**PRACTICAL: 5**

**AIM:** Write a Smart-contract of EVENT PARTICIPATION CERTIFICATE whichincludes the following things

* Write code using 0.8.0 or higher version of solidity.
* Define a struct to define different properties of certificate issues.
* Define mapping function to struct
* Create a constructor to initialize the first transaction using admin only.
* Create function to issue a new certificate with certificate\_id,
* student\_name, branch, course\_name and grade
* Initialize proper datatypes of all variable.
* Display the certificate details by certificate\_id
* Verify the certificate hash.

Use online Ethereum REMIX IDE to compile and deploy the contract.

Integrate the metamask wallet to perform the transaction of certificate.

**CODE:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract EventCertificate {

    // Admin address

    address public admin;

    // Certificate structure

    struct Certificate {

        uint256 certificateId;

        string studentName;

        string branch;

        string courseName;

        string grade;

        bytes32 certificateHash;

    }

    // Mapping certificateId => Certificate

    mapping(uint256 => Certificate) public certificates;

    // Event to log certificate issuance

    event CertificateIssued(

        uint256 indexed certificateId,

        string studentName,

        string branch,

        string courseName,

        string grade,

        bytes32 certificateHash

    );

    // Modifier to restrict access to admin only

    modifier onlyAdmin() {

        require(msg.sender == admin, "Only admin can perform this action");

        \_;

    }

    // Constructor to set the admin at contract deployment

    constructor() {

        admin = msg.sender;

    }

    // Function to issue a new certificate

    function issueCertificate(

        uint256 \_certificateId,

        string memory \_studentName,

        string memory \_branch,

        string memory \_courseName,

        string memory \_grade

    ) public onlyAdmin {

        require(

            certificates[\_certificateId].certificateId == 0,

            "Certificate already exists"

        );

        // Generate a hash for the certificate

        bytes32 certificateHash = keccak256(

            abi.encodePacked(\_certificateId, \_studentName, \_branch, \_courseName, \_grade)

        );

        // Store certificate details

        certificates[\_certificateId] = Certificate(

            \_certificateId,

            \_studentName,

            \_branch,

            \_courseName,

            \_grade,

            certificateHash

        );

        emit CertificateIssued(

            \_certificateId,

            \_studentName,

            \_branch,

            \_courseName,

            \_grade,

            certificateHash

        );

    }

    // Function to get certificate details by certificateId

    function getCertificate(

        uint256 \_certificateId

    ) public view returns (

        uint256,

        string memory,

        string memory,

        string memory,

        string memory,

        bytes32

    ) {

        require(

            certificates[\_certificateId].certificateId != 0,

            "Certificate not found"

        );

        Certificate memory cert = certificates[\_certificateId];

        return (

            cert.certificateId,

            cert.studentName,

            cert.branch,

            cert.courseName,

            cert.grade,

            cert.certificateHash

        );

    }

    // Function to verify the certificate hash

    function verifyCertificate(

        uint256 \_certificateId,

        string memory \_studentName,

        string memory \_branch,

        string memory \_courseName,

        string memory \_grade

    ) public view returns (bool) {

        Certificate memory cert = certificates[\_certificateId];

        require(cert.certificateId != 0, "Certificate not found");

        // Recreate the hash and compare it

        bytes32 expectedHash = keccak256(

            abi.encodePacked(\_certificateId, \_studentName, \_branch, \_courseName, \_grade)

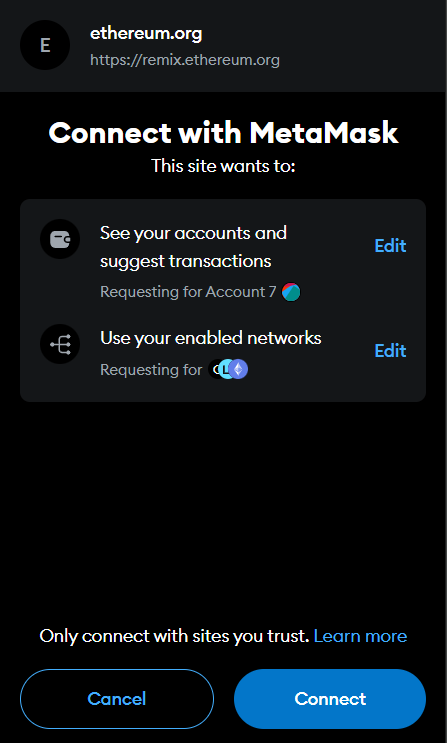
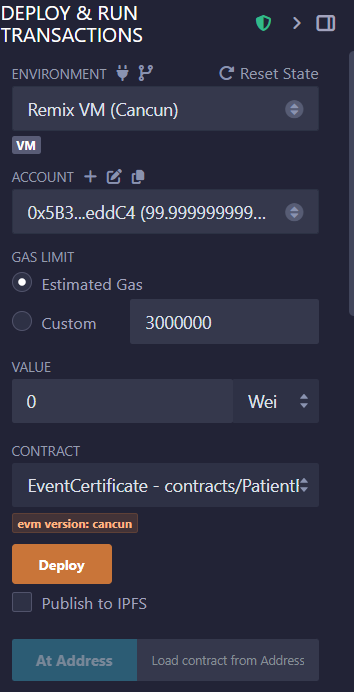
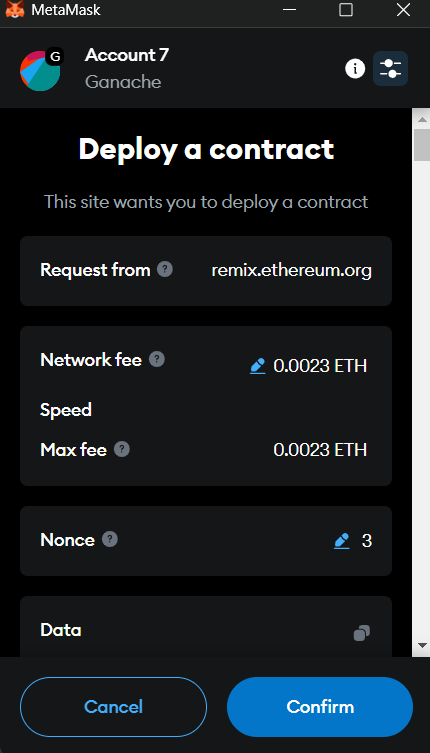
        );

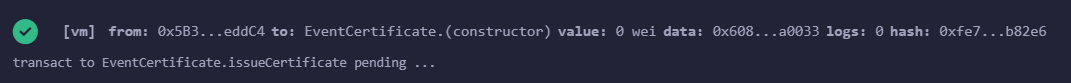
        return (cert.certificateHash == expectedHash);

    }

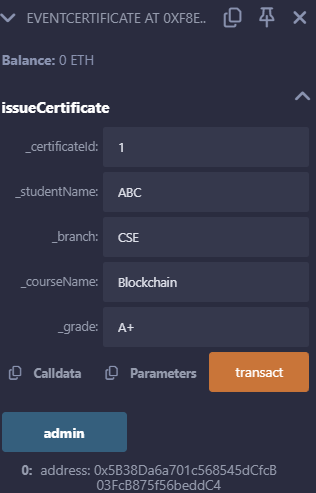
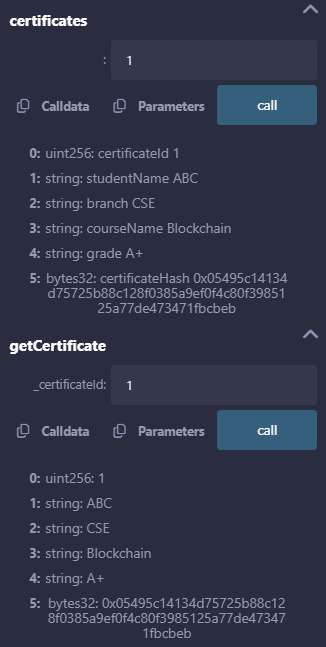
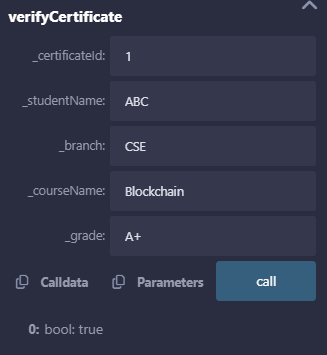
}

**OUTPUT:**

****

****

**  **

**LEARNING OUTCOME:**

A Solidity smart contract for event participation certificates allows secure issuance and verification of certificates on the Ethereum blockchain. The contract uses structs to store certificate details and mappings for efficient data retrieval. An admin-only constructor ensures that only authorized users can initialize transactions. Participants receive certificates with unique IDs, and their authenticity is verified using certificate hashes. The contract is deployed using Remix IDE, and Metamask integration enables secure blockchain transactions, ensuring transparency and immutability.