Practical No - 1

Page No
Date

Problem Statement :-
Installation of metamask & study spending Ether per transaction

Theory :-

Metamask is a type of Ethereum wallet that bridges the gap between user interfaces & ethereum (for example, mist browsers, DAPPS) & regular web (for example, Google chrome, Mozilla firefox, websites). Its function is to inject the Javascript library called web3.js into the namespace of each page of your browser loads. Webjs is written by the Ethereum core team. Metamask is mainly used as ~~pa~~ plugin in the web browser. Metamask is one of the most popular browser extensions that serves as a way of storing your Ethereum & other ERC-20 Tokens. The extension is free & secure, allowing web application to read & interact with Ethereum's blockchain. Let's walk through the steps to install it on google chrome.

Step ① :- Install Metamask on your browser.

To create a new wallet. You have to install the extensions first. Depending on your browser, there are different marketplace to find it. Most browser have metamask on their stores so its not that hard to see it.

- Click on install metamask as a google chrome extension.
- Click ~~on~~ on Add to chrome.
- Click on Add Extension

Page No.
Date

Study Spending Ether per transaction.

- open the Metamask app
- Select "Send"
- Scan your recipient's QR code or paste their public address. After sending to a specific address it will save in your "recents" which you can also rename to remember them easier.
- Complete the transaction, you will have an option to see the network fee before you confirm. Choosing a slower the fee, the longer your transaction will take to be proceed by miners.

Conclusion :-

In this way, we have successfully installed & understood spending transaction on ethereum.

Practical No - 2

Page No.
Date

Problem Statement :-

Create your own wallet using metamask for the crypto-transaction.

Theory :-

- Once, the installation is completed this page will be display click on the get started button.
- This is the first time creating a wallet. So click the create a wallet button. If there is already a wallet then import the already created using the import wallet button.
- Click the I agree button to allow data to be collected to help improve metamask or else click on no task thanks button. The wallet can still be created even if the user will click on the no thanks button.
- Create a password for your wallet. This password is to be entered every time the browser is launched & ~~wat~~ wants to use metamask. A new password needs to be created if chrome is uninstalled or if there is a switching off browser. In that case go through the import wallet button. This is because metamask stores the keys in the browser. Agree to Terms of use.
- click on the dark area which says click here to reveal secret words to get your secret phrase.
- This is the most important step Back up your secret phrase properly. Do not store your secret phrase on your computer. Please read everything on this screen until you understand it completely before preceeding. The secret phrase is the only way to access your wallet.

If you forget your password once done click the next button.

- Click the buttons respective to be order of the words in your seed phrases. In other words, type the seed phrases using the button on screen. If done correctly the confirm button should turn blue.
- Click the confirm button. Please follow the tips mentioned.
- Once you can see the balance & copy the address of the account by clicking on the Account 1 area.
- One can adress metamask in your browser by clicking the foxface icon on the top right. If there the foxface icon is not visible then click on the puzzle piece icon right next to it.

Conclusion :-

In this way, we have learned & understood how to create an account on metamask ? what is a seed phrases ? & how to check the balance of your wallet account.

dude

Practical No - 3

Page No.
Date

Problem Statement :-
Write a smart contract on a test network, for the bank account of a customer for following operations.
- Deposit money
- Withdraw money
- Show balance

Theory :-

What is smart contract ?

Smart contract are simply program that are stored on a blockchain that run when predetermined conditions are met. They typically are used to automate the execution of an agreement so that all participant can be immediately certain of the outcome, without any intermediators are involment or time loss. They can also automate a workflow, triggering the next action when conditions are met.

Benefits of a smart contract :-
- Speed, efficiency & Accuracy
- Trust & Transperency
- Security
- Cost Effective

What is Solidity ?

Solidity is an object-oriented, high-level language for implementing smart contracts. Solidity is a curly-bracket language designed to target the Ethereum virtual machine (EVM) its influenced by C++, Python & JavaScript. You can find more details about which languages Solidity

has been influenced by the language influences section. Solidity is statically typed supports inheritance, libraries & complex user defined types among other features.

We will use solidity to develop our basic account smart contract by following the steps below :-

Step ① :- Define licensing & programs "program" (common instructions for compilers about how to treat the source code) regarding smart contract.

Step ② :- Create an empty smart contract.

Step ③ :- Create mapping (acts like a hashtable or dictionary in any other language) to save data regarding balance & whether user exists or not in our smart contract.

Step ④ :- Write a function (A group of reusable code which can be called anywhere in your program) to create user account.

Here the "require" function call is an inbuilt feature in the solidity which allows us to throw an error & revent the transaction if condition is not met in our case are checking if the user exists before creating an account for them. This will throw an error if the account already exists "msg" is a global various variable identifier available in solidity to have a information regarding the transaction like sender, amount sent etc.

with this we are checking the amount sent by the user, if its zero then we create a user with zero balance in their account, otherwise we get provided value in our user account mapping & along with that in either case we set user exists mapping to true for that particular user wallet address to ensure that this user exists.

Step ⑤ :- Write a function to deposite funds in user account for the deposite functionality we have to make our function payable so the user can send crypto payments with the call of the deposit function. So we have made this function public as our previous functions to make sure we can call this function outside for outside of a contract.

for example, from fronted The "return" keyword specifies what the function will return after successful executions. The "deposit" function implements two "require" functions with conditions as "whether" the user "exist & have the sender of the message sent an amount bigger than the Zero". After successfully meeting those two conditions we add "msg.value" to the user Account mapping lostly we return a message to notify our user the fund has been added successfully as successfully as "Deposited Successfully".

Note :- We don't need to handle payment as making a function payble will automatically handle it.

Page No.
Date

Step ⑥ :- Write a function to withdraw funds in users Account. the "withdraw" function takes one argument "amount" & checks whether the user has more than requested in the bank account already. later a user must exist in our bank. Along with that we check whether the required amount the user wants to withdraw is bigger than zero or not.
If all conditions are met we reduce the balance in user account mapping & send the fund to the requested user. On successful execution we send a "withdrawal successful" message back to the user.

Step ⑦ :- Write a function to show the funds in user's account. Here our function "get Balance" is a view function which means the function can only read & cannot update existing or add new data. We even don't need to check whether user exist or not because "msg sender" is the wallet address of logged in user & user Account has only data of end registered user so even if the unknown user calls the function as user does not exists yet the value will become zero (default value for unit.)

Step ⑧ :- Deploy smart contracts on Gueri Testnet press ctrl + c to compile the code, visit deployment tab & select environment as "injected provider

metamask" Click the deploy button & copy the smart contract deployed address you can see address to create smart contract instances in the frontend code.

Conclusion :-

In this way, we have created a smart contract on a blockchain using solidity language & an injected web3 provider on Ganache test network.

Practical No – 4

Page No.
Date

Problem statement :-

Write a program in solidity to create student data use the following constructors -

- Structures
- Arrays
- Fallback

Deploy this as a smart contract on ethereum & observe the transaction fee & gas values.

Theory :-

What is a "structure (Struct)" in solidity ?

Struct in solidity allows you to create more complicated data types that have multiple properties. You can define your own type by creating a struct.

- They are useful for grouping together related data.
- Structs can be declared outside of a contract & imported in another contract. Generally, its used to represent a record. To define a structure struct keyword is used, which creates a new data type.

What is an "array" in solidity ?

Arrays are data structure that store the fixed collection of elements of the same data types in which each & every element has specific location called index instead of creating numerous individual variable of the same type, we just declare one array of the required size & store the elements in the array & can be accessed using the index. In solidity, an array can be

of fixed size or dynamic size. Arrays have a continuous memory location, where the lowest corresponds to the first elements while the highest represents the last.

What is "fallback" in solidity ?

Fallback is a function that does not take any arguments & does not return anything.

- It is executed either when
  - A function that does not exist is called or
  - Ether is sent directly to a contract but receive() does not exists or msg.data is not empty.

Fallback has 2300 gas limit when called by transfer or send

- Following are the steps to write a program to create student data.

We will used solidity to develop our student data management smart contract by following steps below :-

Step ① :- Define licensing & programs (common instructions for compilers about how to treat the source code) regarding smart contract.

Step ② :- Create an empty smart contract

Step ③ :- Declare struct to create a custom student data type & initialize a student list array using our newly declared data type to store information.

Page No.
Date

Step ④ :- Create a function to register a new student with the smart contract storage "add student" function requires three arguments to store a student first name, last name & roll id, function signatures has a "public" keyword, which means accessible outside of a contract.

If also has a "returns" keyword specifying that the function returns a "string" value. The "memory" keyword along with the "string" tells EVM (Ethereum Virtual Machine) the data location of the "string" value returned. Here we have specified. "Memory" which means after the function execution we don't need returned value & the EVM will clear & empty the RAM. Same also applies to string in arguments.

Initially we use the "require" built-in function to check if the roll id is not zero. We only need to check for zero because "vint" data type will handle negative value by default as it requires an unsigned integer, hence the name "vint" later we create a student structure using provided arguments & add it to the array to store permanently (at least throughout the lifecycle of a smart contract) At last when a student is successfully added we return a response to the caller as "added student successfully".

Step ⑤ :- Create a function to get student data from the smart contract storage for the execution of this function, smart contract requires one argument called "rollid" its the roll id of the student in the class. This function is "public", "view"

only reads data from blockchain. Kind of a read-only functions & returns data in custom student data type. To track student existence in our smart contract storage we have declare a boolean (true/false) identifier called "found" again if the student is present we need to have a copy of student to send back as response so declaring the "student" identifier with custom data type "student".

Now we have to find the index of the student in our list of students. for that purpose we iterate over each student using a "for loop" & check whether the provided id matches we set "found" to "true" & save the found student in the temporary identifier "student" while breaking out of the loop so we don't have to execute unnecessory interactions out the loop so we don't have to execute. We use the "require" function & check whether a student exist or not. If not then "require" will revert the transaction automatically otherwise we return the required student.

Step ⑥ :- Create a function to remove student from the smart contract storage. This function requires the same argument as the "get student" function. However, instead of returning particular student details, this function will return only a success message at last. We perform all steps same as "get student" function to find whether a user exist or not. Instead of having a copy of a student to return we only save its index (positive in array starting from 0). If we find the student

our "require" check will not cause any error & we will change index of the student in array with last student in array in any & later remove the duplicate copy of last user from the last index of the array using "POP (remove last elements from the array)" method of array.

step ⑦ :- Create a fallback function to handle any operation outside defined scope. The "fallback" in-built function, will allow us to handle common cases of our code. No unwanted activity will be performed on our smart contract.

Deploy & observe the transaction fee & Gas values.

- In remix ide, press ctrl + s to compile the code.
- Go to deployment section & click deploy.
  This will be deployed on a local environment provided by the remix ide.
- Check the "deployed contracts" section & perform the transaction.

As per the observation we can clearly see in the following image that the "view" function which was "get student" required no gas. On the other side "remove student" & "add student" functions have cost us an a mount to execute them.

Page No.
Date

Conclusion :-

In this way, we deployed a smart contract for managing students deployed it on test network (ethereum) & understand about gas values & the transaction fee.