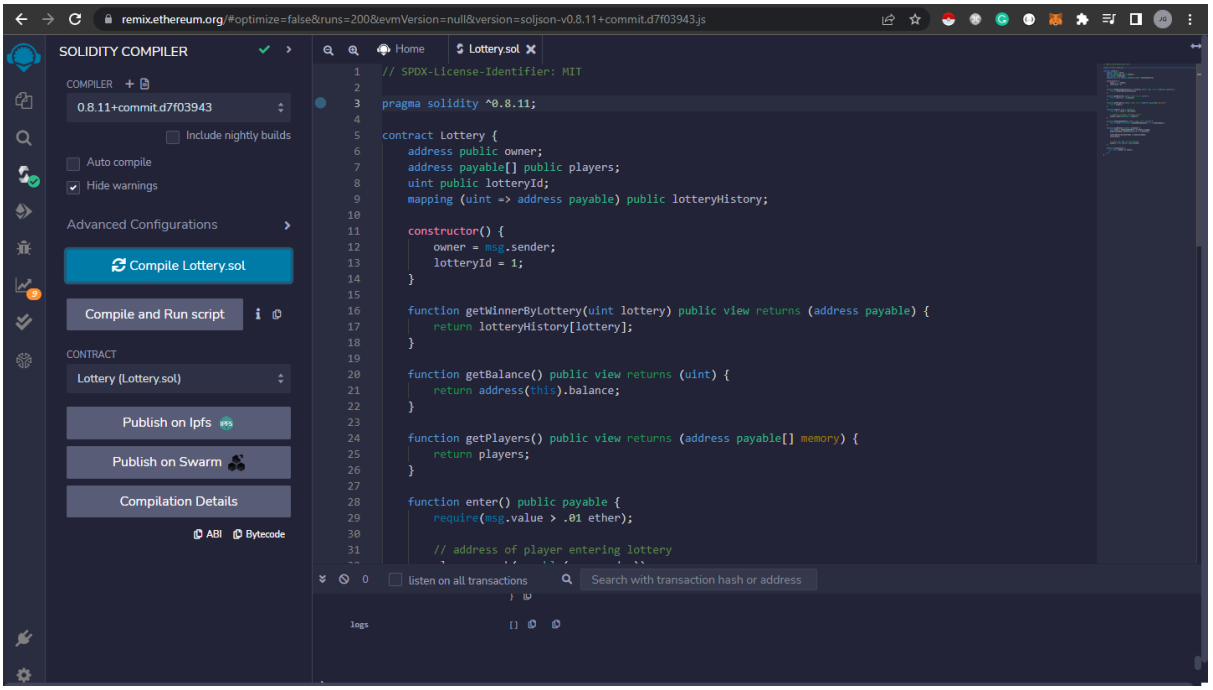
Lottery.sol

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

pragma solidity ^0.8.11;

contract Lottery {
    address public owner;
    address payable[] public players;
    uint public lotteryId;
    mapping (uint => address payable) public lotteryHistory;

    constructor() {
        owner = msg.sender;
        lotteryId = 1;
```

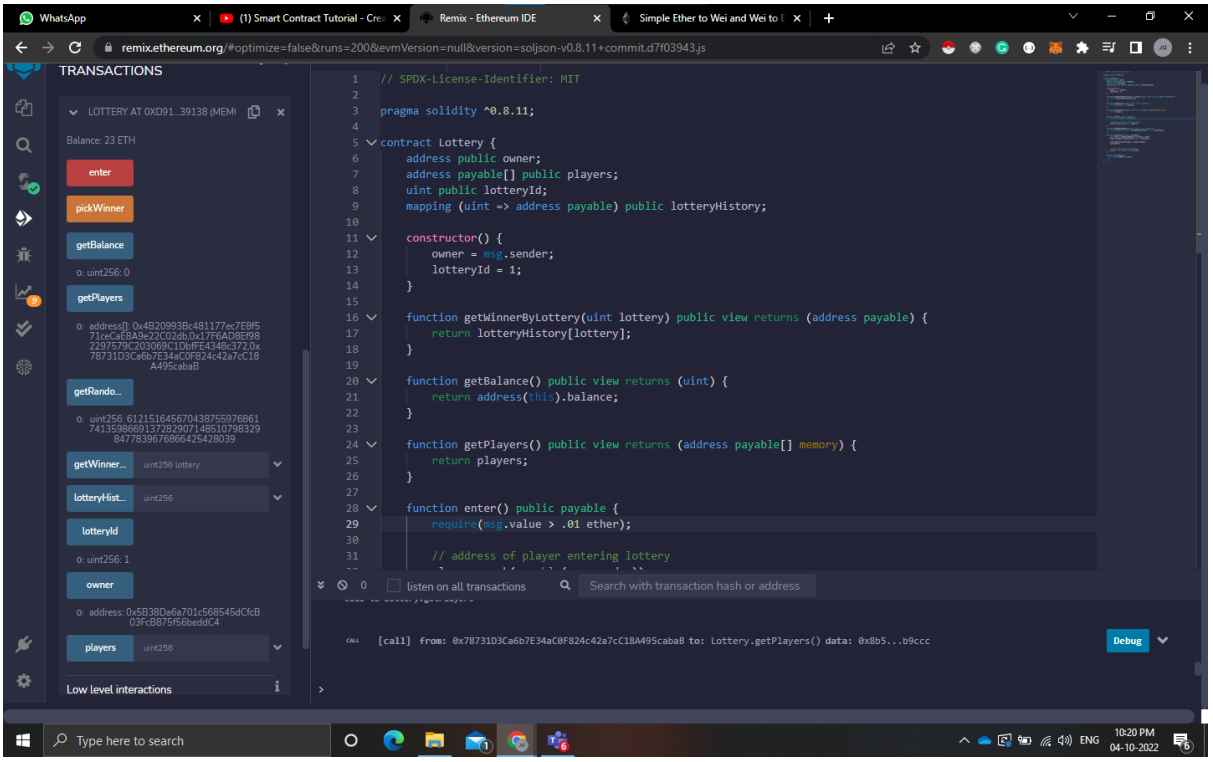
```
    }

    function getWinnerByLottery(uint lottery) public view returns
(address payable) {
        return lotteryHistory[lottery];
    }

    function getBalance() public view returns (uint) {
        return address(this).balance;
    }

    function getPlayers() public view returns (address payable[] memory)
{
        return players;
    }

    function enter() public payable {
        require(msg.value > .01 ether);

        // address of player entering lottery
        players.push(payable(msg.sender));
    }

    function getRandomNumber() public view returns (uint) {
        return uint(keccak256(abi.encodePacked(owner,
block.timestamp)));
    }

    function pickWinner() public onlyowner {
        uint index = getRandomNumber() % players.length;
        players[index].transfer(address(this).balance);

        lotteryHistory[lotteryId] = players[index];
        lotteryId++;


        // reset the state of the contract
        players = new address payable[](0);
    }

    modifier onlyowner() {
      require(msg.sender == owner);
      _;
    }
```

```
}
```

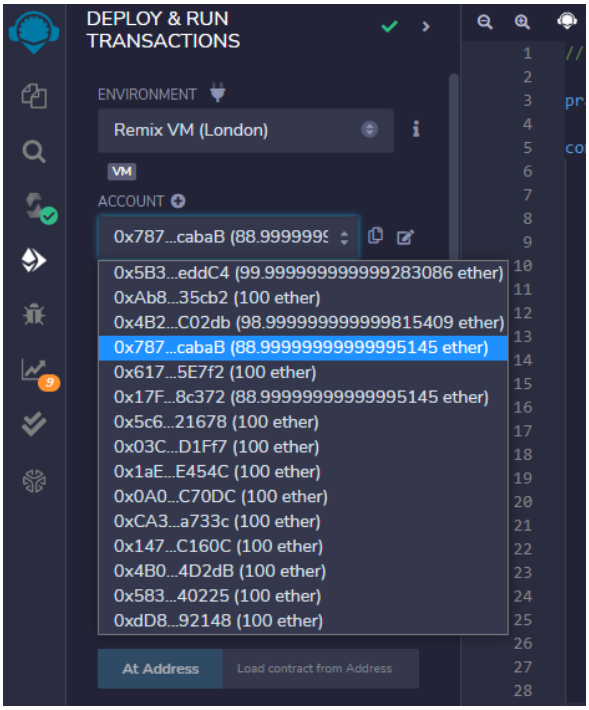
Compile the code:



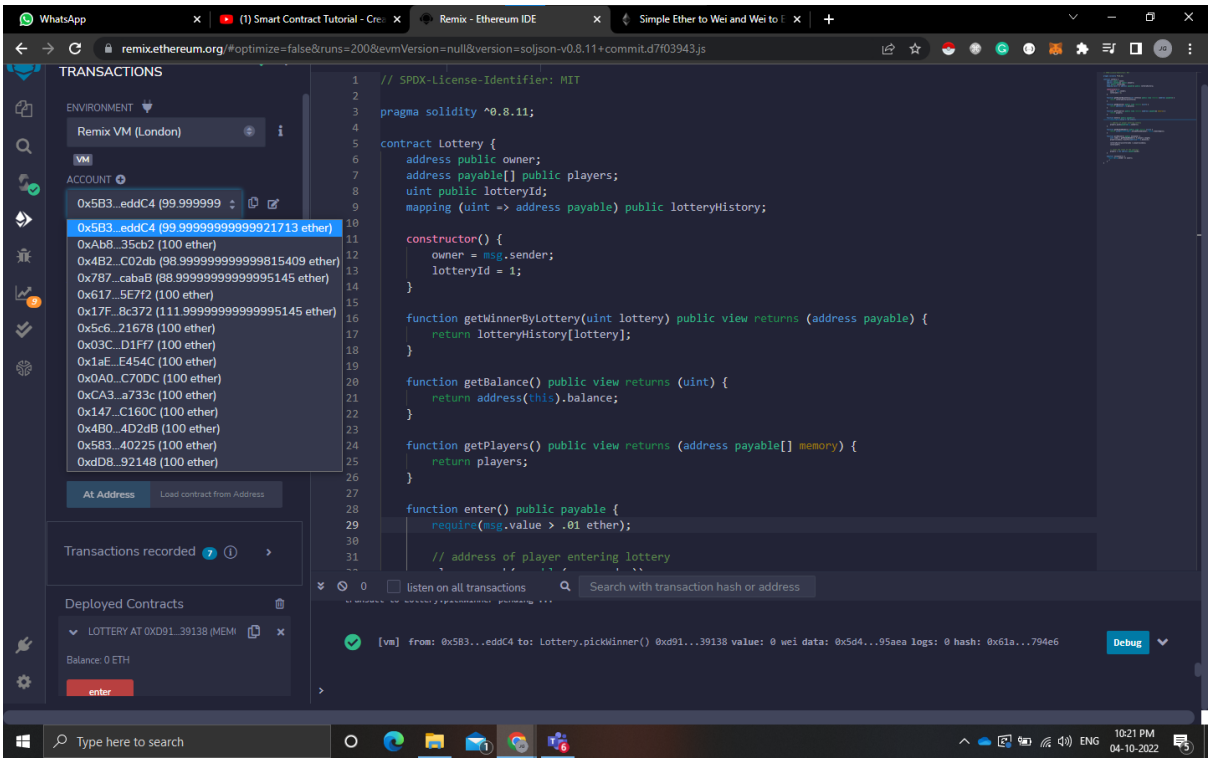
After Deploying the code and entering the lottery through 3 accounts:



Wallet balance of players after entering the lottery:



After picking a winner of the lottery (Account 6 won):



CONCLUSION:

In this experiment, we learnt about cryptocurrency wallets like Metamask and test ethers like Goreli Ethers. We also learnt about smart contracts and implemented them using Solidity in Remid IDE. We did multiple transactions using the metamask wallet and smart contracts and implemented a randomised lottery system.