Group C: BLOCKCHAIN TECHNOLOGY

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Assignment No: 1**

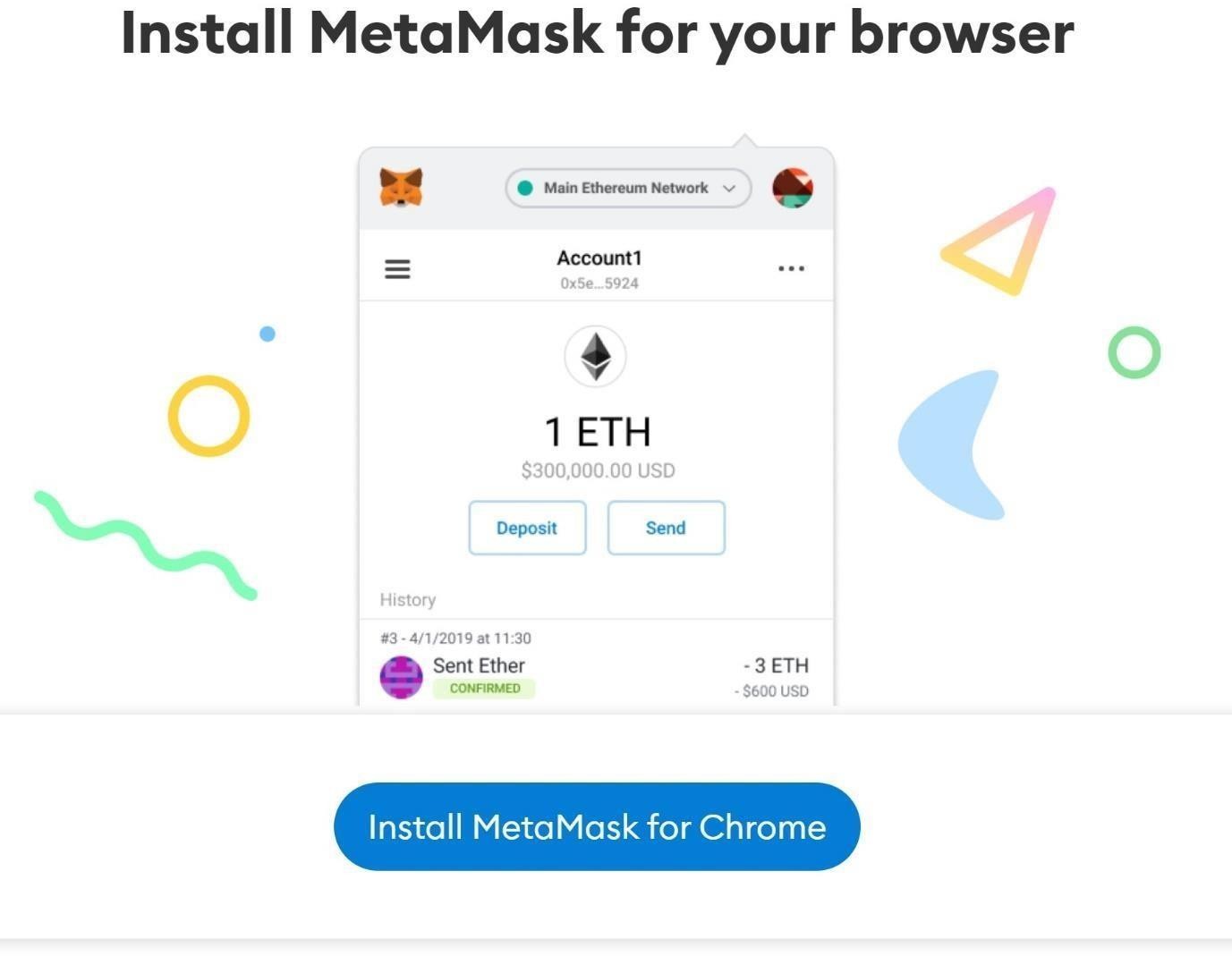
**Title Name:** Installation of Metamask and study spending Ether per transactionlocation.

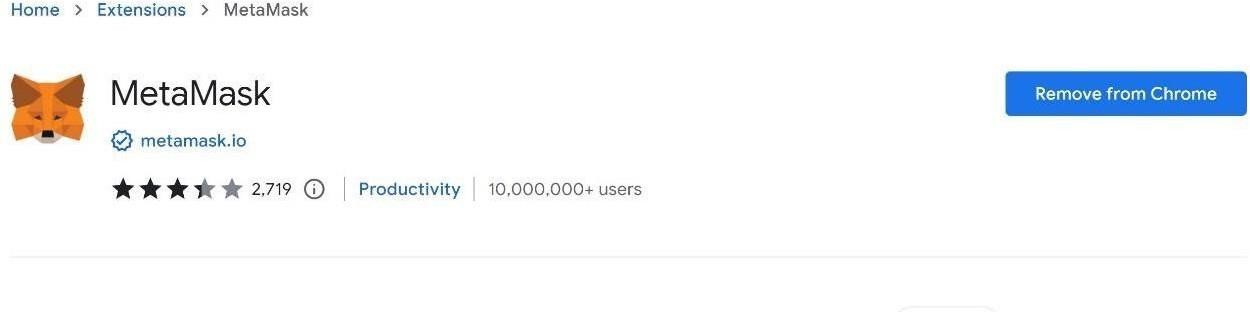
**Name**: Aditi Shivani

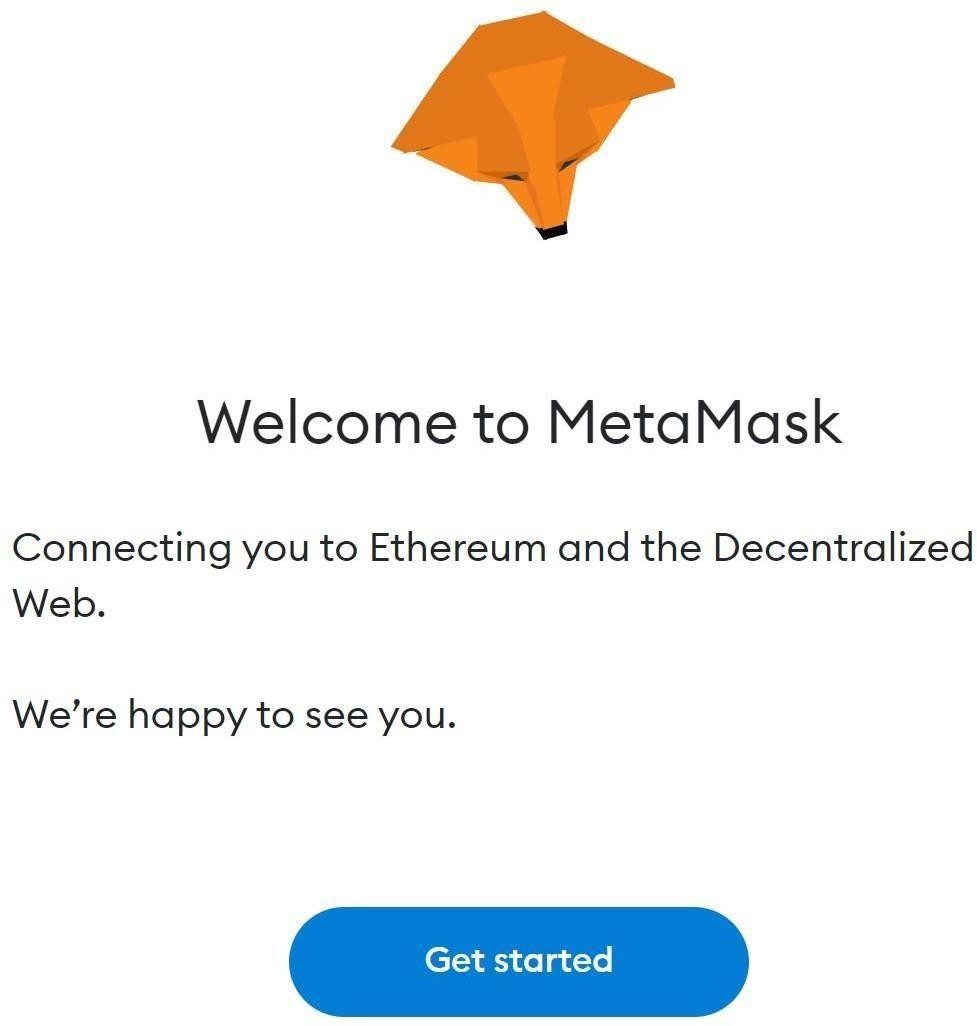
**Class** : BE **Div**: 1 **Batch**: A

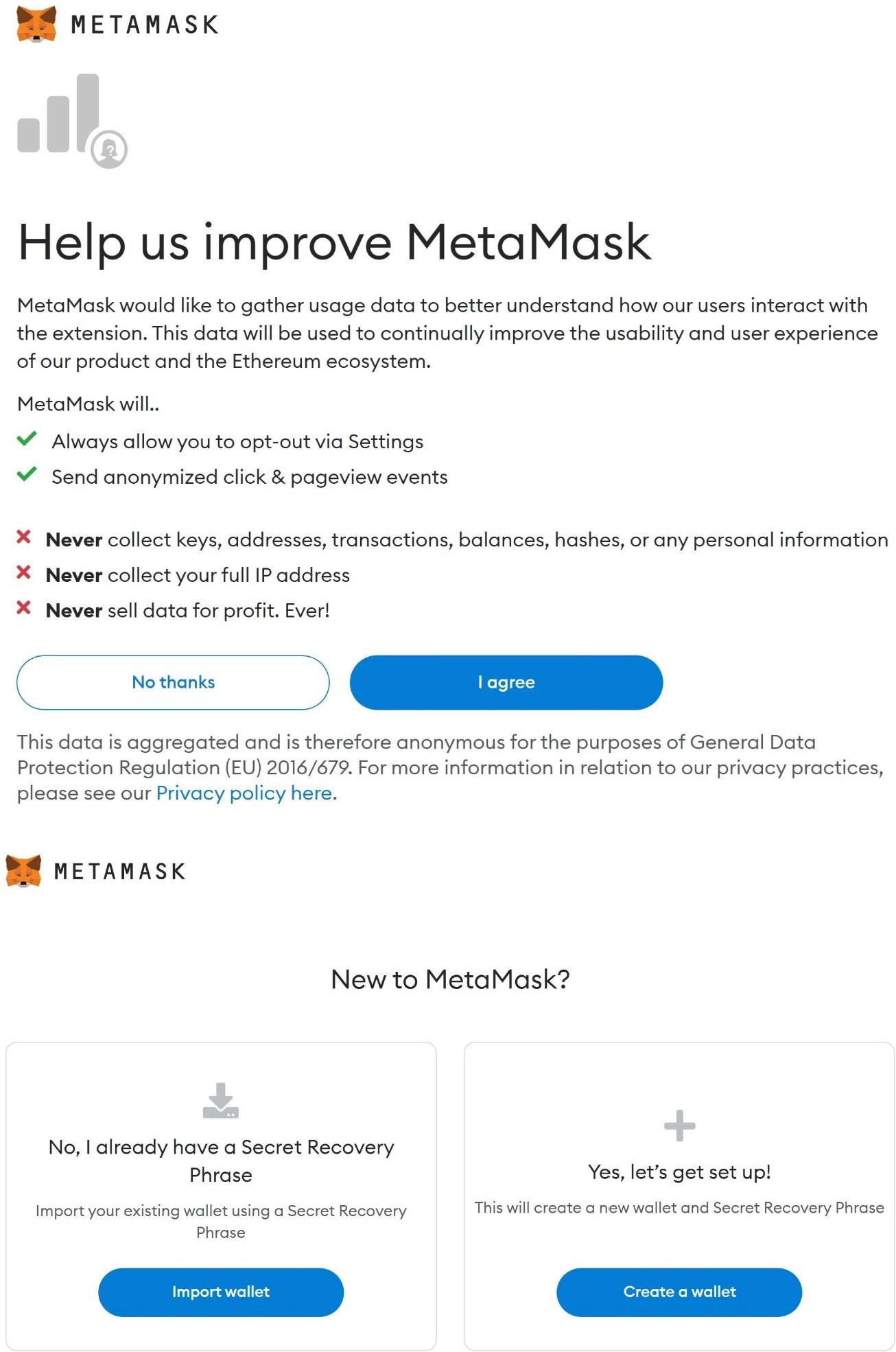
**Roll No: 405A005**

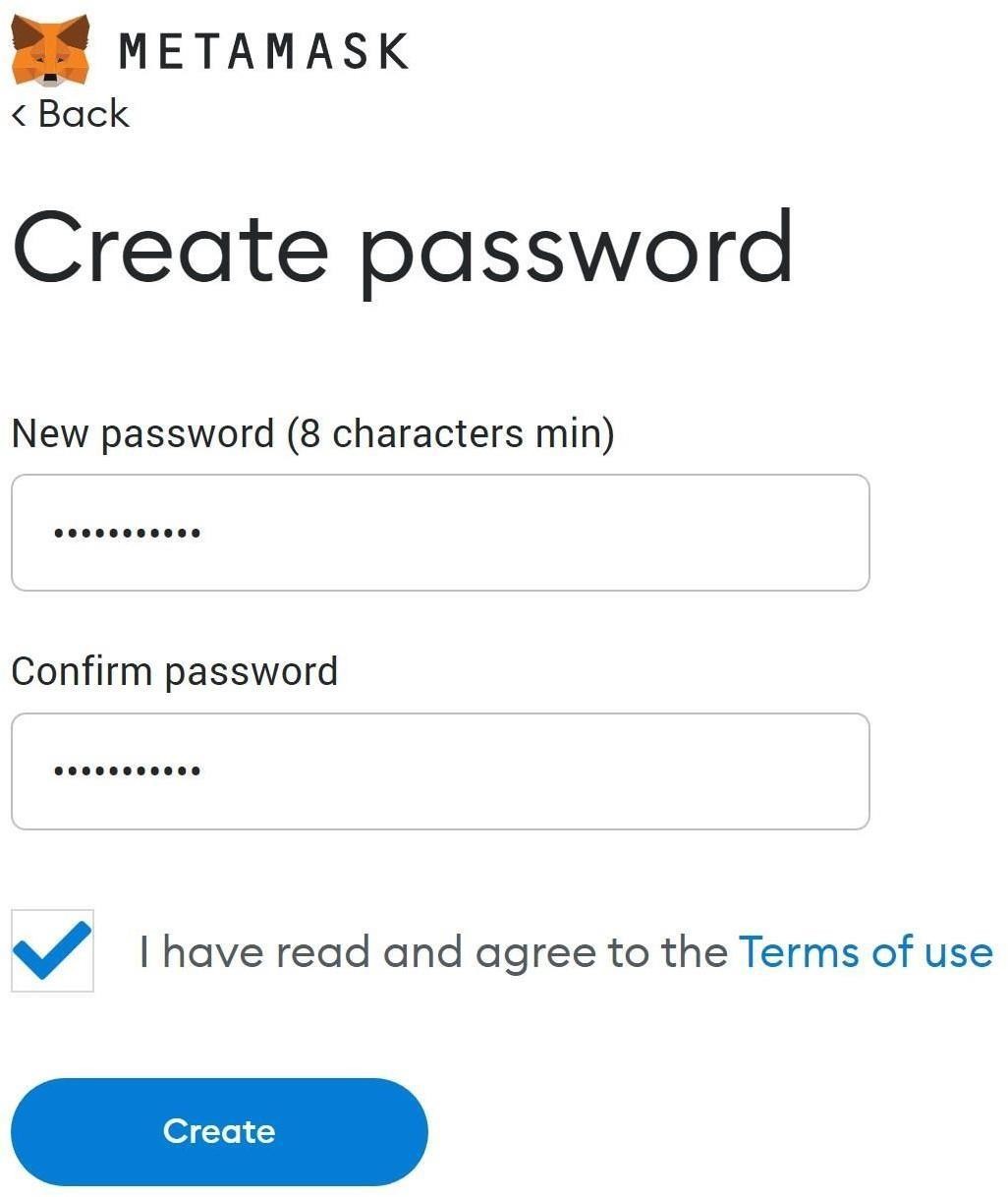
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

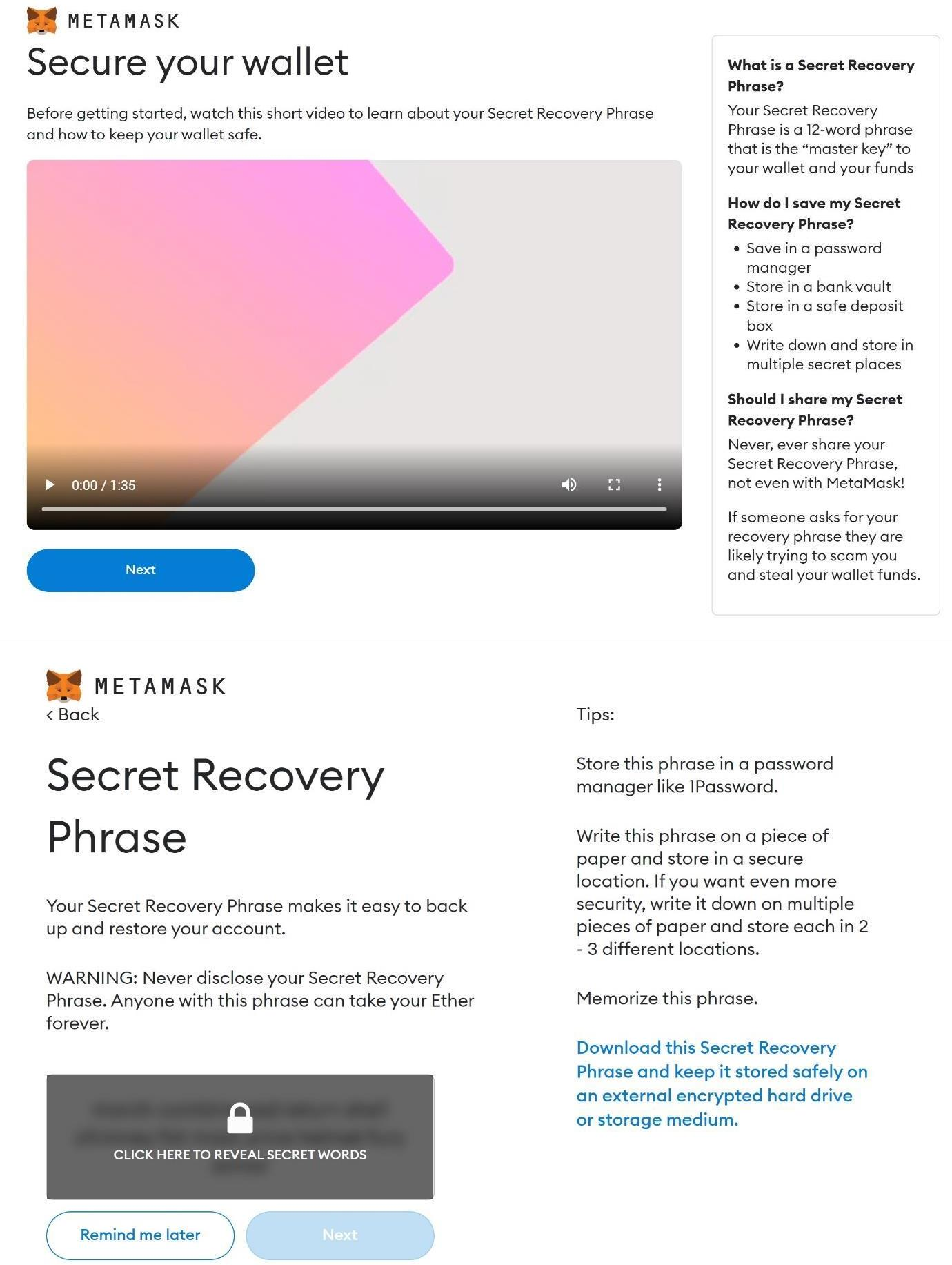


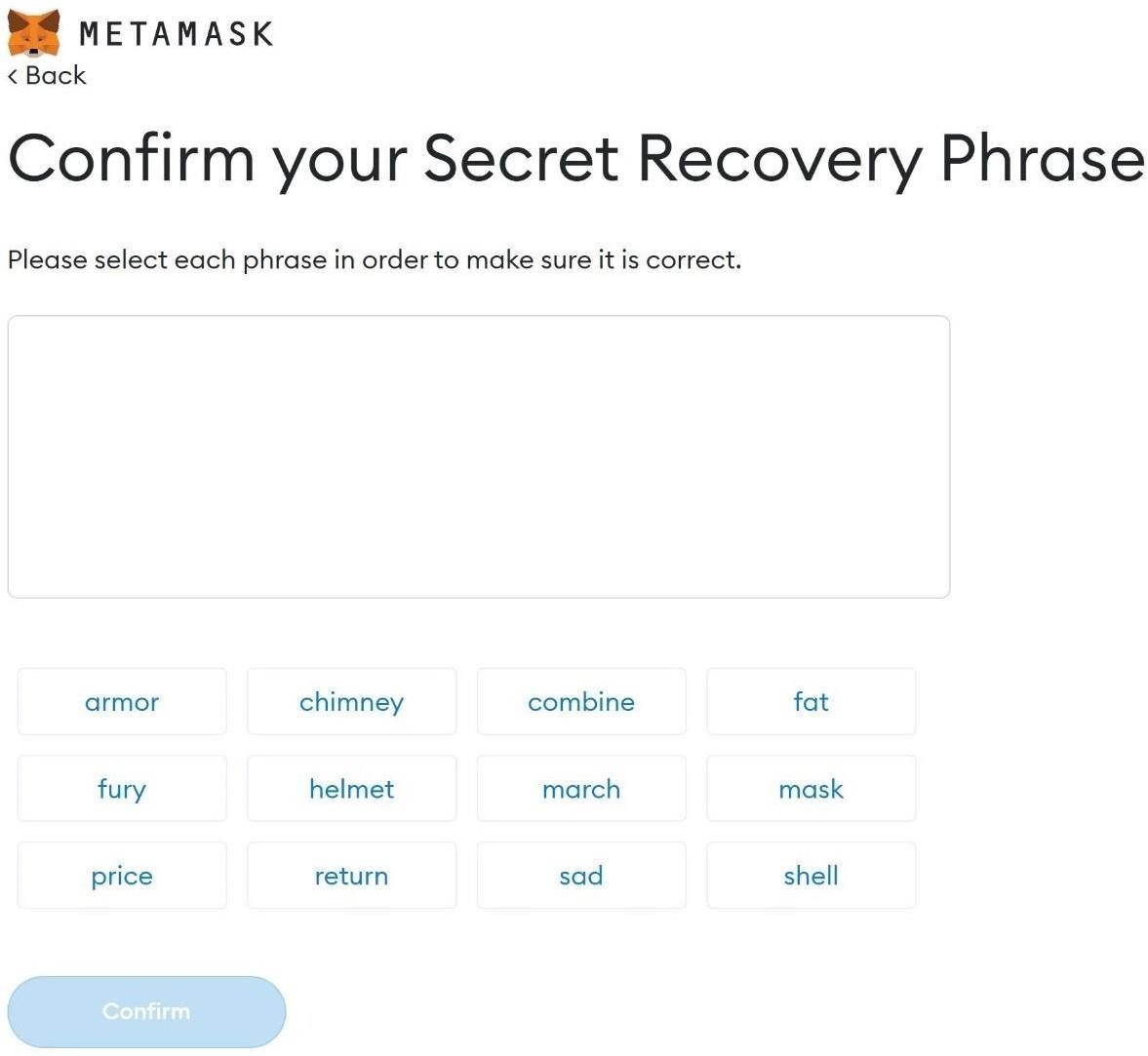


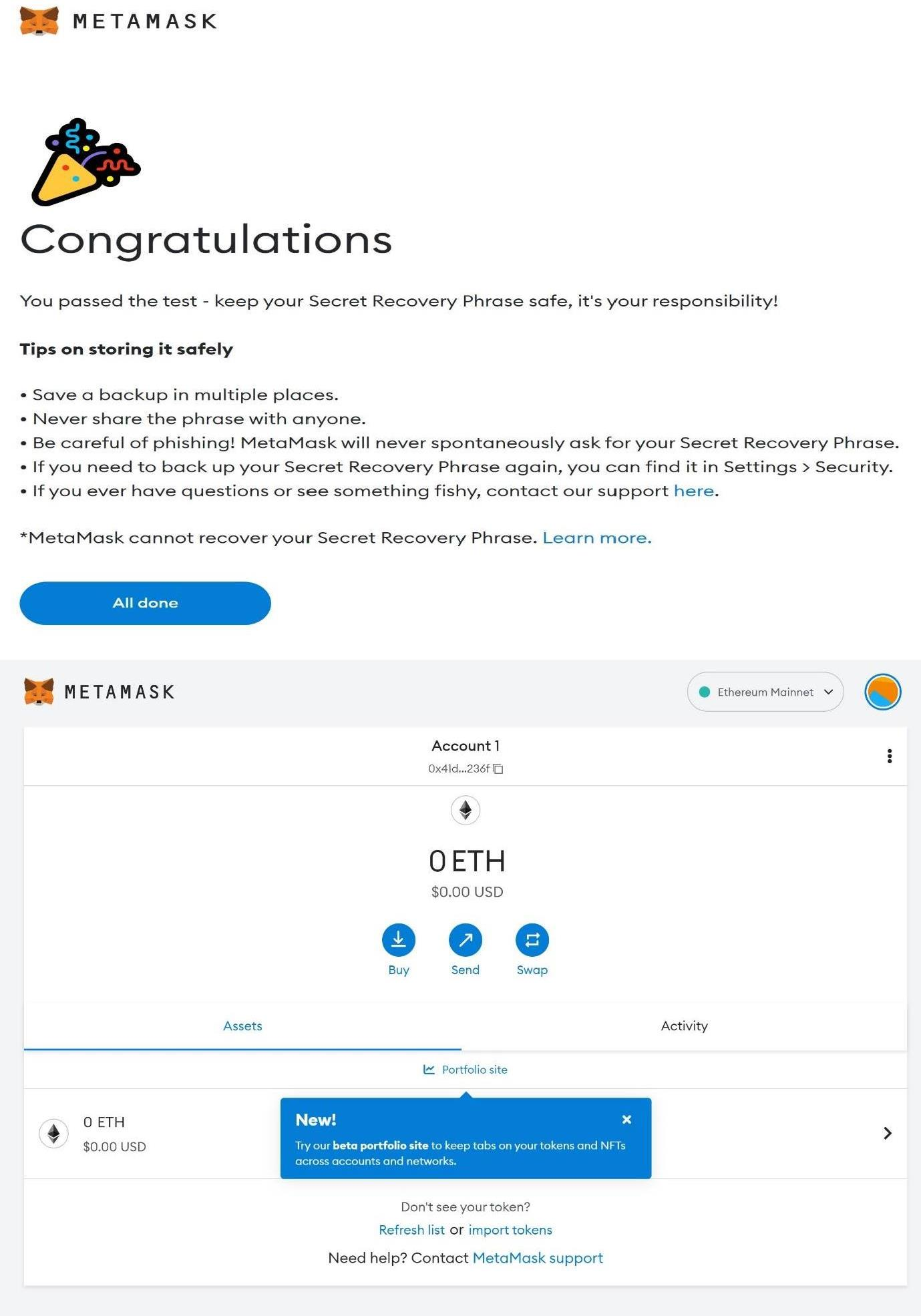












\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Assignment No: 2**

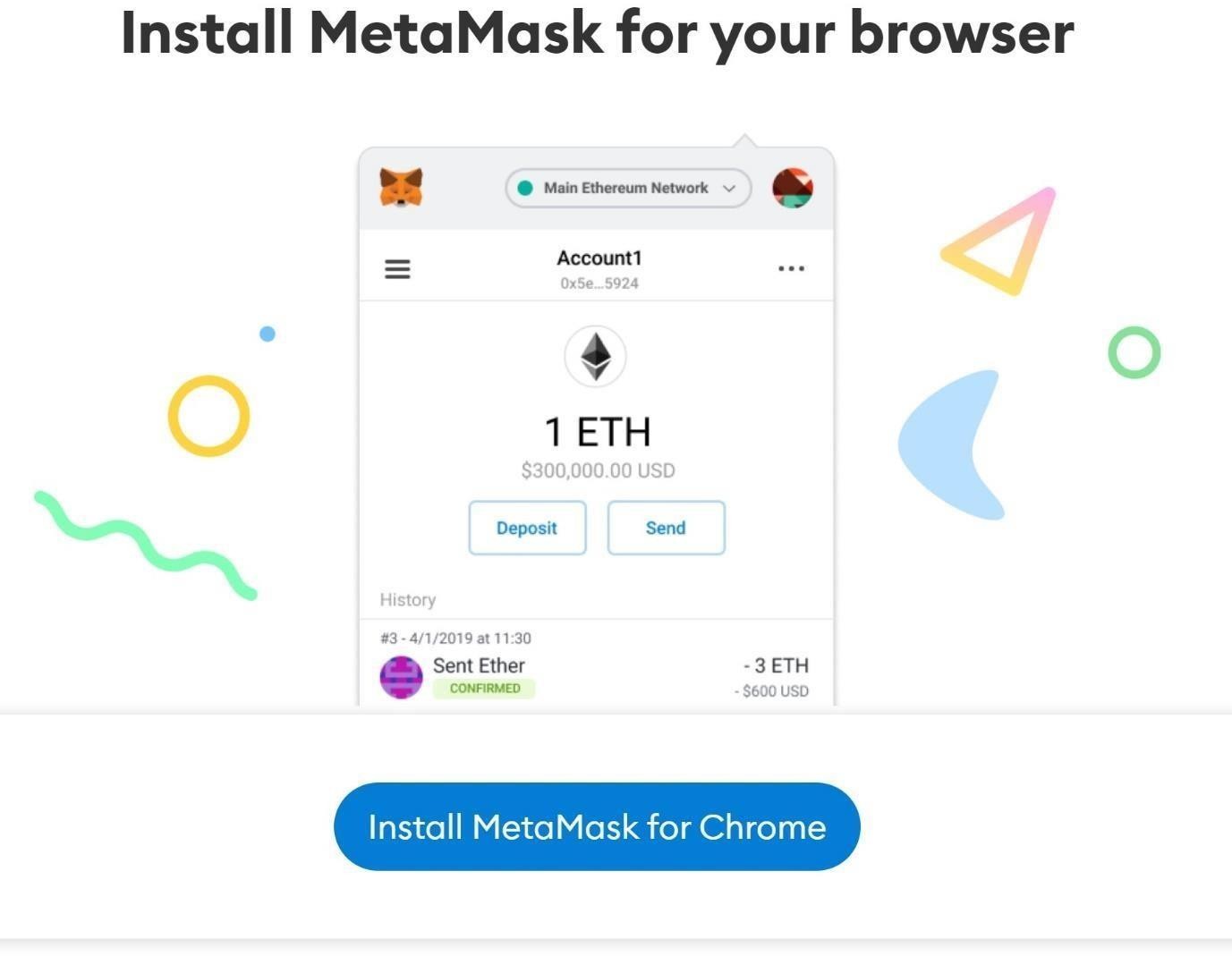
**Title Name:** Create your own wallet using Metamask for crypto transactions.

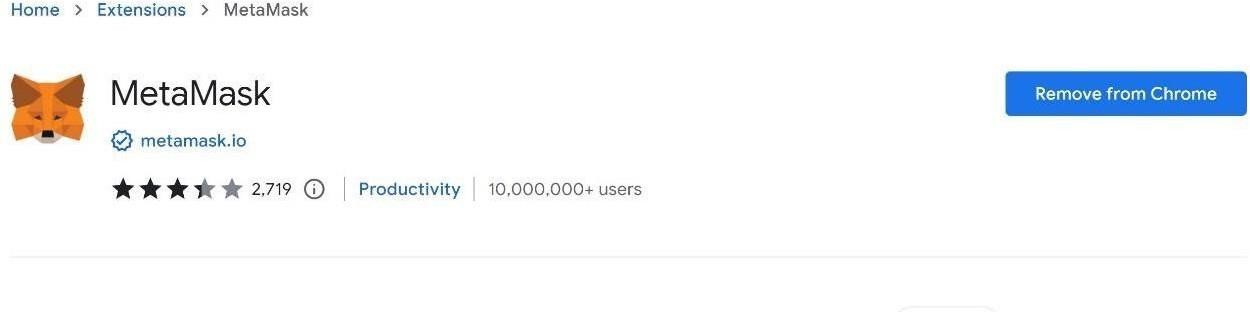
**Name**: Aditi Shivani

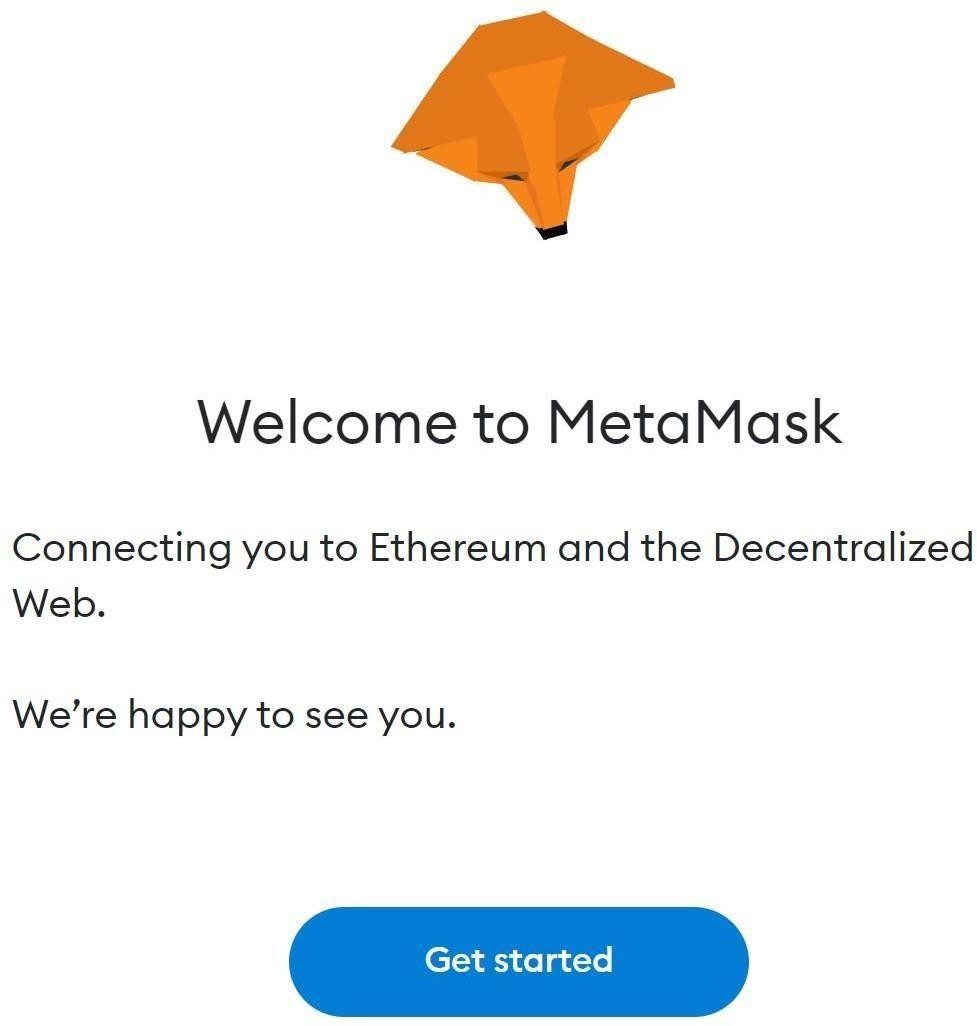
**Class** : BE **Div**: 1 **Batch**: A

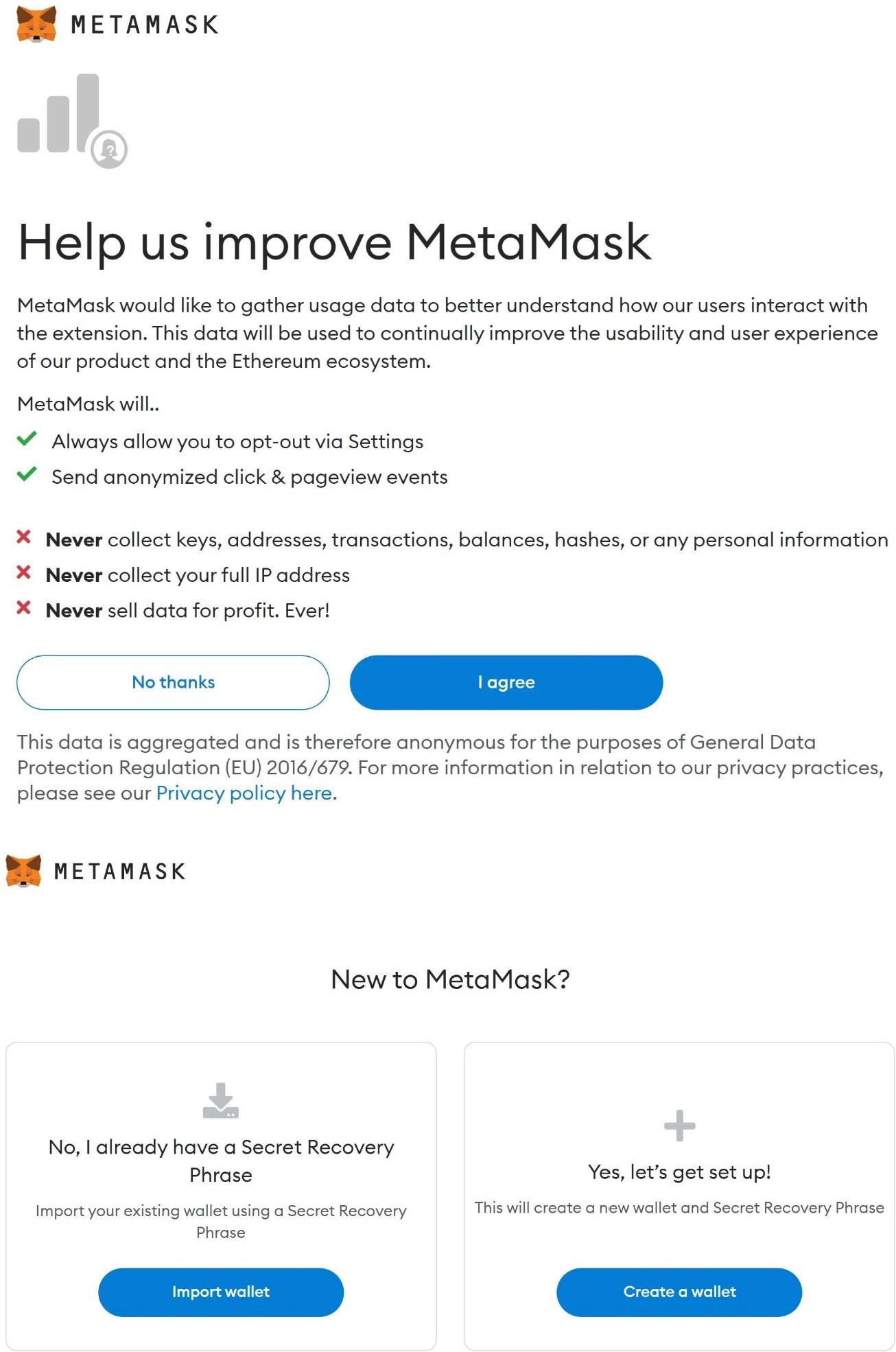
**Roll No: 405A005**

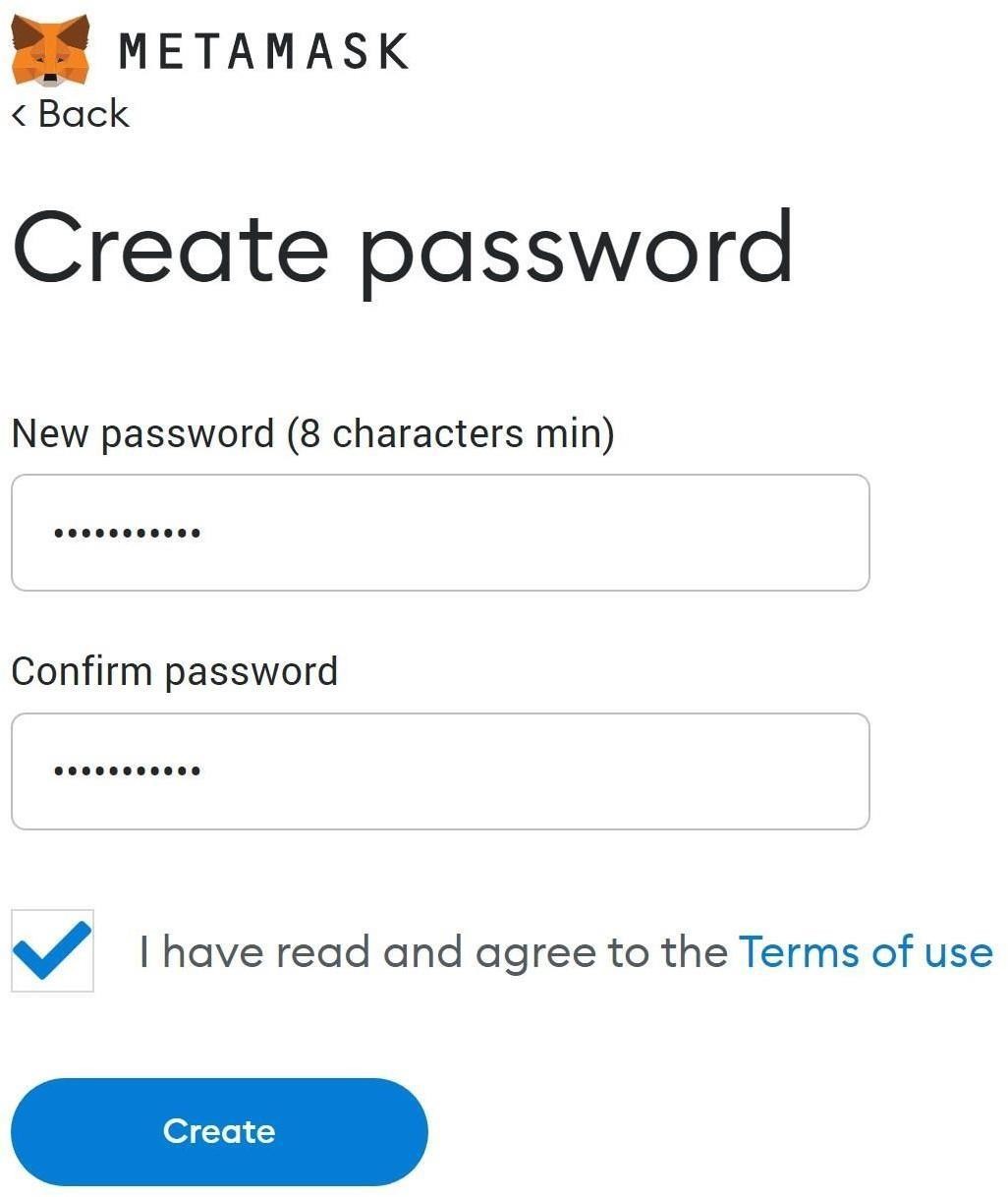
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

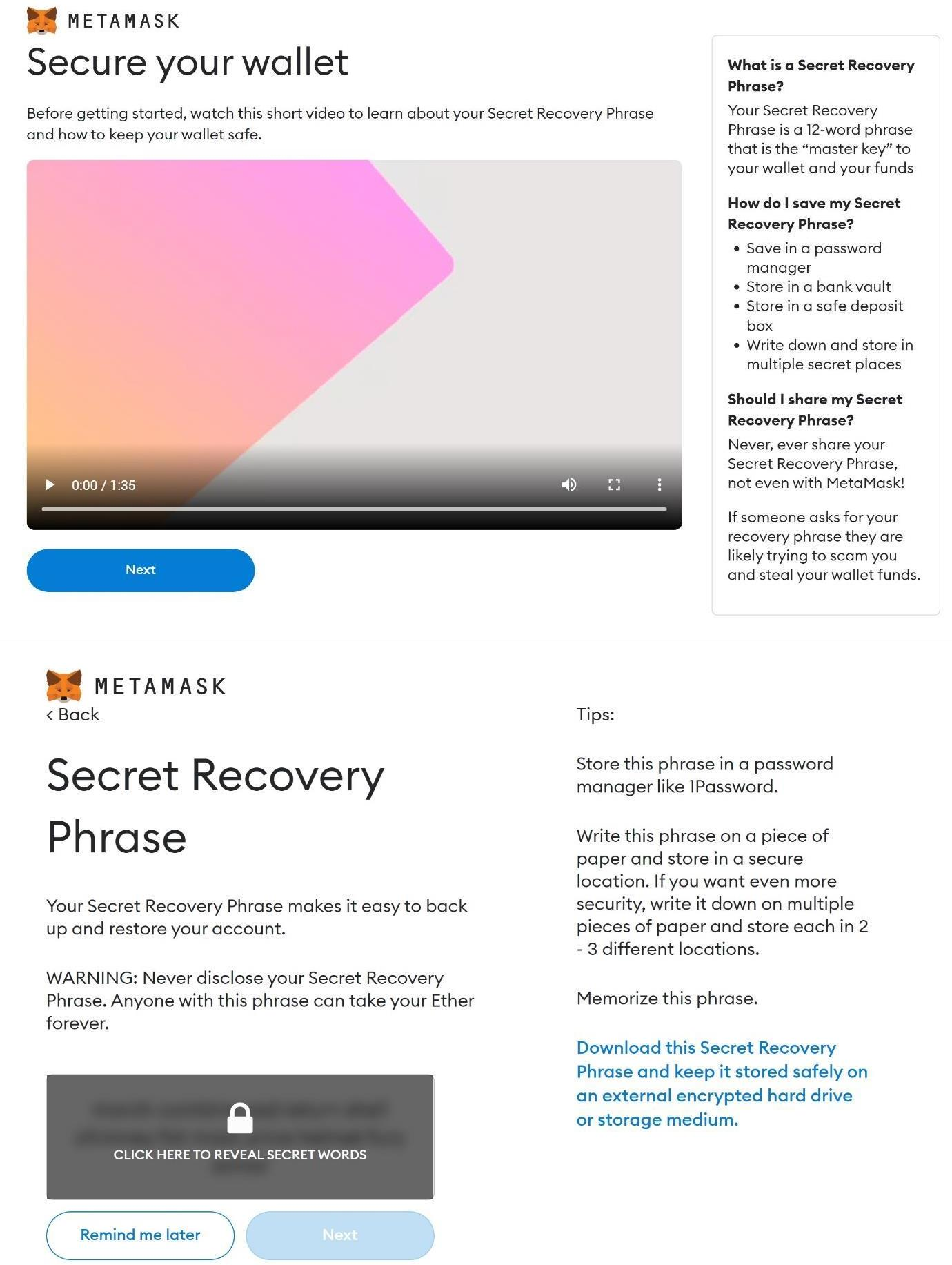


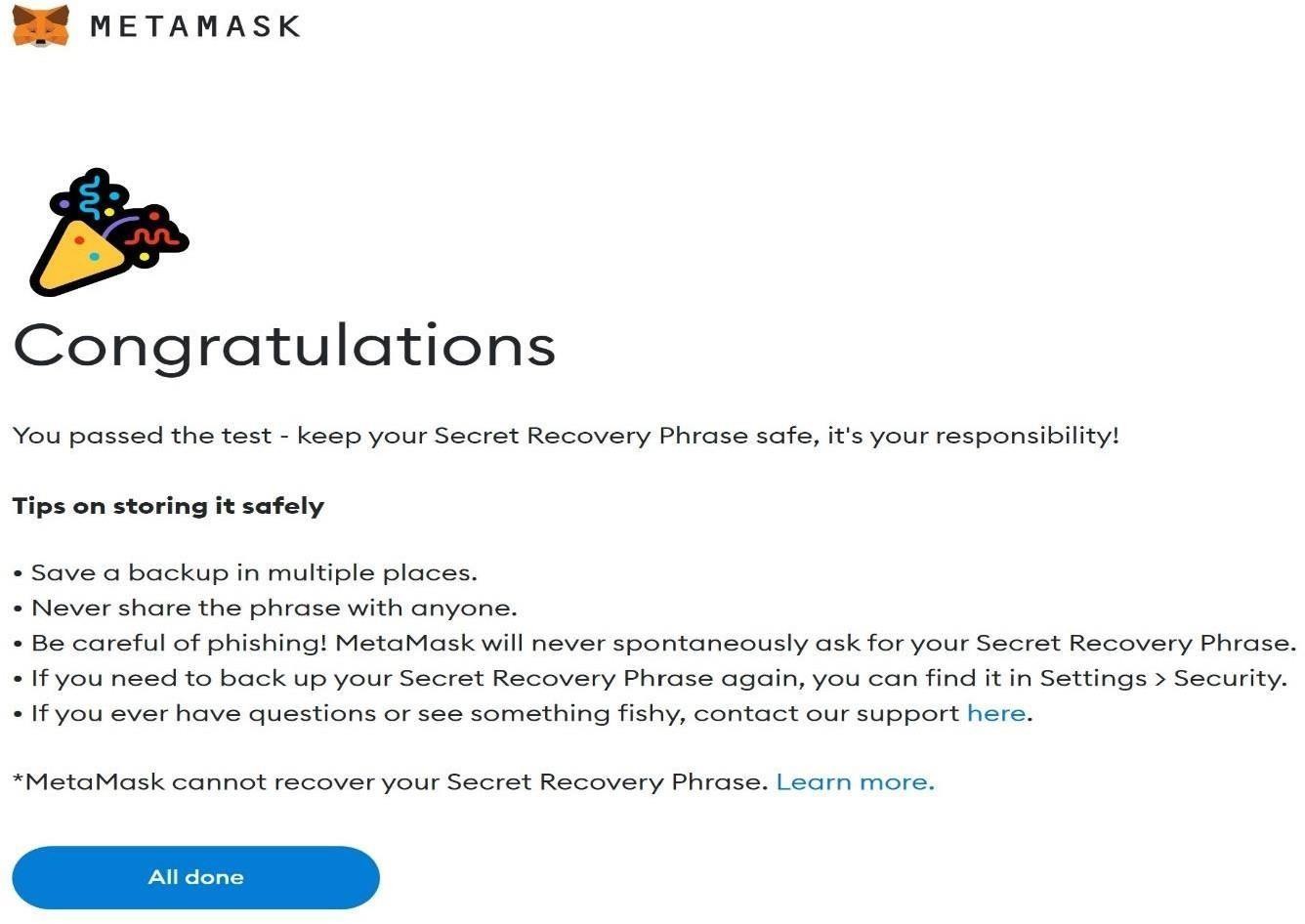


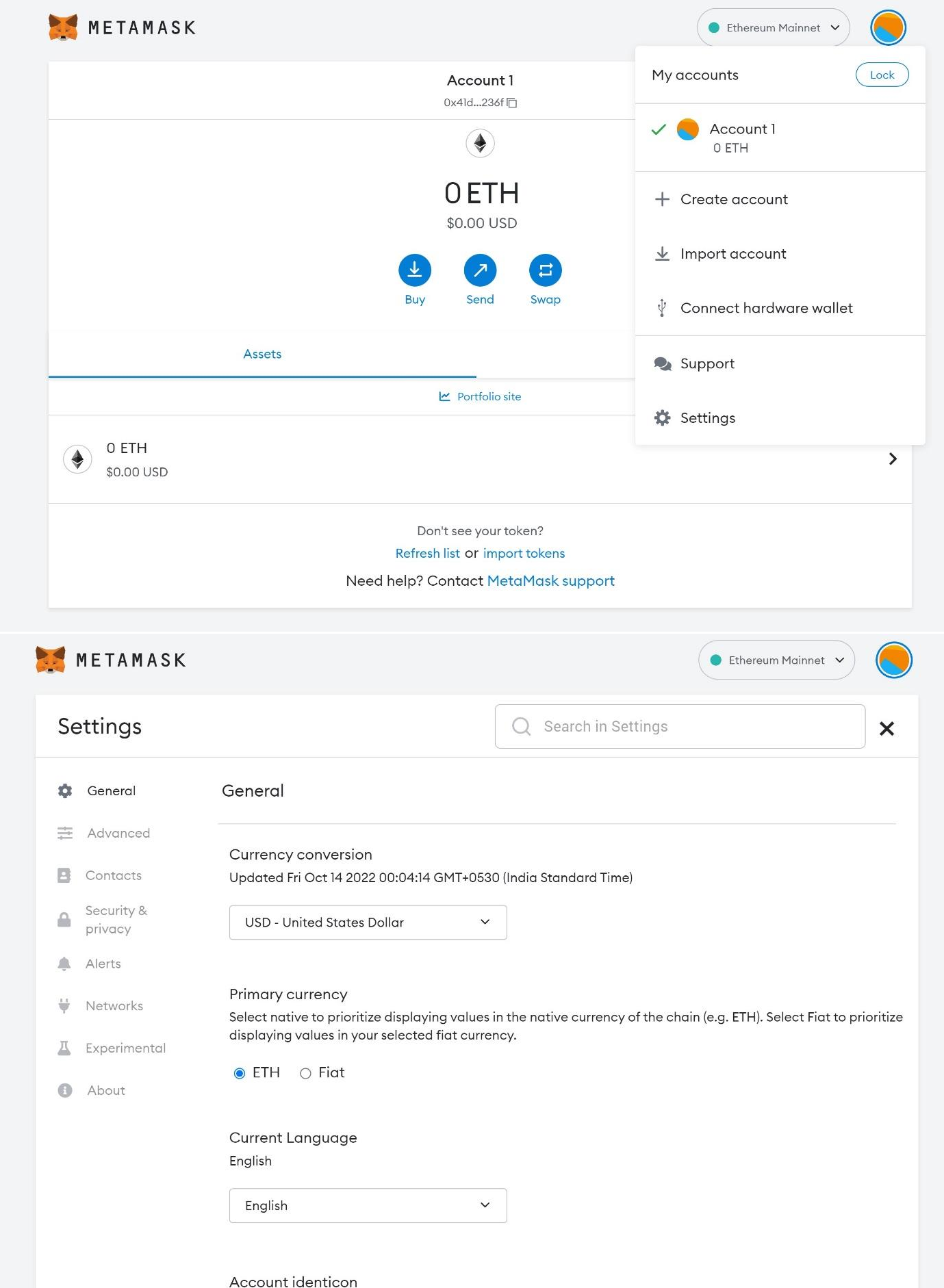


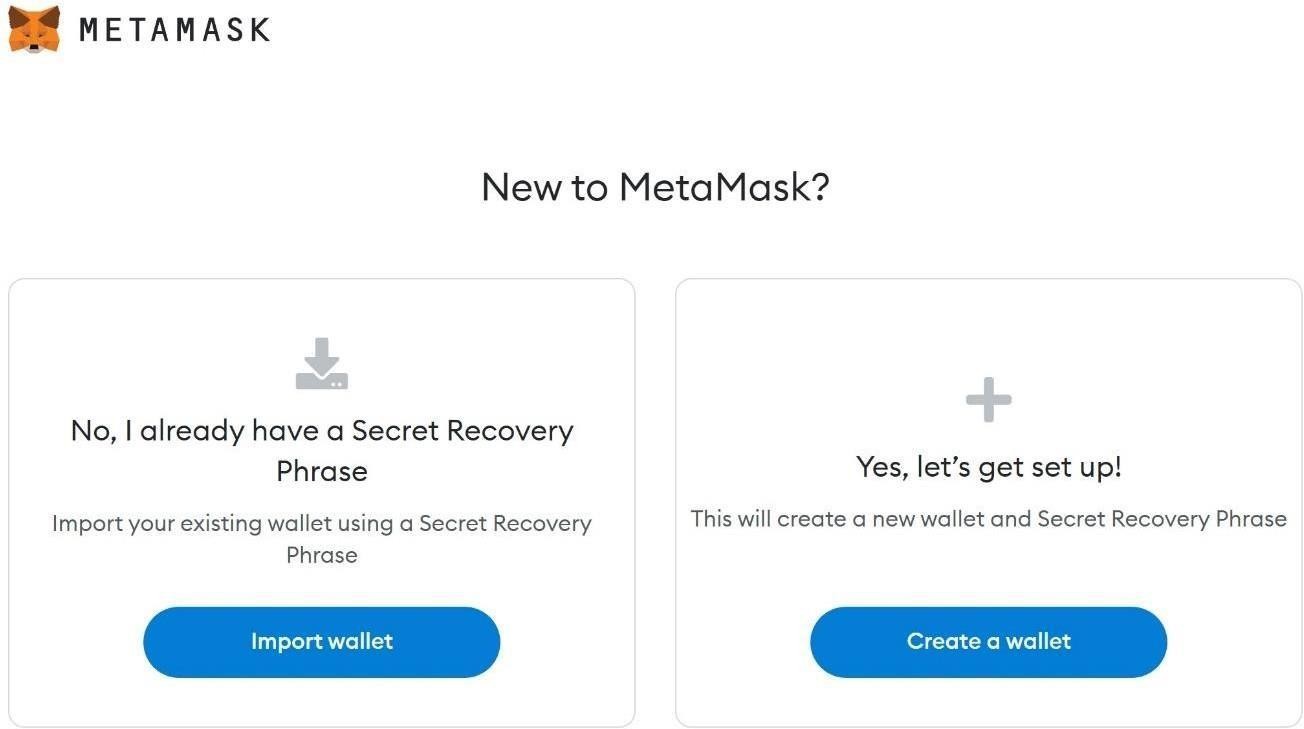


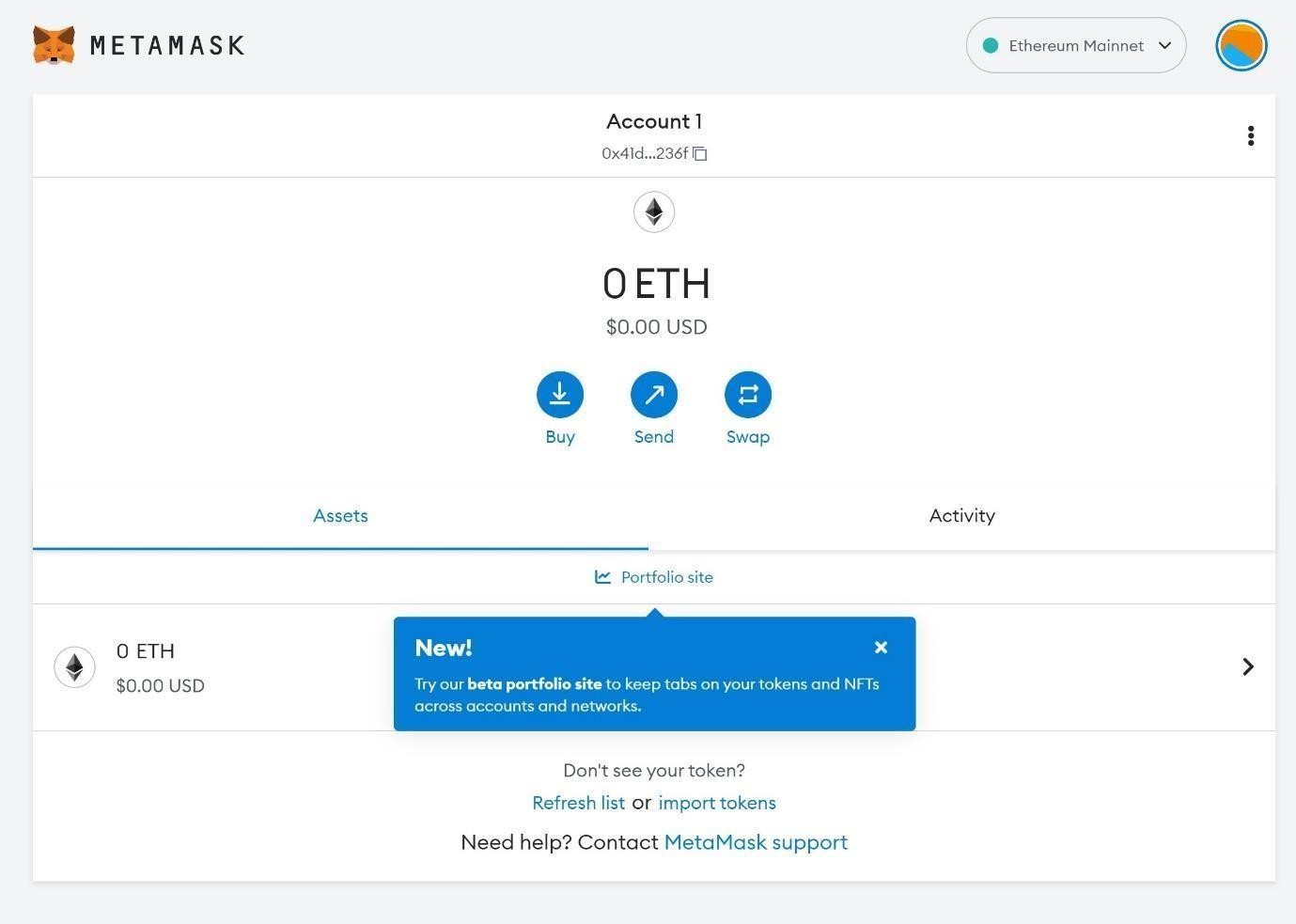












\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Assignment No: 3**

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

**Name**: Aditi Shivani

**Class** : BE **Div**: 1 **Batch**: A

**Roll No: 405A005**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Deposit money into the account:

// pragma is the directive through which we write the smart contract pragma solidity 0.6.0;

// Creating a contract contract wallet{

address payable public Owner;

// mapping is created for mapping address=>uint for transaction mapping(address=>uint) Amount;

// Defining a Constructor constructor() public payable

{

// msg.sender is the address of the person who has currently deployed contract Owner = msg.sender

Amount[Owner] = msg.value;

}

modifier onlyOwner()

{

require(Owner == msg.sender);

}

function sendMoney (address payable receiver, uint amount) public payable onlyOwner

{

require( receiver.balance>0); require(amount>0); Amount[Owner] -= amount; Amount[receiver] += amount;

}

function ReceiveMoney() public payable

{

}

function CheckBalance\_contractAccount() public view onlyOwner returns(uint)

{

return address(this).balance;

}

function CheckBalance() public view onlyOwner returns(uint)

{

return Amount[Owner];

}

}

# Withdrawals and Account Balances:

Here’s the code to accept deposits and track account balances: pragma solidity ^0.4.19;

contract Bank {

mapping(address => uint256) public balanceOf; *// balances, indexed by addresses*

function deposit(uint256 amount) public payable { require(msg.value == amount);

balanceOf[msg.sender] += amount; *// adjust the account's balance*

}

}

Given the balance of mapping from account addresses to ether amounts, the remaining code for a fully- functional bank contract is pretty small. Simply add a withdrawal function

bank.sol

pragma solidity ^0.4.19; contract Bank {

mapping(address => uint256) public balanceOf; // balances, indexed by addresses function deposit(uint256 amount) public payable {

require(msg.value == amount);

balanceOf[msg.sender] += amount; // adjust the account's balance

}

function withdraw(uint256 amount) public {

require(amount <= balanceOf[msg.sender]);balanceOf[msg.sender] -= amount; msg.sender.transfer(amount);

}

}

-The require(amount <= balances[msg.sender]) checks to make sure the sender hassufficient funds to cover the requested withdrawal. If not, then the transaction abortswithout making any state changes or ether transfers.

-The balanceOf mapping must be updated to reflect the lowered residual amount afterthe withdrawal.

-The funds must be sent to the sender requesting the withdrawal.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Assignment No: 4**

**Title Name:** Write a program in solidity to create Student data. and Deploy this as smart contract on Ethereum and Observe the transaction fee and Gas values

**Name**: Aditi Shivani

**Class** : BE **Div**: 1 **Batch**: A

**Roll No: 405A005**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Arrays in Solidity

The array is a special data structure used to create a list of similar type values. The array can be of fixed size and dynamic-sized. With the help of index elements can be accessed easily. below is a sample code to create, and access a fixed-sized array in solidity.

pragma solidity >= 0.5.0 < 0.9.0; contract Array {

uint [4] public arr = [10, 20, 30, 40];

function setter(uint index, uint value) public { arr[index] = value;

}

function length() public view returns(uint) { return arr.length;

}

}

You can compile and deploy the code to try changing the array elements with an index and printing the array length.

Creating Dynamic Array

A dynamic array is an array where we can insert any number of elements and delete the details easily using an index. So solidity has functions like push and pops like python, making it easy to create a dynamic array. Below is a code using which you can create a dynamic array. After writing code, compiles and deploy the code by visiting the deploy section in the left-side navigation bar. After that, try inserting and deleting some elements from an array.

pragma solidity >= 0.5.0 < 0.9.0; contract Array {

uint [] public arr;

function PushElement(uint item) public { arr.push(item);

}

function Length() public view returns(uint) { return arr.length;

}

function PopElement() public { arr.pop();

}

}

Structure in Solidity

The structure is a user-defined data type that stores more than one data member of different data types. As in array, we can only store elements of the same data type, but in structure, you can keep elements of different data types used to create multiple collections. The structure can be made outside and inside the contract storage, and the Structure keyword can be used to declare the form. The structure is storage

type, meaning we use it in-store only, and if we want to use it in function, then we need to use the

struct Student { uint rollNo; string name;

}

contract Demo { Student public s1;

constructor(uint \_rollNo, string memory \_name) { s1.rollNo = \_rollNo;

s1.name = \_name;

}

// to change the value we have to implement a setter function function changeValue(uint \_rollNo, string memory \_name) public { Student memory new\_student = Student( {

rollNo : \_rollNo, name : \_name

});

s1 = new\_student;

}

}

Fallback:

pragma solidity ^0.4.0;

// Creating a contract contract fback

{

// Declaring the state variable uint x;

// Mapping of addresses to their balances mapping(address => uint) balance;

// Creating a constructor constructor() public

{

// Set x to default

// value of 10 x=10;

}

// Creating a function

function SetX(uint \_x) public returns(bool)

{

// Set x to the

// value sent x=\_x;

return true;

}

// This fallback function

// will keep all the Ether function() public payable

{

balance[msg.sender] += msg.value;

}

}

// Creating the sender contract contract Sender

{

function transfer() public payable

{

// Address of Fback contract address \_receiver =

0xbcD310867F1b74142c2f5776404b6bd97165FA56;

// Transfers 100 Eth to above contract

\_receiver.transfer(100);

}

}

Create a Smart Contract with CRUD Functionality

We have excellent theoretical and hands-on practical knowledge about solidity, and now you can create a primary smart contract like hello world, getter, and setter contracts. So it’s a great time to try making some functional smart contracts, and the best way to try all the things in one code is to create one program that performs all CRUD operations.

A sample smart contract code to create ERC20 tokens pragma solidity ^0.4.0;

import "./ERC20.sol"; contract myToken is ERC20{

mapping(address =&gt;uint256) public amount; uint256 totalAmount;

string tokenName; string tokenSymbol; uint256 decimal;

constructor() public{ totalAmount = 10000 \* 10\*\*18;

amount[msg.sender]=totalAmount; tokenName="Mytoken"; tokenSymbol="Mytoken"; decimal=18;

}

function totalSupply() public view returns(uint256){ return totalAmount;

}

function balanceOf(address to\_who) public view returns(uint256){

return amount[to\_who];

}

function transfer(address to\_a,uint256 \_value) public returns(bool){ require(\_value&lt;=amount[msg.sender]); amount[msg.sender]=amount[msg.sender]-\_value; amount[to\_a]=amount[to\_a]+\_value;

return true;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Assignment No: 7 (Mini Project)**

**Title Name:** Develop a Blockchain based application for transparent and genuine charity

**Name**: Aditi Shivani

**Class** : BE **Div**: 1

**Batch**: A

**Roll No: 405A005**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Title**: Develop a Blockchain based application for transparent and genuine charity

**Problem Statement**: Develop a Blockchain based application for transparent and genuine charity

**Prerequisites**: Blockchain Technology

**Objectives**: Develop a Blockchain based application for transparent and genuine charity

**Theory**:

A series of scandals have rocked the way that the public perceives the typical charity, and trust in charitable organizations. It’s no coincidence that headline-grabbing scandals in the world of philanthropy coincided with the decline in faith toward these organizations.

The Blockchain build trust with donors, recipients, and other stakeholders reach the right people and improve administration costs and efficacy. Show donors the difference their donation makes, acquire funds rapidly through crowdfunding and hand control to the people you help.

The primary administrative cost for charities is associated with fundraising and marketing or getting the word out about the charity. Blockchain-based platforms are aiming to provide charities with a marketplace to reach a ready-to-give audience, and these platforms take far fewer fees than traditional marketing and fundraising agencies.

Facilities: Meta-mask Google Extension Input:

pragma solidity ^0.8.6;

import "@openzeppelin/contracts/utils/Counters.sol"; contract Charity {

using Counters for Counters.Counter;

event CampaignStarted(bytes32 campaignId, address initiator);

event WithdrawFunds(bytes32 campaignId, address initiator, uint256 amount); event FundsDonated(bytes32 campaignId, address donor, uint256 amount);

Counters.Counter public \_campaignCount;

struct Campaign { string title; string imgUrl;

string description; uint256 fundsRaised;

bool isLive; address initiator; uint256 deadline; uint256 balance;

}

mapping(bytes32=>Campaign) public \_campaigns; mapping(address=>mapping(bytes32=>uint256)) public userCampaignDonations;

constructor(){

}

function generateCampaignId(address initiator, string calldata title, string calldata description) public pure returns(bytes32) {

bytes32 campaignId = keccak256(abi.encodePacked(title, description, initiator)); return campaignId;

}

function startCampaign( string calldata title, string calldata description, string calldata imgUrl, uint256 deadline ) public {

// generate a campaignID

// using the title, description and the address of the initiator

bytes32 campaignId = generateCampaignId(msg.sender, title, description);

// get a reference to the campaign with the generated Id Campaign storage campaign = \_campaigns[campaignId];

// require that the campaign is not live yet. require(!campaign.isLive, "Campaign exists");

// require the current time to be less than the campaign deadline require(block.timestamp < deadline, "Campaign ended");

campaign.title = title; campaign.description = description; campaign.initiator = msg.sender; campaign.imgUrl = imgUrl; campaign.deadline = deadline; campaign.isLive = true;

// increment the total number of charity campaigns created

\_campaignCount.increment();

// emit an event to the blockchain

emit CampaignStarted(campaignId, msg.sender);

}

function endCampaign() public {

}

// allows users to donate to a charity campaign of their choice function donateToCampaign(bytes32 campaignId) public payable {

// get campaign details with the given campaign Campaign storage campaign = \_campaigns[campaignId];

// end the campaign if the deadline is exceeded if(block.timestamp > campaign.deadline){

campaign.isLive = false;

}

// require the campaign has not ended

require(block.timestamp < campaign.deadline, "Campaign has ended");

uint256 amountToDonate = msg.value; require(amountToDonate > 0, "Wrong ETH value");

// increase the campaign balance by the amount donated; campaign.fundsRaised += amountToDonate; campaign.balance += amountToDonate;

// keep track of users donation history userCampaignDonations[msg.sender][campaignId] = amountToDonate;

// emit FundsDonated event

emit FundsDonated(campaignId, msg.sender, amountToDonate);

}

// returns the details of a campaign given the campaignId

function getCampaign(bytes32 campaignId) public view returns(Campaign memory) {

return \_campaigns[campaignId];

}

function withdrawCampaignFunds(bytes32 campaignId) public { Campaign storage campaign = \_campaigns[campaignId];

// require the msg.sender is the creator of the campaign require(msg.sender == campaign.initiator, "Not campaign initiator");

// require the campaign has ended require(!campaign.isLive, "campaign is still active");

require(block.timestamp > campaign.deadline, "Campaign is still active");

// require the campaign has funds to be withdrawn require(campaign.balance > 0, "No funds to withdraw");

uint256 amountToWithdraw = campaign.balance;

// zero the campaign balance campaign.balance = 0;

// transfer the balance to the initiator address; payable(campaign.initiator).transfer(amountToWithdraw);

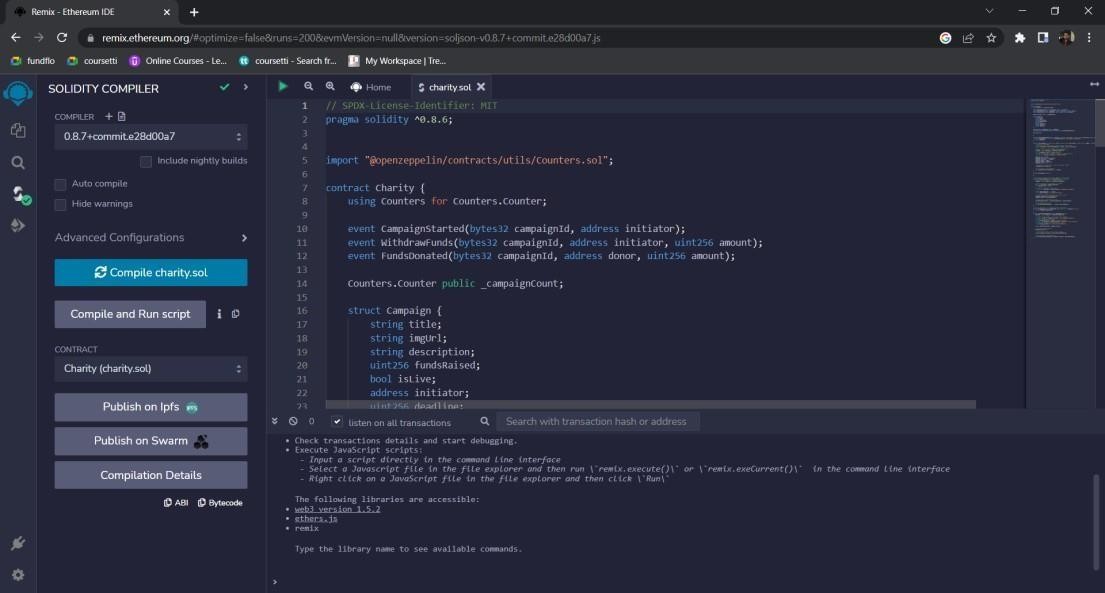
// emit an event to the blockchain

emit WithdrawFunds(campaignId, campaign.initiator, amountToWithdraw);

}

}

Output:



Conclusion: We have developed a developed a Blockchain based application for transparent and genuine charity successfully.