Setting Up for local Ganche and Truffle

Start with installing truffle using **npm install -g truffle** . This will install truffle globally

Error:-

**truffle : File c:\users\91966\appdata\roaming\npm\truffle.ps1 cannot be loaded because running scripts is disabled on this system.**

Solution :-

**Set-ExecutionPolicy -ExecutionPolicy Bypass -Scope CurrentUser**

paste this in IDE terminal

Check truffle version -

Using **truffle --version**  
for this project these are the configrationss

Truffle v5.11.5 (core: 5.11.5)

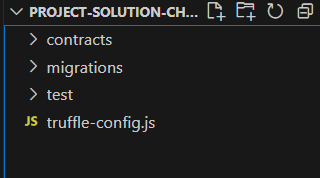
Ganache v7.9.1

Solidity v0.5.16 (solc-js)

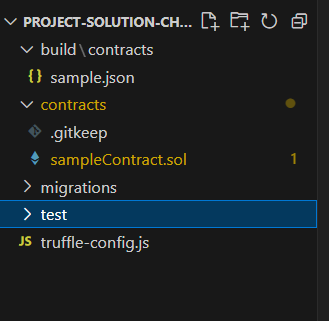
Node v18.17.0

Web3.js v1.10.0

Initial structure folder structure :-



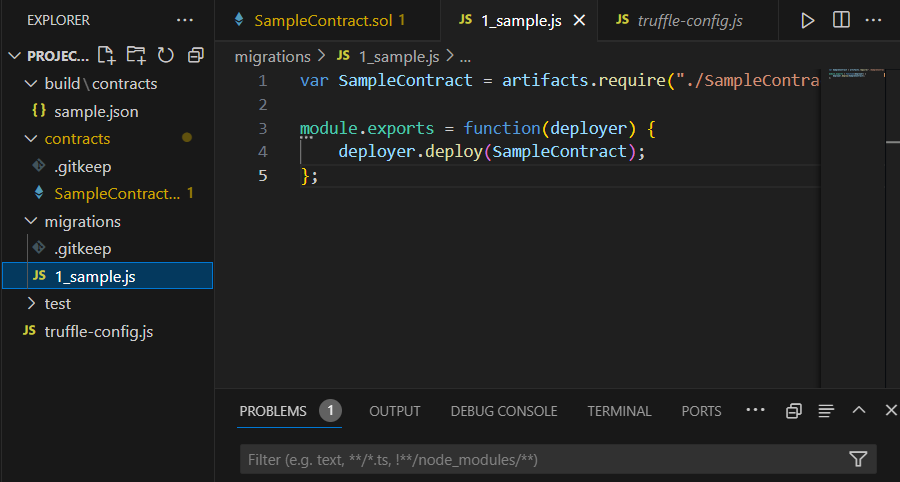
After Compilation



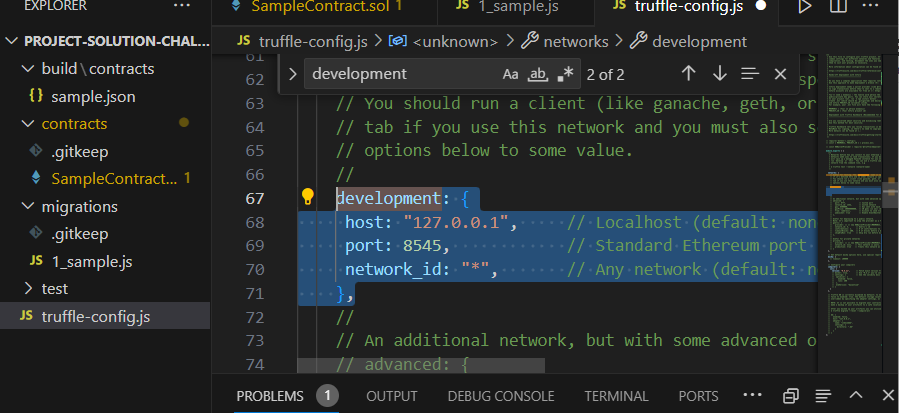
ABI and byte code get combined into a JSON file in ./build/sample.json

Migrating the smart contract

This means creating a script to deploy your smart contract on a network

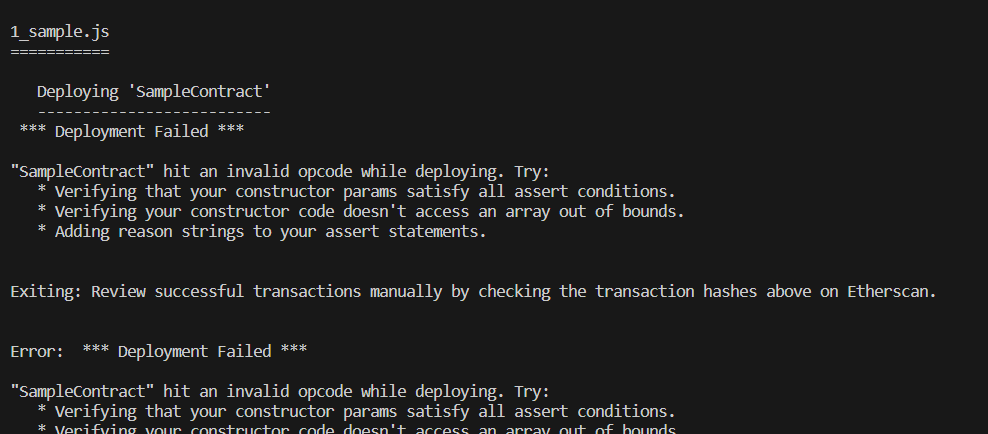


Deployment of smart contract on ganache



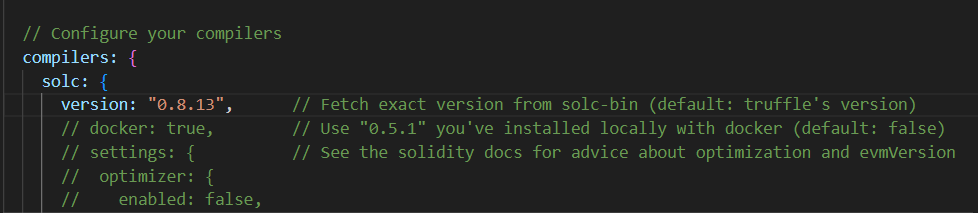
Uncomment the configration

Error :-

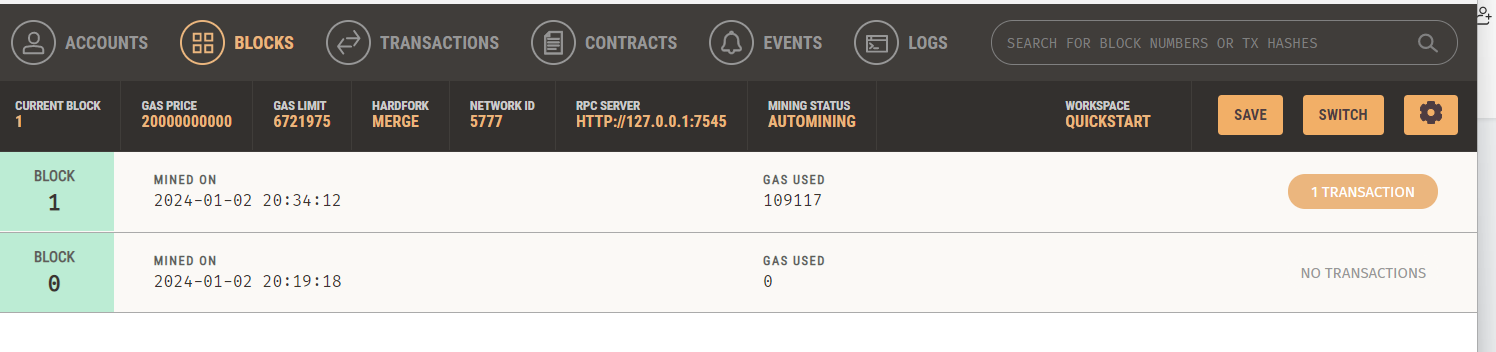


Solution :-

Downgrade Solidity compiler to 0.8.13 in truffle.js



Mined transaction in ganache



Result :- The Project is ready to develop Smart contract locally

Developing Smart Contracts

Contract for pushing student details to chain

contract StuDetails

{

    // Create a structure for

    // student details

    struct Student

    {

        uint256 ID;

        string FirstName;

        string LastName;

        address payable Address;

    }

    //Define variables

    address stuAdd;

    mapping(uint256 => Student) internal stuRecords;

    Student[] internal student;

    // Create a function to add

    // the new records

    function addStuRecords(uint256 \_ID,

                        string memory \_FirstName,

                        string memory \_LastName) public

    {   //error message if student try to enter using the same ID again

        require(stuRecords[\_ID].ID != \_ID,"\_\_\_\_Student ID already registered and cannot be altered\_\_\_\_");

        // Get the student address

        stuAdd = msg.sender;

        //Add to array

        student.push(Student({ID: \_ID, FirstName: \_FirstName,

                    LastName: \_LastName, Address: payable(stuAdd)}));

        // Fetch the student details

        // with the student ID

        stuRecords[\_ID] = Student(\_ID, \_FirstName,

                                    \_LastName, payable(stuAdd));

    }

    //function to view the student array

    function getStuDetails(uint256 \_ID) public view returns(uint256, string memory, string memory, address payable) {

        uint256 ID = stuRecords[\_ID].ID;

        string memory FirstName = stuRecords[\_ID].FirstName;

        string memory LastName = stuRecords[\_ID].LastName;

        address payable Address = stuRecords[\_ID].Address;

        return (ID, FirstName, LastName, Address);

    }

}

Scholarship Distribution Contract

// SPDX-License-Identifier: MIT

pragma solidity >=0.8.2 <0.9.0;

import "./priceConvertRinkeby.sol";

import "./StudentDetailsContract.sol";

contract ScholarDetails {

    //Get the addresses of another 3 contracts

    address PriceContractAdd; //this only relevant for running on rinkeby testnet to get value in GBP

    address StuContractAdd;

    address payable StaffContractAdd;

    constructor(address \_StuContractAdd, address \_PriceContractAdd) payable {

        StuContractAdd = \_StuContractAdd;

        PriceContractAdd = \_PriceContractAdd;

    }

    function storeContractAdd(address payable \_StaffContractAdd) public {

        StaffContractAdd = \_StaffContractAdd;

    }

    //This is mandatory for receiving ETH

    event ReceivedEth(uint256 amount);

    receive() external payable {

        emit ReceivedEth(msg.value);

    }

    fallback() external payable {

        emit ReceivedEth(msg.value);

    }

    //view the amount converted

    function viewGBPWEI(uint256 \_number) public view returns (uint256) {

        // PriceConversion p = PriceConversion(PriceContractAdd);

        // uint256 price = p.getGBPWEI(\_number);

        uint256 price = (\_number) \* 62539;

        return price;

    }

    // defining scholarship details struct

    struct Scholarship {

        uint256 ID;

        string ScholarshipName;

        uint256 Amount;

        address Provider;

        uint256 Attendance;

        uint256 AvgMark;

        string Status;

    }

    //Define variables

    address payable provAdd;

    string Status;

    mapping(uint256 => Scholarship) internal schlRecords;

    Scholarship[] internal scholarship;

    // Create a function to add

    // the new scholarship records

    function addSchlRecords(

        uint256 \_ID,

        string memory \_ScholarshipName,

        uint256 \_Amount,

        uint256 \_Attendance,

        uint256 \_AvgMark

    ) public payable {

        //Convert the amount from GBP to wei

        // PriceConversion p = PriceConversion(PriceContractAdd);

        // uint256 cvrAmt = p.getGBPWEI(\_Amount);

        uint256 cvrAmt = (\_Amount)\*62539;

        //Show error if there is not enough balance to offer the scholarship

        require(

            cvrAmt <= address(this).balance,

            "\_\_\_\_This wallet does not have enough balance to offer the scholarship\_\_\_\_"

        );

        //Get the student ID stored on student contract

        uint256 stuID;

        StuDetails stu = StuDetails(StuContractAdd);

        (stuID, , , ) = stu.getStuDetails(\_ID);

        //Show error if the student is not found

        require(\_ID == stuID, "\_\_\_\_The student ID entered is not found\_\_\_\_");

        // Get the provider address

        provAdd = payable(msg.sender);

        //Set the default scholarship status as true

        Status = "active";

        //Add to array

        scholarship.push(

            Scholarship({

                ID: \_ID,

                ScholarshipName: \_ScholarshipName,

                Amount: cvrAmt,

                Provider: provAdd,

                Attendance: \_Attendance,

                AvgMark: \_AvgMark,

                Status: Status

            })

        );

        // Fetch the sscholarship details

        // using the student ID

        schlRecords[\_ID] = Scholarship(

            \_ID,

            \_ScholarshipName,

            cvrAmt,

            provAdd,

            \_Attendance,

            \_AvgMark,

            Status

        );

        //send the scholarship amount to staff contract for payment disbursement

        StaffContractAdd.transfer(cvrAmt);

    }

    //function to cancel scholarship

    function cancelScholarship(uint256 \_ID) public {

        Scholarship storage \_scholarship = schlRecords[\_ID];

        //Show error if the student is not found

        require(

            \_ID == \_scholarship.ID,

            "\_\_\_\_The student ID entered is either not found or no scholarship\_\_\_\_"

        );

        //only same address can cancel the scholarship

        require(

            \_scholarship.Provider == msg.sender,

            "\_\_\_\_Only scholarship owner can cancel the scholarship\_\_\_\_"

        );

        //only can cancel active scholarship

        require(

            keccak256(abi.encodePacked(\_scholarship.Status)) ==

                keccak256(abi.encodePacked(Status)),

            "\_\_\_\_The scholarship is already cancelled, no further cancelation needed\_\_\_\_"

        );

        \_scholarship.Status = "pending\_refund";

    }

    //function to view the scholarship array

    function getSchlDetails(

        uint256 \_ID

    )

        public

        view

        returns (

            uint256,

            uint256,

            address payable,

            uint256,

            uint256,

            string memory

        )

    {

        uint256 ID = schlRecords[\_ID].ID;

        uint256 Amount = schlRecords[\_ID].Amount;

        address payable Provider = payable(schlRecords[\_ID].Provider);

        uint256 Attendance = schlRecords[\_ID].Attendance;

        uint256 AvgMark = schlRecords[\_ID].AvgMark;

        string memory Stat = schlRecords[\_ID].Status;

        return (ID, Amount, Provider, Attendance, AvgMark, Stat);

    }

    //function to get the scholarship status

    function getStatus(uint256 \_ID) public view returns (string memory) {

        string memory Stat = schlRecords[\_ID].Status;

        return Stat;

    }

    //function to update the scholarship status to cancel

    function updStatCancel(uint256 \_ID) public {

        Scholarship storage \_scholarship = schlRecords[\_ID];

        \_scholarship.Status = "cancel";

    }

    //function to update the scholarship status to paid

    function updStatPaid(uint256 \_ID) public {

        Scholarship storage \_scholarship = schlRecords[\_ID];

        \_scholarship.Status = "paid";

    }

    //function to update the scholarship status to failed

    function updStatFailed(uint256 \_ID) public {

        Scholarship storage \_scholarship = schlRecords[\_ID];

        \_scholarship.Status = "failed";

    }

    //function to update the scholarship status to active

    function updStatActive(uint256 \_ID) public {

        Scholarship storage \_scholarship = schlRecords[\_ID];

        \_scholarship.Status = "active";

    }

}

Staff Distribution Contract

// SPDX-License-Identifier: MIT

pragma solidity >=0.8.2 <0.9.0;

import "./1\_Storage.sol";

import "contracts/4\_SchloarshipProvider.sol";

contract Staff{

    //get the other 2 contract addresses

    address StuContractAdd;

    address payable SchlContractAdd;

    constructor (address \_StuContractAdd, address \_SchlContractAdd) payable {

        StuContractAdd = \_StuContractAdd;

        SchlContractAdd = payable(\_SchlContractAdd);

    }

    //This is mandatory for receiving ETH

    event ReceivedEth(uint256 amount);

    receive() external payable  {

        emit ReceivedEth(msg.value);

    }

    fallback() external payable {

        emit ReceivedEth(msg.value);

    }

    // defining result details struct

    struct Result {

        uint256 ID;

        uint256 Attendance;

        uint256 AvgMark;

    }

    //define variables

    uint256 id;

    address payable receiver;

    address payable provider;

    uint256 payAmt;

    string status;

    mapping(uint256 => Result) internal rsltRecords;

    Result[] internal result;

    // Create a function to staff to add

    // the student result

    function resultNpay(uint256 \_ID,

                        uint256 \_Attendance,

                        uint256 \_AvgMark) payable public

    {

        //Add to array

        result.push(Result({ID: \_ID, Attendance: \_Attendance,

                                    AvgMark: \_AvgMark}));

        // Fetch the result details

        // using the student ID

        rsltRecords[\_ID] = Result(\_ID, \_Attendance,

                                    \_AvgMark);

        //variables to capture info from Scholarship contract

        uint256 reqAtt;

        uint256 reqMark;

        //Get required info from Student contract

        StuDetails stu = StuDetails(StuContractAdd);

        (, , , receiver) = stu.getStuDetails(\_ID);

        //Get required info from Scholarship contract

        ScholarDetails sc = ScholarDetails(SchlContractAdd);

        (id, payAmt, , reqAtt, reqMark, status) = sc.getSchlDetails(\_ID);

        //error message if no student ID found either in student or scholarhsip contract

        require(\_ID == id, "\_\_\_\_The student ID entered is either not found or no scholarship\_\_\_\_");

        //scholarship need to be active for the payment disbursement

        require(keccak256(abi.encodePacked(status)) == keccak256(abi.encodePacked('active')), "\_\_\_\_The scholarship is not active\_\_\_\_");

        //check is the attendance and mark meet the requirements from scholarhsip provider

        //if failed

        if (reqAtt > \_Attendance || reqMark > \_AvgMark) {

            //Update the status to failed

            sc.updStatFailed(\_ID);

        } else {

            //if pass

            require(payAmt <= address(this).balance,"\_\_\_\_This wallet does not have enough balance to pay to student\_\_\_\_");

            receiver.transfer(payAmt);

            //Update the status to paid

            sc.updStatPaid(\_ID);

        }

    }

    //function to return the ETH back to provider if provider cancel the scholarship

    function processRefund(uint256 \_ID) payable public {

        //Get required info from Scholarship contract

        ScholarDetails sc = ScholarDetails(SchlContractAdd);

        (id, payAmt, provider, , ,status) = sc.getSchlDetails(\_ID);

        //Make sure the ID has scholarship record

        require(\_ID == id, "\_\_\_\_The student ID entered is either not found or no scholarship\_\_\_\_");

        //Make sure the scholarship is pending refund

        require(keccak256(abi.encodePacked(status)) == keccak256(abi.encodePacked('pending\_refund')), "\_\_\_\_No refund pending\_\_\_\_");

        //Make sure there is enough money for refund

        require(payAmt <= address(this).balance,"\_\_\_\_This wallet does not have enough balance to perform refund\_\_\_\_");

        //perform refund back to provider address

        provider.transfer(payAmt);

        //Update the status to cancel

        sc.updStatCancel(\_ID);

    }

    //function to activate back failed scholarship

    function processActivation(uint256 \_ID) public {

        //Get required info from Scholarship contract

        ScholarDetails sc = ScholarDetails(SchlContractAdd);

        (id, payAmt, , , ,status) = sc.getSchlDetails(\_ID);

        //Make sure the ID has scholarship record

        require(\_ID == id, "\_\_\_\_The student ID entered is either not found or no scholarship\_\_\_\_");

        //Make sure the scholarship is failed

        require(keccak256(abi.encodePacked(status)) == keccak256(abi.encodePacked('failed')), "\_\_\_\_The scholarship is not in failed status, no activation required\_\_\_\_");

        //Update the status to active

        sc.updStatActive(\_ID);

    }

    //function to view the scholarship status in order to return correct messaging on frontend

    function viewStatus(uint256 \_ID) public view returns(string memory) {

        //Get status info from Scholarship contract

        ScholarDetails sc = ScholarDetails(SchlContractAdd);

        string memory Stat = sc.getStatus(\_ID);

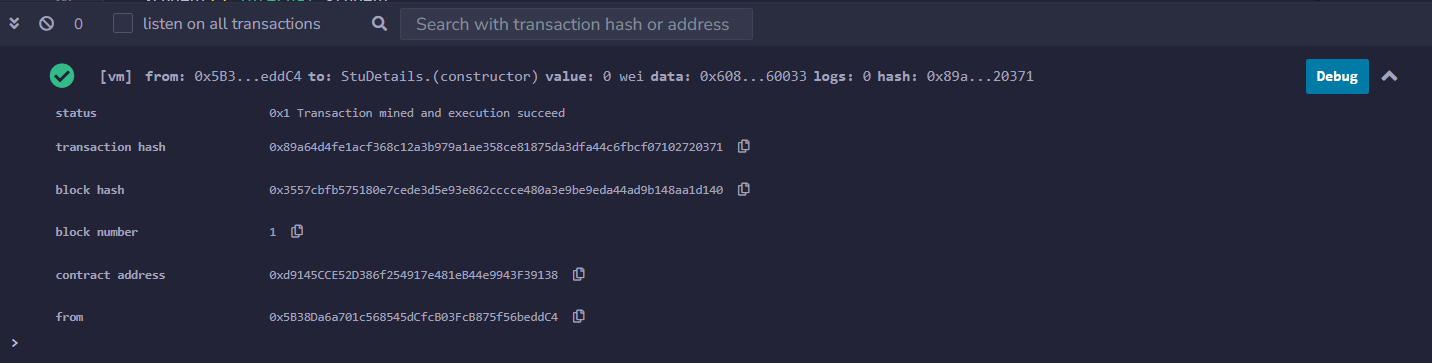
        return Stat;

    }

}

Deploying Smart Contracts

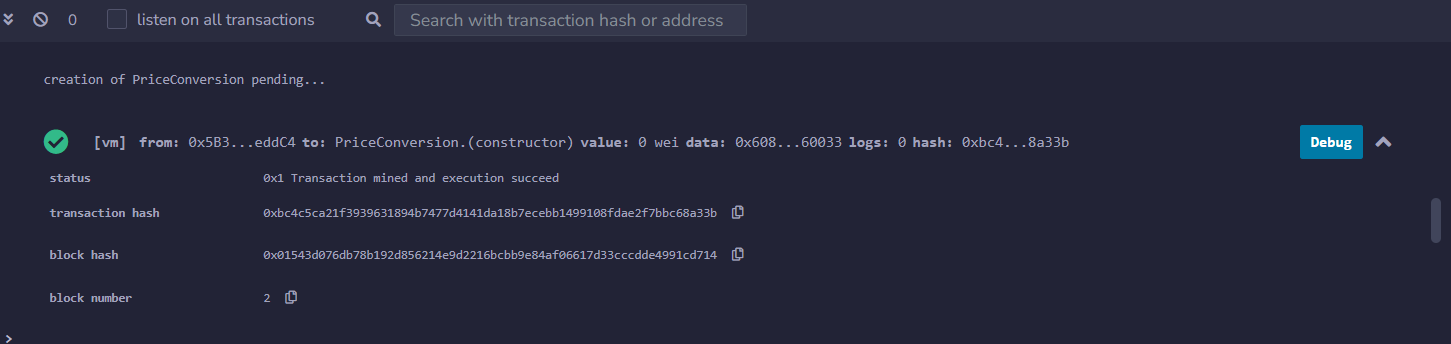
Deploying Student Details Contract



Account :- 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

Gas Fees :- 688155

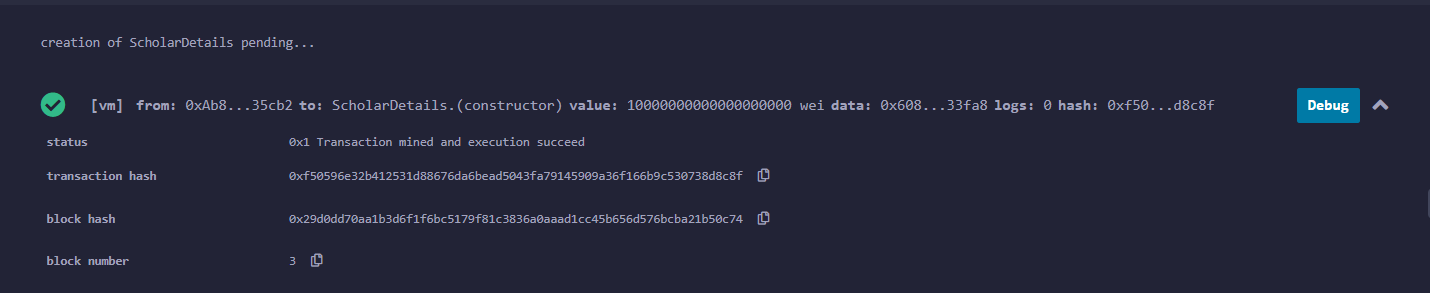
Deploying Price Conversion SC: -

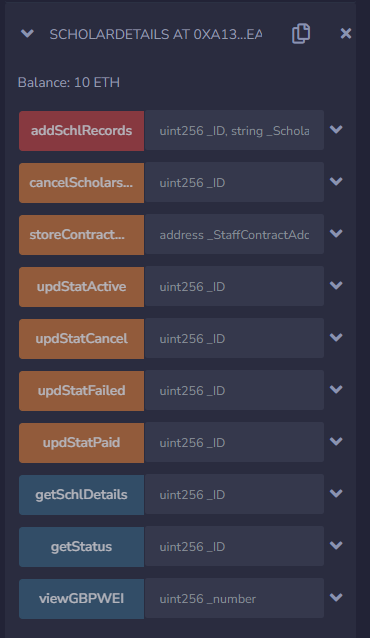


Account :- 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

Gas Fees :- 345463

Deploying Schlorship SC :-



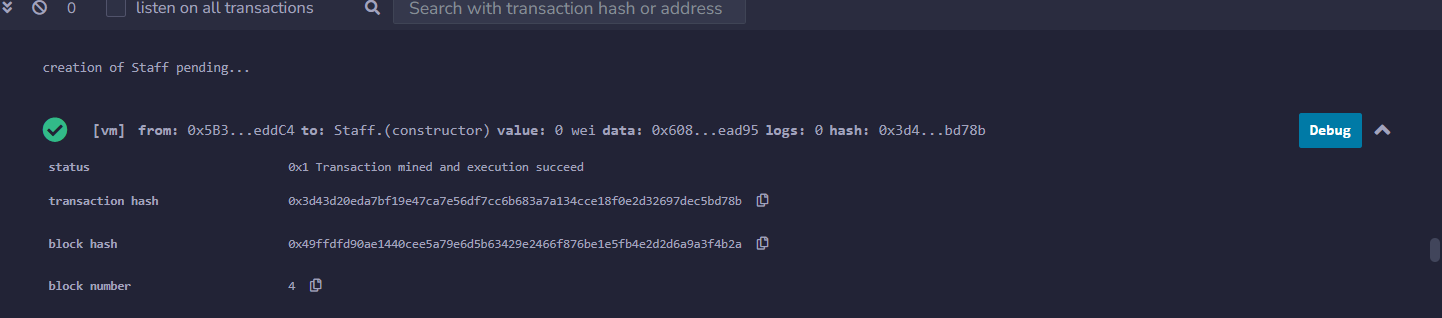


The Contract has an initial balance of 10ETH

Account :- 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2

Gas Fees + Store ETH :- 1561223 + 10ETH

Deploying Staff Contract :-



Account :- 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

Gas Fees:- 1481596

Testing Functions

Major Functions :-

Contract :- Student Details

Function to Test :- addStuDetails

What it should achieve:- Push the details of the student including Id, FN , LN to the array and add/update the array record on the chain. PS :- The Student Wallet Address is also stored.

Contract :- SchlorDetails

Function to Test :- storeContractAddress

Aim:- As the Staff contract is deployed after SchlorDetails contract this should be provided after staff SC deployment. It should be able to store the contract.

Contract :- SchlorDetails

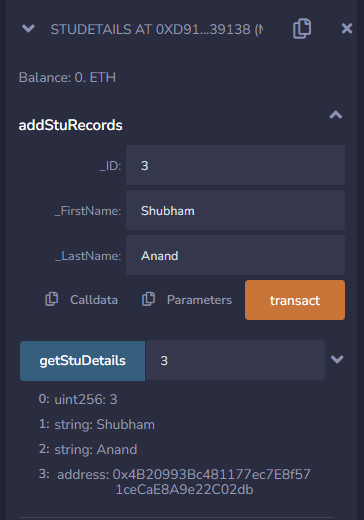
Function to Test :- addSchlDetails:- Add the schlorship name, min requirement for the schlorship. It should also be able to figure out the schlorship receipt from the student contract. Then it should be able to sent the the schlorship to Staff SC to further distribution for futher checks before finally sending it to Student Wallet.

Contract :- Staff

Function to Test :- resultNpay :- Check If the Student passes the requirement for the schlorship. Then it should transfer the schlorship to student contract.

Student Contract Test :-

Function to Test :- addStuDetails



Status :- Successful

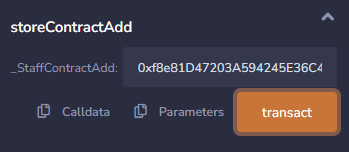
Student Wallet Address :- 0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db

Balance After :-



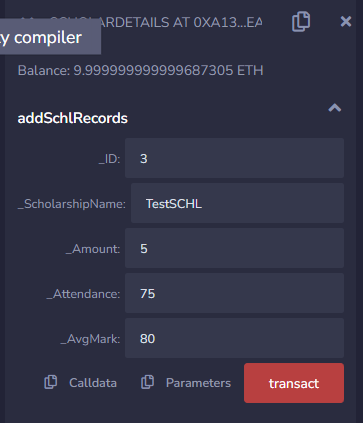
Note that this is the Wallet for Student.

Function to Test :- storeContractAddress



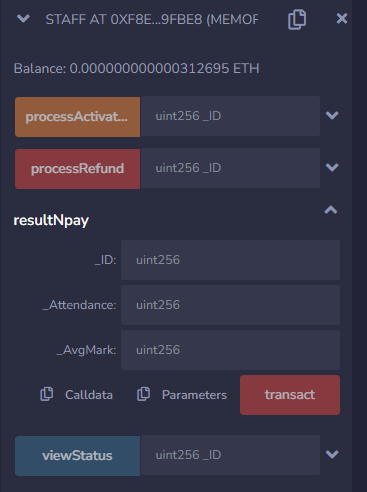
Successfully Stored Staff SC Address

Contract :- SchlorDetails

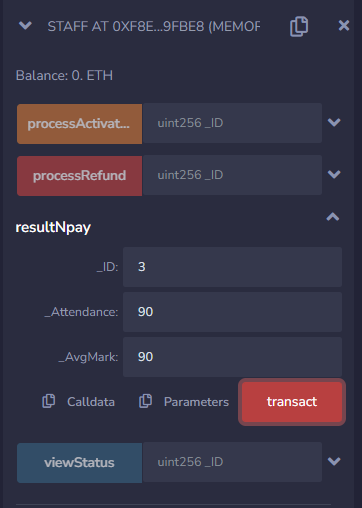


Sent ETH to the staff SC

Function to Test :- resultNpay



Initially the ETH from SCHL SC get stored in this smart contract



After Successfully verifying the student details and successful transaction the ETH is dropped to zero



And finally the the Student Wallet has an updated value of the ETH value