BITS, Pilani -Pilani Campus
CS F452 (Blockchain Technology)
Laboratory 4 (Week-4)

AIM: To deploy and test Smart Contracts in Remix IDE using Ganache Server and MetaMask Wallet

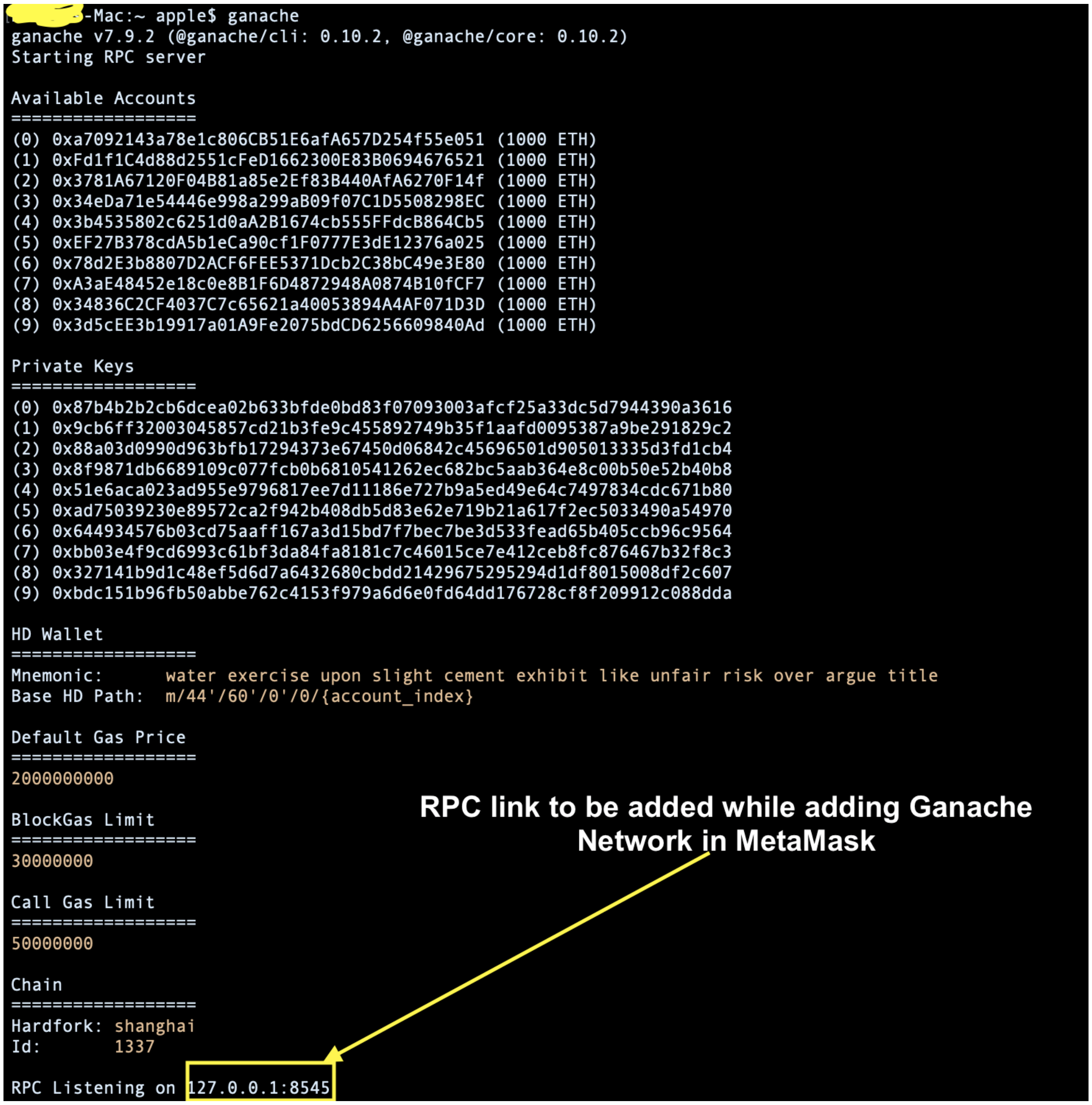
Objective: In this laboratory exercise, we will learn how to deploy a smart contract using the Remix IDE and interact with it using Ganache as a local Ethereum blockchain. We will deploy a simple banking smart contract that allows users to deposit, withdraw, and check balances.

Smart Contract: We will be using the SimpleBanking.sol file as the contract to deploy and test on remix.

Steps for Deployment and Testing:

Step 1: **Install and Set Up Ganache:**

- You can either use the Ganache GUI or Ganache CLI version , we will work with the CLI version in this lab.
- In lab-1 we discussed that by **running this command:"npm install -g ganache-cli"** we can install ganache-cli in our system globally.
- Then by running **"ganache"** we will successfully setup ganache.

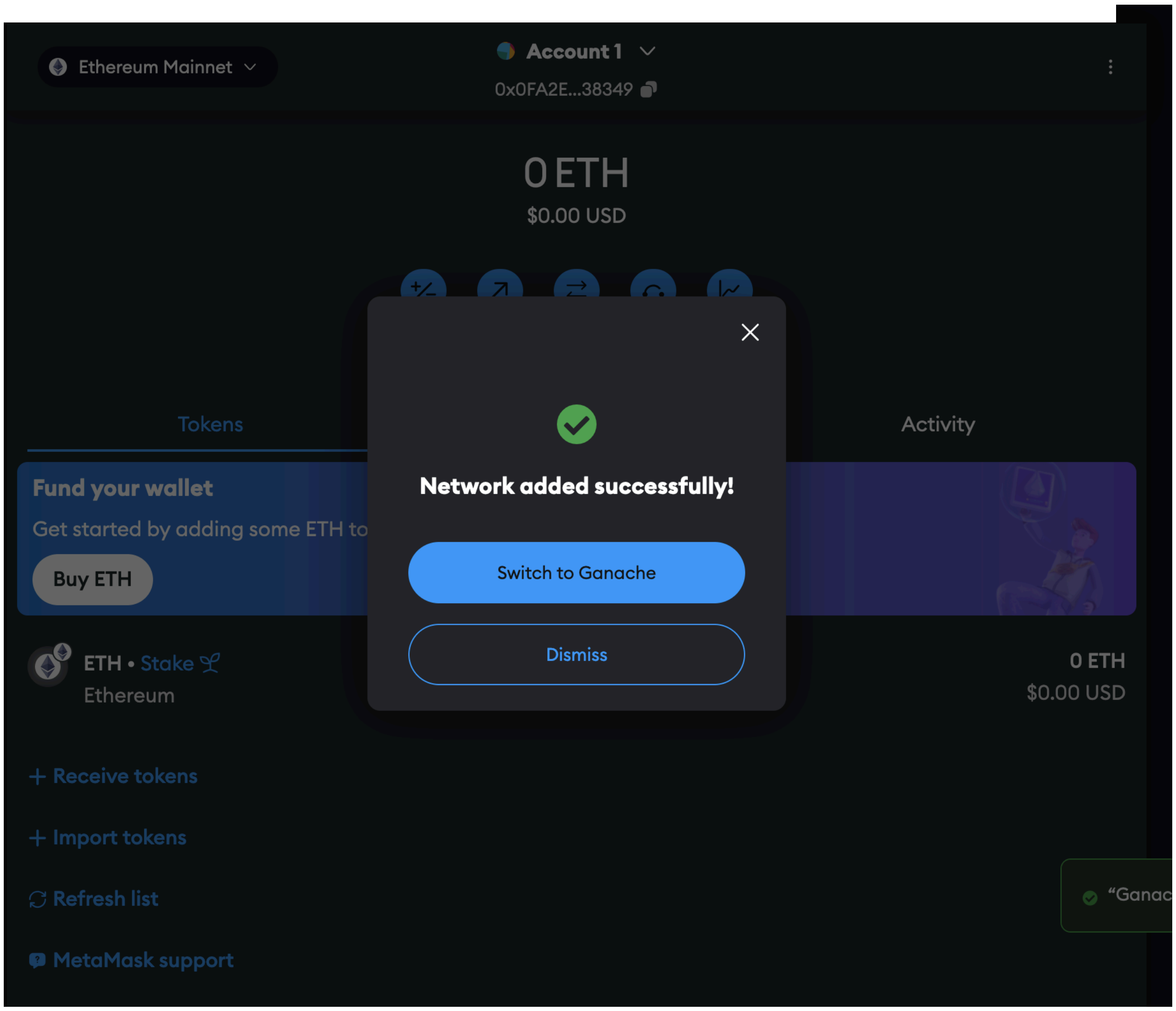


```
[redacted]-Mac:~ apple$ ganache
ganache v7.9.2 (@ganache/cli: 0.10.2, @ganache/core: 0.10.2)
Starting RPC server

Available Accounts
==================
(0) 0xa7092143a78e1c806CB51E6afA657D254f55e051 (1000 ETH)
(1) 0xFd1f1C4d88d2551cFeD1662300E83B0694676521 (1000 ETH)
(2) 0x3781A67120F04B81a85e2Ef83B440AfA6270F14f (1000 ETH)
(3) 0x34eDa71e54446e998a299aB09f07C1D5508298EC (1000 ETH)
(4) 0x3b4535802c6251d0aA2B1674cb555FFdcB864Cb5 (1000 ETH)
(5) 0xEF27B378cdA5b1eCa90cf1F0777E3dE12376a025 (1000 ETH)
(6) 0x78d2E3b8807D2ACF6FEE5371Dcb2C38bC49e3E80 (1000 ETH)
(7) 0xA3aE48452e18c0e8B1F6D4872948A0874B10fCF7 (1000 ETH)
(8) 0x34836C2CF4037C7c65621a40053894A4AF071D3D (1000 ETH)
(9) 0x3d5cEE3b19917a01A9Fe2075bdCD6256609840Ad (1000 ETH)

Private Keys
==================
(0) 0x87b4b2b2cb6dcea02b633bfde0bd83f07093003afcf25a33dc5d7944390a3616
(1) 0x9cb6ff32003045857cd21b3fe9c455892749b35f1aafd0095387a9be291829c2
(2) 0x88a03d0990d963bfb17294373e67450d06842c45696501d905013335d3fd1cb4
(3) 0x8f9871db6689109c077fcb0b6810541262ec682bc5aab364e8c00b50e52b40b8
(4) 0x51e6aca023ad955e9796817ee7d11186e727b9a5ed49e64c7497834cdc671b80
(5) 0xad75039230e89572ca2f942b408db5d83e62e719b21a617f2ec5033490a54970
(6) 0x644934576b03cd75aaff167a3d15bd7f7bec7be3d533fead65b405ccb96c9564
(7) 0xbb03e4f9cd6993c61bf3da84fa8181c7c46015ce7e412ceb8fc876467b32f8c3
(8) 0x327141b9d1c48ef5d6d7a6432680cbdd21429675295294d1df8015008df2c607
(9) 0xbdc151b96fb50abbe762c4153f979a6d6e0fd64dd176728cf8f209912c088dda

HD Wallet
==================
Mnemonic:      water exercise upon slight cement exhibit like unfair risk over argue title
Base HD Path:  m/44'/60'/0'/0/{account_index}

Default Gas Price
==================
2000000000

BlockGas Limit
==================
30000000

Call Gas Limit
==================
50000000

Chain
==================
Hardfork: shanghai
Id:       1337

RPC Listening on 127.0.0.1:8545
```

RPC link to be added while adding Ganache Network in MetaMask

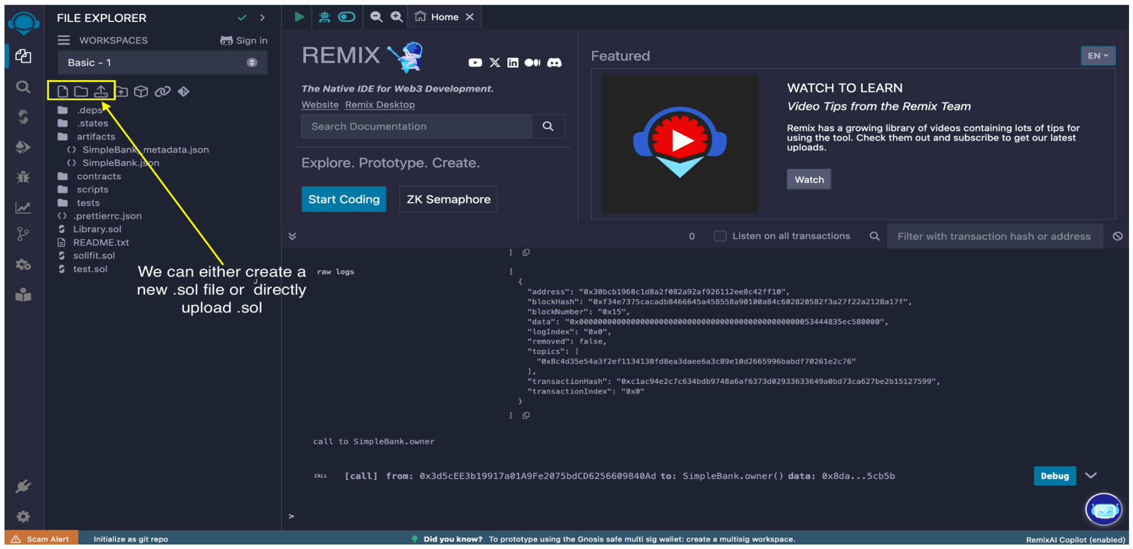
Step 2:**MetaMask connection with Ganache**

- After we have selected the ganache network in MetaMask , we can paste the RPC link as displayed in our terminal/Cmd.(SetUp of MetaMask discussed in Lab-1)
- Click on **Swtich to Ganache**



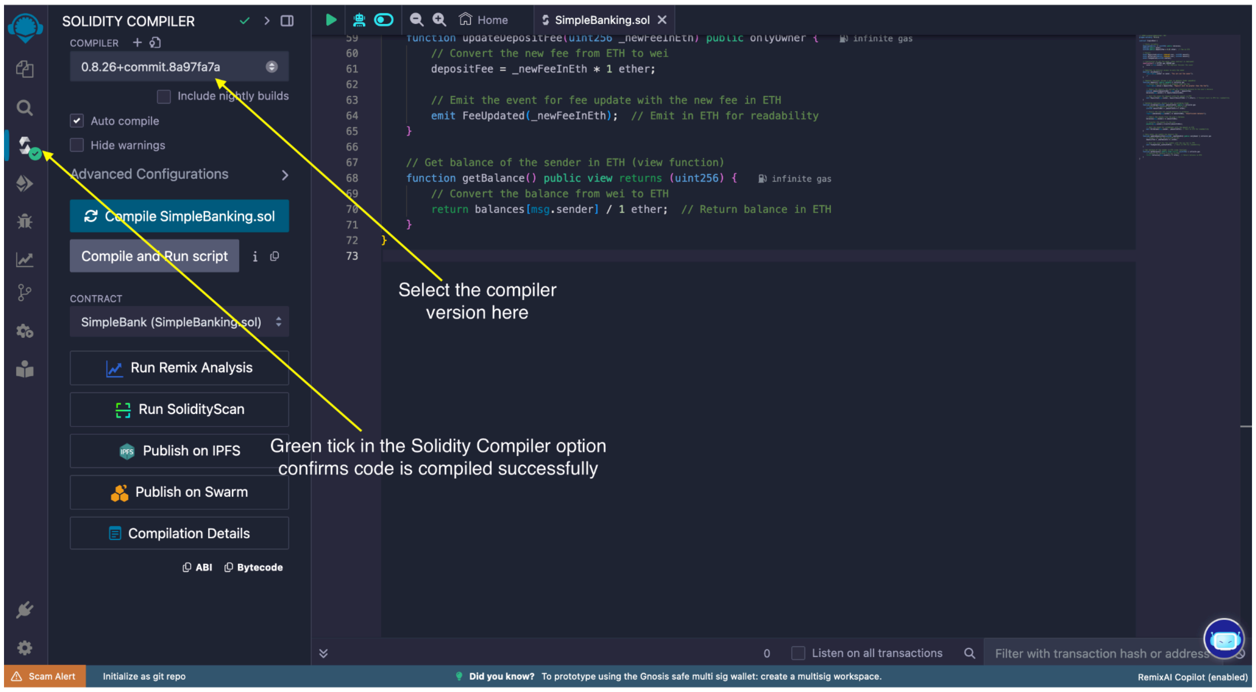
Step3:**Deploy the Smart Contract**

- Once you have created a .sol file in the Remix IDE you should compile the file using solidity Compiler options in the leftsidebar.


FILE EXPLORER
WORKSPACES
Sign in
Basic - 1
.deps
.states
artifacts
SimpleBank_metadata.json
SimpleBank.json
contracts
scripts
tests
.prettierrc.json
Library.sol
README.txt
solifit.sol
test.sol
Home
REMIX
The Native IDE for Web3 Development.
Website
Remix Desktop
Search Documentation
Explore. Prototype. Create.
Start Coding
ZK Semaphore
Featured
EN
WATCH TO LEARN
Video Tips from the Remix Team
Remix has a growing library of videos containing lots of tips for using the tool. Check them out and subscribe to get our latest uploads.
Watch
0
Listen on all transactions
Filter with transaction hash or address
raw logs
Debug
Scam Alert
Initialize as git repo
Did you know? To prototype using the Gnosis safe multi sig wallet: create a multisig workspace.
RemixAI Copilot (enabled)

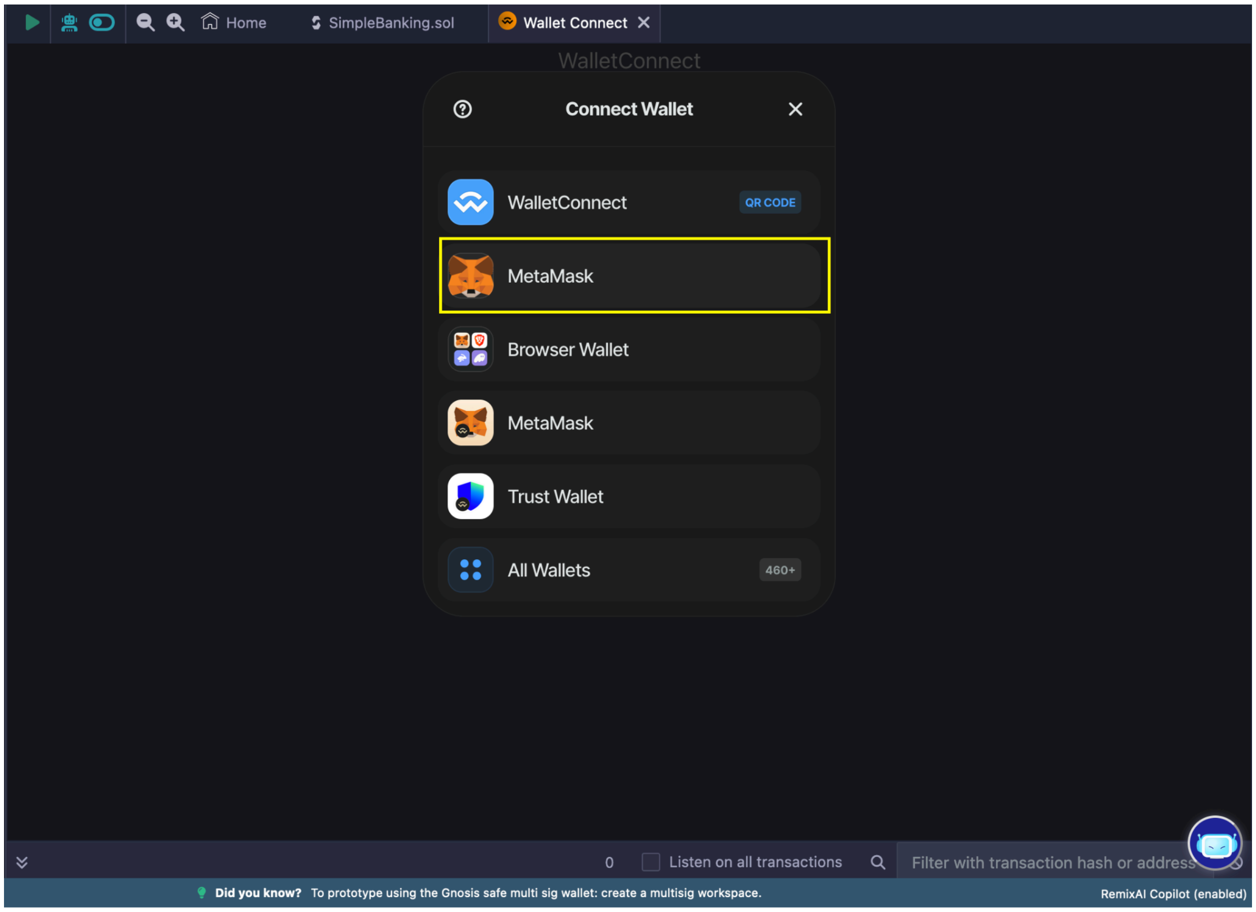
We can either create a new .sol file or directly upload .sol

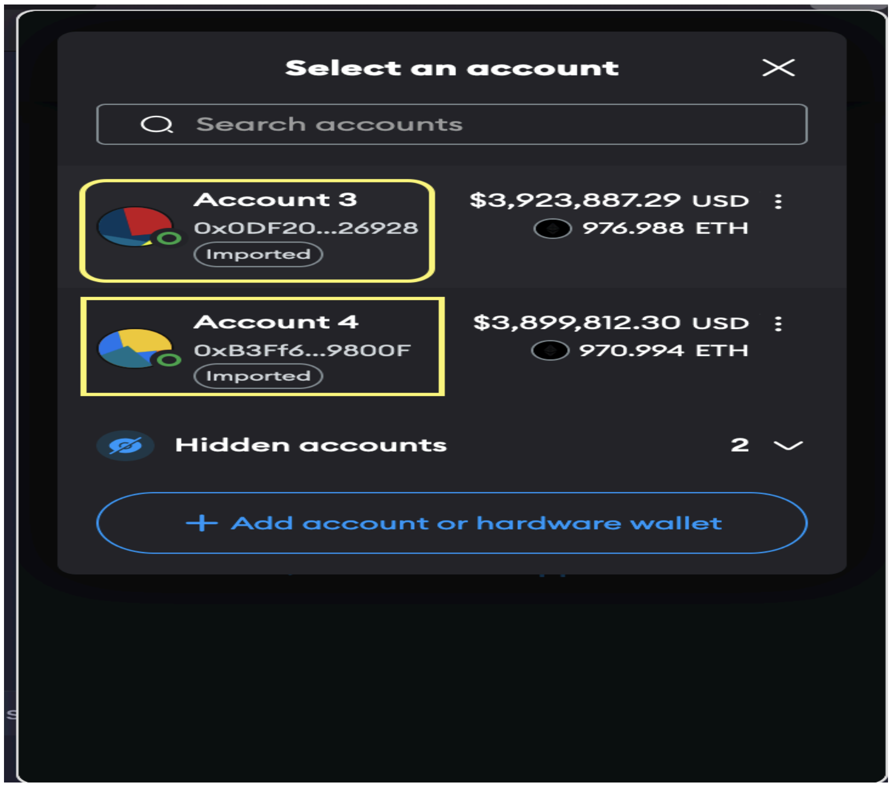
```
]
[
  {
    "address": "0x30bcb1968c1d8a2f082a92af926112ee8c42ff10",
    "blockHash": "0xf34e7375cacadb8466645a458558a90100a84c602820582f3a27f22a2128a17f",
    "blockNumber": "0x15",
    "data": "0x000000000000000000000000000000000000000000000000053444835ec580000",
    "logIndex": "0x0",
    "removed": false,
    "topics": [
      "0x8c4d35e54a3f2ef1134138fd8ea3daee6a3c89e10d2665996babdf70261e2c76"
    ],
    "transactionHash": "0xc1ac94e2c7c634bdb9748a6af6373d02933633649a0bd73ca627be2b15127599",
    "transactionIndex": "0x0"
  }
]

call to SimpleBank.owner

CALL  [call] from: 0x3d5cEE3b19917a01A9Fe2075bdCD6256609840Ad to: SimpleBank.owner() data: 0x8da...5cb5b

>
```


SOLIDITY COMPILER
COMPILER
0.8.26+commit.8a97fa7a
Include nightly builds
Auto compile
Hide warnings
Advanced Configurations
Compile SimpleBanking.sol
Compile and Run script
CONTRACT
SimpleBank (SimpleBanking.sol)
Run Remix Analysis
Run SolidityScan
Publish on IPFS
Publish on Swarm
Compilation Details
ABI
Bytecode
Home
SimpleBanking.sol
Select the compiler version here
Green tick in the Solidity Compiler option confirms code is compiled successfully
0
Listen on all transactions
Filter with transaction hash or address
Scam Alert
Initialize as git repo
Did you know? To prototype using the Gnosis safe multi sig wallet: create a multisig workspace.
RemixAI Copilot (enabled)

```
function updateDepositFee(uint256 _newFeeInEth) public onlyOwner {    infinite gas
    // Convert the new fee from ETH to wei
    depositFee = _newFeeInEth * 1 ether;

    // Emit the event for fee update with the new fee in ETH
    emit FeeUpdated(_newFeeInEth);  // Emit in ETH for readability
}

// Get balance of the sender in ETH (view function)
function getBalance() public view returns (uint256) {    infinite gas
    // Convert the balance from wei to ETH
    return balances[msg.sender] / 1 ether;  // Return balance in ETH
}
}
```

- Now we have to deploy and run transactions in the MetaMask wallet environment. By going to environment dropdown select WalletConnect->Choose MetaMask.


Home
SimpleBanking.sol
Wallet Connect
WalletConnect
Connect Wallet
WalletConnect
QR CODE
MetaMask
Browser Wallet
MetaMask
Trust Wallet
All Wallets
460+
0
Listen on all transactions
Filter with transaction hash or address
Did you know? To prototype using the Gnosis safe multi sig wallet: create a multisig workspace.
RemixAI Copilot (enabled)



Select an account

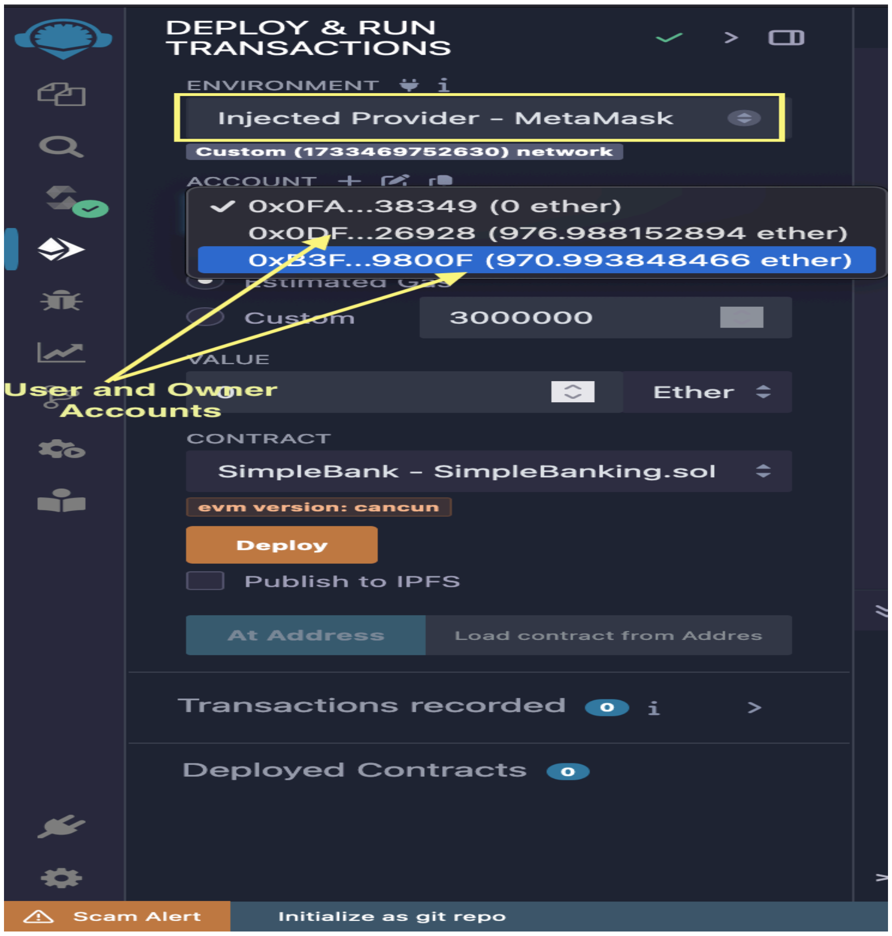
Search accounts

Account 3
0x0DF20...26928
Imported
$3,923,887.29 USD
976.988 ETH

Account 4
0xB3Ff6...9800F
Imported
$3,899,812.30 USD
970.994 ETH

Hidden accounts 2

+ Add account or hardware wallet



DEPLOY & RUN TRANSACTIONS

ENVIRONMENT
Injected Provider - MetaMask
Custom (1733469752630) network

ACCOUNT
✓ 0x0FA...38349 (0 ether)
0x0DF...26928 (976.988152894 ether)
0xB3F...9800F (970.993848466 ether)

Custom 3000000

VALUE
Ether

User and Owner Accounts

CONTRACT
SimpleBank - SimpleBanking.sol
evm version: cancun

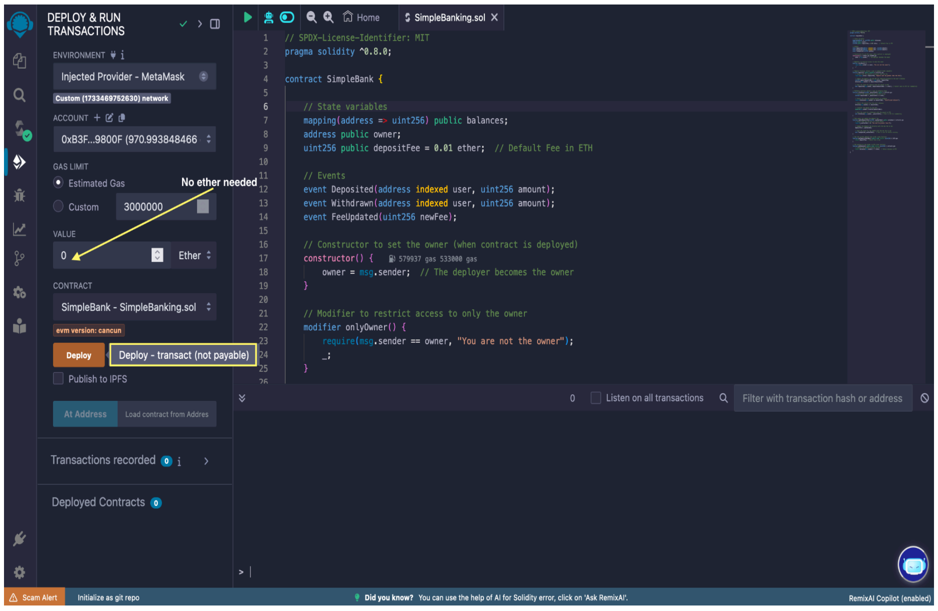
Deploy
Publish to IPFS
At Address Load contract from Addres

Transactions recorded 0

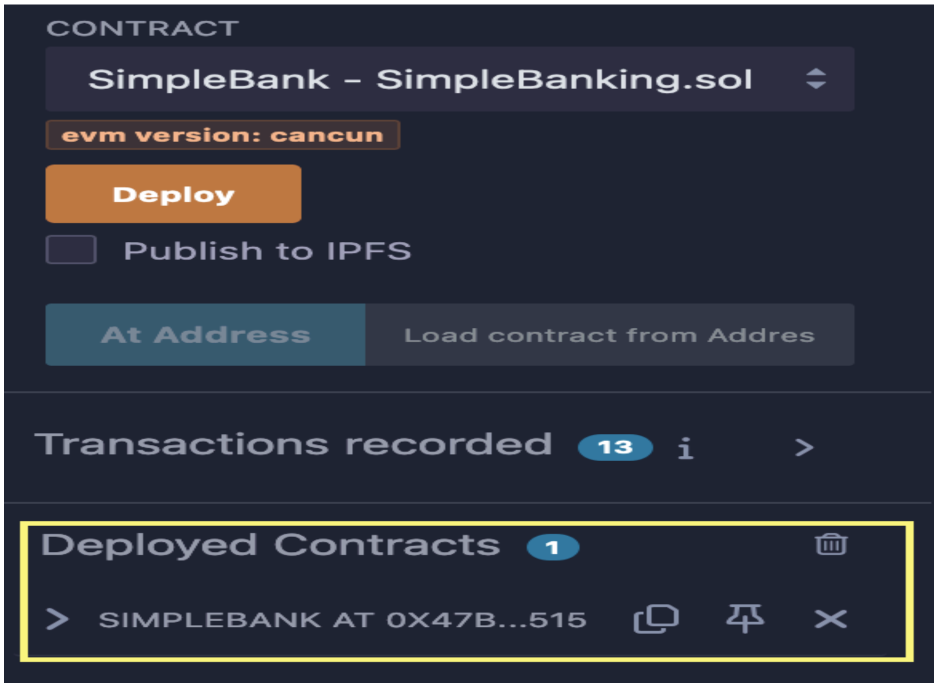
Deployed Contracts 0

Scam Alert
Initialize as git repo

- Injected Provider -MetaMask option will come where we can see linked Ganache Accounts(One account for owner , One for user)
- Ensure the account displayed in Remix matches the **owner account** imported in MetaMask.
- Select `SimpleBank` from the contract dropdown.Click "Deploy".
- MetaMask will prompt you to confirm the transaction. Review the details and confirm.
- Deploy is a non payable function inherently and thus we can simply deploy the contract.
- After deployment, the contract will appear under **"Deployed Contracts"** in Remix.
- Note the contract address for future interactions


DEPLOY & RUN TRANSACTIONS
ENVIRONMENT
Injected Provider - MetaMask
Custom (1733469752630) network
ACCOUNT
0xB3F...9800F (970.993848466
GAS LIMIT
Estimated Gas
Custom
3000000
No ether needed
VALUE
0
Ether
CONTRACT
SimpleBank - SimpleBanking.sol
evm version: cancun
Deploy
Deploy - transact (not payable)
Publish to IPFS
At Address
Load contract from Addres
Transactions recorded 0
Deployed Contracts 0
Home
SimpleBanking.sol
0
Listen on all transactions
Filter with transaction hash or address
Scam Alert
Initialize as git repo
Did you know? You can use the help of AI for Solidity error, click on 'Ask RemixAI'.
RemixAI Copilot (enabled)

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

contract SimpleBank {

    // State variables
    mapping(address => uint256) public balances;
    address public owner;
    uint256 public depositFee = 0.01 ether;  // Default Fee in ETH

    // Events
    event Deposited(address indexed user, uint256 amount);
    event Withdrawn(address indexed user, uint256 amount);
    event FeeUpdated(uint256 newFee);

    // Constructor to set the owner (when contract is deployed)
    constructor() {    579937 gas 533000 gas
        owner = msg.sender;  // The deployer becomes the owner
    }

    // Modifier to restrict access to only the owner
    modifier onlyOwner() {
        require(msg.sender == owner, "You are not the owner");
        _;
    }
```

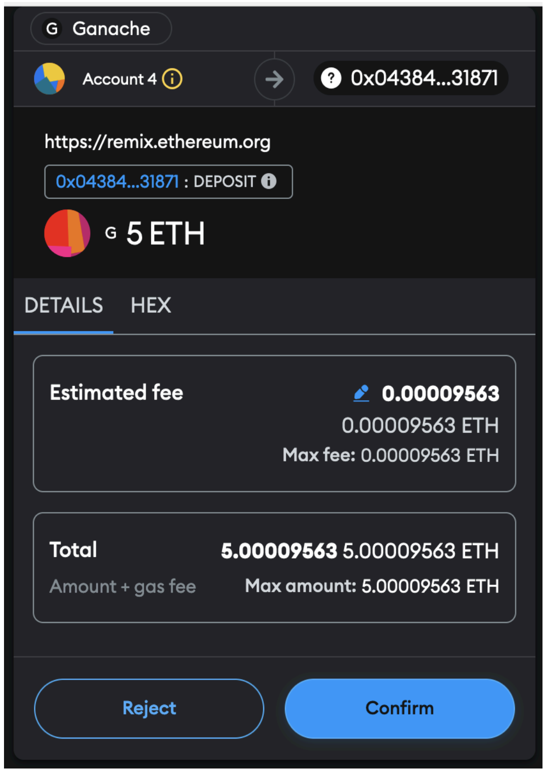


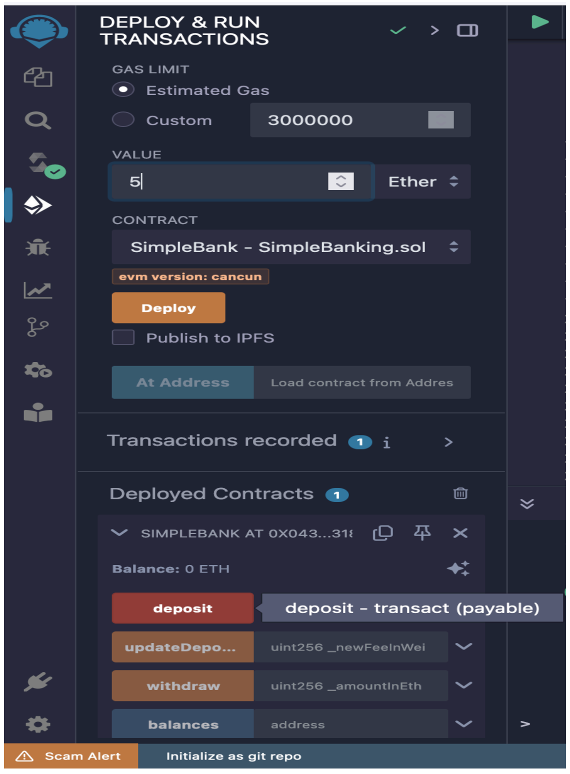
Step4 **Access the Deployed Contract Functions:**

- Under "Deployed Contracts", expand the `SimpleBank` contract to view available functions.

1.**Deposit Funds:**

- **Function:** `deposit`
- **Type:** Payable
- **Procedure:**
  - Click on the `deposit` function.
  - Enter the amount of ETH you wish to deposit (ensure it's greater than `0.01 ETH`).
  - Click **"Transact"**.
  - MetaMask will prompt for transaction confirmation. Review and confirm.
- **Verification:**
  - After the transaction is mined, the `Deposited` event will emit, showing the deposited amount.
  - The user's balance in the contract should increase by the deposited amount minus the deposit fee.

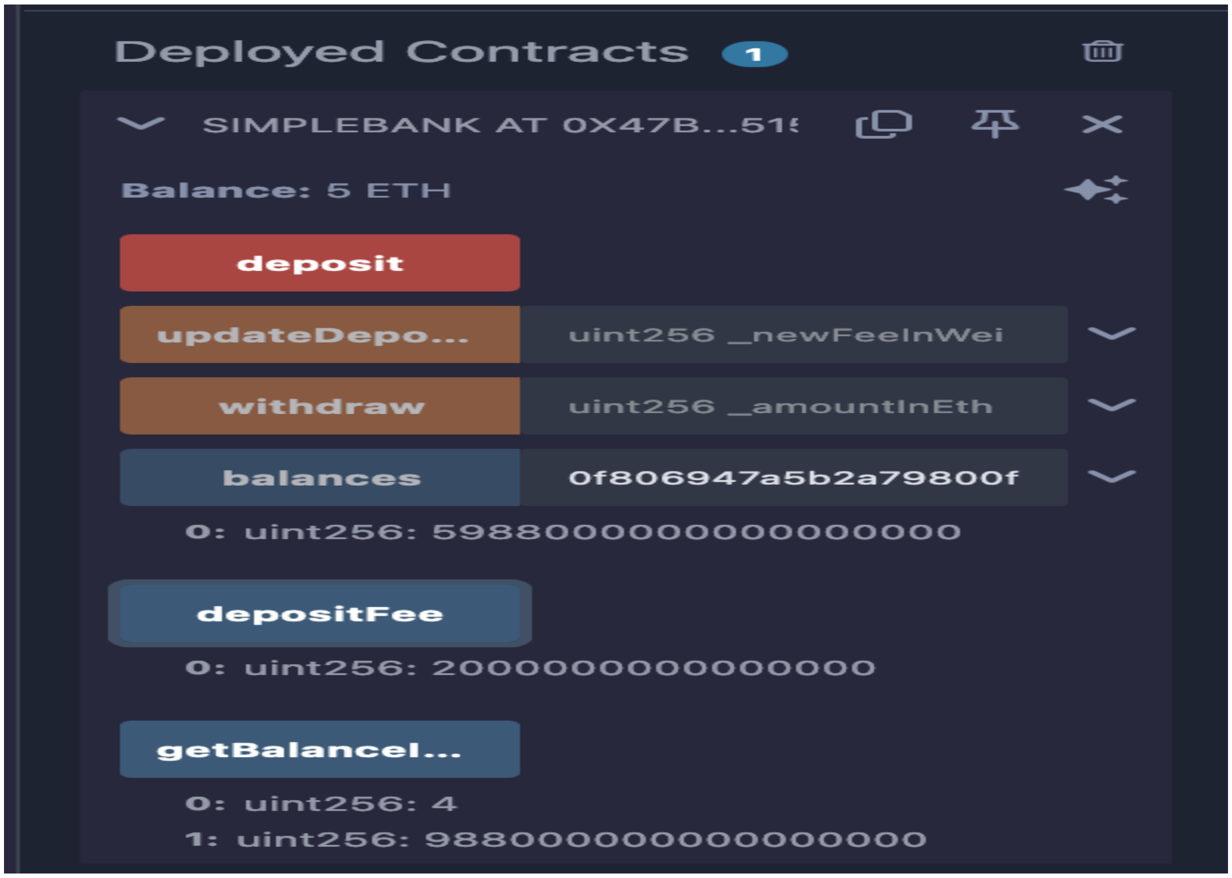

G Ganache
Account 4 → 0x04384...31871
https://remix.ethereum.org
0x04384...31871 : DEPOSIT
G 5 ETH
DETAILS HEX
Estimated fee 0.00009563
0.00009563 ETH
Max fee: 0.00009563 ETH
Total 5.00009563 5.00009563 ETH
Amount + gas fee Max amount: 5.00009563 ETH
Reject Confirm


DEPLOY & RUN TRANSACTIONS
GAS LIMIT
Estimated Gas
Custom 3000000
VALUE
5 Ether
CONTRACT
SimpleBank - SimpleBanking.sol
evm version: cancun
Deploy
Publish to IPFS
At Address Load contract from Addres
Transactions recorded 1
Deployed Contracts 1
SIMPLEBANK AT 0X043...31E
Balance: 0 ETH
deposit deposit - transact (payable)
updateDepo... uint256 _newFeeInWei
withdraw uint256 _amountInEth
balances address
Scam Alert Initialize as git repo

MetaMask Prompt ⬆️ **Deposit Function**

2. Check Balance:

- **Function:** `getBalance`
- **Type:** View
- **Procedure:**
  - Click on the `getBalance` function.
  - The balance will be displayed in **ETH**(with integer and fractional value of ETH separately)
- **Conversion to ETH:**
  - Since `1 ETH = 10^18 wei`, convert the returned value off-chain (e.g., using JavaScript or a calculator) to view the balance in ETH with decimals.(balance variable is converted to ETH using the same conversion)



Balance: 5 ETH

balances: 0f806947a5b2a79800f

```
0: uint256: 59880000000000000000
```

depositFee

```
0: uint256: 20000000000000000
```

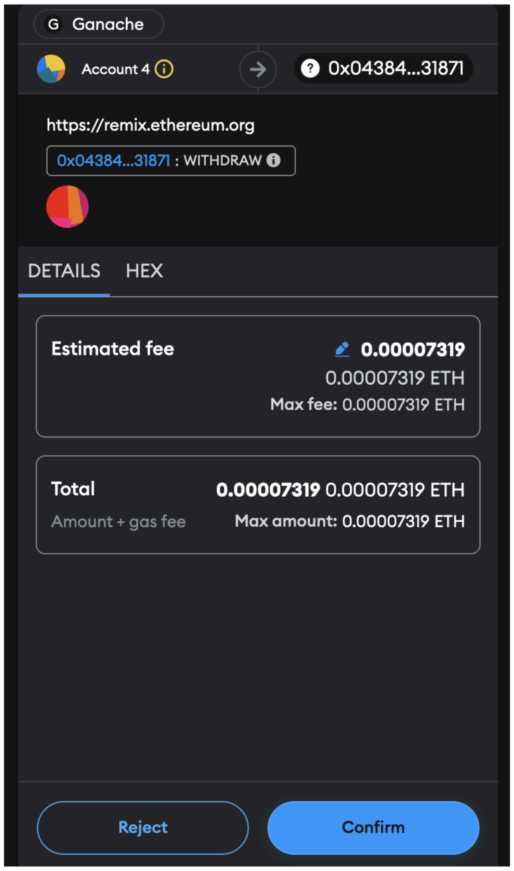
getBalanceI...

```
0: uint256: 4
1: uint256: 9880000000000000000
```

As you can see getBalance returns 4.988 ETH value after deducting the depositFee of 0.02ETH

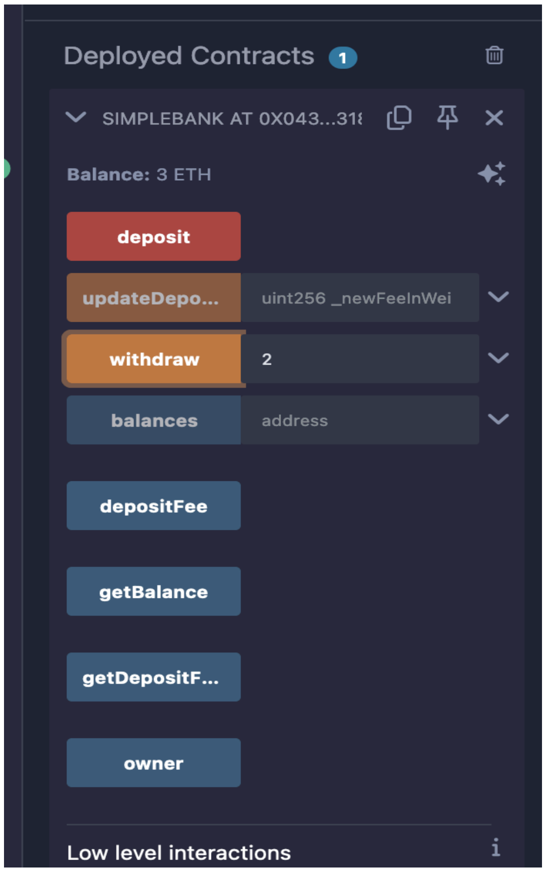
Withdraw Funds:

- **Function:** `withdraw`
- **Type:** Transaction
- **Procedure:**
  - Click on the `withdraw` function.
  - Enter the amount of ETH you wish to withdraw.
  - Click **"Transact"**.
  - MetaMask will prompt for transaction confirmation. Review and confirm.
- **Verification:**
  - After the transaction is mined, the `Withdrawn` event will emit, showing the withdrawn amount.
  - The user's balance in the contract should decrease by the withdrawn amount.



Estimated fee 0.00007319
0.00007319 ETH
Max fee: 0.00007319 ETH

Total 0.00007319 0.00007319 ETH
Amount + gas fee Max amount: 0.00007319 ETH



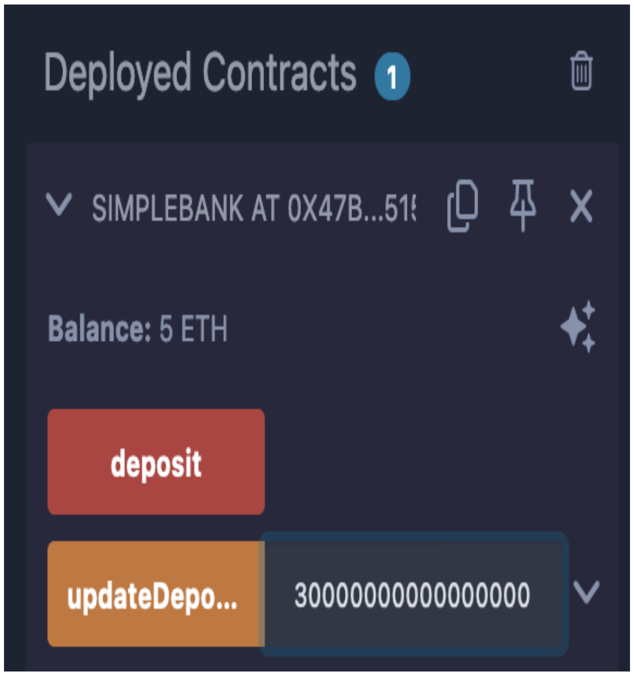
Balance: 3 ETH

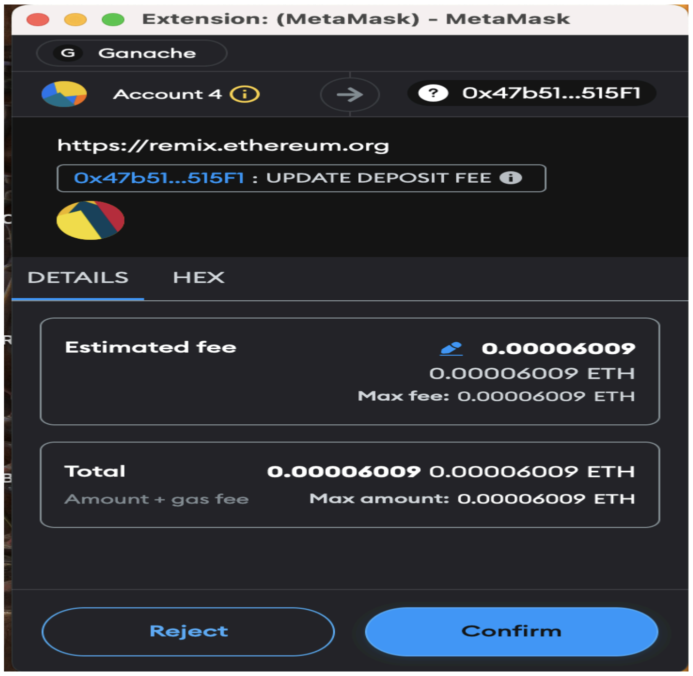
withdraw: 2

MetaMask Prompt ⬆️ **Withdraw 2ETH, balance becomes 3ETH**

Update Deposit Fee (Owner Only):

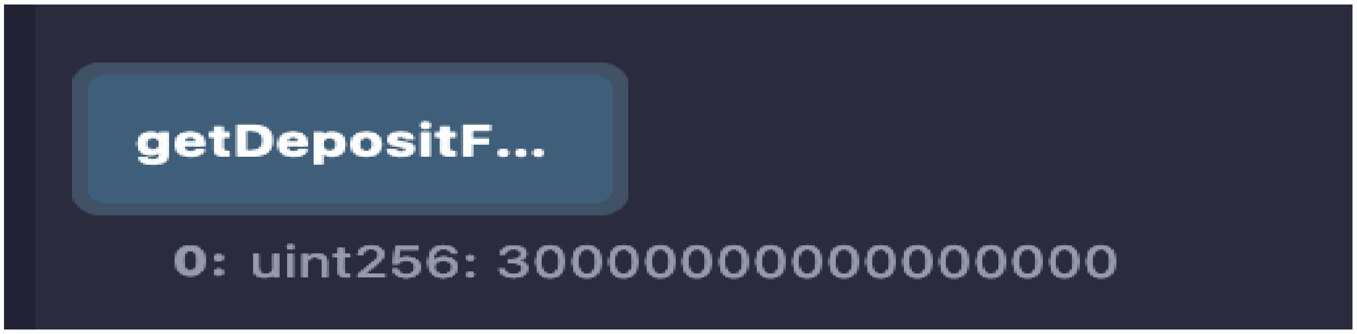
- **Function:** `updateDepositFee`
- **Type:** Transaction
- **Procedure:**
  - Ensure you are using the **owner account**.
  - Click on the `updateDepositFee` function.
  - Enter the new fee in **wei** (e.g., `30000000000000000` for `0.03 ETH`).
  - Click **"Transact"**.
  - MetaMask will prompt for transaction confirmation. Review and confirm.
- **Verification:**
  - After the transaction is mined, the `FeeUpdated` event will emit, showing the new fee.
  - The `depositFee` state variable will reflect the updated fee.




Extension: (MetaMask) - MetaMask
Ganache
Account 4
0x47b51...515F1
https://remix.ethereum.org
0x47b51...515F1 : UPDATE DEPOSIT FEE
DETAILS
HEX
Reject
Confirm

Estimated fee 0.00006009
0.00006009 ETH
Max fee: 0.00006009 ETH

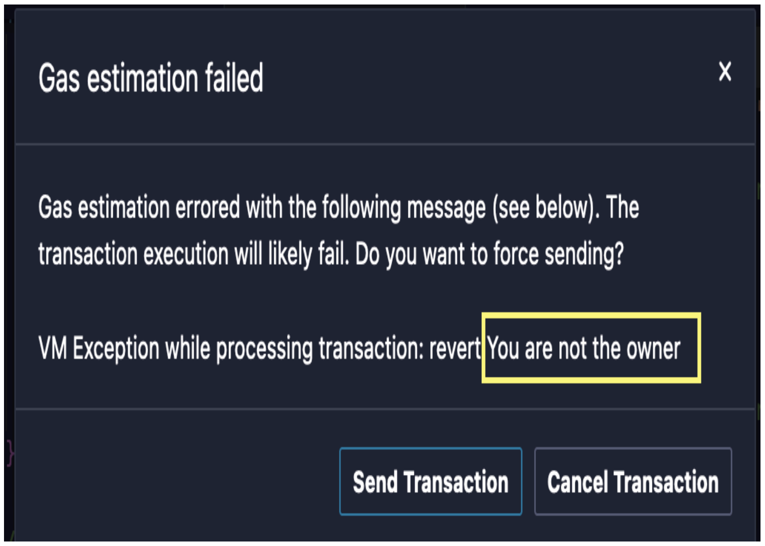
Total 0.00006009 0.00006009 ETH
Amount + gas fee Max amount: 0.00006009 ETH



```
0: uint256: 30000000000000000
```

Non-Owner Attempts to Update Fee

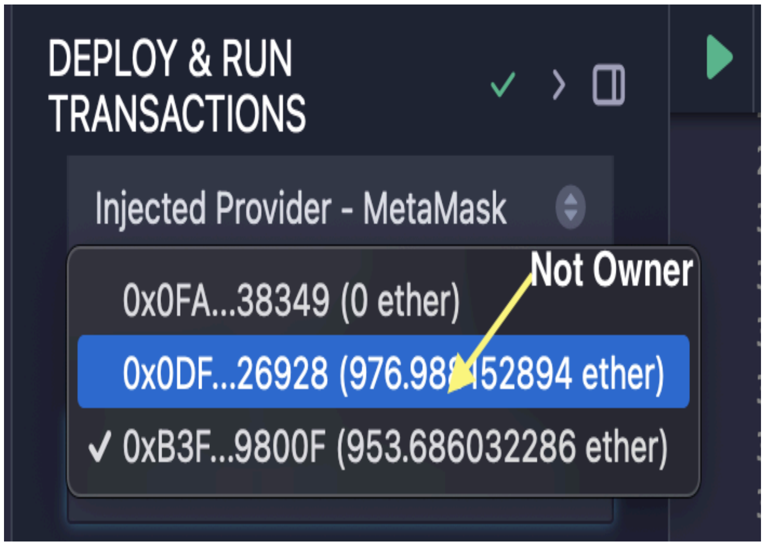
- **Action:** Use a non-owner account to call `updateDepositFee`.
- **Expected Result:** Transaction fails with the error message `"You are not the owner"`.



Gas estimation failed

Gas estimation errored with the following message (see below). The transaction execution will likely fail. Do you want to force sending?

VM Exception while processing transaction: revert You are not the owner



DEPLOY & RUN TRANSACTIONS
Injected Provider - MetaMask
0x0FA...38349 (0 ether)
0x0DF...26928 (976.988152894 ether)
✓ 0xB3F...9800F (953.686032286 ether)
Not Owner

In this laboratory session, we successfully deployed and interacted with the `SimpleBank` smart contract using Remix IDE, Ganache CLI, and MetaMask. These tools provided hands-on experience in setting up a local Ethereum blockchain, managing accounts, and executing transactions efficiently. Deploying the contract deepened our understanding of key blockchain concepts such as contract deployment, Ether transactions, and access control through modifiers.

By implementing deposit and withdrawal functionalities, we saw firsthand how smart contracts manage user balances and enforce transaction fees, ensuring both functionality and security. Updating the deposit fee as the contract owner emphasized the importance of role-based access control in smart contract management.

This lab highlighted the importance of precise unit handling in Solidity, particularly the use of wei for accurate financial computations. Additionally, we addressed balance display issues by performing off-chain conversions, demonstrating a common practice in real-world blockchain applications.

Overall, this exercise provided a solid foundation in smart contract deployment and interaction, equipping us with essential skills to develop and test more complex decentralized applications in the future.