Pravara Rural Education Society's

SIR VISVESVARAYA INSTITUTE OF TECHNOLOGY, Nashik

DEPARTMENT OF INFORMATION TECHNOLOGY

LAB MANUAL

BEIT

SEMESTER-VIII

LABORATORY PRACTICE-VI

BLOCKCHAIN TECHNOLOGY

ACADEMIC YEAR 2022-2023

Ms. Ashwini V. Waje
Assistant Professor

Prerequisites: Programming skills: javascript, react.js

Course Objectives:

1. To acquaint students with the basic skills required for adopting to crypto currency & block chain

2. To acquire knowledge about consensus algorithms and its working.

Course Outcomes:

On completion of the course, students will be able to–

1.To implement small blockchain experimentations.

2. Identify Consensus mechanism for Blockchain Application

Guidelines for Instructor's Manual

The instructor's manual is to be developed as hands - on resource and reference. The instructor's manual need to include prologue (about university/program/ institute/ department/foreword/ preface etc), University syllabus, conduction & Assessment guidelines, topics under consideration - concept, objectives, outcomes, set of typical applications/assignments/ guidelines, and references.

Guidelines for Student's Lab Journal

The laboratory assignments are to be submitted by student in the form of journal. Journal consists of prologue, Certificate, table of contents, and handwritten write-up of each assignment.

Guidelines for Lab /TW Assessment

Faculty member should frame Practical Assignments based on given list of assignments. Students will submit term work in the form of journal containing handwritten write-ups/ source code and output. Staff in-charge should maintain a record of continuous assessment and produced at the time of examination

Guidelines for Laboratory Conduction

The instructor is expected to frame the assignments by understanding the prerequisites, technological aspects, utility and recent trends related to the topic. Use of open-source software is to be encouraged. All the assignments should be conducted on Latest version of open-Source Operating Systems, tools and Multi-core CPU supporting Virtualization and Multi-Threading.

Guidelines for Practical Examination

Both internal and external examiners should jointly set problem statements. During practical assessment, the expert evaluator should give the maximum weightage to the satisfactory implementation of the problem statement

Online References

- https://buildspace.so/p/build-solidity-web3-app/lessons/welcome
- https://www.theinsaneapp.com/2022/05/best-web3-projects.html
- https://www.coinbase.com/learn/tips-and-tutorials/how-to-set-up-a-crypto-wallet

List of Experiments

Expt. No.	Aim / Objective of the Experiment
01	To setup a crypto wallet a) i) hosted wallets ii) self-custody wallet iii) hardware wallets (optional) and evaluate each of these.
	b) Understand the basic operations in the wallet on bitcoin such as 1) buy 2) sell 3) send 4) receive 5) exchange 6) mining.
02	Create a local Ethereum network using Hardhat or any other tool, build a smart contract that lets you send a (wave) to your contract and keep track of the total # of waves. Compile it to run locally.
	Connect to any Ethereum walleteg.Metamask. Deploy the contract with testnet. Connect wallet with your webapp. Call the deployed contract through your web app. Then store the wave messages from users in arrays using structs
03	Prepare your build system and Building Bitcoin Core. a. Write Hello World smart contract in a higher programming language (Solidity).
	b. Solidity example using arrays and functions
04	Deploy a simple contract to the Ethereum blockchain.
05	Polling / voting system using Solidity, Ethereum and a data structure hashmap(optional)

INDEX

Assignment No: 1(A,B)

Title of the Assignment: To setup a crypto wallet model,

Objective of the Assignment :-

To setup a crypto wallet

a) i) hosted wallets ii) self-custody wallet iii) hardware wallets (optional) and evaluate each of these.

b) Understand the basic operations in the wallet on bitcoin such as 1) buy 2) sell 3) send 4) receive 5) exchange 6) mining.

Prerequisite: Programming skills: javascript, react.js,

What is crypto Wallet?

A cryptocurrency wallet is a device, physical medium, program or a service which stores the public and/or private keys for cryptocurrency transactions.

A crypto wallet provides a way for users to validate an account balance to provide visibility into how much cryptocurrency the user owns. A crypto wallet enables users to send and receive cryptocurrency transactions -- an approach that's similar in concept to how a traditional bank account enables users to conduct transactions. For many users, a crypto wallet is a primary mechanism for managing cryptocurrency balances.

Why are crypto wallets important?

- Management of cryptocurrency. Crypto wallets provide users with the ability to monitor a balance for cryptocurrency assets.
- Transactions. Sending and receiving cryptocurrency payments is an important feature of crypto wallets.
- Connection to decentralized apps (dApps). A crypto wallet is required to connect and interact with Web 3.0 dApps.
- Username identities. All cryptocurrencies are stored on a blockchain. A crypto wallet enables transactions with a username that can be associated with a public key address on a blockchain.

- Key management. Functionally, cryptocurrency exists on the blockchain as a public key address. A crypto wallet helps users manage the private encryption keys used to access a given address and enable a transaction.

Types of crypto wallets

Crypto wallet users get to choose not just the service or vendor that supplies a crypto wallet, but the deployment approach as well.There are functionally two core types of crypto wallets: hot wallets and cold wallets. Hot wallets are generally always on and connected to the internet, while cold wallets, sometimes also referred to as cold storage, are typically disconnected and only connect online as needed.Within the category of cold wallets are two primary types:

1. Hardware wallets. With a hardware-based crypto wallet, the private key for the user's cryptocurrency balance is stored on a physical medium, which is typically a USB drive. Because it's a secured device that isn't always connected, the hardware wallet ensures a form of isolation when the user pulls out the key.

2. Paper wallets. A paper wallet is truly a low-tech solution, whereby the user writes down the public and private key information on a piece of paper.Within the hot wallet category are three types:

1. Online (web) wallets. Perhaps the most common and widely used form of crypto wallet is found in online services. With an online wallet, an online service such as a crypto exchange holds the user's public and private keys. Users access the wallet by logging in to the online service.

2. Desktop wallets. With a desktop wallet, the cryptographic keys are stored in an application on a user's desktop system.

3. Mobile wallets. A mobile app can be used to store a user's public and private keys for accessing and using cryptocurrency.

Custodial vs. noncustodial wallets

Crypto wallet types are either custodial or noncustodial. The fundamental difference between custodial and noncustodial wallets is control:

- Custodial wallets are crypto wallets in which the custody -- that is, the control and operations of the wallet -- is managed by a third party. That third party is typically the cryptocurrency exchange itself, where users buy and sell cryptocurrency tokens and other crypto assets.

  - Noncustodial wallets are crypto wallets where the custody is held by the individual who has the private keys for the crypto assets on the blockchain and is responsible for securing them. Noncustodial wallets include paper wallets, as well software wallets, that are managed by users.

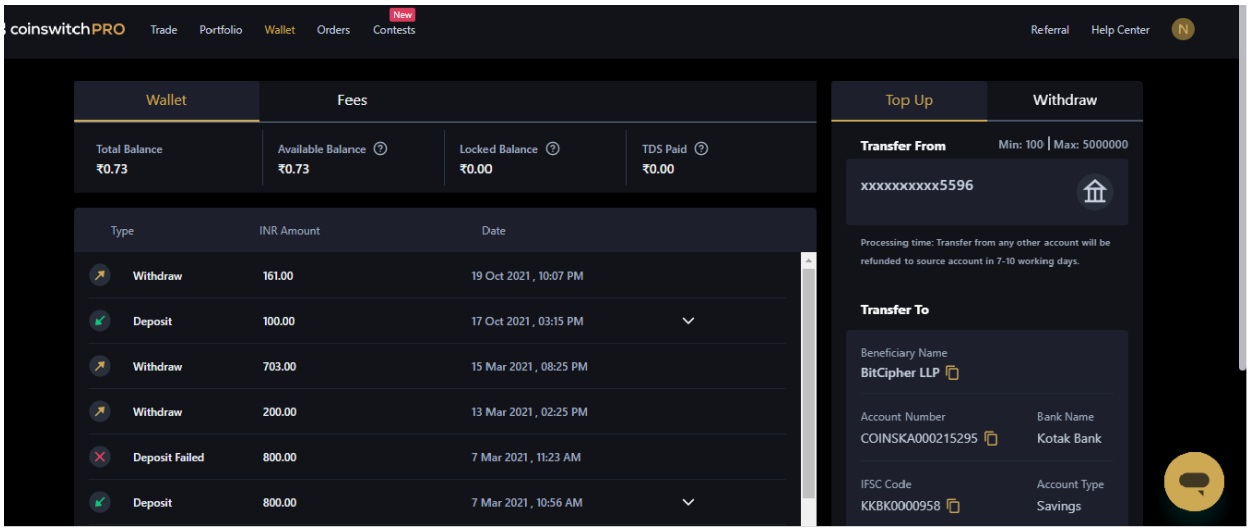
❖ **List of Crypto Wallets:-**

- Coinbase Wallet.
- Metamask.
- ZenGo Wallet.
- Trust Wallet.
- Ambire Wallet.
- Exodus.
- Trezor Wallet.
- BitGo Cryptocurrency Wallet

Hosted wallets

How to set up a hosted wallet:

1. **Choose a platform you trust.** Your main considerations should be security, ease of use, and compliance with government and financial regulations.
2. **Create your account.** Enter your personal info and choose a secure password. It's also recommended to use 2-step verification (also called 2FA) for an extra layer of security.



Self-custody wallets

A self-custody wallet, like Coinbase Wallet, puts you in complete control of your crypto. Non-custodial wallets don't rely on a third party — or a "custodian" — to keep your crypto safe. While they provide the software necessary to store your crypto, the responsibility of remembering and safeguarding your password falls entirely on you. If you lose or forget your password — often referred to as a "private key" or "seed phrase" — there's no way to access your crypto. And if someone else discovers your private key, they'll get full access to your assets.

Why have a non-custodial wallet? In addition to being in full control of the security of your crypto, you can also access more advanced crypto activities like yield farming, staking, lending, borrowing, and more. But if all you want to do is buy, sell, send, and receive crypto, a hosted wallet is the easiest solution.

How to set up a non-custodial wallet:

1. **Download a wallet app.** Popular options include Coinbase Wallet.
2. **Create your account.** Unlike a hosted wallet, you don't need to share any personal info to create a non-custodial wallet. Not even an email address.
3. **Be sure to write down your private key.** It's presented as a random 12-word phrase. Keep it in a secure location. If you lose or forget this 12-word phrase you won't be able to access your crypto.
4. **Transfer crypto to your wallet**. It's not always possible to buy crypto using traditional currencies (like US dollars or Euros) with a non-custodial wallet, so you'll need to transfer crypto into your non-custodial wallet from elsewhere.

Hardware wallets

How to set up a hardware wallet:

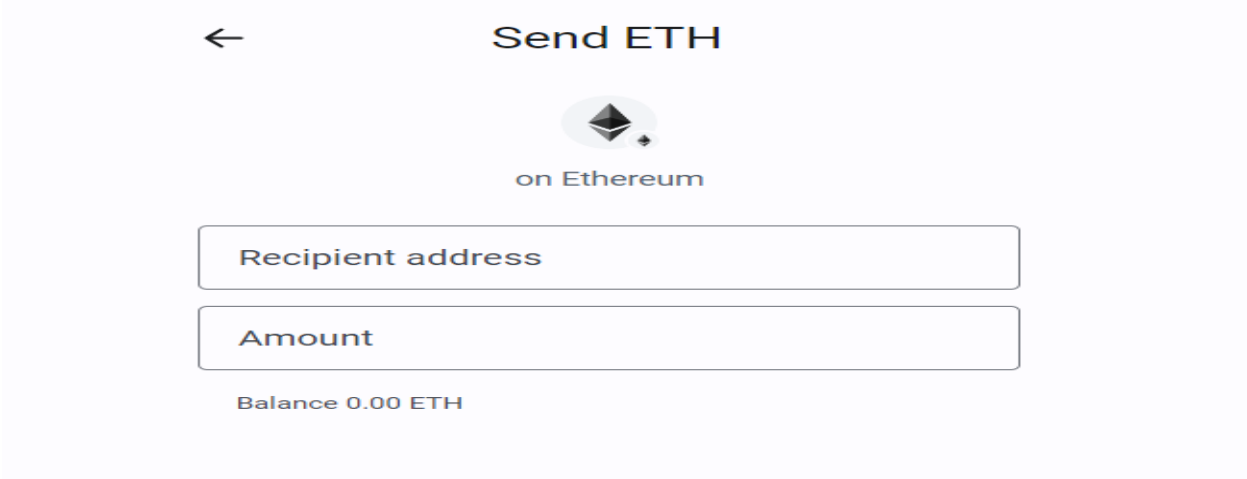
1. **Buy the hardware.** The two most well-known brands are Ledger and Trezor.
2. **Install the software.** Each brand has their own software that's needed to set up your wallet. Download the software from the official company website and follow the instructions to create your wallet.
3. **Transfer crypto to your wallet.** Similar to a non-custodial wallet, a hardware wallet typically doesn't allow you to buy crypto using traditional currencies (like US dollars or Euros), so you'll need to transfer crypto to your wallet.

Basic Operations in a Bitcoin Wallet:

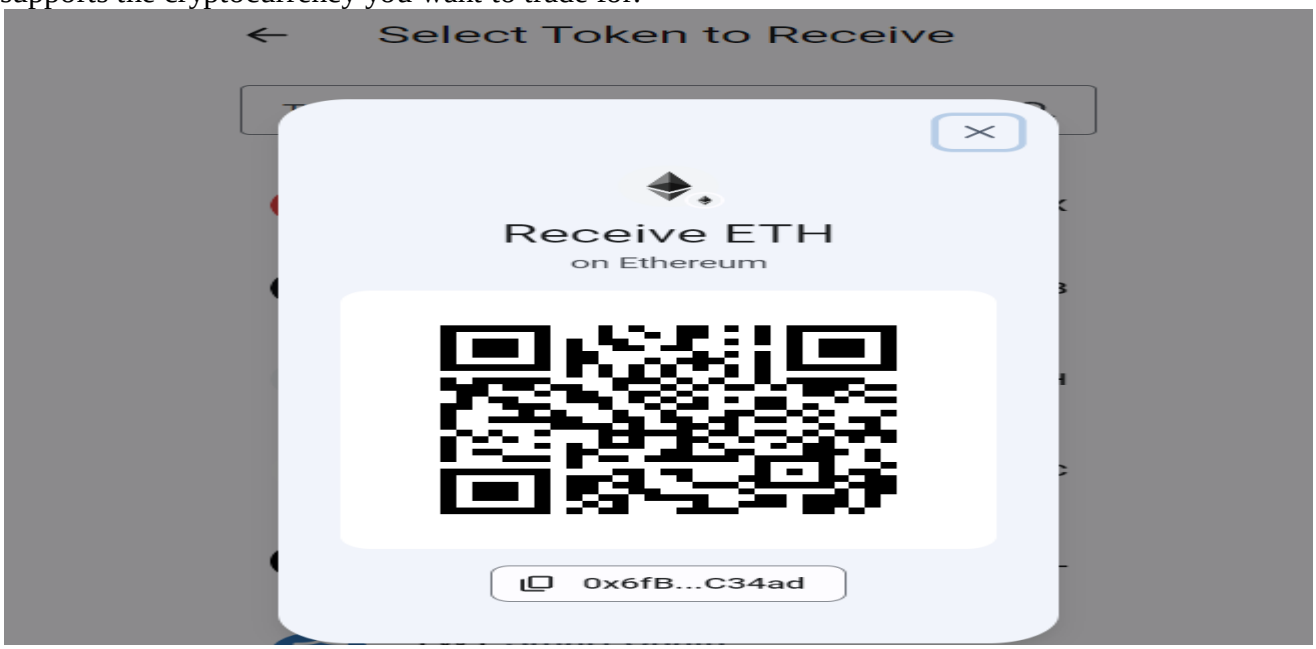
Buy: To buy Bitcoin, link your bank account or credit card to your Bitcoin wallet, and then use your wallet to purchase Bitcoin from an exchange or broker.

Sell: To sell Bitcoin, transfer your Bitcoin to an exchange or broker, and then sell it for fiat currency.

Send: To send Bitcoin, enter the recipient's Bitcoin address and the amount you want to send in your Bitcoin wallet, and then confirm the transaction.

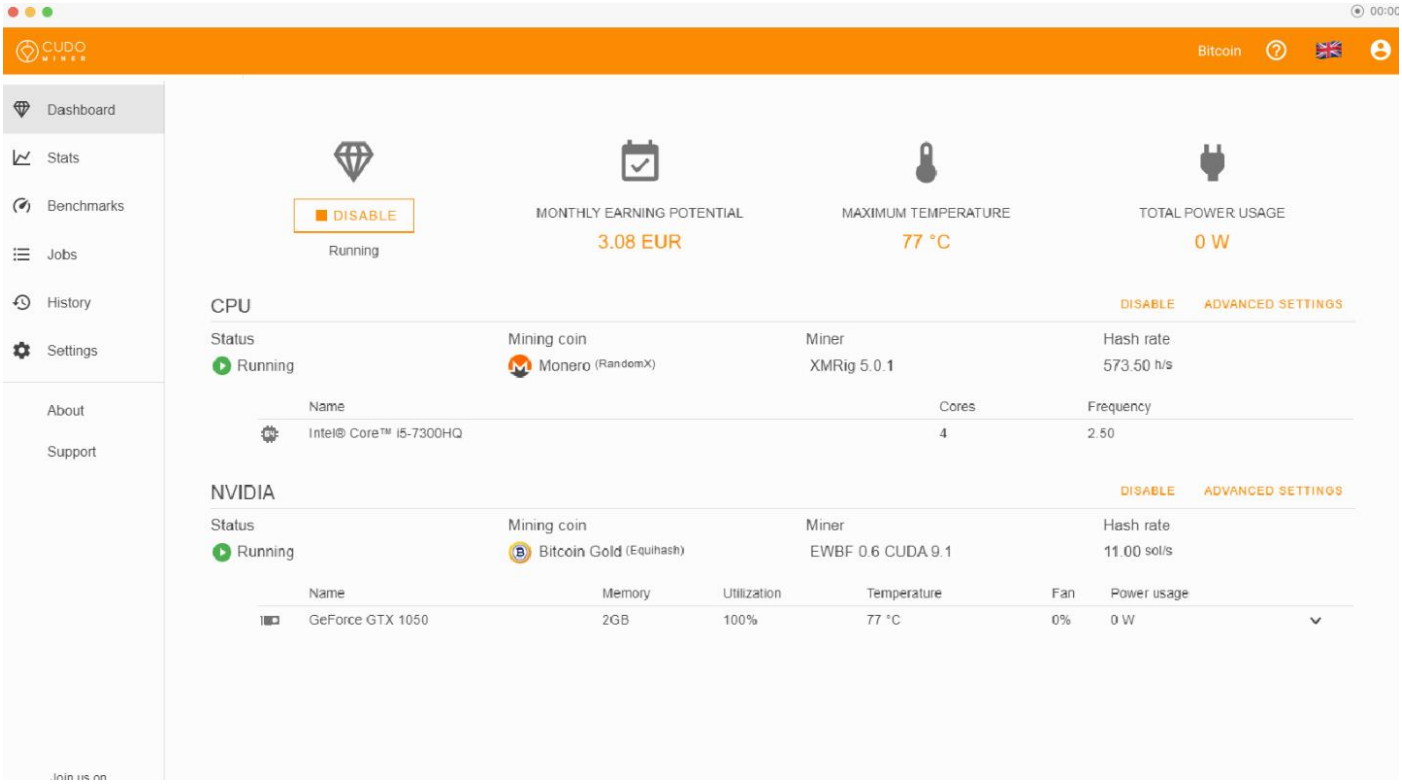


Receive: To receive Bitcoin, give the sender your Bitcoin address, which can be found in your Bitcoin wallet. Once the sender sends Bitcoin to your address, it will appear in your wallet. Exchange: To exchange Bitcoin for another cryptocurrency, use a cryptocurrency exchange that supports the cryptocurrency you want to trade for.



Mining:

Mining is the process of validating Bitcoin transactions and adding them to the blockchain. To mine Bitcoin, you need specialized mining equipment and software.



4. Oral Questions:

1. What do u mean by self Custody wallet?.
2. What do u mean by hosted wallets
3. What is bitcoin

5. Assignment Question

Study Various template for Creating a wallet

Conclusion:

Creating a cryptocurrency wallet is a crucial step in the cryptocurrency world. Hosted, self-custody, and hardware wallets each have their own advantages and disadvantages, and users should choose the wallet that best suits their needs. Basic operations in a Bitcoin wallet include buying, selling, sending, receiving, exchanging, and mining. By understanding these operations, users can effectively manage their cryptocurrency holdings.

Assignment No: 2(A,B)

Title of the Assignment: local Ethereum network using Hardhat .

Objective of the Assignment:

1. Create a local Ethereum network using Hardhat or any other tool, build a smart contract that lets you send a □(wave) to your contract and keep track of the total # of waves. Compile it to run locally.

2) Connect to any Ethereum walleteg.Metamask. Deploy the contract with testnet. Connect wallet with your webapp. Call the deployed contract through your web app. Then store the wave messages from users in arrays using struct.

Prerequisite:

- Basic Tool Node.js installed on your system
- CLI/Terminal
- Text Editor

Theory:

A local Ethereum network

A local Ethereum network designed for development. It allows you to deploy your contracts, run your tests and debug your code. Hardhat Network comes built-in with Hardhat, an Ethereum development environment for professionals.

What is Hardhat?

Hardhat is a development environment that helps developers compile, deploy, test, and debug their Ethereum applications. It has some of the cleanest, most detailed documentation. Hardhat also provides console.log() functionality, similar to javascript for debugging purposes. Hardhat also has many plugins, which further increases its functionality.finite time to complete. Action state is a special state of activity state. An activity state is composite and its flow of control is made up of other activity states and action states. Activity states have additional parts likeentry and exit actions and some machine specifications.

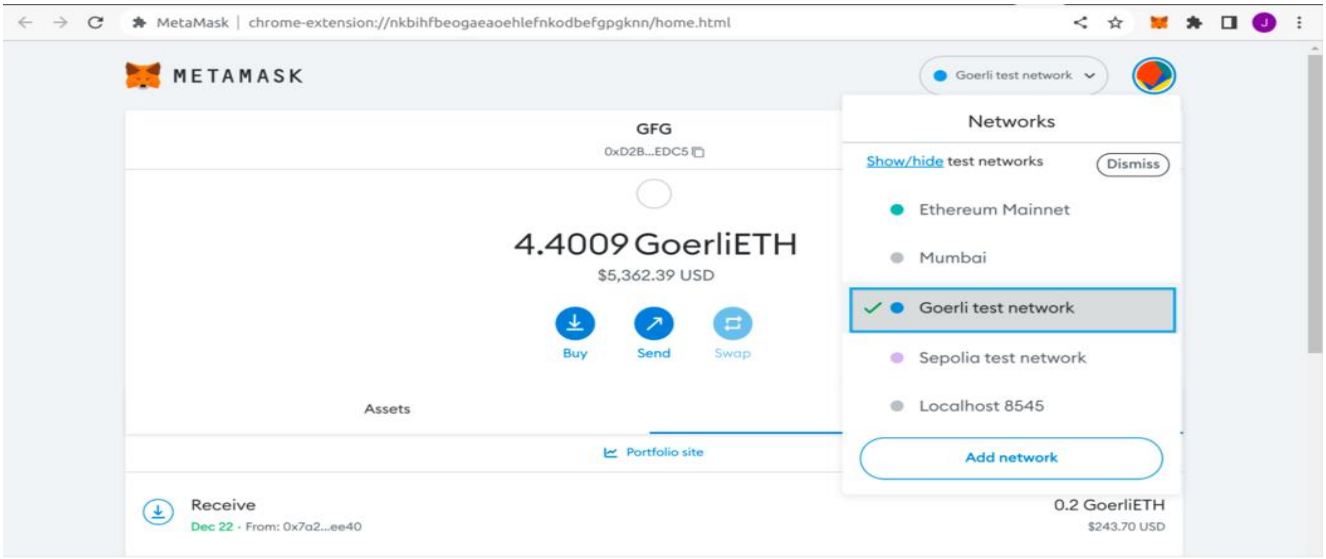
MetaMask

MetaMask is a type of Ethereum wallet that bridges the gap between the user interfaces for Ethereum (e.g. Mist browsers, DApps) and the regular web (e.g. Chrome, Firefox, websites).

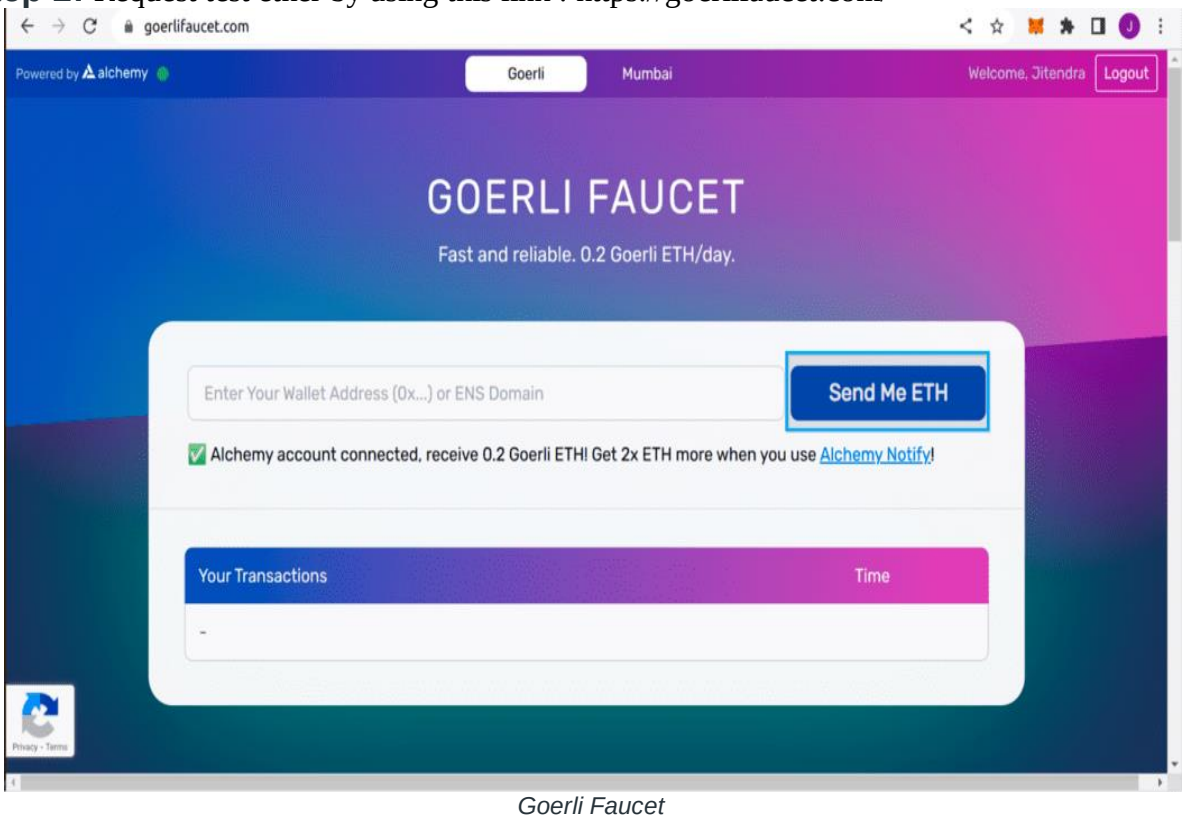
Its function is to inject a JavaScript library called web3.js into the namespace of each page your browser loads. Web3.js is written by the Ethereum core team. MetaMask is mainly used as a plugin in chrome. You can add it from here or download it directly from this link.

After adding MetaMask as an extension in chrome and creating an account, set up your account as follows –

Step 1: Select Goerli Test Network from a list of available networks as below:



Step 2: Request test ether by using this link : https://goerlifaucet.com/

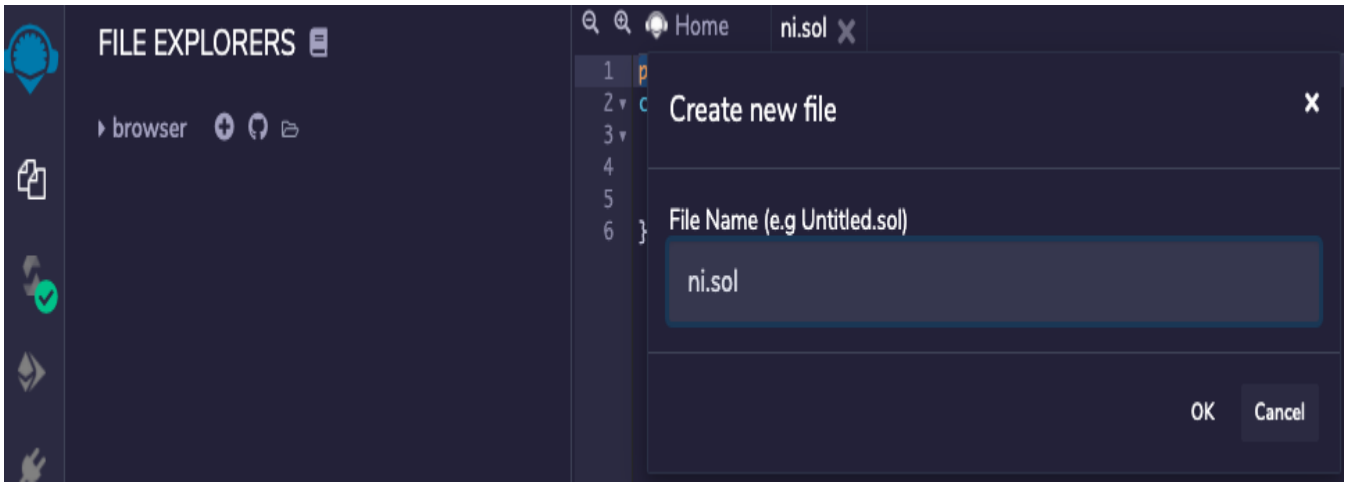


Goerli Faucet

Step 3: MetaMask is ready for deployment. To know more about MetaMask visit the MetaMask official guide.

Steps to deploy your contract

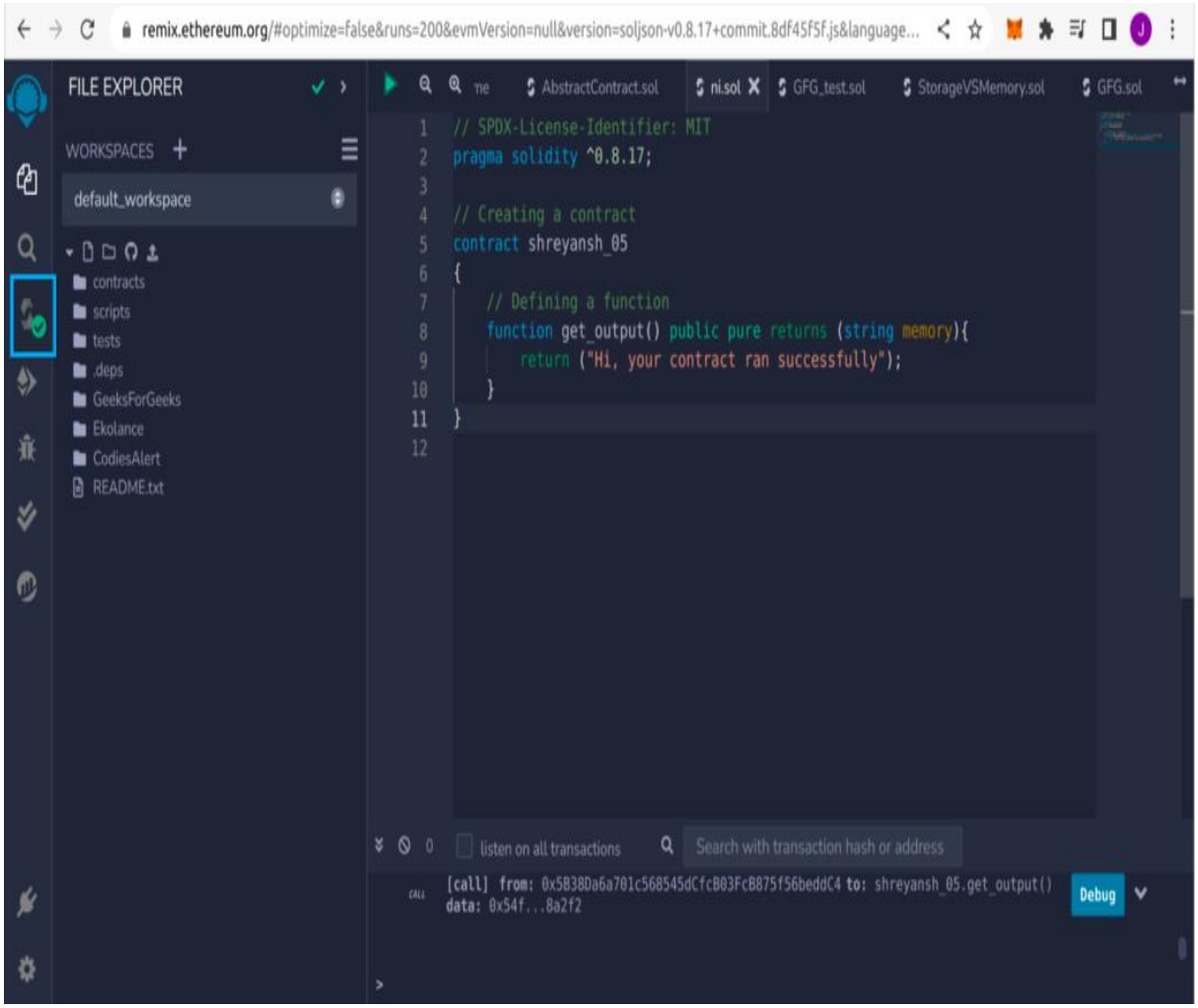
Step 1: Open Remix IDE in your browser. After opening click on + and write the filename as follows:



Step 2: Write the following sample code for testing and compile by clicking on the compile button as shown:

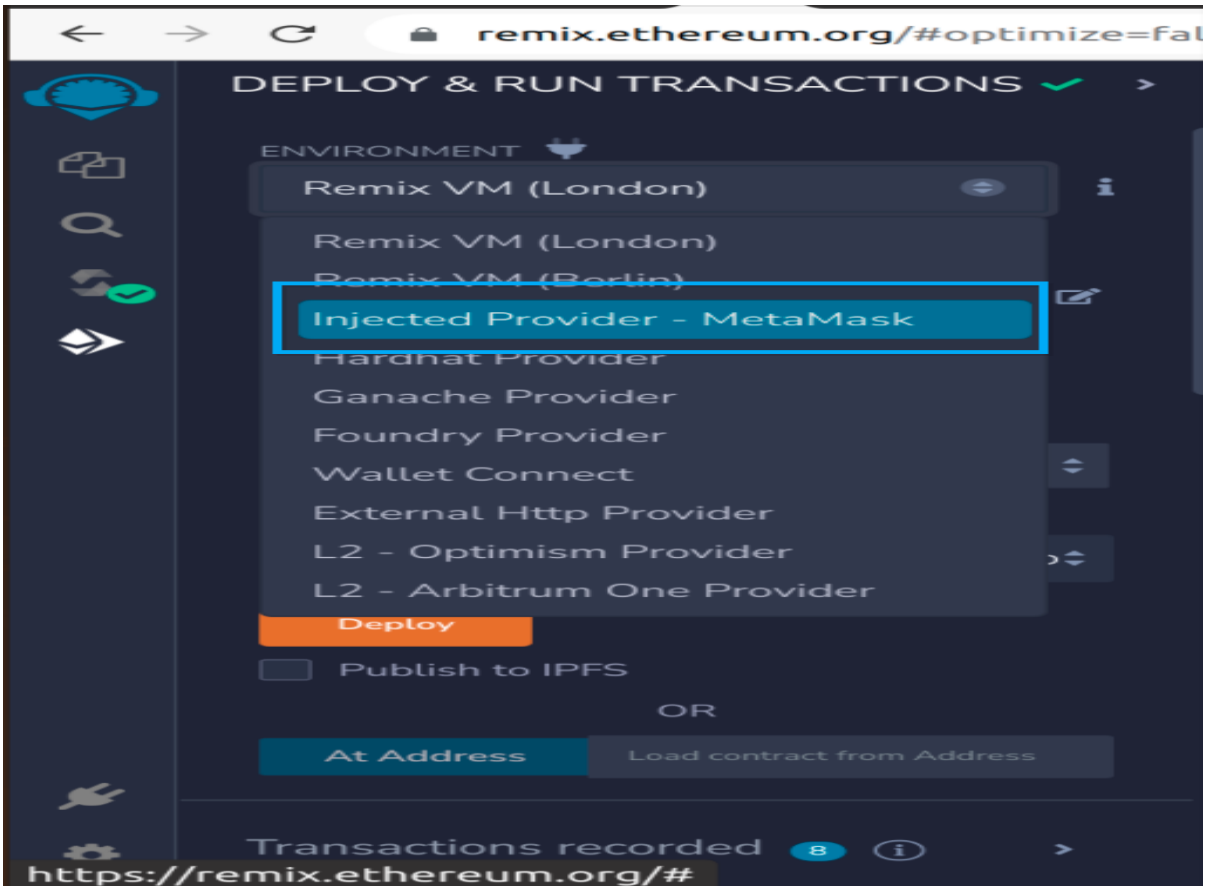
- Solidity

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

pragma solidity ^0.8.17;



// Creating a contract

contract shreyansh_05

{

    // Defining a function

    function get_output() public pure returns (string memory){

        return ("Hi, your contract ran successfully");

    }

}
```

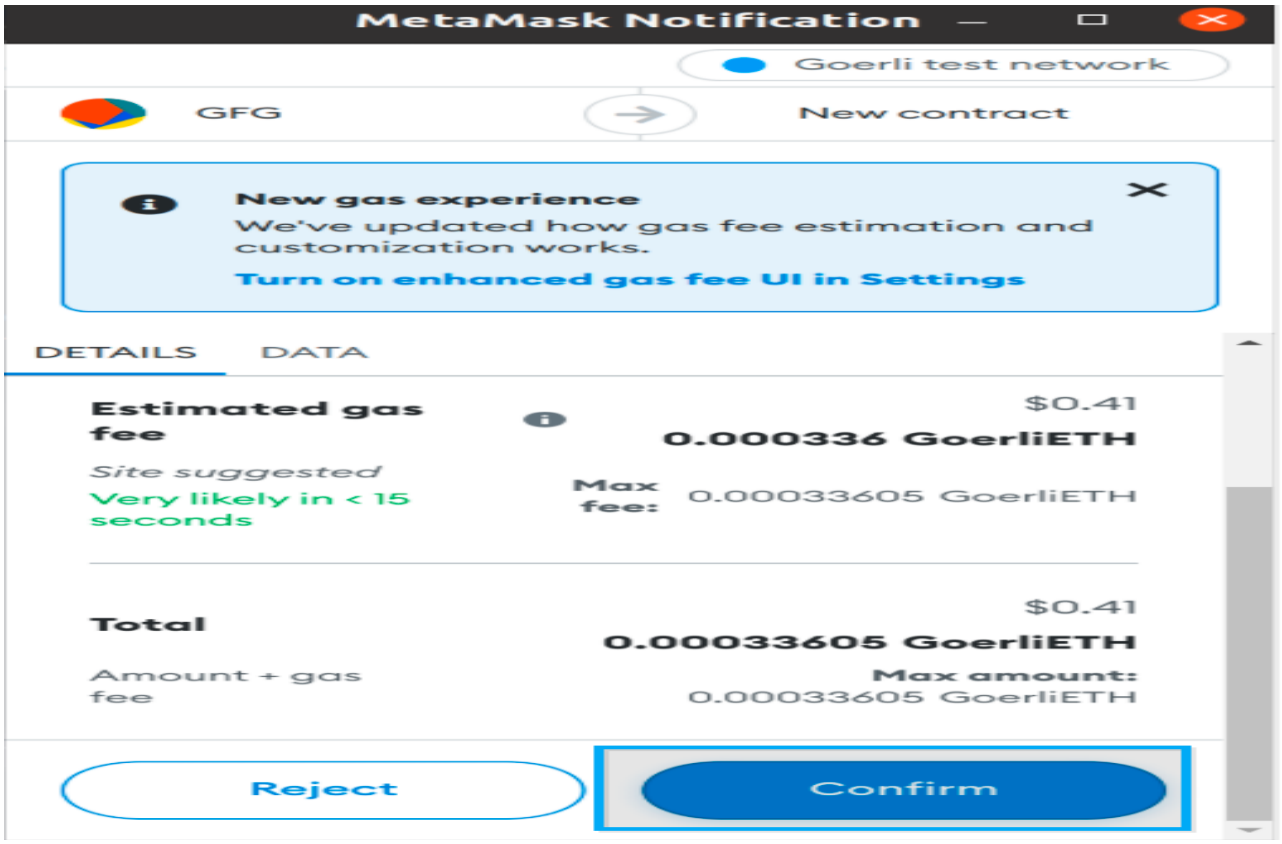


Compile the ni.sol file

Step 3: After compilation and move to deploy section just below the compilation and select Injected Provider – MetaMask in place of Remix VM as shown below –

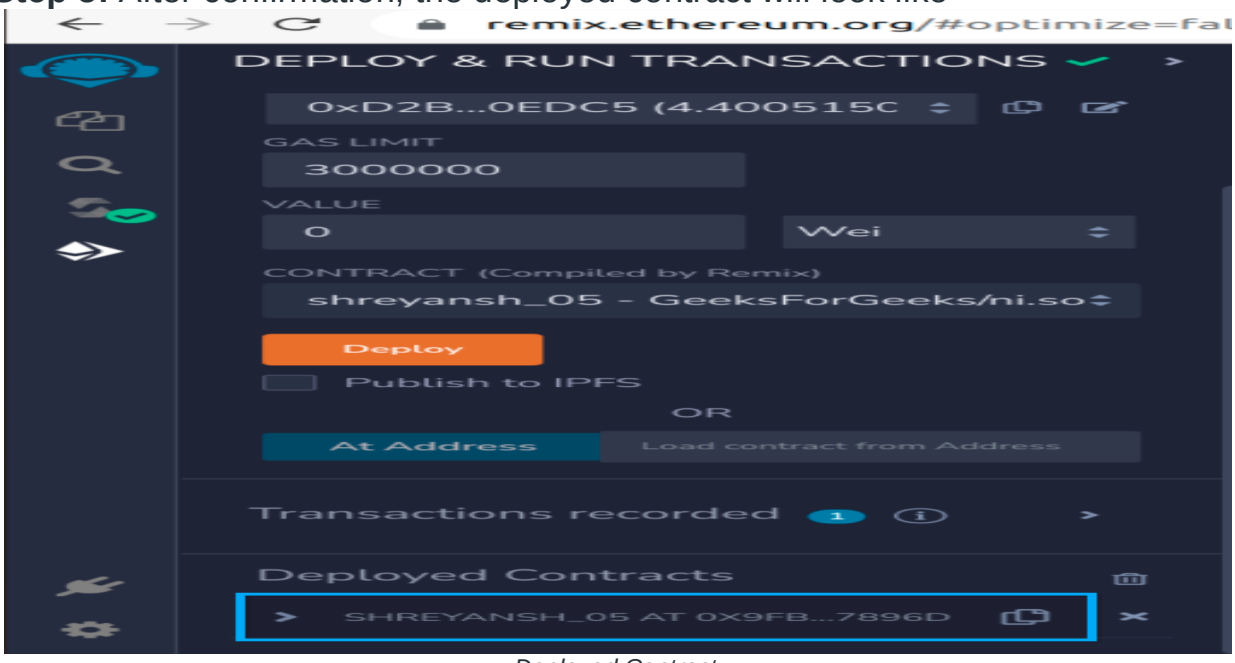


Step 4: **Now your contract is ready to be deployed. Click on deploy button and the MetaMask will ask for confirmation as follows –**



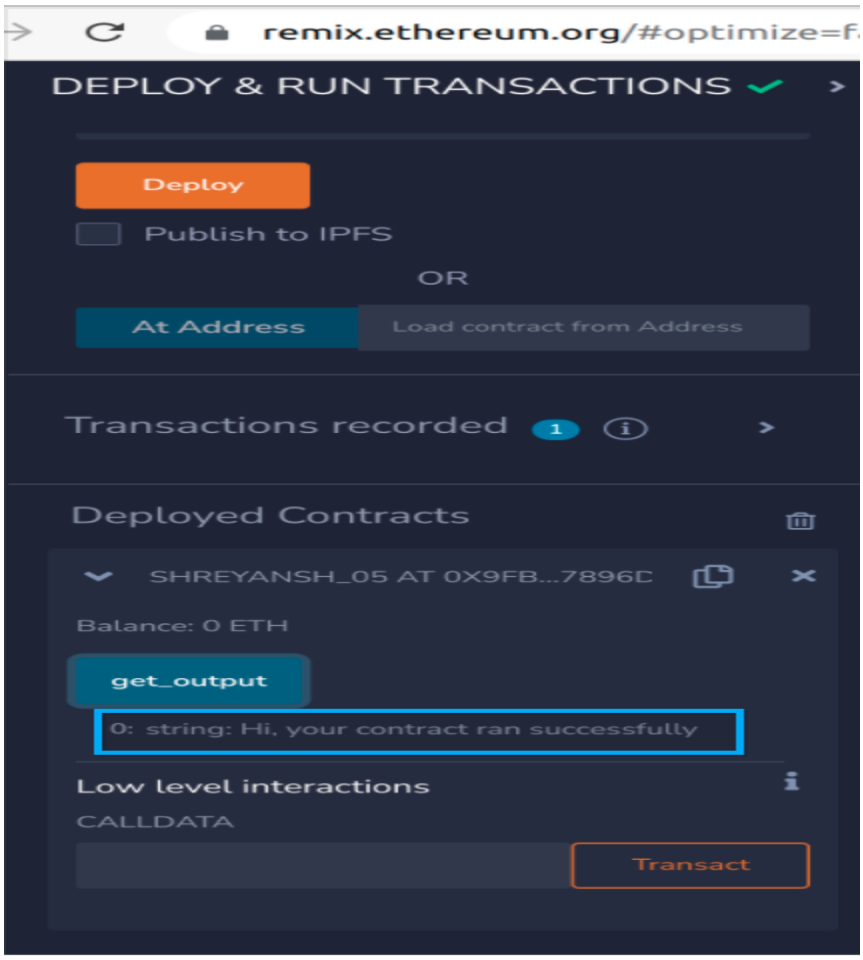
Click the Confirm Button

Step 5: After confirmation, the deployed contract will look like –



Deployed Contract

Step 6: Expand the deployed contract as below and get the output using the *get_output() function:*



Output

1. **Step 7:** Now, to verify whether your transaction (process) executed successfully, you can check your balance on MetaMask.
2. Now your contract is completely ready to function. Make sure the compiler version matches the version of your solidity code. This is the basic implementation of MetaMask with solidity.

Hard chat

Installing hardhat and other dependencies

We'll install hardhat using npm, which comes with node.js.

First, create a new project directory and cd into it. Feel free to use your own names here instead:

```
mkdir HardhatTutorial
cd HardhatTutorial
```

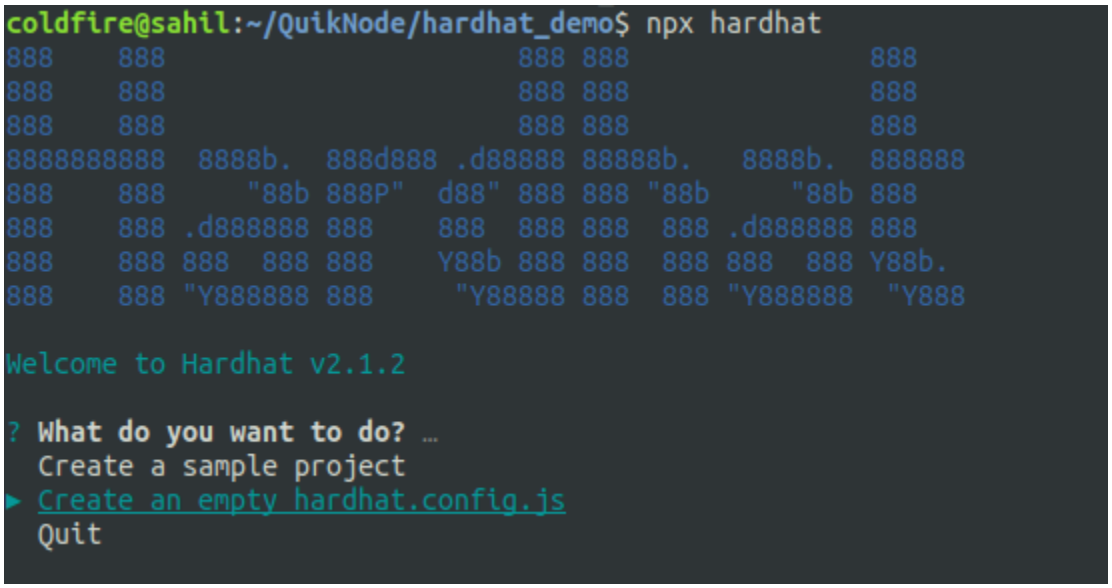
Open terminal/cmd in your project directory and type the following.

```
npm install -d hardhat
```

Now that we have hardhat installed let's start a new hardhat project. We'll use npx to do so. Npx helps process node.js executables.

```
npx hardhat
```

You'll be greeted with a CLI hardhat interface. Select the second option, "Create an empty hardhat.config.js", and press enter.



Now, let's install other dependencies required to work with hardhat.

```
npm install --save-dev @nomiclabs/hardhat-ethers ethers @nomiclabs/hardhat-waffle ethereum-waffle chai
```

We need these dependencies to write automated tests for contracts.

The most common issue while installing packages with npm can be an internal failure with `node-gyp`. You can follow node-gyp installation instructions here.

Another common issue is a stale cache. Clear your npm cache by simply typing the below into your terminal:

```
npm cache clean
```

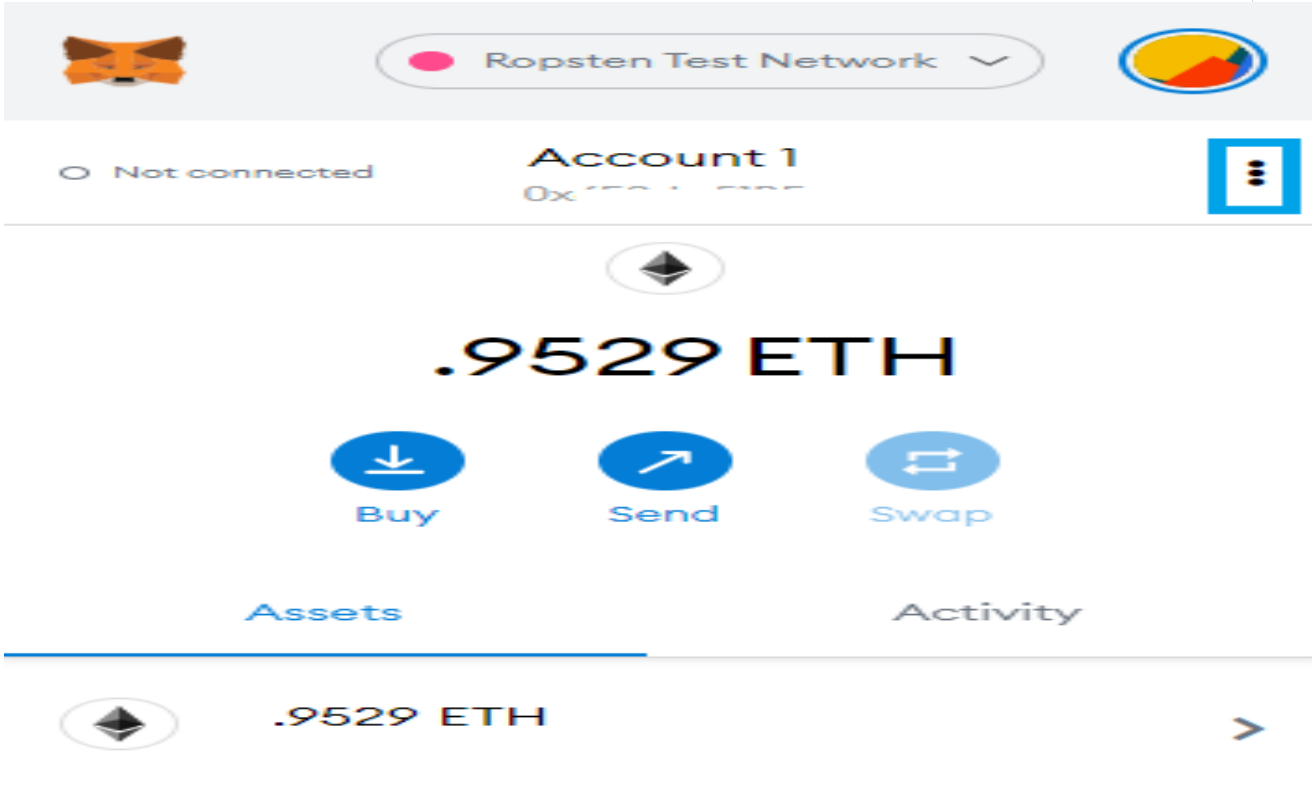
Now, let's get a private key for our wallet, some test ETH for gas, and an Ethereum node in place; all of which we'll need to deploy our smart contract.

Getting the private key

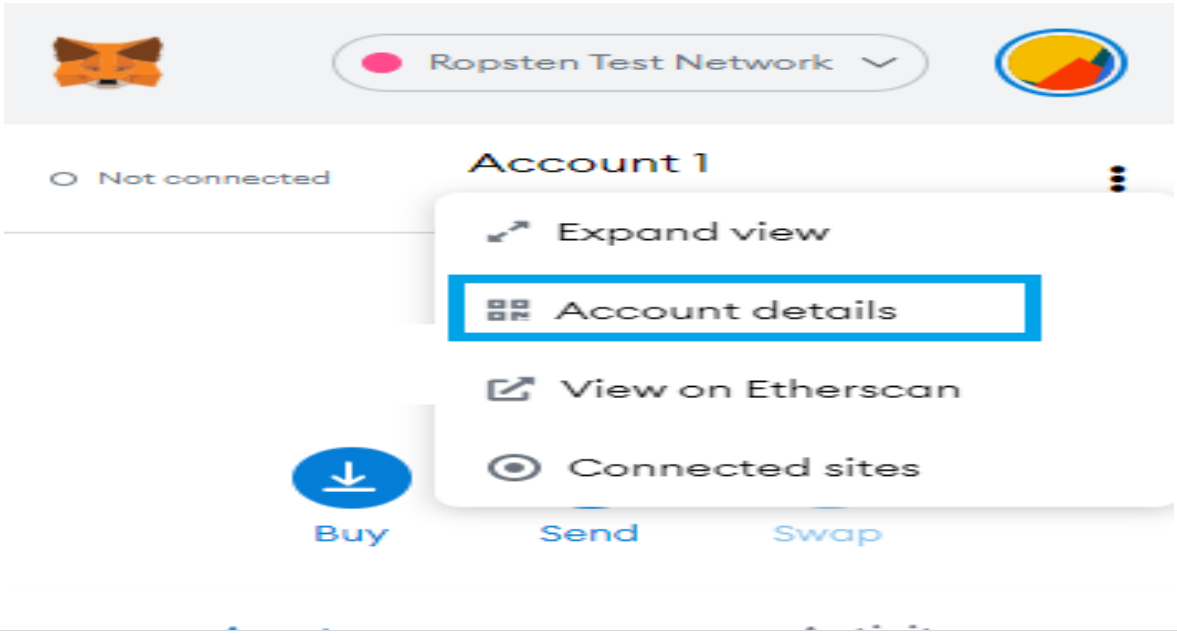
We'll need an Ethereum wallet/address to sign the smart contract deployment transaction. To set up the wallet, get its private key and add it to the config file of hardhat.

You can follow QuikNode guides to generate a private key and an Ethereum address in JavaScript, Ruby, Python, Go, PHP.

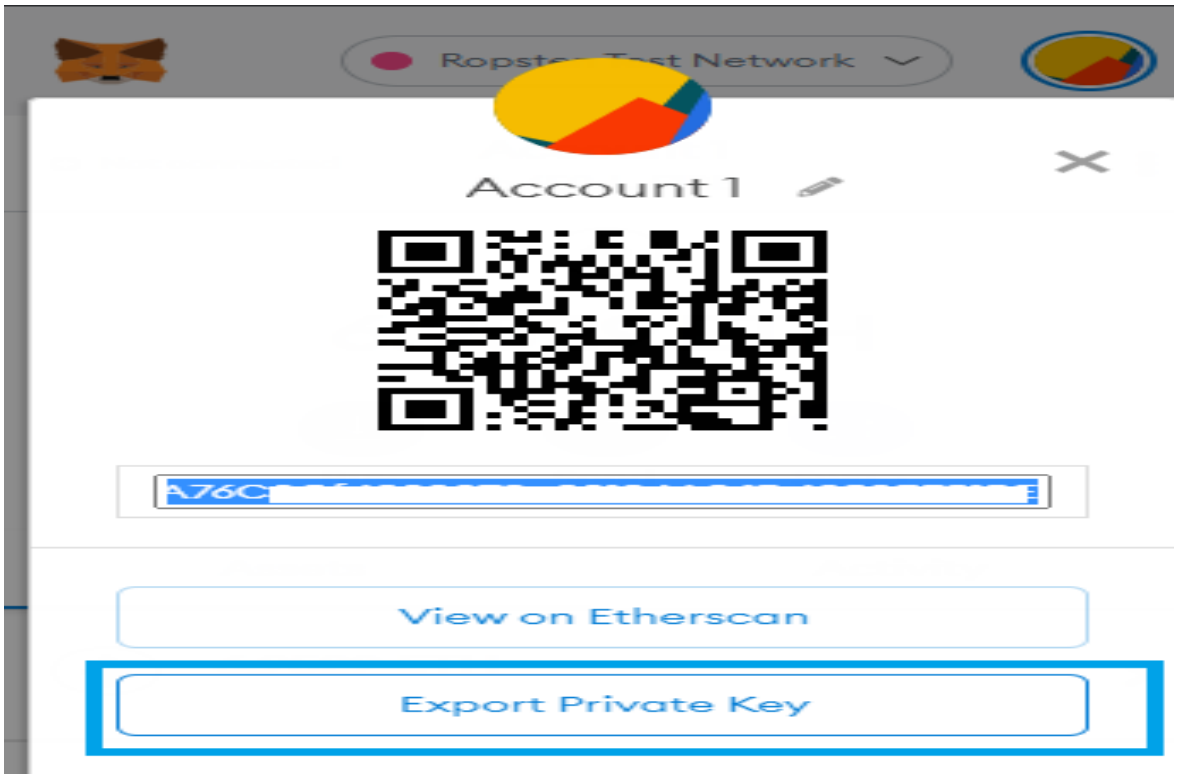
You can also get a private key from your MetaMask wallet. To do so, open your MetaMask browser plugin, select Ropsten network and click on the three dots below the favicon.



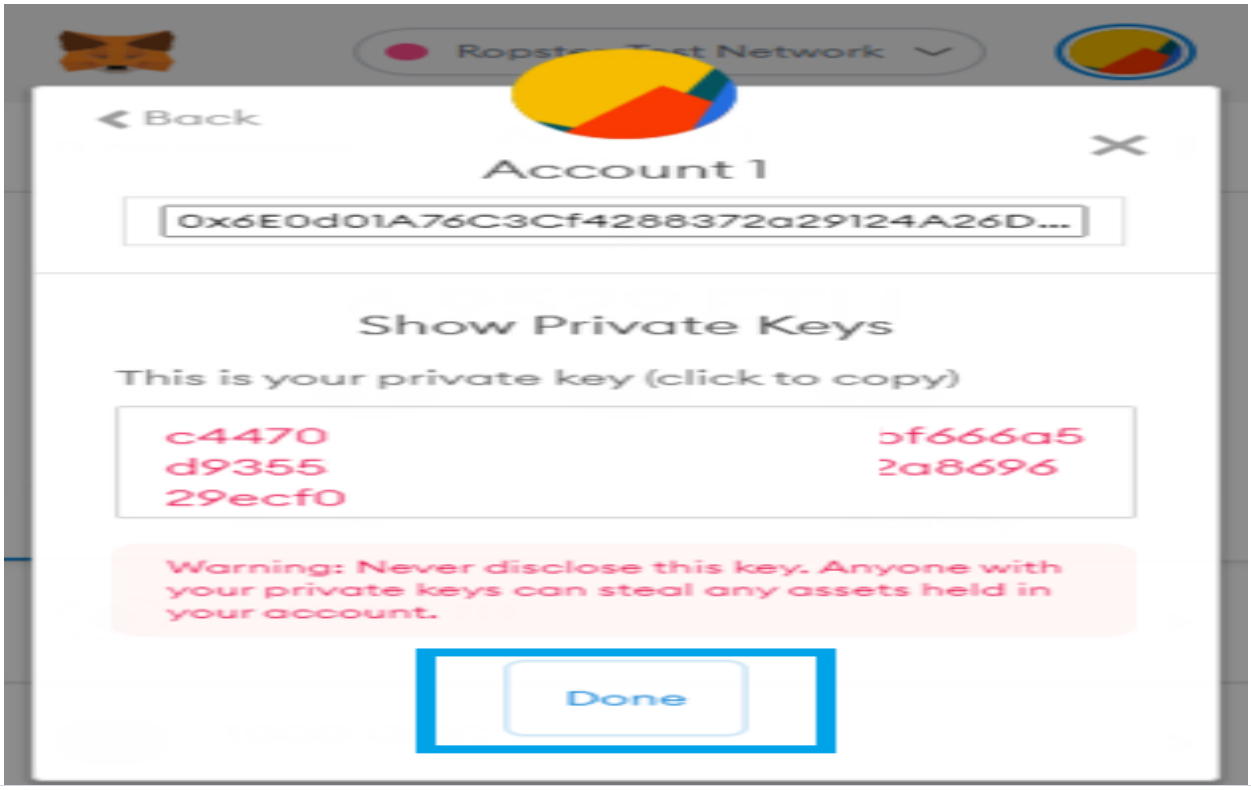
Now, click on “Account details”



Then click on “Export Private Key”



Enter your password and click on confirm. It’ll display your private key (similar to the above image).



Copy and save the private key; we'll need it in the later steps.

1. Oral Questions

1. What is mean by private key in meta mask
2. What is hard chat
3. How to install meta mask

Assignment Question

- Installed hard chat
- Install meta mask and create account

Conclusion: We conclude that installing hard meta mask and Crete account on that and deploy smart contract

---

Assignment No: 3

Prerequisite: Basic Blockchain Concepts

Title of the Assignment: Prepare your build system and Building Bitcoin Core.

Objective of the Assignment: a. Write Hello World smart contract in a higher programming language (Solidity). b. Solidity example using arrays and functions..

.

Theory:

What Are Smart Contracts?

Smart contracts are computer programs published and executed in a blockchain environment. As they run on blockchains, they can be run without a central party or server.

Once a smart contract has been published, it is not possible to update or make any changes to its code due to the immutable nature of blockchains. However, the smart contract may have been programmed with functions to change data. This information can be recorded in one block and deleted in another, but the history remains and it is possible to audit. Solidity is a high level, object-oriented programming language for writing smart contracts in the Ethereum Blockchain. Smart contracts are used to manipulate the Ethereum Blockchain and govern the behaviour of the accounts within the Ethereum Blockchain.

Solidity is highly influenced by Javascript, C++ and Python. It is designed to target the EVM (Ethereum Virtual Machine). With Solidity, you can create interesting Web3.0 projects like a crowdfunding system, blind auctions, multi-signature wallets. Solidity keeps receiving many regular updates which means that you have to adapt to the new version as quickly as you can. The most recent Solidity version is 0.8x.

As Web3.0 has just started to gain traction, many companies are ahead of the curve and have already started to adapt to the change and have started implementing Solidity in their development processes. Some of the top companies using Solidity are:

- LeewayHertz
- EngineerBabu
- TreeHouse Technology Group
- OpenGeek Slab
- Altoros
- AppInvetiv
- Abes Lab
- Arc Touch

Solidity Programing Language

Solidity is an object-oriented, high-level language for implementing smart contracts. It is a curly-bracket language, which means that the characters "{" and "}" define statement blocks.

Solidity is influenced by C++, Python, and JavaScript, and is designed to run on the Ethereum Virtual Machine (EVM). It is statically typed and supports inheritance, libraries, and complex user-defined types, among other features.

Remix

Remix is an online web tool. It is an IDE (integrated development environment) used to write, compile, deploy, and debug Solidity code. Remix has an environment called JavaScriptVM, which is a blockchain simulator that runs in your browser. This tutorial will use it. Go to remix.ethereum.org to get started.

Create the Smart Contract

Click on the second icon on the left bar, “File Explorers”.

Click on the button “Create a new file”.

File name: HelloWord.sol

Files written in Solidity use the extension “.sol”.

Copy and paste this example:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.13;
contract HelloWorld {
    function sayHelloWorld() public pure returns (string memory) {
        return "Hello World";
    }
}
```

Now let’s look at what’s in the smart contract.

// SPDX-License-Identifier

“//” means that this is a comment, not a code line.

The SPDX License List Specification is a list of common licenses used in free and open or collaborative software.

Solidity 0.6.8 introduces SPDX license identifiers so developers can specify the license the smart contract uses.

SPDX license identifiers should be added to the top of contract files, using the identifier “//”

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

Pragma

This specifies the version of Solidity, using semantic versioning. Learn more here.

```
pragma solidity 0.8.13;
```

Contract HelloWorld

This defines a contract named “HelloWorld”.

A contract is a collection of functions and data (its state).

Once deployed, a contract will exist at an address on the Ethereum blockchain. Learn more here.

Function sayHelloWorld

This is a public function that returns the string “Hello World”. It is declared pure because it doesn’t read or modify the blockchain state.

Compile a Smart Contract

Locate in the left bar a button called “Solidity compiler”.

Click on the button “Compile HelloWorld.sol”.

It is useful to enable the auto-compile option.

Check the green sign at the button with the message compilation successful.

Deploy a Smart Contract

In the left side panel, go to the button “Deploy and run transactions”.

For now, we have only one smart contract, so it is automatically selected in the dropdown “Contracts”.

Click on the button “Deploy”.

Interact With the Smart Contract

When a smart contract is deployed in Remix, we can see it in the left panel under “deploy and run transactions”:

1. Scroll down the left side until you reach “Deployed Contracts”.
2. Expand the “HelloWorld”.
3. Click on the button “sayHelloWorld”.
4. It will return the message recorded in the smart contract: “Hello World”.

Congratulations, you’ve created a “Hello World” smart contract

b. Solidity example using arrays and functions

array is a data structure, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

In Solidity, an array can be of compile-time fixed size or of dynamic size. For storage array, it can have different types of elements as well. In case of memory array, element type can not be mapping and in case it is to be used as function parameter then element type should be an ABI type.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

Declaring Arrays

To declare an array of fixed size in Solidity, the programmer specifies the type of the elements and the number of elements required by an array as follows −

type arrayName [arraySize];

This is called a single-dimension array. The arraySize must be an integer constant greater than zero and type can be any valid Solidity data type. For example, to declare a 10-element array called balance of type uint, use this statement −

uint balance[10];

To declare an array of dynamic size in Solidity, the programmer specifies the type of the elements as follows −

type[] arrayName;

Initializing Arrays

You can initialize Solidity array elements either one by one or using a single statement as follows −

uint balance[3] = [1, 2, 3];

The number of values between braces [ ] can not be larger than the number of elements that we declare for the array between square brackets [ ]. Following is an example to assign a single element of the array −

If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write −

uint balance[] = [1, 2, 3];

You will create exactly the same array as you did in the previous example.

balance[2] = 5;

The above statement assigns element number 3rd in the array a value of 5.

Creating dynamic memory arrays

Dynamic memory arrays are created using new keyword.

uint size = 3;

uint balance[] = new uint[](size);

Accessing Array Elements

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example −

uint salary = balance[2];

The above statement will take 3rd element from the array and assign the value to salary variable. Following is an example, which will use all the above-mentioned three concepts viz. declaration, assignment and accessing arrays −

Members

length − length returns the size of the array. length can be used to change the size of dynamic array be setting it.

push − push allows to append an element to a dynamic storage array at the end. It returns the new length of the array.

Example

Try the following code to understand how the arrays works in Solidity.

```
pragma solidity ^0.5.0;

contract test {
  function testArray() public pure{
    uint len = 7;

    //dynamic array
    uint[] memory a = new uint[](7);

    //bytes is same as byte[]
    bytes memory b = new bytes(len);

    assert(a.length == 7);
    assert(b.length == len);

    //access array variable
    a[6] = 8;
```

```
    //test array variable
    assert(a[6] == 8);

    //static array
    uint[3] memory c = [uint(1) , 2, 3];
    assert(c.length == 3);
  }
}
```

Function Definition

Before we use a function, we need to define it. The most common way to define a function in Solidity is by using the function keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

Syntax

The basic syntax is shown here.

```
function function-name(parameter-list) scope returns() {
  //statements
}
```

Example

Try the following example. It defines a function called getResult that takes no parameters −

```
pragma solidity ^0.5.0;

contract Test {
  function getResult() public view returns(uint){
    uint a = 1; // local variable
    uint b = 2;
    uint result = a + b;
    return result;
  }
}
```

Calling a Function

To invoke a function somewhere later in the Contract, you would simply need to write the name of that function as shown in the following code.

Try the following code to understand how the string works in Solidity.

```
pragma solidity ^0.5.0;

contract SolidityTest {
   constructor() public{
   }
   function getResult() public view returns(string memory){
      uint a = 1;
      uint b = 2;
      uint result = a + b;
      return integerToString(result);
   }
   function integerToString(uint _i) internal pure
      returns (string memory) {

      if (_i == 0) {
         return "0";
      }
      uint j = _i;
      uint len;

      while (j != 0) {
         len++;
         j /= 10;
      }
      bytes memory bstr = new bytes(len);
      uint k = len - 1;

      while (_i != 0) {
         bstr[k--] = byte(uint8(48 + _i % 10));
```

```
      _i /= 10;
    }
    return string(bstr);//access local variable
  }
}
```

Run the above program using steps provided in Solidity First Application chapter.

Output

0: string: 3

Function Parameters

Till now, we have seen functions without parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

Example

Try the following example. We have used a uint2str function here. It takes one parameter.

```
pragma solidity ^0.5.0;

contract SolidityTest {
  constructor() public{
  }
  function getResult() public view returns(string memory){
    uint a = 1;
    uint b = 2;
    uint result = a + b;
    return integerToString(result);
  }
  function integerToString(uint _i) internal pure
    returns (string memory) {

    if (_i == 0) {
      return "0";
```

```
    }
    uint j = _i;
    uint len;

    while (j != 0) {
      len++;
      j /= 10;
    }
    bytes memory bstr = new bytes(len);
    uint k = len - 1;

    while (_i != 0) {
      bstr[k--] = byte(uint8(48 + _i % 10));
      _i /= 10;
    }
    return string(bstr);//access local variable
  }
}
```

Run the above program using steps provided in Solidity First Application chapter.

Output

0: string: 3

The return Statement

A Solidity function can have an optional return statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.

As in above example, we are using uint2str function to return a string.

In Solidity, a function can return multiple values as well. See the example below −

```
pragma solidity ^0.5.0;

contract Test {
  function getResult() public view returns(uint product, uint sum){
    uint a = 1; // local variable
    uint b = 2;
```

```
        product = a * b;
        sum = a + b;

        //alternative return statement to return
        //multiple values
        //return(a*b, a+b);
    }
}
```

Run the above program using steps provided in Solidity First Application chapter.

Output

0: uint256: product 2

1: uint256: sum 3

a. Write Hello World smart contract in a higher programming language (Solidity).
b. Solidity example using arrays and functions

1. Write Hello World smart contract in a higher programming language (Solidity).

HelloWorld.sol

```
// SPDX-License-Identifier: MIT
// compiler version must be greater than or equal to 0.8.17 and less than 0.9.0
pragma solidity ^0.8.17;

contract HelloWorld {
   string public greet = "Hello World!";
}
```

2. Solidity example using arrays and functions

Functions.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

contract Function {
   // Functions can return multiple values.
   function returnMany() public pure returns (uint, bool, uint) {
      return (1, true, 2);
   }

   // Return values can be named.
   function named() public pure returns (uint x, bool b, uint y) {
```

```
        return (1, true, 2);
    }

    // Return values can be assigned to their name.
    // In this case the return statement can be omitted.
    function assigned() public pure returns (uint x, bool b, uint y) {
        x = 1;
        b = true;
        y = 2;
    }

    // Use destructuring assignment when calling another
    // function that returns multiple values.
    function destructuringAssignments()
        public
        pure
        returns (uint, bool, uint, uint, uint)
    {
        (uint i, bool b, uint j) = returnMany();

        // Values can be left out.
        (uint x, , uint y) = (4, 5, 6);

        return (i, b, j, x, y);
    }

    // Cannot use map for either input or output

    // Can use array for input
    function arrayInput(uint[] memory _arr) public {}

    // Can use array for output
    uint[] public arr;

    function arrayOutput() public view returns (uint[] memory) {
        return arr;
    }
}

// Call function with key-value inputs
contract XYZ {
    function someFuncWithManyInputs(
        uint x,
        uint y,
        uint z,
        address a,
        bool b,
        string memory c
    ) public pure returns (uint) {}

    function callFunc() external pure returns (uint) {
        return someFuncWithManyInputs(1, 2, 3, address(0), true, "c");
    }
```

```
    function callFuncWithKeyValue() external pure returns (uint) {
        return
            someFuncWithManyInputs({a: address(0), b: true, c: "c", x: 1, y: 2, z: 3});
    }
}
```

Array.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

contract Array {
    // Several ways to initialize an array
    uint[] public arr;
    uint[] public arr2 = [1, 2, 3];
    // Fixed sized array, all elements initialize to 0
    uint[10] public myFixedSizeArr;

    function get(uint i) public view returns (uint) {
        return arr[i];
    }

    // Solidity can return the entire array.
    // But this function should be avoided for
    // arrays that can grow indefinitely in length.
    function getArr() public view returns (uint[] memory) {
        return arr;
    }

    function push(uint i) public {
        // Append to array
        // This will increase the array length by 1.
        arr.push(i);
    }

    function pop() public {
        // Remove last element from array
        // This will decrease the array length by 1
        arr.pop();
    }

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

    function remove(uint index) public {
        // Delete does not change the array length.
        // It resets the value at index to it's default value,
        // in this case 0
        delete arr[index];
    }

    function examples() external {
        // create array in memory, only fixed size can be created
```

```
        uint[] memory a = new uint[](5);
    }
}
```

Oral Questions

1. What is Solidity?
2. What is Remix IDE
3. How to work Solidity Compilation Steps.
4. Difference Mainnet vs Testnet
5. What is solidity array
6. What funcation?

Assignment Questions

Solidity example pass parameter using function

Conclusion:

In this way we have studied basics of Solidity example using arrays and functions.
And how to create hello word using solidly programming

Assignment No: 4

Title of the Assignment: Deploy a simple contract to the Ethereum blockchain..

Objective of the Assignment: Deploy a simple contract to the Ethereum blockchain.

Prerequisite: Basic simple contract to the Ethereum blockchain.

Theory:

Ethereum is an open-source, public, blockchain-based distributed computing platform featuring smart contract (scripting) functionality.

Ethereum provides a cryptocurrency token called "ether", which acts as a vehicle for moving around on the Ethereum platform.

Ether is the value token of the Ethereum blockchain, though Ethereum's network supports other digital currencies.

The majority of ICO tokens issued on Ethereum are ERC 20 tokens that have built-in compliance with the set standard. Ethereum was proposed in late 2013 by Vitalik Buterin, a cryptocurrency researcher and programmer.

The development was funded by an online crowd sale during July–August 2014.

The system went live in 2015, with 72 million coins "premined" for the crowd sale.

This accounts for about 68 percent of the total circulating supply in 2018. Ethereum is currently the second-largest blockchain after Bitcoin by market cap, with a value nearing $14 billion as of May 2018.

The rapid growth of Ethereum has brought about an explosion in ICOs, which have been crowdfunding millions in capital for new projects on smart contract creation platform ethereum.

Its key feature is that it allows smart contracts - any digital agreements that are self-executing for either party involved in an agreement.

Because it's an open-source platform, anyone can create their own cryptocurrency that runs on the ethereum network - but they are independent of the ethereum team.

Ether is the currency that fuels transactions on ethereum, which means you need to buy Ether if you want to run dapps or make smart contracts. Ether has been referred to as fuel for the ethereum ecosystem.

In the future, that could very well change as Ether is not just a currency but also acts as a commodity as it will have to be consumed to access specific dapps on the network

Ethereum vs. Bitcoin

Ethereum's network is like Bitcoin in many ways, except that it has one major improvement - smart contracts. It was created as an improvement on bitcoin, but due to its popularity and success, it now competes against other currencies like Litecoin (LTC), Ripple (XRP), etc.

How to Buy Ether?

In order to buy Ether, you'll need to open an account on a crypto exchange. Many exchanges are available, but the most popular ones for buying Ether are Robinhood, Coinbase, Kraken, Bitstamp, and Gemini.

The History of Ethereum

Vitalik Buterin first described Ethereum in late 2013.

The system provides a decentralized Turing-complete virtual machine, the Ethereum Virtual Machine (EVM), which can execute scripts using an international network of public nodes.

The system went live on July 30, 2015, with 72 million coins premined for the crowd sale and 14 million coins more mined every year after that.

This accounts for about 68 percent of the total circulating supply in 2018. Ether is a cryptocurrency whose blockchain is generated by the Ethereum platform.

It is listed under the code ETH and traded on cryptocurrency exchanges, and the Greek uppercase Xi character (Ξ) is generally used for its currency symbol.

Create and Deploy your Smart Contract

Step 1: Connect to the Ethereum network

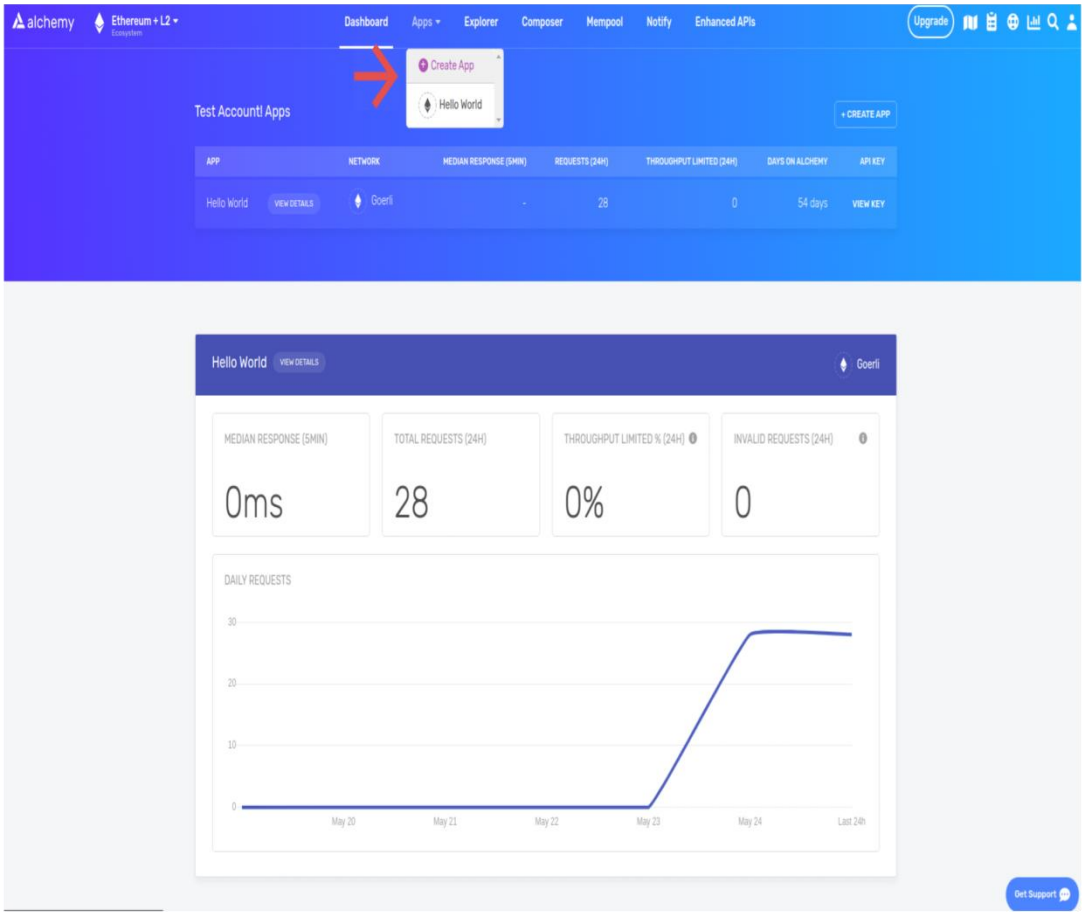
There are many ways to make requests to the Ethereum chain. For simplicity, we'll use a free account on Alchemy, a blockchain developer platform and API that allows us to communicate with the Ethereum chain without having to run our own nodes. The platform also has developer tools for monitoring and analytics that we'll take advantage of in this tutorial to understand what's going on under the hood in our smart contract deployment.

If you don't already have an Alchemy account, sign up for free here.

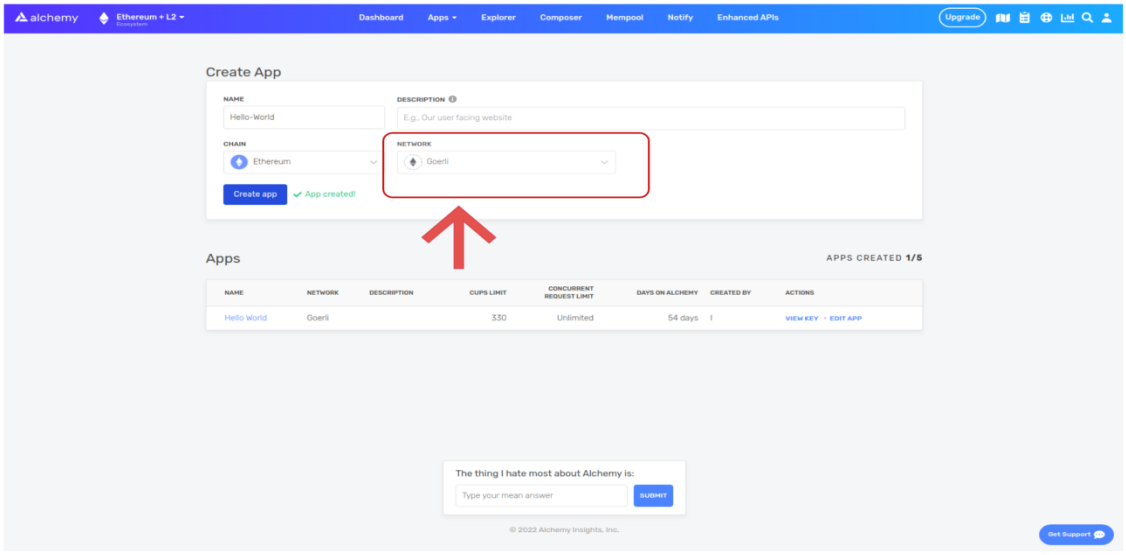
Step 2: Create your app (and API key)

Once you've created an Alchemy account, you can generate an API key by creating an app. This will allow us to make requests to the Goerli test network. If you're not familiar with testnets, check out this guide.

Navigate to the "Create App" page in your Alchemy Dashboard by hovering over "Apps" in the nav bar and clicking "Create App"



Name your app "Hello World", offer a short description, select "Staging" for the Environment (used for your app bookkeeping), and choose "Goerli" for your network.



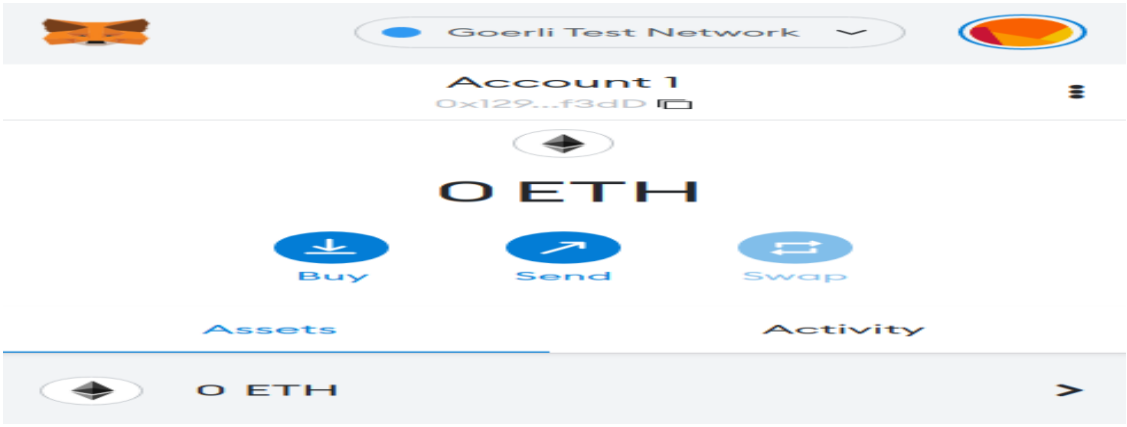
Double check that you're selecting the Goerli testnet!

Click “Create app” and that’s it! Your app should appear in the table below.

Step 3: Create an Ethereum account (address)

We need an Ethereum account to send and receive transactions. For this tutorial, we’ll use Metamask, a virtual wallet in the browser used to manage your Ethereum account address. If you want to understand more about how transactions on Ethereum work, check out this page from the Ethereum foundation.

You can download and create a Metamask account for free here. When you are creating an account, or if you already have an account, make sure to switch over to the “Goerli Test Network” in the upper right (so that we’re not dealing with real money).



Step 4: Add ether from a Faucet

In order to deploy our smart contract to the test network, we'll need some fake Eth. To get Eth you can go to the Goerli faucet and enter your Goerli account address, then click "Send Me Eth." It may take some time to receive your fake Eth due to network traffic. (At the time of writing this, it took around 30 minutes.) You should see Eth in your Metamask account soon after!

Step 5: Check your Balance

To double check our balance is there, let's make an eth_getBalance request using Alchemy's composer tool. This will return the amount of Eth in our wallet. Check out this video for instructions on how to use the composer tool!

After you input your Metamask account address and click "Send Request", you should see a response that looks like this:

{"jsonrpc": "2.0", "id": 0, "result": "0x2B5E3AF16B1880000"}

NOTE: This result is in wei not eth. Wei is used as the smallest denomination of ether. The conversion from wei to eth is: 1 eth = 10^18 wei. So if we convert 0x2B5E3AF16B1880000 to decimal we get 5*10^18 which equals 5 eth. Phew! Our fake money is all there □.

Step 6: Initialize our project

mkdir hello-world
cd hello-world

First, we'll need to create a folder for our project. Navigate to your command line and type:

Now that we're inside our project folder, we'll use npm init to initialize the project. If you don't already have npm installed, follow these instructions (we'll also need Node.js so download that too!).

npm init # (or npm init --yes)

It doesn’t really matter how you answer the installation questions, here is how we did it for reference:

```
package name: (hello-world)
version: (1.0.0)
description: hello world smart contract
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)

About to write to /Users/.../.../.../hello-world/package.json:

{
  "name": "hello-world",
  "version": "1.0.0",
  "description": "hello world smart contract",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Approve the package.json and we’re good to go!

Step 7: Download **Hardhat**

Hardhat is a development environment to compile, deploy, test, and debug your Ethereum software. It helps developers when building smart contracts and dApps locally before deploying to the live chain.

Inside our hello-world project run:

npm install --save-dev hardhat

Check out this page for more details on installation instructions.

Step 8: Create Hardhat project

Inside our hello-world project folder, run:

npx hardhat

You should then see a welcome message and option to select what you want to do. Select “create an empty hardhat.config.js”:

```
888    888                  888 888            888
888    888                  888 888            888
888    888                  888 888            888
8888888888  8888b.  888d888 .d88888 88888b.   8888b.  888888
888    888     "88b 888P"  d88" 888 888 "88b     "88b 888
888    888 .d888888 888    888  888 888  888 .d888888 888
888    888 888  888 888    Y88b 888 888  888 888  888 Y88b.
888    888 "Y888888 888     "Y88888 888  888 "Y888888  "Y888

 Welcome to Hardhat v2.0.11 

What do you want to do? …
Create a sample project
❯ Create an empty hardhat.config.js
Quit
```

This will generate a hardhat.config.js file for us, which is where we’ll specify all of the set up for our project (on step 13).

Step 9: Add project folders

To keep our project organized we’ll create two new folders. Navigate to the root directory of your hello-world project in your command line and type

mkdir contracts

mkdir scripts

contracts/ is where we'll keep our hello world smart contract code file

scripts/ is where we'll keep scripts to deploy and interact with our contract

Step 10: Write our contract

You might be asking yourself, when the heck are we going to write code?? Well, here we are, on Step 10 □

Open up the hello-world project in your favorite editor (we like VSCode). Smart contracts are written in a language called Solidity which is what we will use to write our HelloWorld.sol smart contract.

1. Navigate to the "contracts" folder and create a new file called HelloWorld.sol

2. Below is a sample Hello World smart contract from the Ethereum Foundation that we will be using for this tutorial. Copy and paste in the contents below into your HelloWorld.sol file, and be sure to read the comments to understand what this contract does:

```
// Specifies the version of Solidity, using semantic versioning.
// Learn more: https://solidity.readthedocs.io/en/v0.5.10/layout-of-source-files.html#pragma
pragma solidity >=0.7.3;

// Defines a contract named `HelloWorld`.
// A contract is a collection of functions and data (its state). Once deployed, a contract resides at a specific address on the Ethereum blockchain. Learn more: https://solidity.readthedocs.io/en/v0.5.10/structure-of-a-contract.html
contract HelloWorld {

  //Emitted when update function is called
  //Smart contract events are a way for your contract to communicate that something happened on the blockchain to your app front-end, which can be 'listening' for certain events and take action when they happen.
  event UpdatedMessages(string oldStr, string newStr);
```

```
  // Declares a state variable `message` of type `string`.
  // State variables are variables whose values are permanently stored in contract storage. The keyword `public` makes variables accessible from outside a contract and creates a function that other contracts or clients can call to access the value.
  string public message;

  // Similar to many class-based object-oriented languages, a constructor is a special function that is only executed upon contract creation.
  // Constructors are used to initialize the contract's data. Learn more:https://solidity.readthedocs.io/en/v0.5.10/contracts.html#constructors
  constructor(string memory initMessage) {

    // Accepts a string argument `initMessage` and sets the value into the contract's `message` storage variable).
    message = initMessage;
  }

  // A public function that accepts a string argument and updates the `message` storage variable.
  function update(string memory newMessage) public {
    string memory oldMsg = message;
    message = newMessage;
    emit UpdatedMessages(oldMsg, newMessage);
  }
}
```

This is a super simple smart contract that stores a message upon creation and can be updated by calling the update function.

Step 11: Connect Metamask & Alchemy to your project

We've created a Metamask wallet, Alchemy account, and written our smart contract, now it's time to connect the three.

Every transaction sent from your virtual wallet requires a signature using your unique private key. To provide our program with this permission, we can safely store our private key (and Alchemy API key) in an environment file.

To learn more about sending transactions, check out this tutorial on sending transactions using web3.

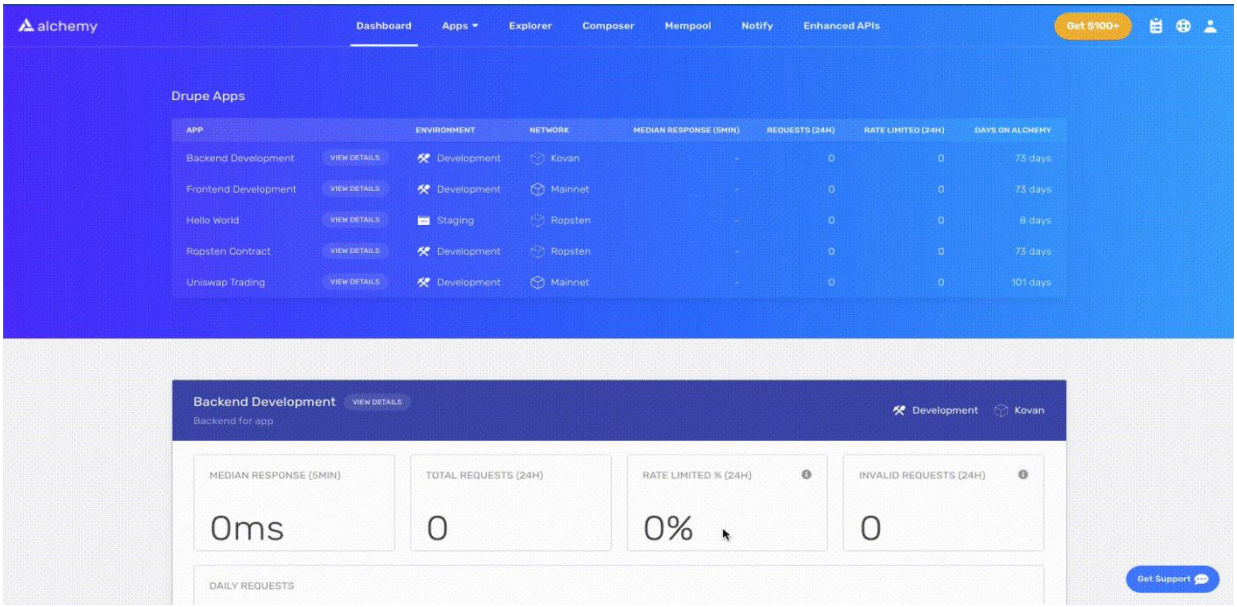
First, install the dotenv package in your project directory:

npm install dotenv --save

This is a super simple smart contract that stores a message upon creation and can be updated by calling the update function.

Your environment file must be named .env or it won't be recognized as an environment file.Do not name it process.env or .env-custom or anything else.

- Follow these instructions to export your private key
- See below to get HTTP Alchemy API URL



Your .env should look like this:

API_URL = "https://eth-goerli.alchemyapi.io/v2/your-api-key"
PRIVATE_KEY = "your-metamask-private-key"

To actually connect these to our code, we'll reference these variables in our hardhat.config.js file on step 13.

To actually connect these to our code, we'll reference these variables in our hardhat.config.js file on step 13.

Step 12: Install Ethers.js

Ethers.js is a library that makes it easier to interact and make requests to Ethereum by wrapping standard JSON-RPC methods with more user friendly methods.

Hardhat makes it super easy to integrate Plugins for additional tooling and extended functionality. We'll be taking advantage of the Ethers plugin for contract deployment (Ethers.js has some super clean contract deployment methods).

In your project directory type:

npm install --save-dev @nomiclabs/hardhat-ethers "ethers@^5.0.0"

We'll also require ethers in our hardhat.config.js in the next step.

Step 13: Update hardhat.config.js

We've added several dependencies and plugins so far, now we need to update hardhat.config.js so that our project knows about all of them.

Update your hardhat.config.js to look like this:

```
/**
* @type import('hardhat/config').HardhatUserConfig
*/

require('dotenv').config();
require("@nomiclabs/hardhat-ethers");

const { API_URL, PRIVATE_KEY } = process.env;

module.exports = {
  solidity: "0.7.3",
  defaultNetwork: "goerli",
  networks: {
```

```
    hardhat: {},
    goerli: {
      url: API_URL,
      accounts: [`0x${PRIVATE_KEY}`]
    }
  },
}
```

Step 14: Compile our contract

To make sure everything is working so far, let's compile our contract. The compile task is one of the built-in hardhat tasks.

From the command line run:

```
npx hardhat compile
```

You might get a warning about SPDX license identifier not provided in source file, but no need to worry about that — hopefully everything else looks good!

If not, you can always message in the Alchemy discord.

Step 15: Write our deploy script

Now that our contract is written and our configuration file is good to go, it's time to write our contract deploy script.

Navigate to the /scripts folder and create a new file called deploy.js, adding the following contents to it:

```
async function main() {
  const HelloWorld = await ethers.getContractFactory("HelloWorld");

  // Start deployment, returning a promise that resolves to a contract object
  const hello_world = await HelloWorld.deploy("Hello World!");
  console.log("Contract deployed to address:", hello_world.address);
```

```
}

main()
  .then(() => process.exit(0))
  .catch(error => {
    console.error(error);
    process.exit(1);
  });
```

Hardhat does an amazing job of explaining what each of these lines of code does in their Contracts tutorial, we've adopted their explanations here.

```
const HelloWorld = await ethers.getContractFactory("HelloWorld");
```

A ContractFactory in ethers.js is an abstraction used to deploy new smart contracts,
so HelloWorld here is a factory for instances of our hello world contract. When using the hardhat-ethers plugin ContractFactory and Contract, instances are connected to the first signer (owner) by default.

```
const hello_world = await HelloWorld.deploy();
```

Calling deploy() on a ContractFactory will start the deployment, and return a Promise that resolves to a Contract object. This is the object that has a method for each of our smart contract functions.

Step 16: Deploy our contract

We're finally ready to deploy our smart contract! Navigate to the command line and run:

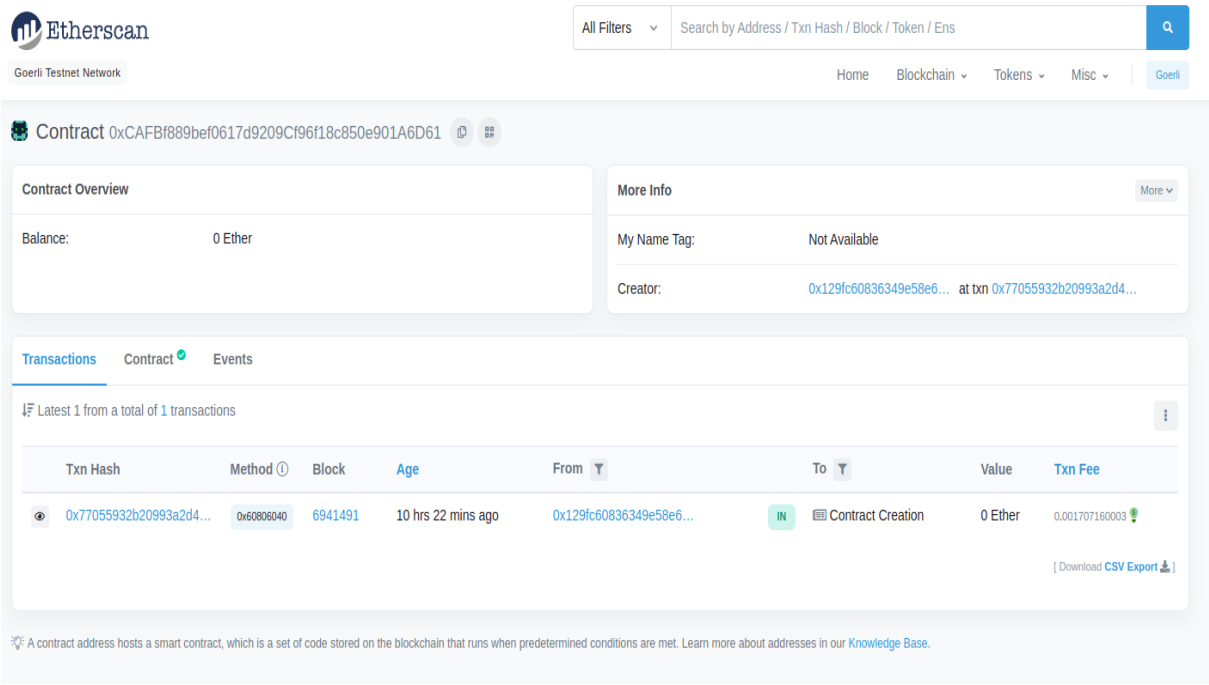
```
npx hardhat run scripts/deploy.js --network goerli
```

You should then see something like:

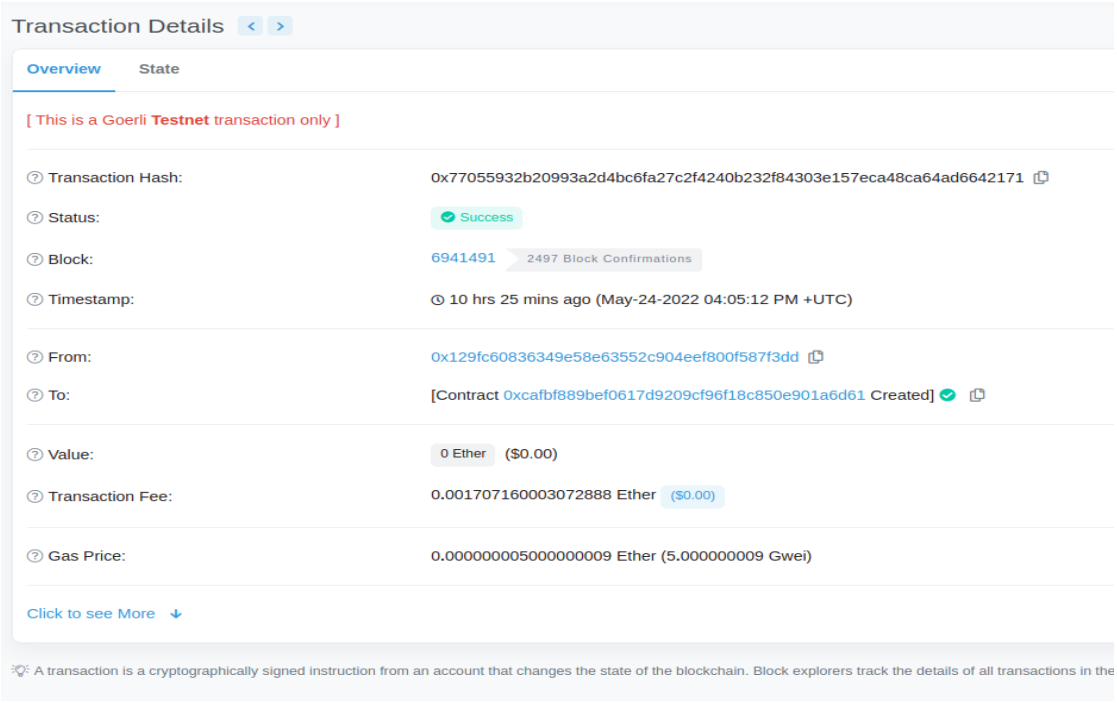
Contract deployed to address: 0xCAFBf889bef0617d9209Cf96f18c850e901A6D61

Please copy and paste this address to save it somewhere, as we will be using this address for later tutorials, so you don't want to lose it.

If we go to the Goerli etherscan and search for our contract address we should able to see that it has been deployed successfully. The transaction will look something like this:



The From address should match your Metamask account address and the To address will say "Contract Creation" but if we click into the transaction we'll see our contract address in the To field:



Congrats! You just deployed a smart contract to the Ethereum chain □

Oral Questions

1. What is Ethereum chain?
2. What is Remix IDE
3. What is solidity array

Assignment Questions

deployed a smart contract to the Ethereum chain

Conclusion:

In this way we have studied how to deployed a smart contract to the Ethereum blockchain

Assignment No: 5

Title of the Assignment: Polling / voting system using Solidity, Ethereum and a data structure hashmap(optional)..

Objective of the Assignment: Polling / voting system using Solidity, Ethereum and a data structure hashmap(optional).

Prerequisite: Basic simple contract Polling / voting system using Solidity, Ethereum and a data structure hashmap(optional)

Theory:

Ethereum is an open-source, public, blockchain-based distributed computing platform featuring smart contract (scripting) functionality.

Ethereum provides a cryptocurrency token called "ether", which acts as a vehicle for moving around on the Ethereum platform.

Ether is the value token of the Ethereum blockchain, though Ethereum's network supports other digital currencies.

The majority of ICO tokens issued on Ethereum are ERC 20 tokens that have built-in compliance with the set standard. Ethereum was proposed in late 2013 by Vitalik Buterin, a cryptocurrency researcher and programmer.

The development was funded by an online crowd sale during July–August 2014.

The system went live in 2015, with 72 million coins "premined" for the crowd sale.

This accounts for about 68 percent of the total circulating supply in 2018. Ethereum is currently the second-largest blockchain after Bitcoin by market cap, with a value nearing $14 billion as of May 2018.

The rapid growth of Ethereum has brought about an explosion in ICOs, which have been crowdfunding millions in capital for new projects on smart contract creation platform ethereum.

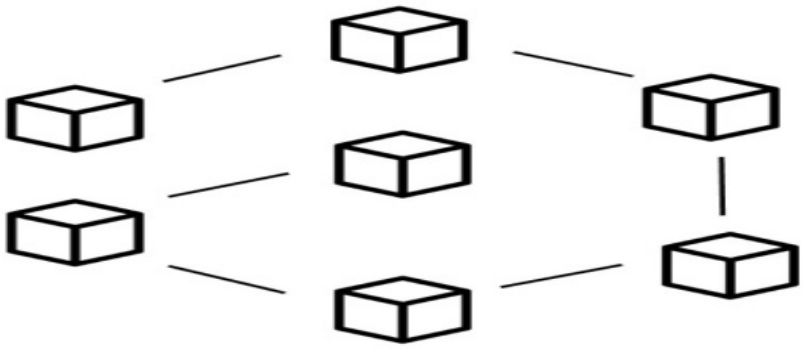
Its key feature is that it allows smart contracts - any digital agreements that are self-executing for either party involved in an agreement.

Because it's an open-source platform, anyone can create their own cryptocurrency that runs on the ethereum network - but they are independent of the ethereum team.

Ether is the currency that fuels transactions on ethereum, which means you need to buy Ether if you want to run dapps or make smart contracts. Ether has been referred to as fuel for the ethereum ecosystem.

In the future, that could very well change as Ether is not just a currency but also acts as a commodity as it will have to be consumed to access specific dapps on the network

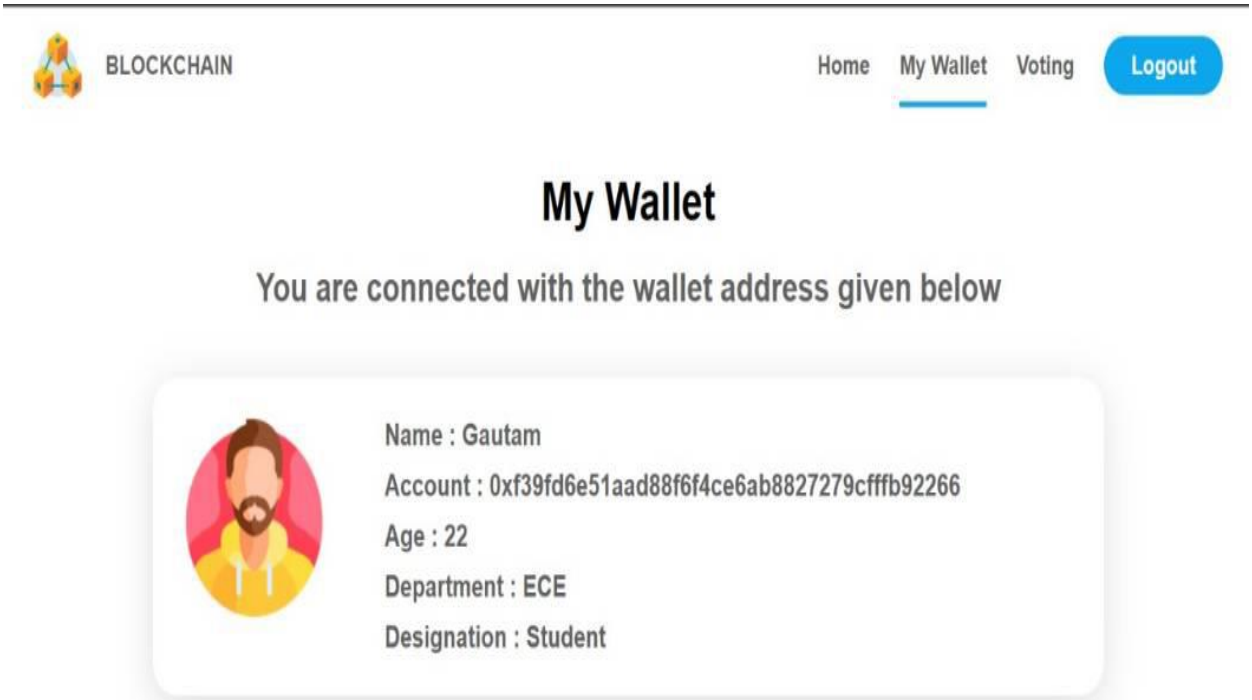
- Block chain technology offers a decentralized node for online voting or electronic voting.
- Recently distributed ledger technologies like block chain were used to produce electronic voting systems mainly due to their end-to-end verification advantages.

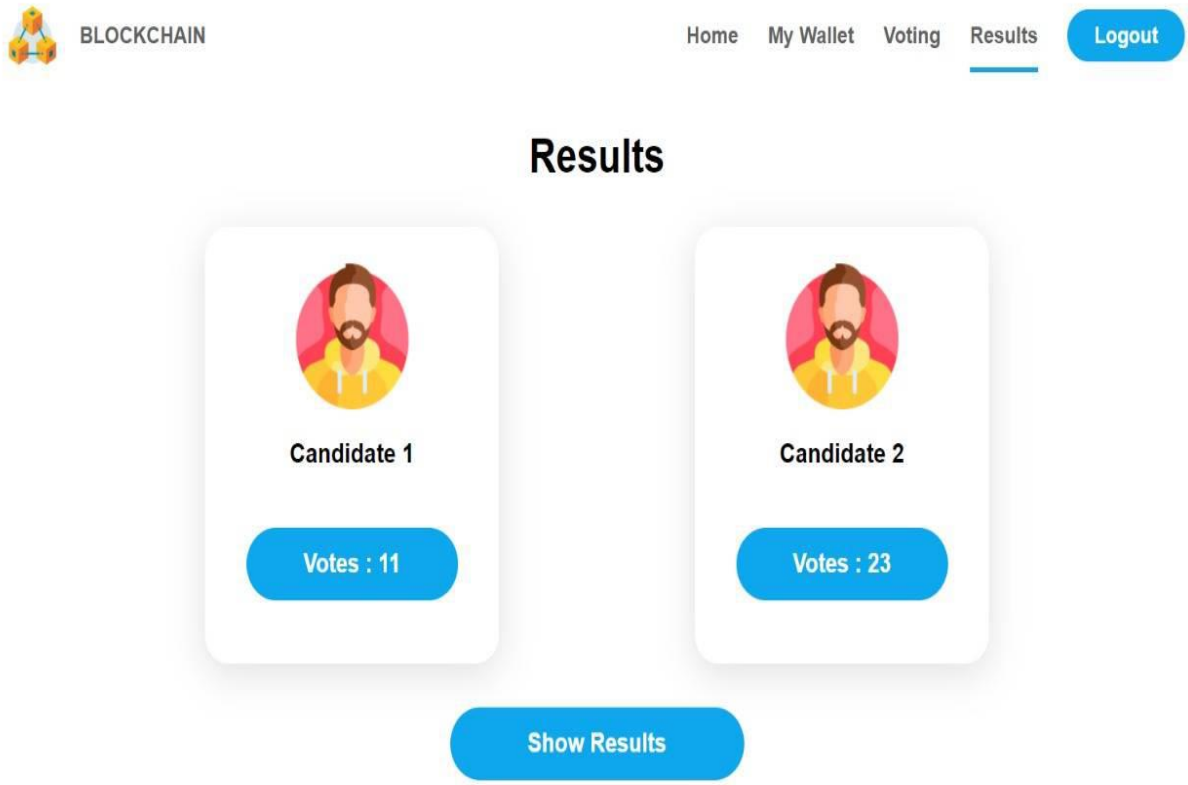


Program for system

```
contract Voting {
    int256 public candidate1 = 0;
    int256 public candidate2 = 0;

    mapping(address => bool) public voters;

    function vote1(address voterAddress) public {
        bool checkIfAlreadyVoted = voters[voterAddress];
        if (!checkIfAlreadyVoted) {
            candidate1++;
            voters[voterAddress] = true;
        }
    }

    function vote2(address voterAddress) public {
        bool checkIfAlreadyVoted = voters[voterAddress];
        if (!checkIfAlreadyVoted) {
            candidate2++;
            voters[voterAddress] = true;
        }
    }

    function getResults() public view returns (int256[2] memory) {
        return [candidate1, candidate2];
    }
}
```

```
const { address, candidateNo } = req.body;

    const existingUser = await User.findOne({ address });
    if (!existingUser) {
      throw httpError("User not found");
    }

    if (existingUser.age < 18) {
      throw httpError("User is below 18 yrs of age");
    }

    if (existingUser.department != "ECE") {
      throw httpError("User must belong to ECE Department");
    }

    if (existingUser.designation != "Student") {
      throw httpError("User must be a student");
    }

    res.status(422).send({
      message: "Successfully voted",
    });
```


BLOCKCHAIN
Home
My Wallet
Voting
Logout
My Wallet
You are connected with the wallet address given below
Name : Gautam
Account : 0xf39fd6e51aad88f6f4ce6ab8827279cfffb92266
Age : 22
Department : ECE
Designation : Student


BLOCKCHAIN
Home
My Wallet
Voting
Results
Logout
Results
Candidate 1
Votes : 11
Candidate 2
Votes : 23
Show Results

Oral Questions

1. What is Ethereum chain?
2. What is Remix IDE
3. What is solidity array

Conclusion:
In this way we have studied Basic simple contract Polling / voting system using Solidity, Ethereum

We tried to shift traditional physical voting methods to a block chain-based environment.

We were able to do all this with the help of the Ethereum network. Which provided us with the necessary tools required to achieve the desired result