

Project Description:

Blockchain is a technology designed to manage education data that has the potential to support transparency and accountability. A blockchain is a ledger of transactions where an identical copy is visible to all the members of a computer network. Network members validate the data entered into the ledger, and once entered, the data is immutable.

Design a solution where you can store the digital certificates of the students in a distributed and decentralized network. You should be able to

- add the certificated details into the blockchain
- query the certificate details from the blockchain.

Introduction:

A Smart Contract for educational institution is to add the student's certificate details like 'Name', 'Institution name', 'Date of Issue' in the block chain network, and can able to query the details of particular certificate details by calling their 'Name', or "id".

Approach:

- ✓ The first step is to deploy the smart contract using the Remix IDE. After writing the code compile the code. When it is successfully compiled then deploy it.
- ✓ After deploying the contract a deployed Contract is obtained and then add the Certificate details to the network. And hit the **transact** button.
- ✓ Once the details were added to the network, we can verify the details by **call** the details by using the student name.

Algorithm:

Step-1: Choose a blockchain platform:

Step-2: Define the data structure. The data should include details such as the student's name, the name of the institution that issued the certificate, the date of issue, the type of certificate, and any other relevant information.

Step-3: Develop a smart contract that defines the rules for adding and querying certificate details. The smart contract should include functions for adding new certificates to the blockchain and for querying the details of a specific certificate.

Step-4: Deploy the smart contract on the blockchain network.

Step-5: Add certificates to the blockchain. The smart contract will validate the data and add it to the blockchain.

Step-6: Query certificate details: A user can access the blockchain network and search for the certificate using the student's name or other relevant information. The blockchain will return the certificate details, which can be verified by anyone on the network.

Step-7: By entering the student's name or certificate id, we can able to get the details of that particular certificate from the block chain network.

Certificate Smart Contract

This smart contract will have the basic functionalities like,

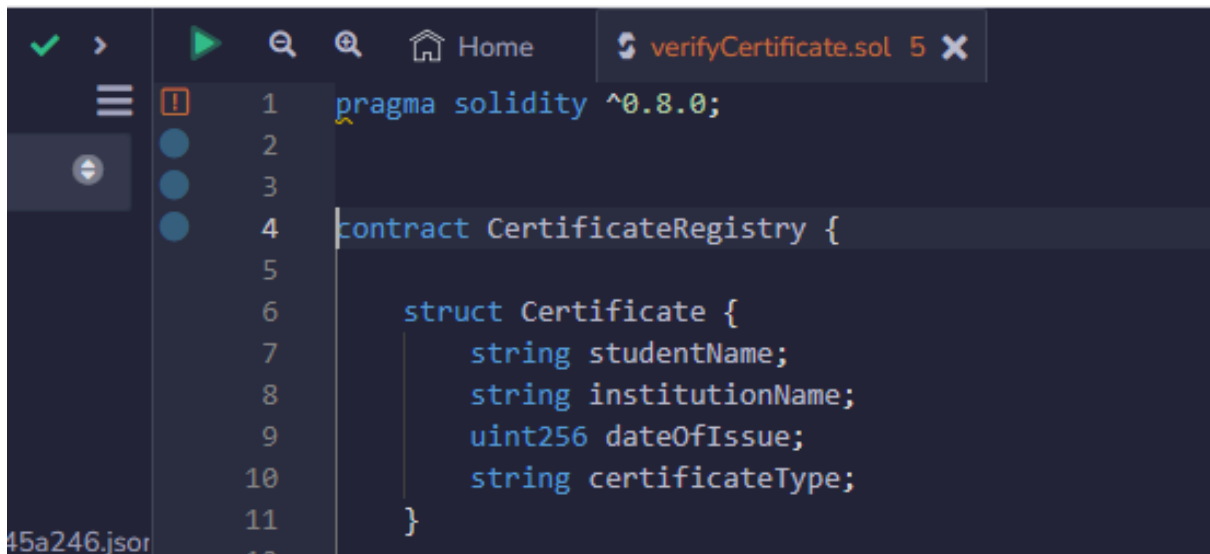
- Adding the certificate details to the network
- Getting the certificate details from network by using student 's name.

First we need to add solidity compiler version

A screenshot of a code editor interface. At the top, there is a search bar, a magnifying glass icon, a home icon, and a tab labeled 'verifyCertificate.sol' with a close button. The main editing area has a dark background with light blue text. Line numbers 1, 2, 3, and 4 are visible on the left. Line 1 contains the code 'pragma solidity ^0.8.0;'.

```
1  pragma solidity ^0.8.0;  
2  
3  
4
```

Then creating certificate contract

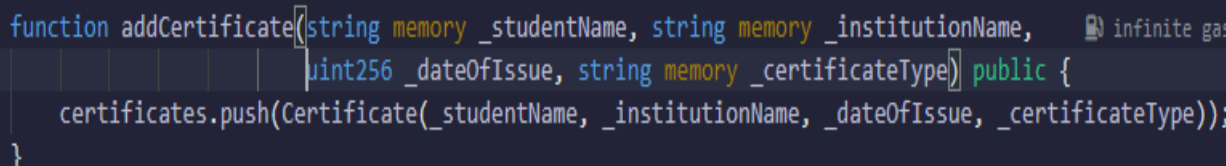


```
1 pragma solidity ^0.8.0;
2
3
4 contract CertificateRegistry {
5
6     struct Certificate {
7         string studentName;
8         string institutionName;
9         uint256 dateOfIssue;
10        string certificateType;
11    }
12
```

Functions used in certificate contract

1. addCertificate()

- ✓ Define all the required parameters with respective datatype inside the public key arguments.
- ✓ Append the collected data inside the network.
- ✓



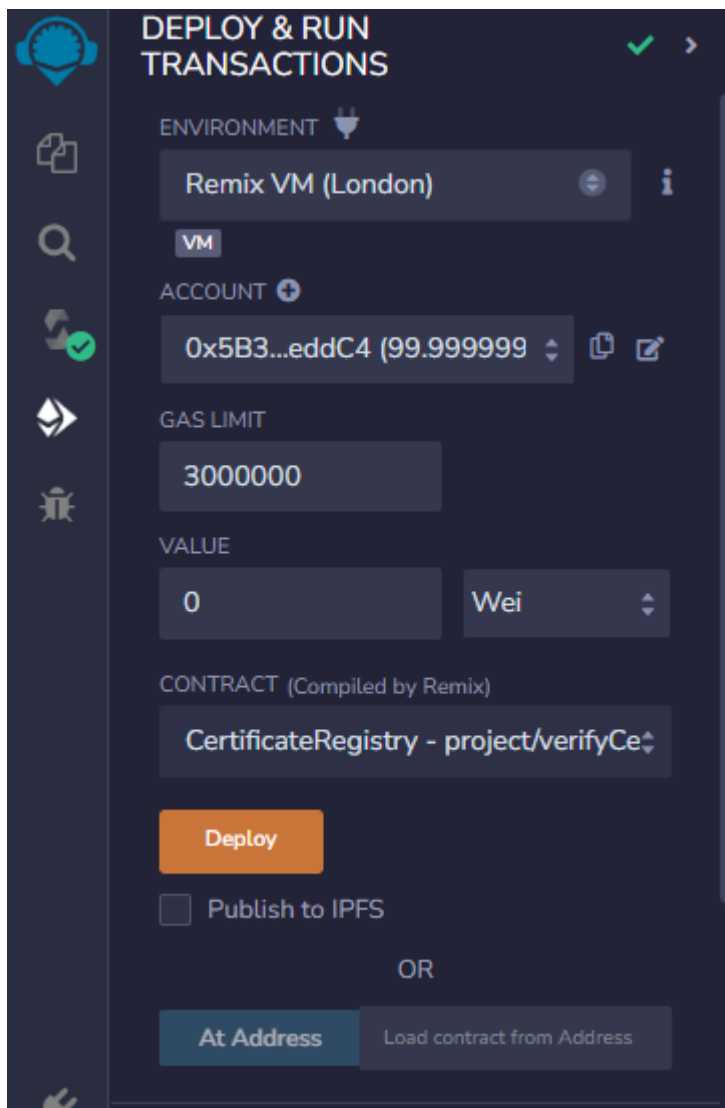
```
function addCertificate(string memory _studentName, string memory _institutionName, uint256 _dateOfIssue, string memory _certificateType) public {
    certificates.push(Certificate(_studentName, _institutionName, _dateOfIssue, _certificateType));
}
```

2. getCertificateDetailsByName()



- ✓ Use the cryptographic hash function (keccak256) takes an input of an arbitrary length and produces a fixed length output of 256 bits.
- ✓ Get the value either as name or certificate id as input from the user and compare it with the database and return the targeted certificate details.

```
function getCertificateDetailsByName(string memory _studentName) public view returns
    ((string memory, string memory, uint256, string memory)) {
    for (uint i=0; i<certificates.length; i++) {
        if (keccak256(bytes(certificates[i].studentName)) == keccak256(bytes(_studentName))) {
            return (certificates[i].studentName, certificates[i].institutionName, certificates[i].dateOfIssue,
                certificates[i].certificateType);
        }
    }
}
```




Deploy the file and then insert the multiple certificate details to the network using the addCertificate Transact method.




DEPLOY & RUN TRANSACTIONS

ENVIRONMENT  **Remix VM (London)** 

VM

ACCOUNT  **0x5B3...eddC4 (99.999999)**  

GAS LIMIT **3000000**

VALUE **0** **Wei** 

CONTRACT (Compiled by Remix) **CertificateRegistry - project/verifyCe**

Deploy

☐ Publish to IPFS

OR

At Address

[vm] from: 0x5B3...eddC4 to: CertificateRegistry.(constructor) value: 0 wei data: 0x608...20033 logs: 0

hash: 0x688...b601e

Debug

status

true Transaction mined and execution succeed

transaction hash

0x68833a81a49045cda3c1a6734bd8d338416a8b8b98df5e5f2762ded2675b601e

from

0x5B380a6a701c568545dCfc803Fc8875F56beddC4

to

CertificateRegistry.(constructor)

gas

956631 gas

transaction cost

831853 gas

execution cost

723553 gas

input

0x608...20033

Deployed Contracts

▼

CERTIFICATEREGISTRY AT 0X5F

Balance: 0 ETH

addCertificate

string _studentName, string _i

▼

certificates

uint256

▼

getCertificate

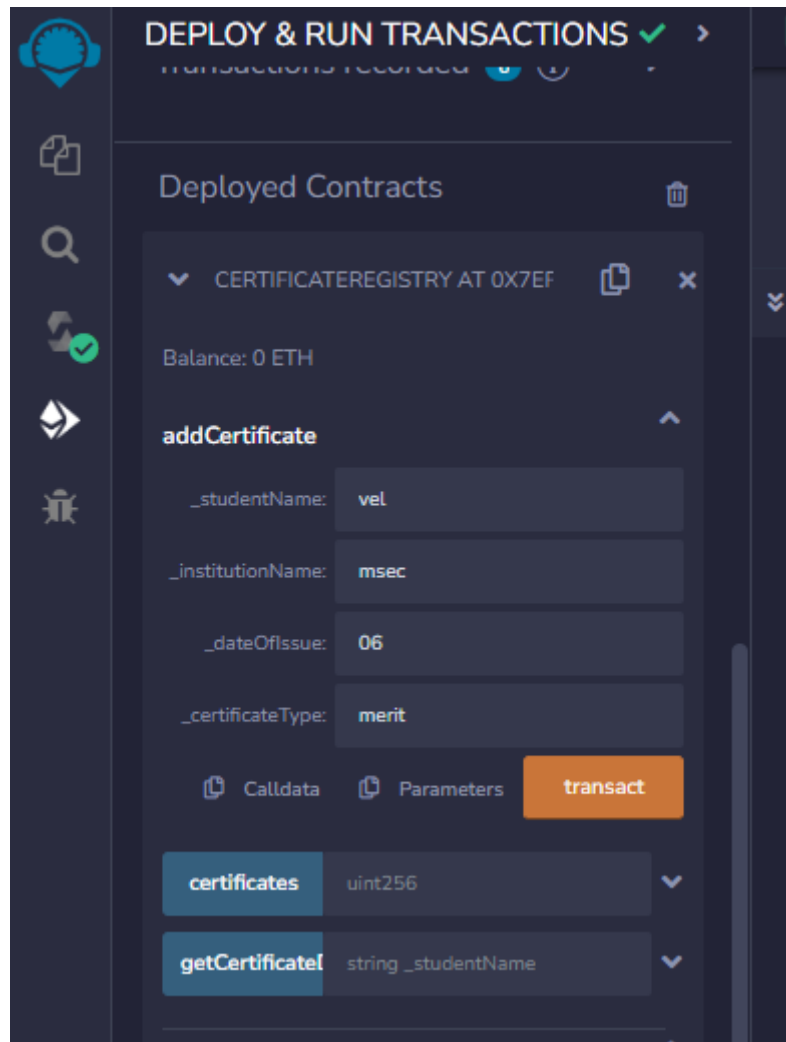
string _studentName

▼

Low level interactions

CALLDATA

Transact



To **view** the particular certificate details from the network, use **the call method** by entering the certificate id or name of student as input.

