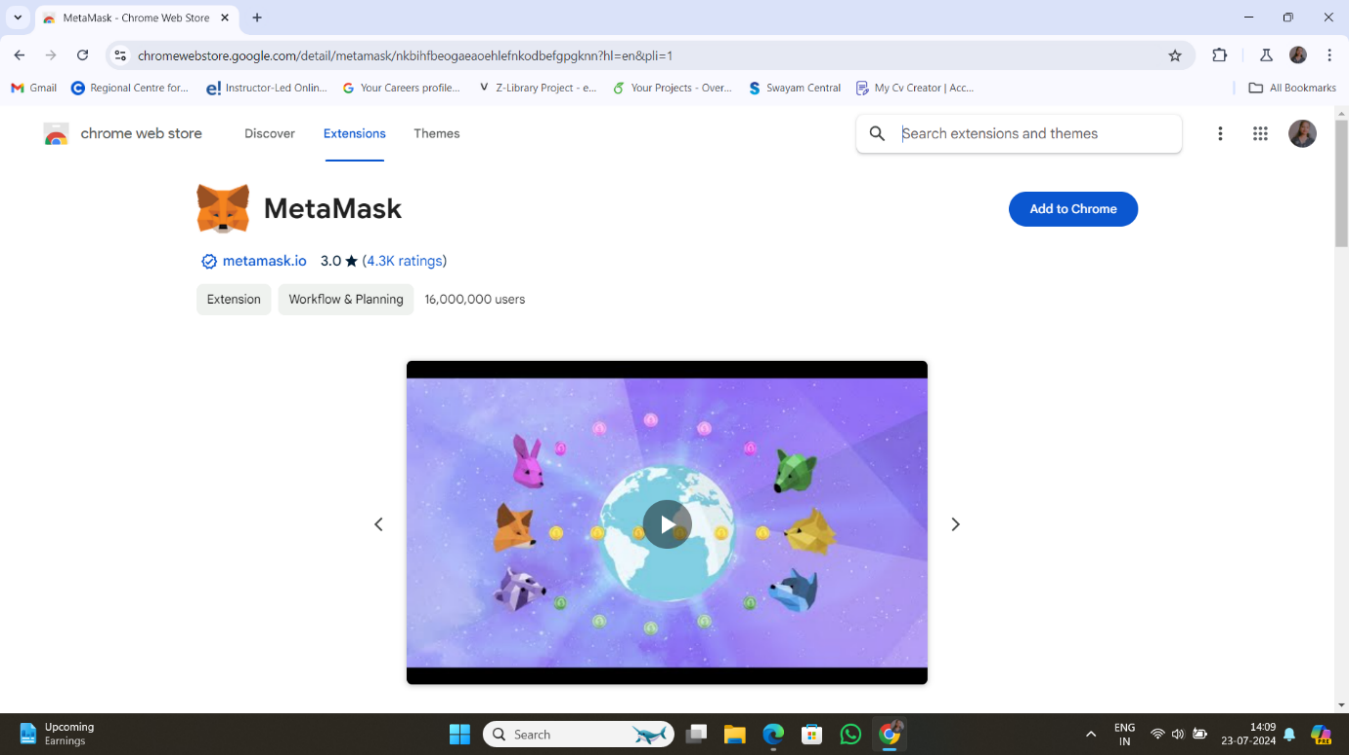
**Practical No. 01**

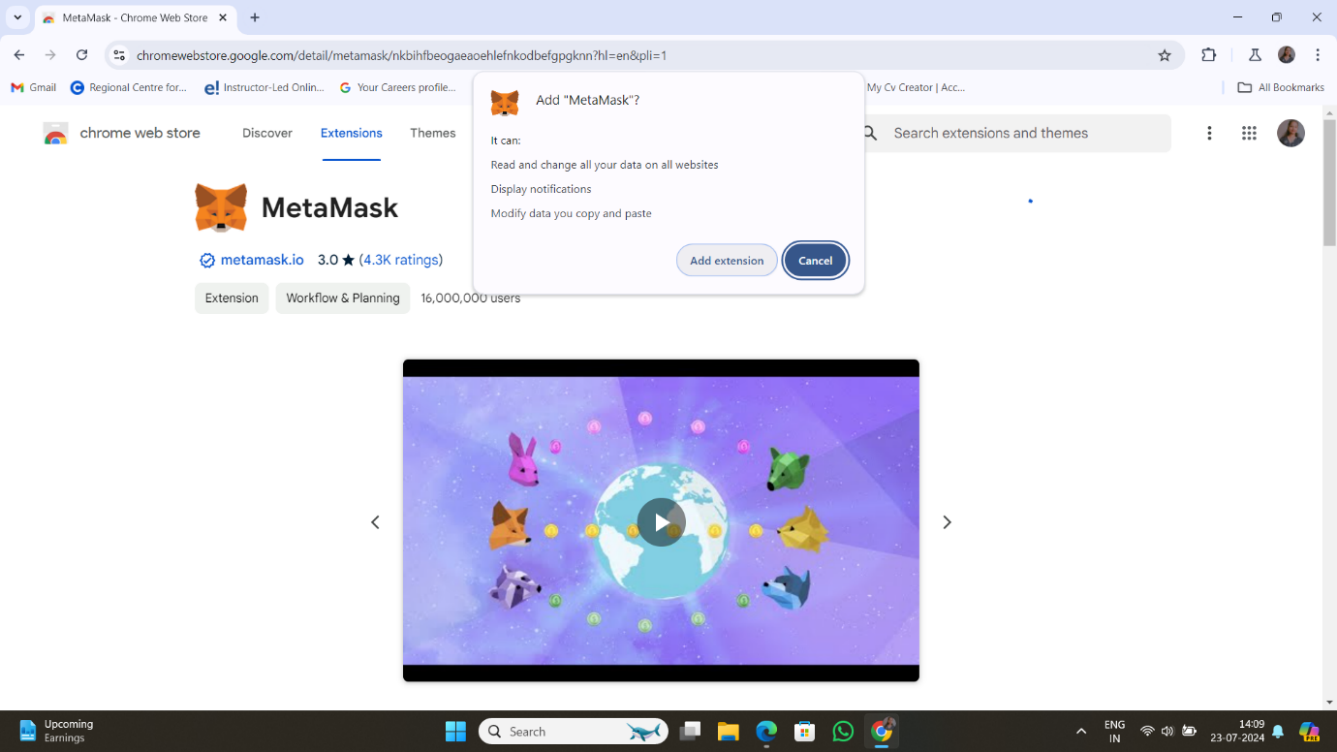
**Title: Installation of MetaMask and study spending Ether per transaction**

**Add MetaMask Extension to chrome:**

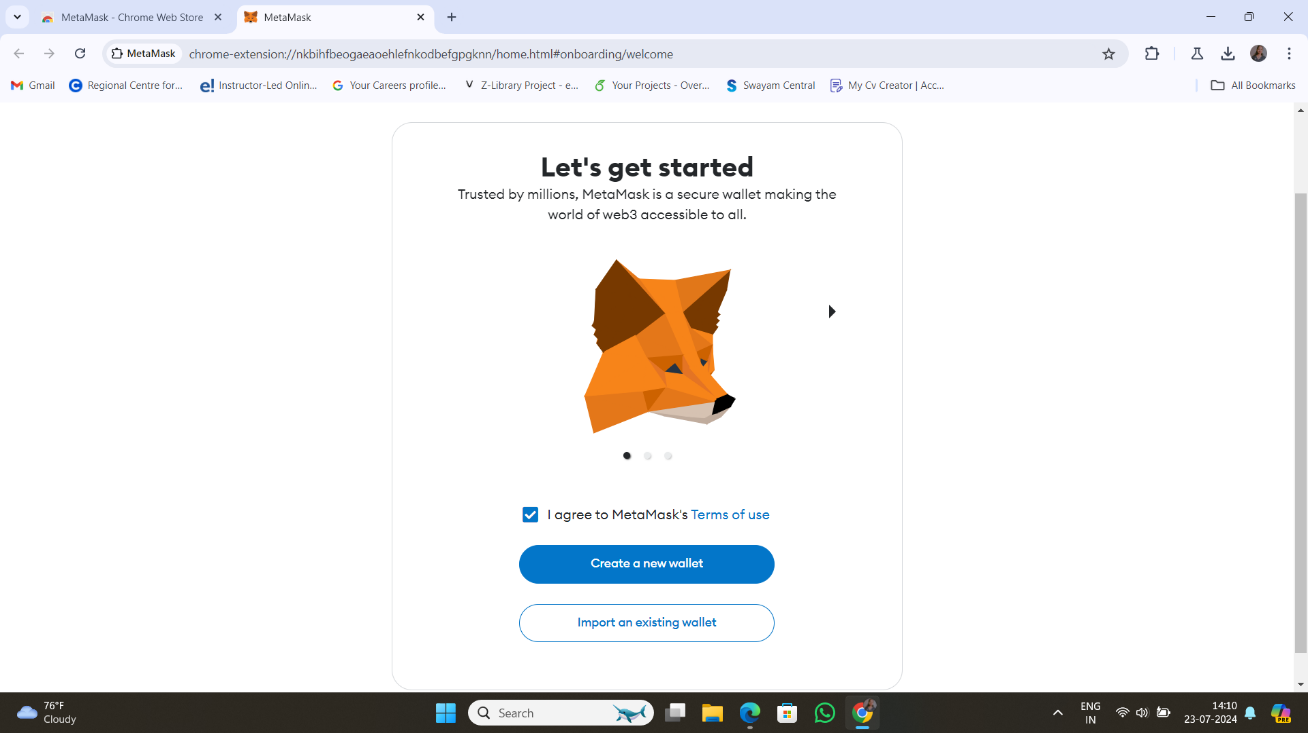
1. **Search MetaMask in Chrome Web Store Extension Section.**



1. **Click on “Add Extension”.**

****

1. **Click on “Create new Wallet”.**

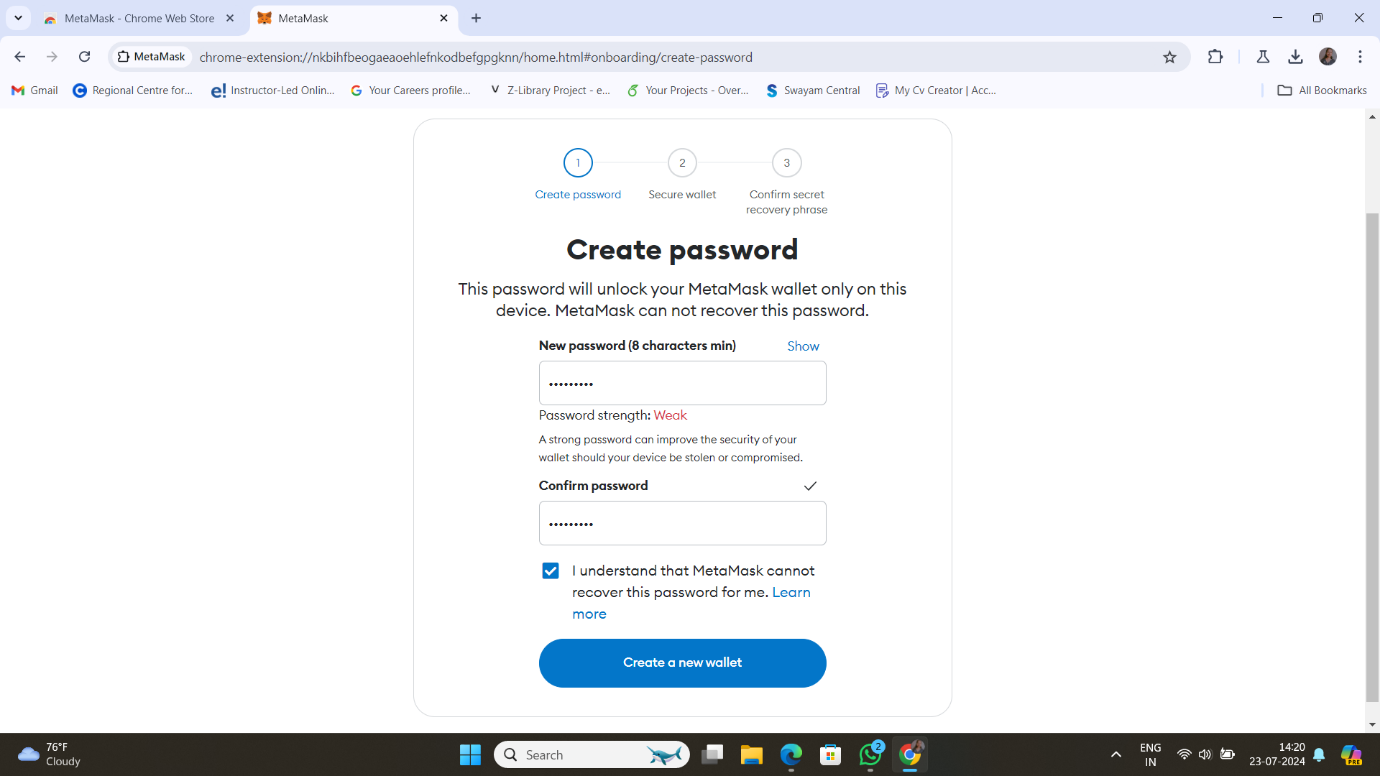
****

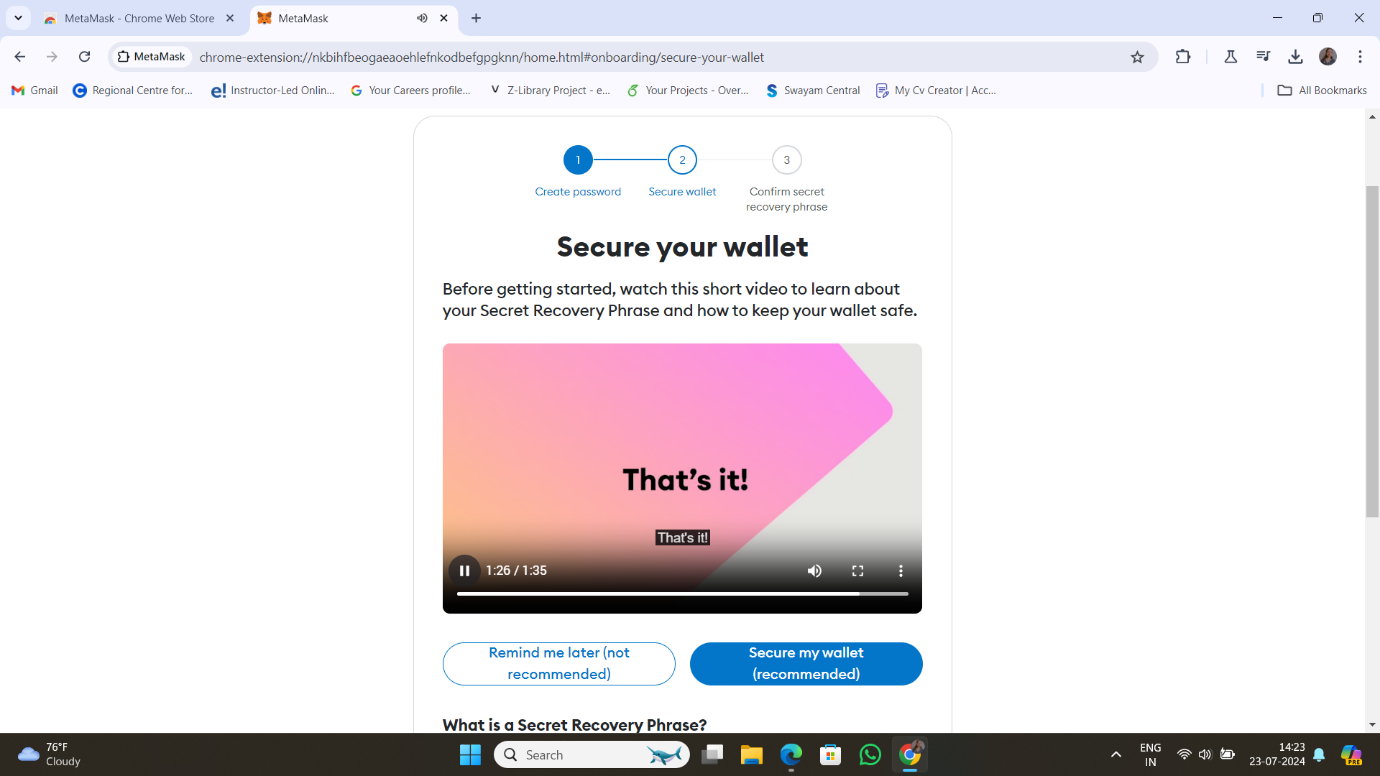
1. **Click “I Agree”.**

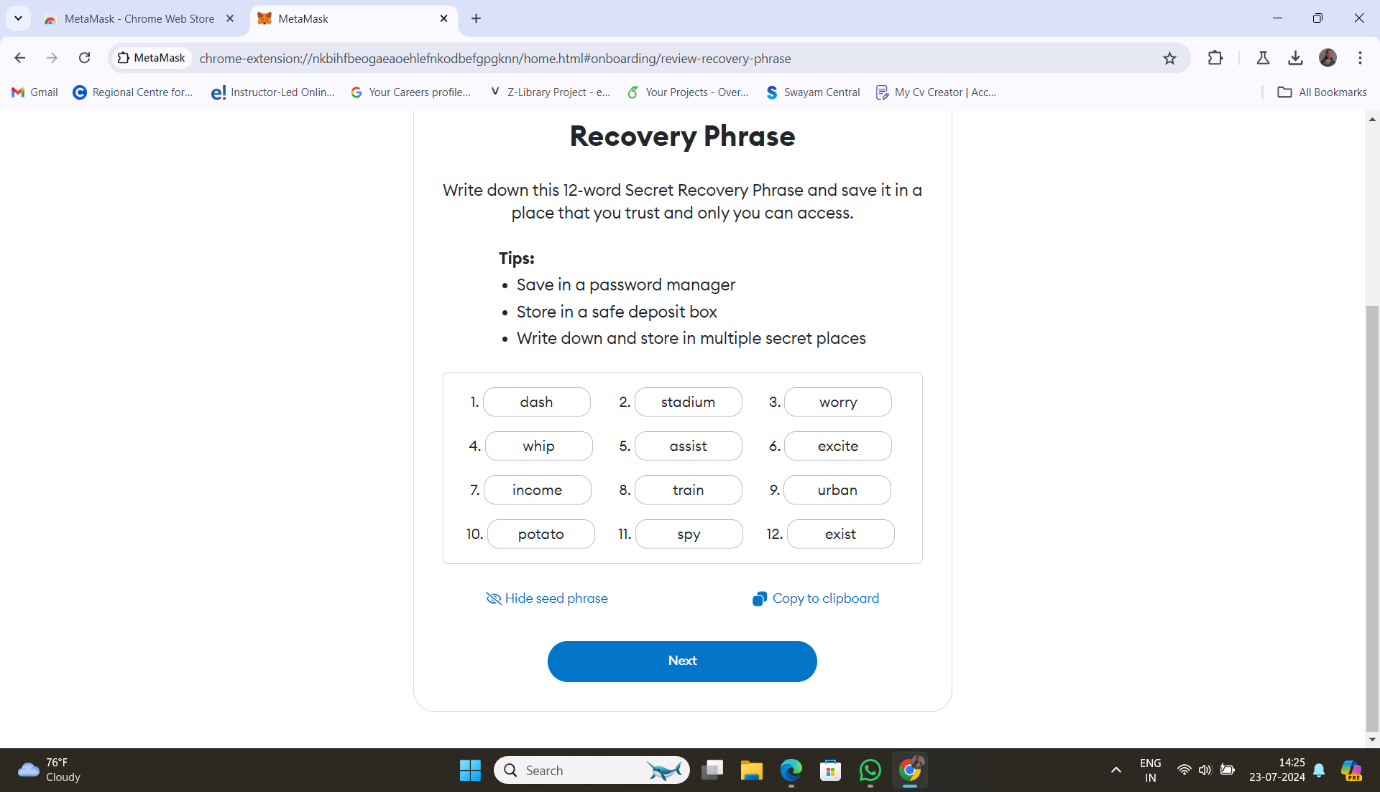
****

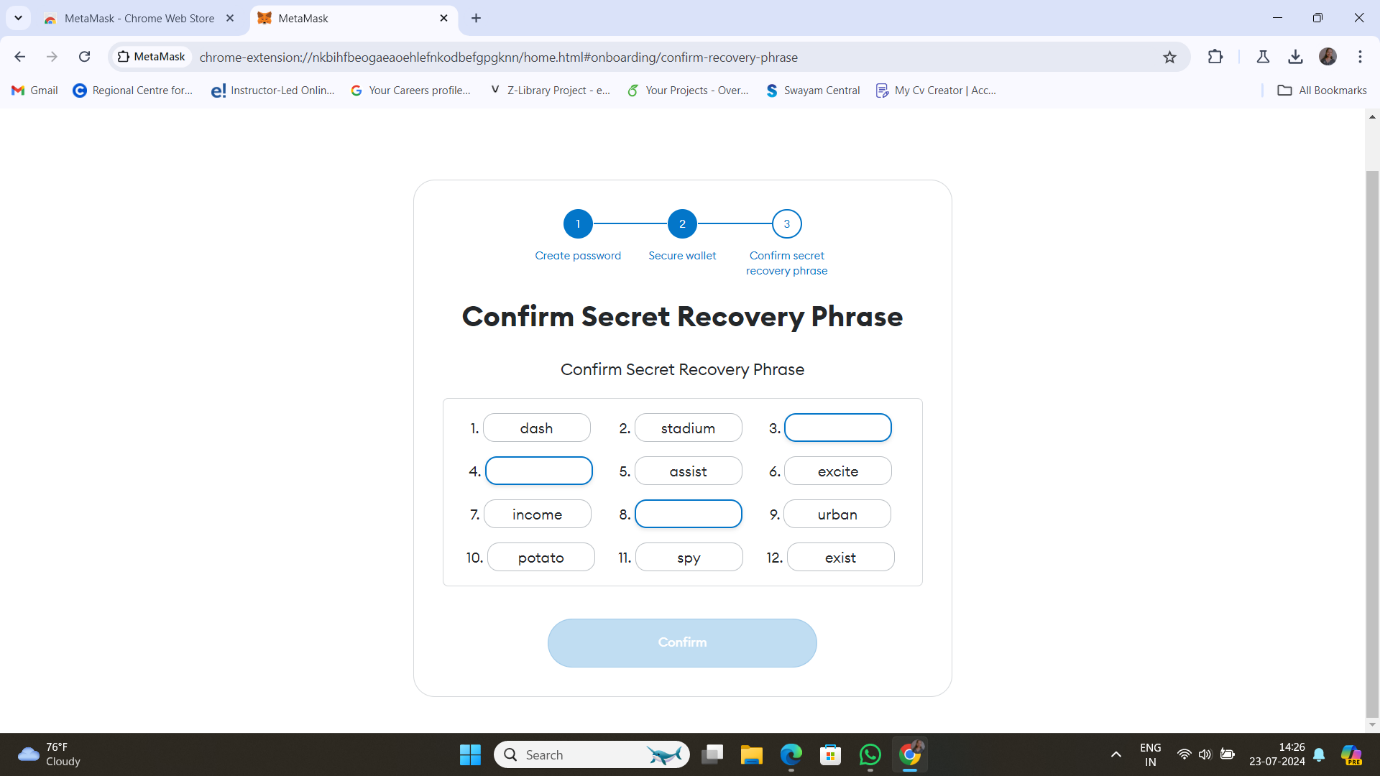
**Practical No. 02**

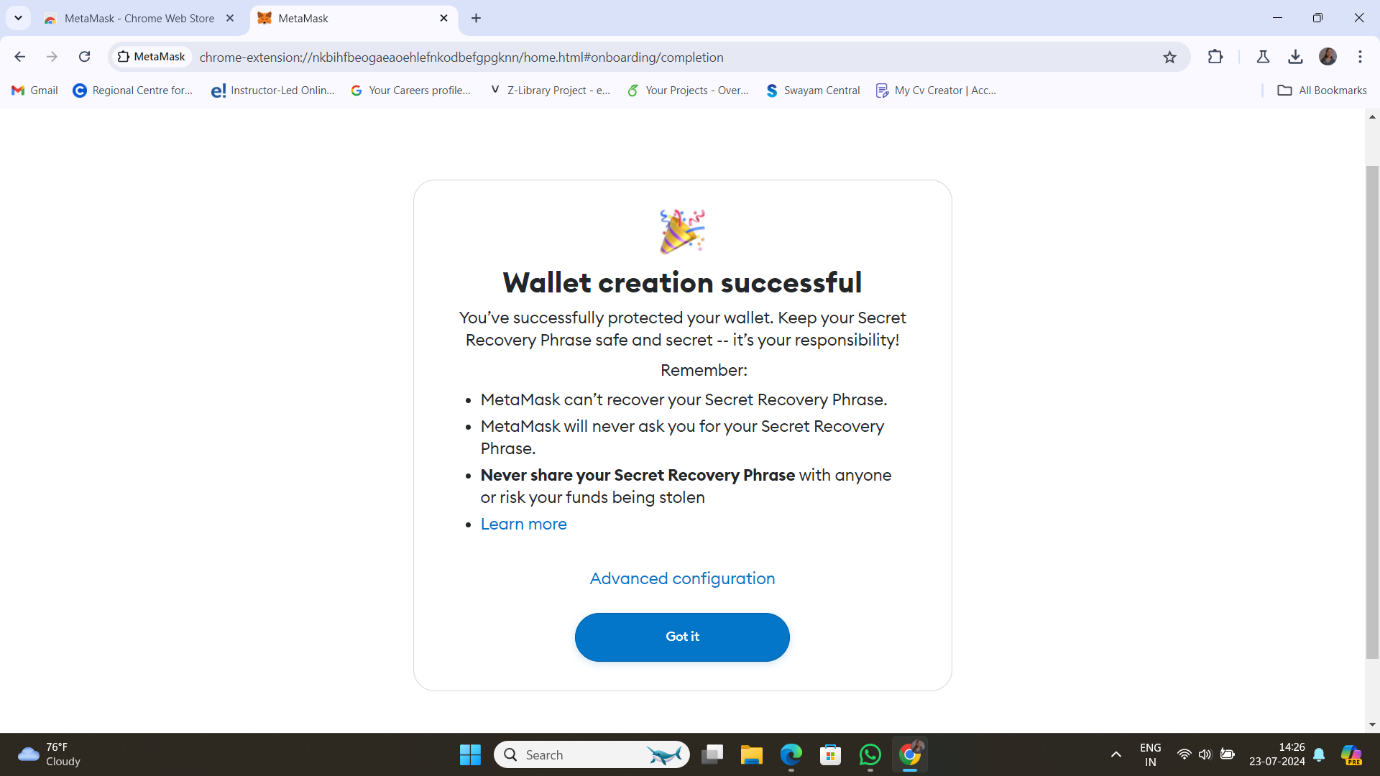
**Title: Create your own wallet using MetaMask for crypto transactions.**

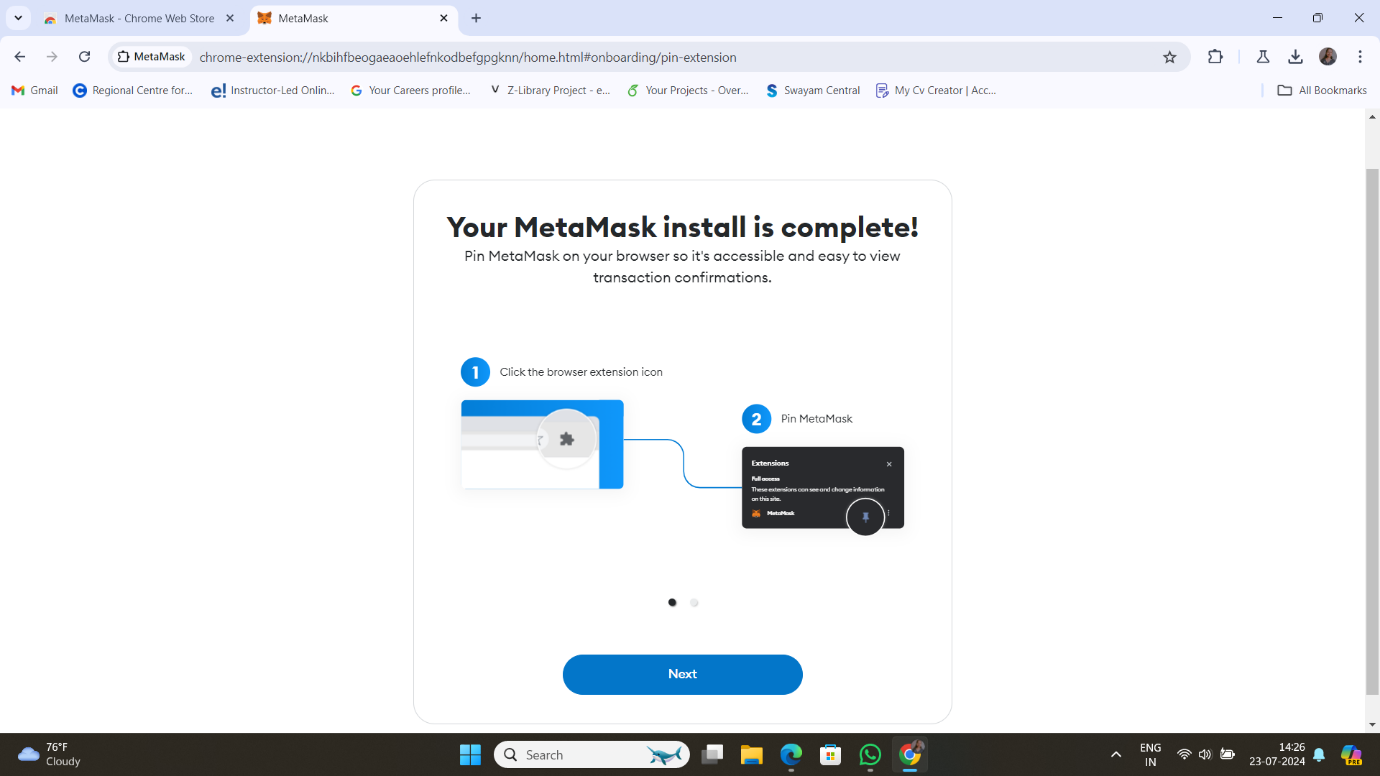
****

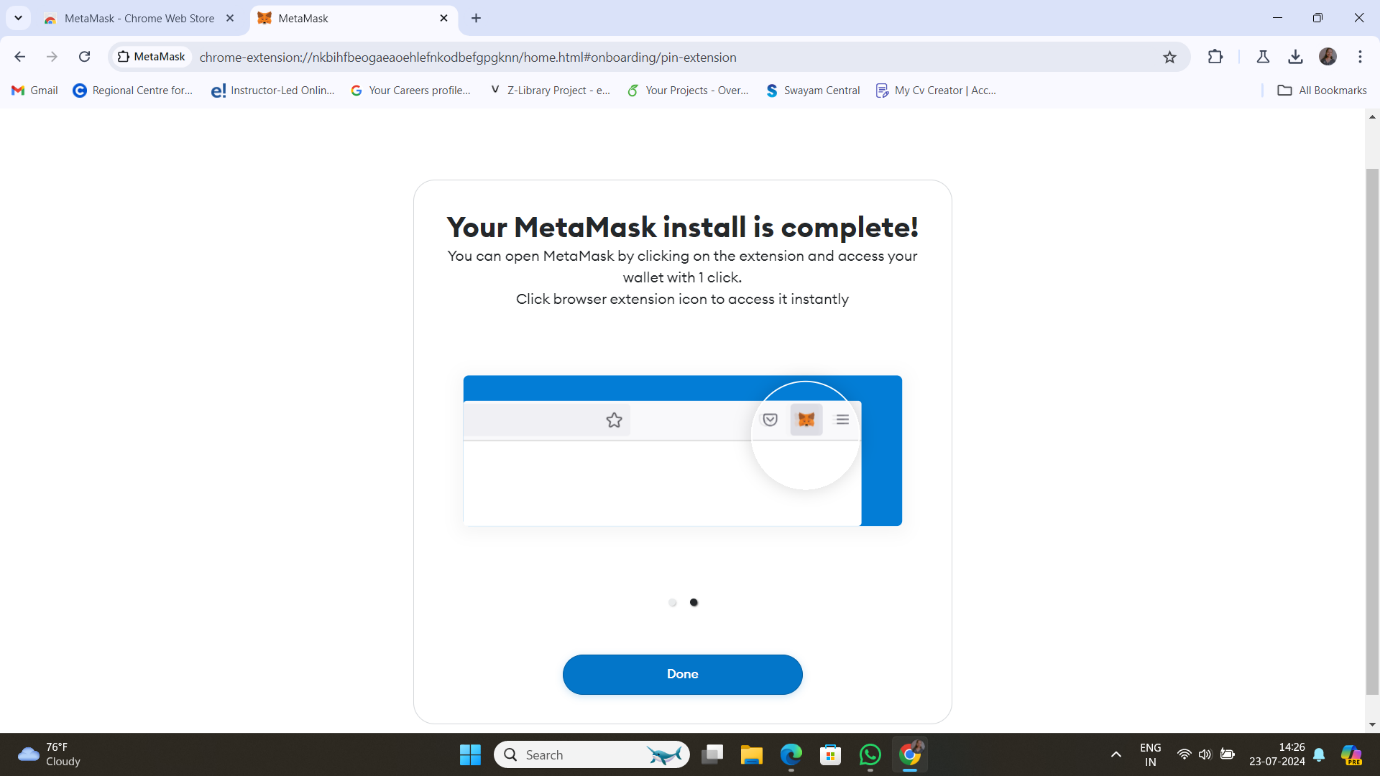
****

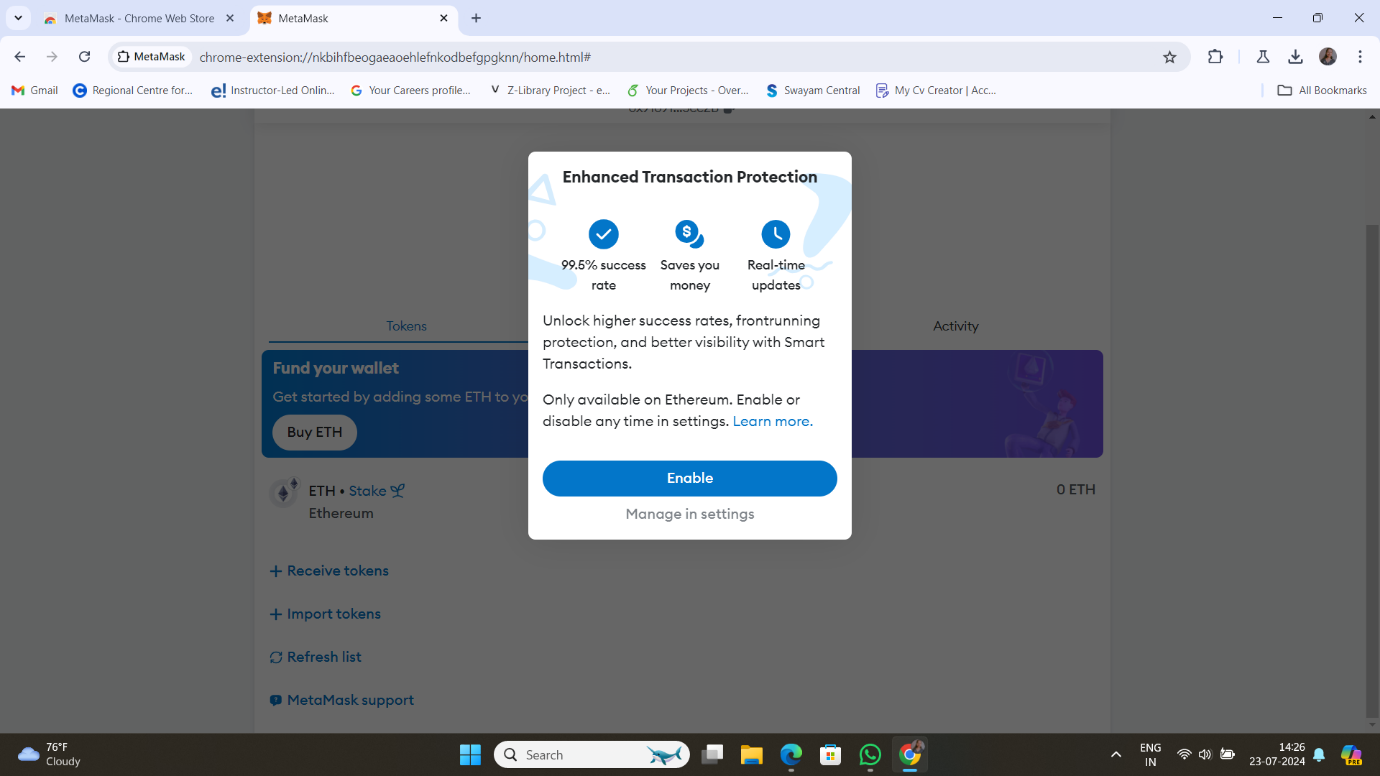
****

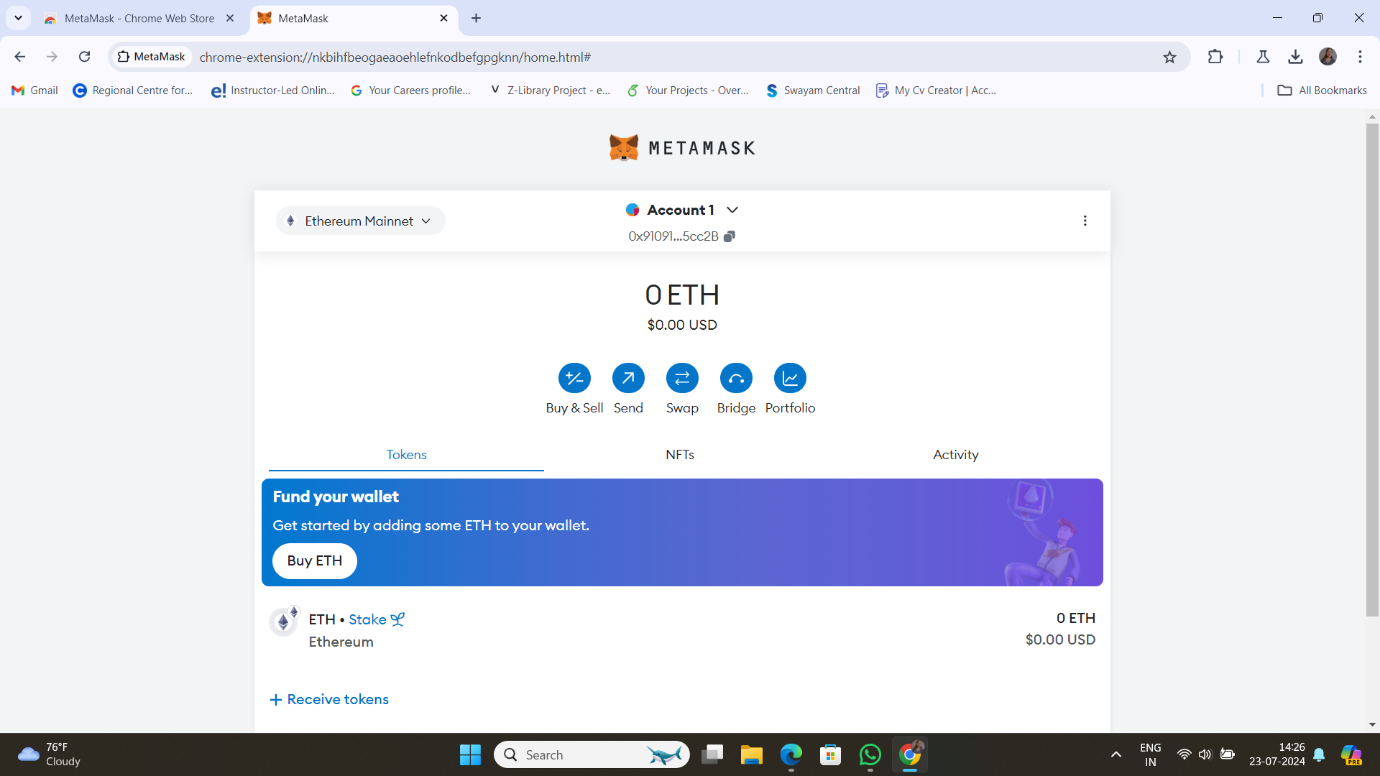
****

****

****

****

****

****

**Practical No. 03**

**Title: Write a smart contract on a test network, for Bank account of a customer for following operations:**

**• Deposit money • Withdraw Money • Show balance**

**Code:** //SPDX-License-Identifier: MIT

pragma solidity ^0.6;

contract banking {

mapping(address=>uint) public user\_account;

mapping(address=>bool) public user\_exists;

function create\_account() public payable returns(string memory) {

require(user\_exists[msg.sender]==false,'Account already created');

if(msg.value==0) {

user\_account[msg.sender]=0;

user\_exists[msg.sender]=true;

return "Account created";

}

require(user\_exists[msg.sender]==false,"Account already created");

user\_account[msg.sender]=msg.value;

user\_exists[msg.sender]=true;

return "Account created";

}

function deposit() public payable returns(string memory) {

require(user\_exists[msg.sender]==true,"Account not created");

require(msg.value>0,"Value for deposit is Zero");

user\_account[msg.sender]=user\_account[msg.sender]+msg.value;

return "Deposited Successfully";

}

function withdraw(uint amount) public payable returns(string memory) {

require(user\_account[msg.sender]>amount,"Insufficient Balance");

require(user\_exists[msg.sender]==true,"Account not created");

require(amount>0,"Amount should be more than zero");

user\_account[msg.sender]=user\_account[msg.sender]-amount;

msg.sender.transfer(amount);

return "Withdrawl Successful";

}

function transfer(address payable userAddress, uint amount) public returns(string memory) {

require(user\_account[msg.sender]>amount,"Insufficient balance in Bank account");

require(user\_exists[msg.sender]==true,"Account is not created");

require(user\_exists[userAddress]==true,"Transfer account does not exist");

require(amount>0,"Amount should be more than zero");

user\_account[msg.sender]=user\_account[msg.sender]-amount;

user\_account[userAddress]=user\_account[userAddress]+amount;

return "Transfer Successful";

}

function send\_amt(address payable toAddress, uint256 amount) public payable returns(string memory) {

require(user\_account[msg.sender]>amount,"Insufficeint balance in Bank account");

require(user\_exists[msg.sender]==true,"Account is not created");

require(amount>0,"Amount should be more than zero");

user\_account[msg.sender]=user\_account[msg.sender]-amount;

toAddress.transfer(amount);

return "Transfer Success";

}

function user\_balance() public view returns(uint) {

return user\_account[msg.sender];

}

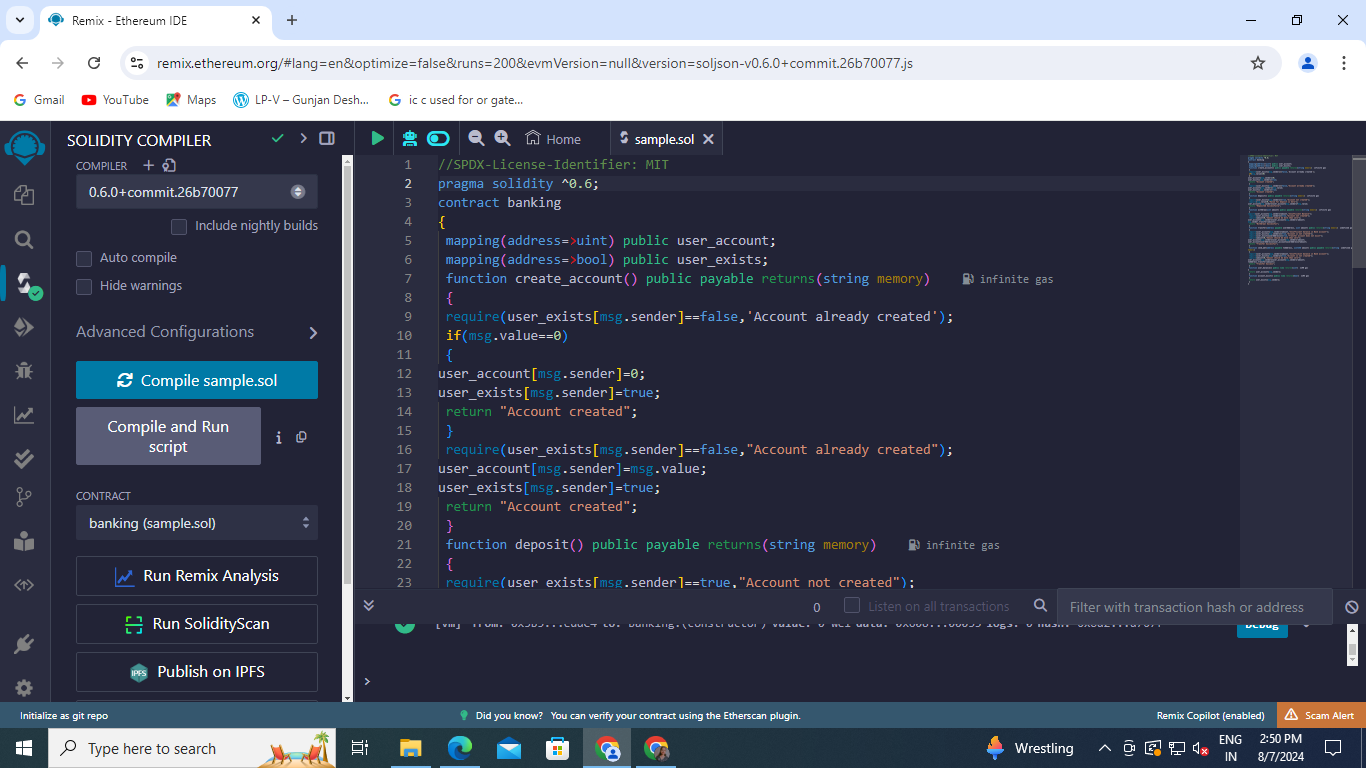
function account\_exist() public view returns(bool) {

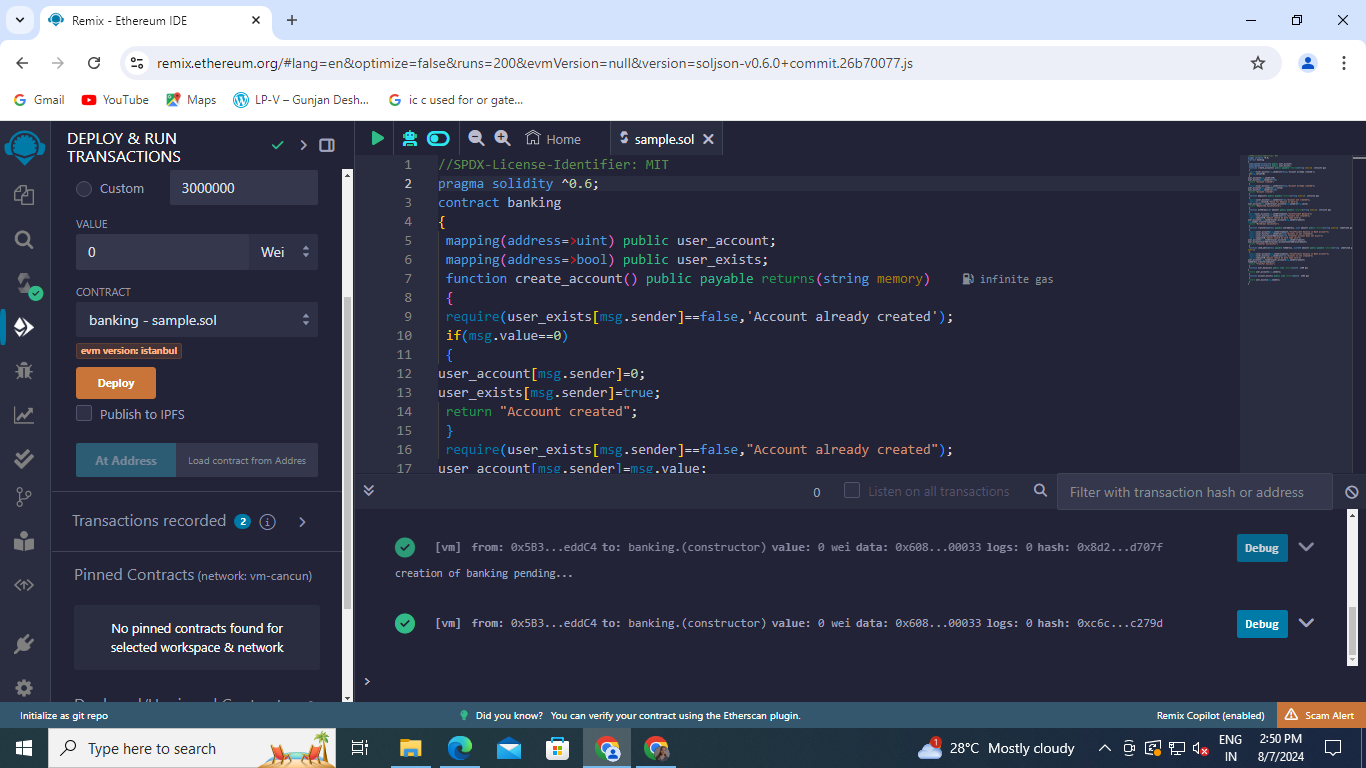
return user\_exists[msg.sender];

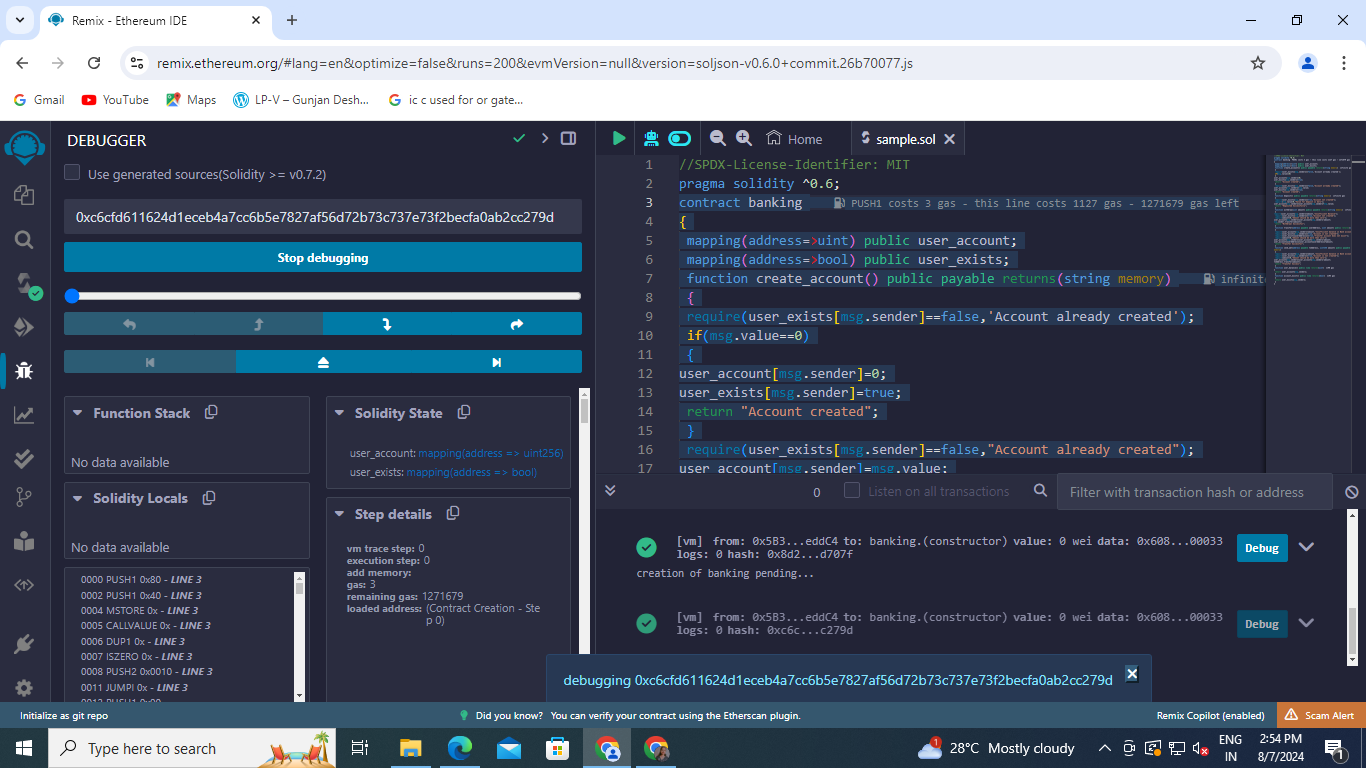
}

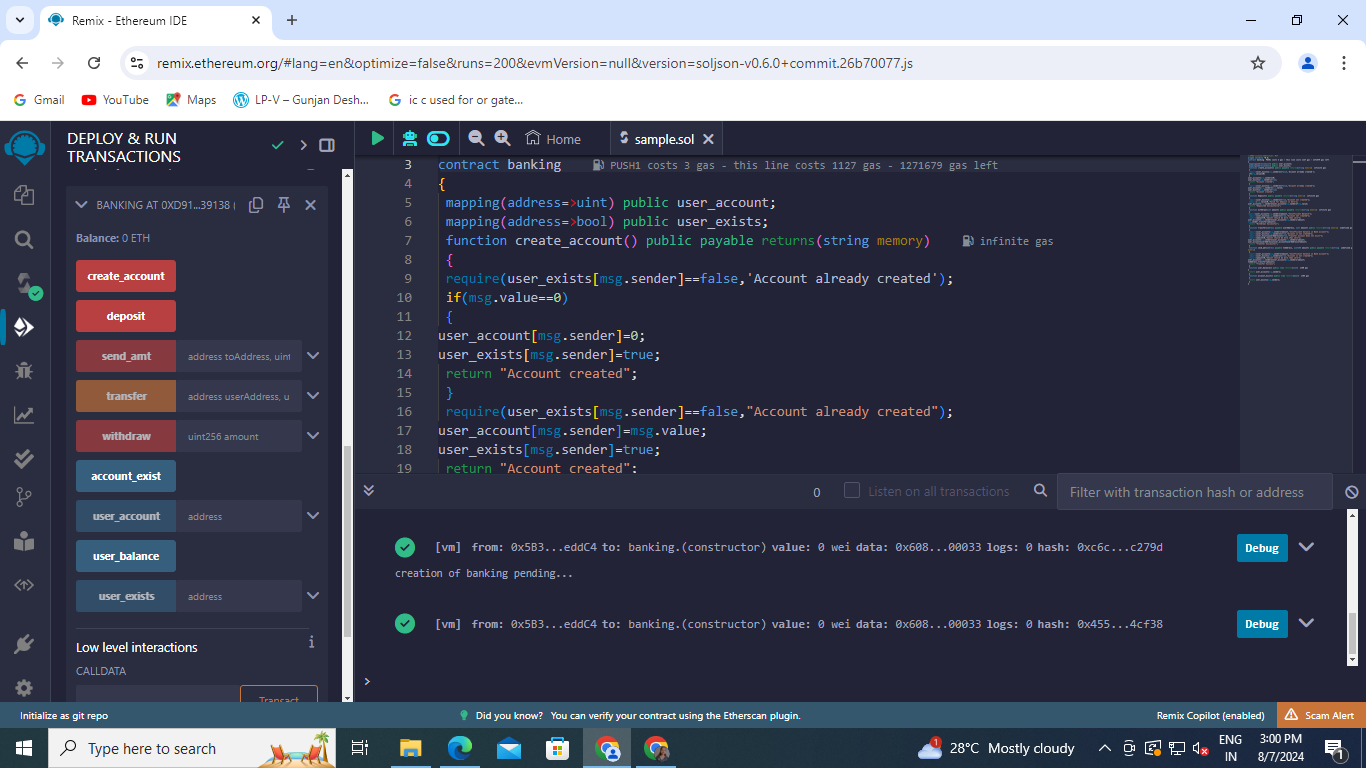
}

**Output:**

****

****

****

****

**Practical No. 04**

**Title: Write a program in solidity to create Student data. Use the following constructs:**

**• Structures • Arrays • Fallback**

**Deploy this as smart contract on Ethereum and Observe the transaction fee and Gas values.**

**Code:** // SPDX-License-Identifier: MIT

pragma solidity ^0.6.0;

contract StudentManagement {

struct Student {

uint stud\_id;

string name;

string department;

}

Student[] public students;

function addStudent(uint stud\_id, string memory name, string memory department) public {

Student memory newStudent = Student(stud\_id, name, department);

students.push(newStudent);

}

function getStudent(uint stud\_id) public view returns (string memory, string memory) {

for (uint i = 0; i < students.length; i++) {

Student memory stud = students[i];

if (stud.stud\_id == stud\_id) {

return (stud.name, stud.department);

}

}

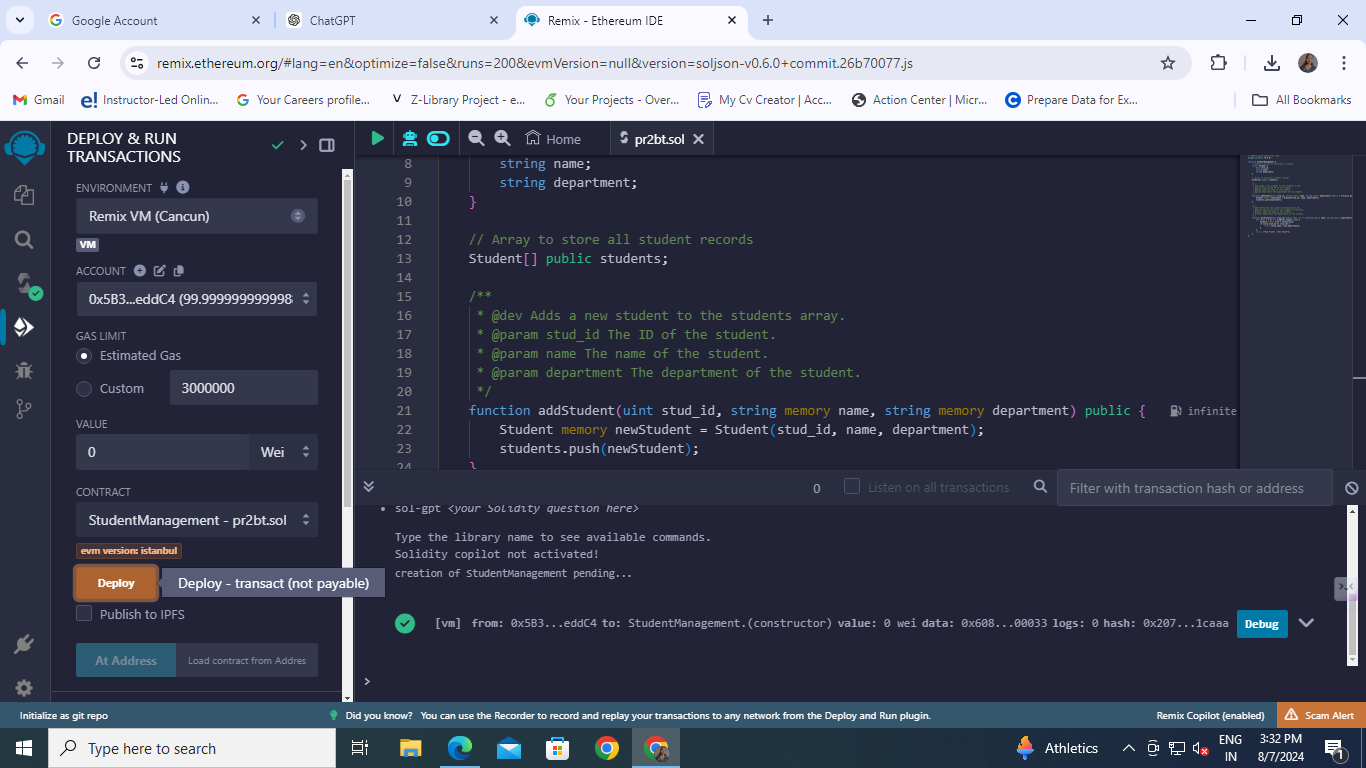
return ("Not Found", "Not Found");

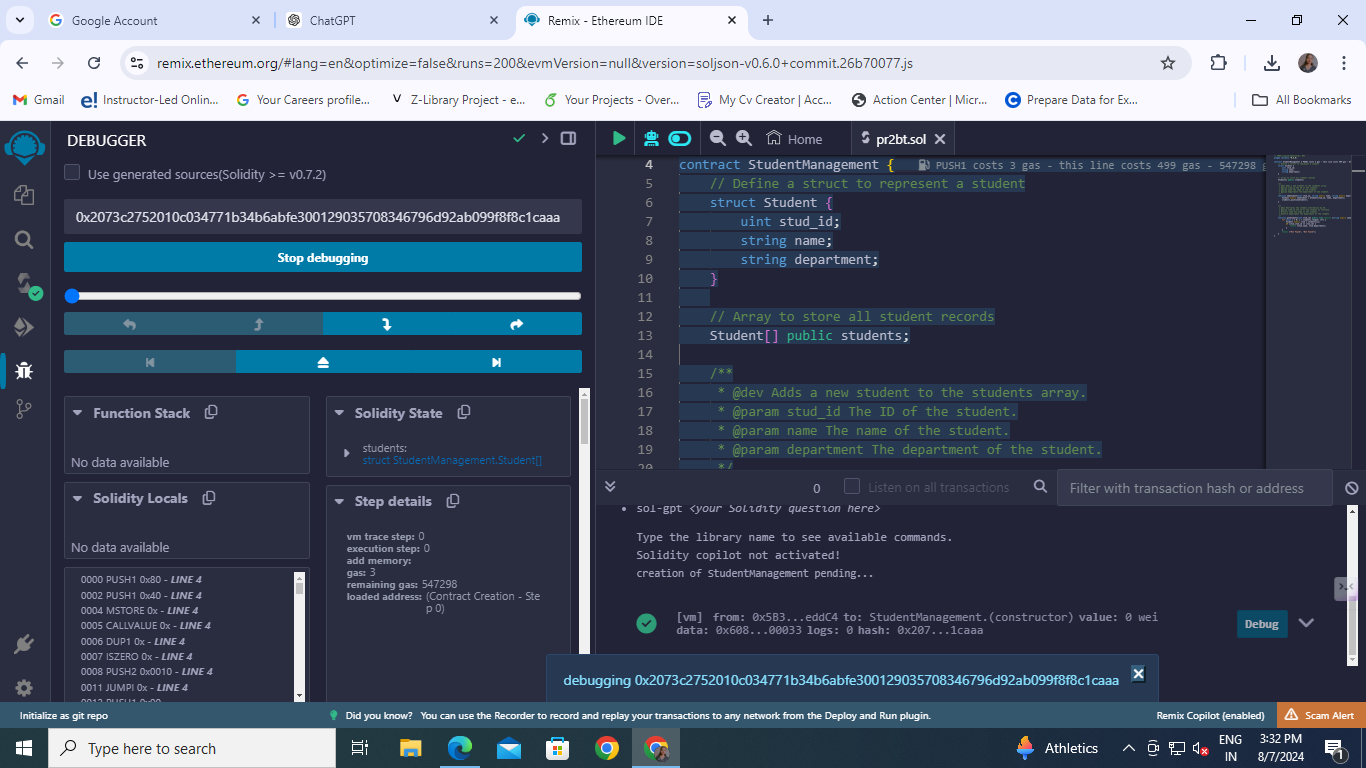
}

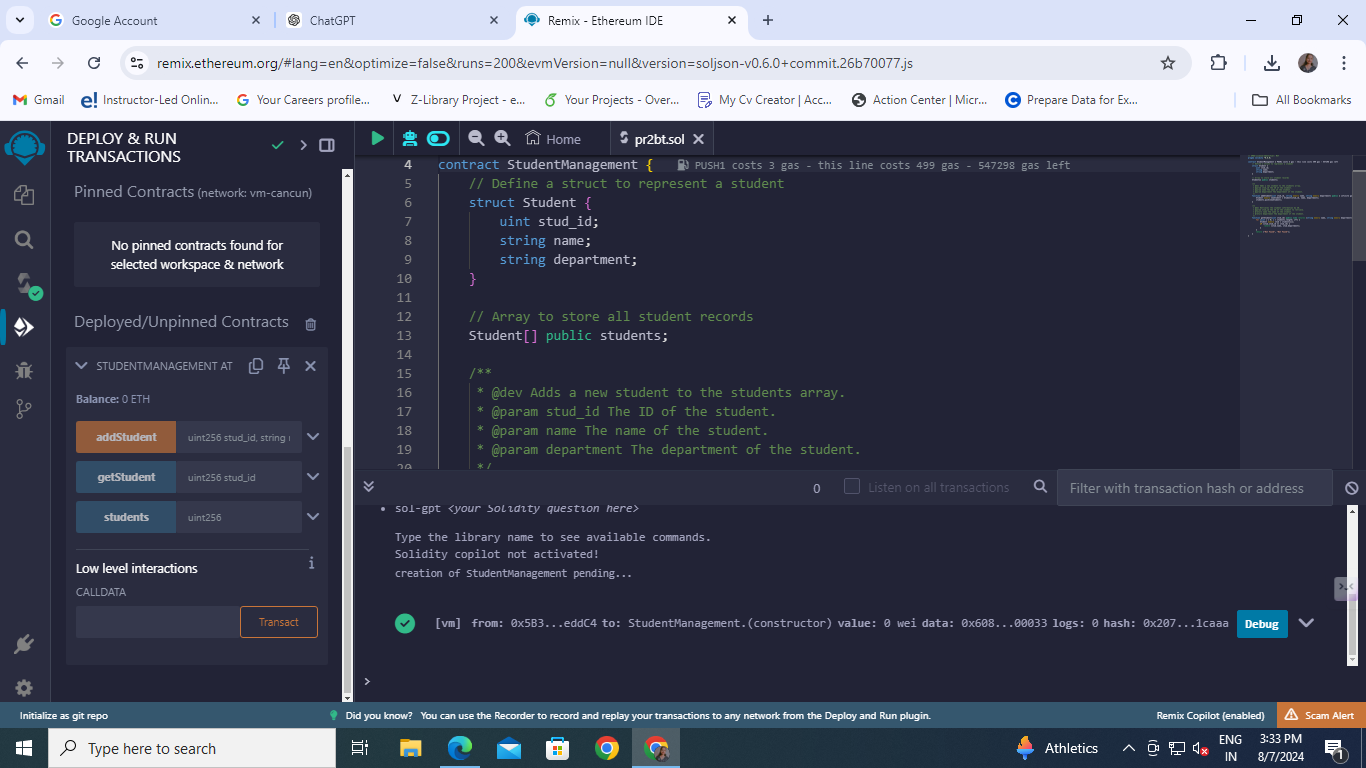
}

**Output:**

****

****

****

****