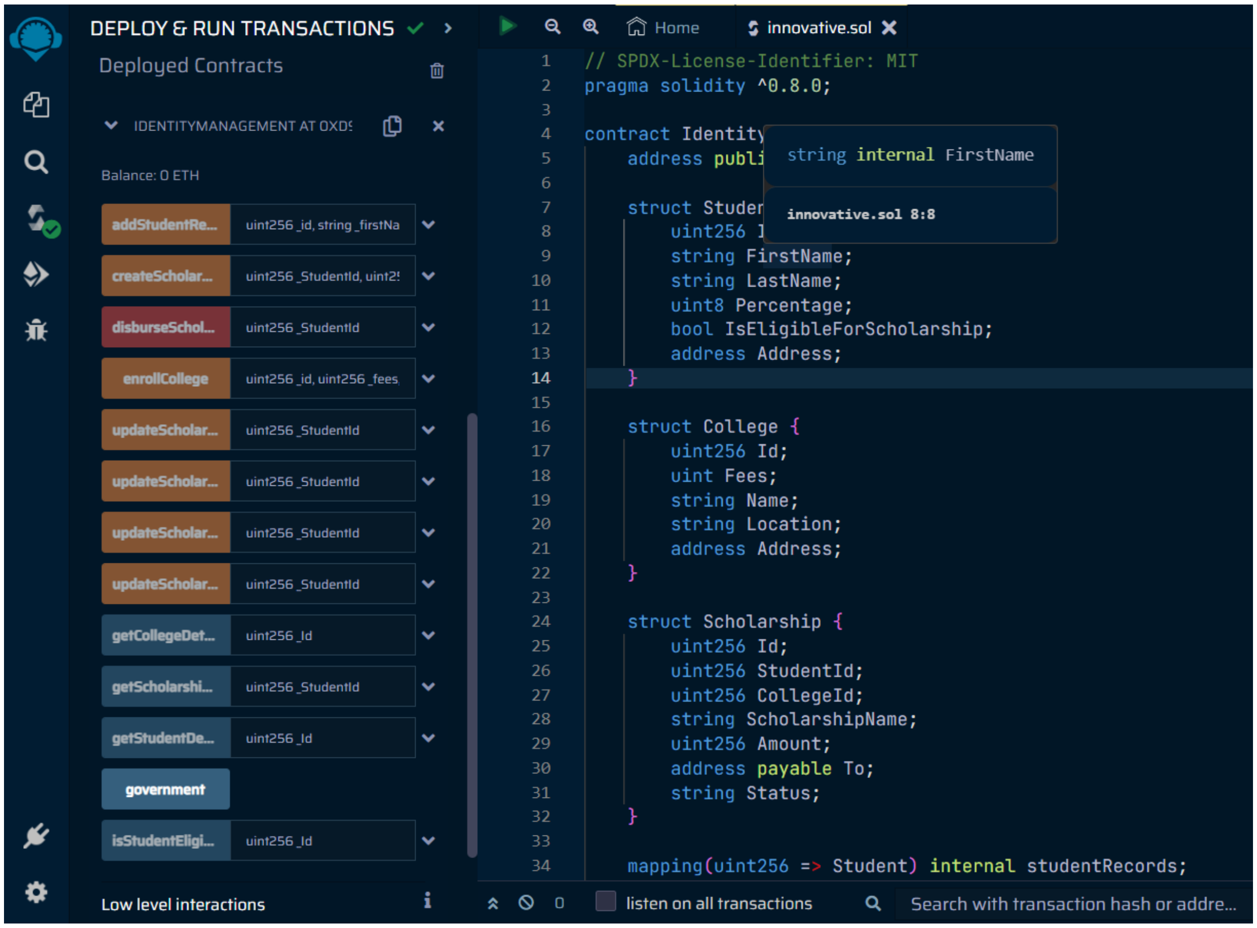
Name: Raj K Patel
Roll No: 20BCE218
Course: 2CSDE93 - Blockchain Technology
Practical No: 5
Aim: To perform thorough study and installation of Remix IDE and Truffle IDE for deploying Smart Contracts and Decentralized Applications (dapps) and create and deploy a Smart Contract for any application such as finance, healthcare etc.

Code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract IdentityManagement {
    address public government;

    struct Student {
        uint256 Id;
        string FirstName;
        string LastName;
        uint8 Percentage;
        bool IsEligibleForScholarship;
        address Address;
    }

    struct College {
        uint256 Id;
        uint Fees;
        string Name;
        string Location;
        address Address;
    }

    struct Scholarship {
        uint256 Id;
        uint256 StudentId;
        uint256 CollegeId;
        string ScholarshipName;
        uint256 Amount;
        address payable To;
        string Status;
    }
```

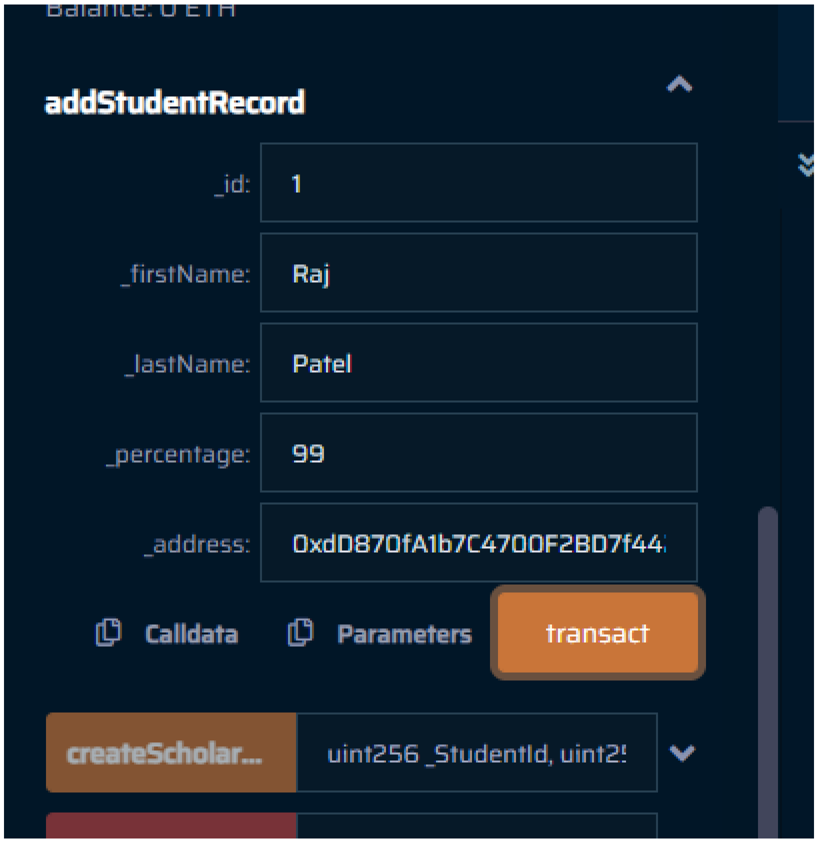
```
mapping(uint256 => Student) internal studentRecords;
Student[] internal students;

mapping(uint256 => College) internal collegeRecords;
College[] internal colleges;

mapping(uint256 => Scholarship) internal scholarshipRecords;
Scholarship[] internal scholarships;

struct StudentFeesPaid {
    mapping(uint256 => uint) studentFeesPaid;
}
mapping(uint => StudentFeesPaid) college_student_fees_paid;

modifier onlyGovernment() {
    require(
        msg.sender == government,
        "Only government can perform this action"
    );
    _;
}

constructor() {
    government = msg.sender;
}

function enrollCollege(
    uint256 _id,
    uint256 _fees,
    string memory _name,
    string memory _location,
    address payable _address
) public onlyGovernment {
    require(_address != address(0), "Invalid college address");
    require(
        collegeRecords[_id].Id == 0,
        "College with the given ID already exists"
    );
    require(bytes(_name).length > 0, "College name cannot be empty");
    require(
        bytes(_location).length > 0,
        "College location cannot be empty"
        );

        College memory c = College({
            Id: _id,
            Fees: _fees,
            Name: _name,
            Location: _location,
            Address: _address
        });
        collegeRecords[c.Id] = c;
        colleges.push(c);
    }

    function getCollegeDetails(
        uint256 _Id
    ) public view returns (College memory) {
        require(
            collegeRecords[_Id].Id != 0,
            "No college with the given ID found"
        );
        return collegeRecords[_Id];
    }

    function addStudentRecord(
        uint256 _id,
        string memory _firstName,
        string memory _lastName,
        uint8 _percentage,
        address _address
    ) public onlyGovernment {
        require(
            studentRecords[_id].Id == 0,
            "Student with the given ID already exists"
        );
        Student memory student = Student({
            Id: _id,
            FirstName: _firstName,
            LastName: _lastName,
            Percentage: _percentage,
            IsEligibleForScholarship: false,
            Address: _address
        });
        students.push(student);
        studentRecords[student.Id] = student;
    }

    function getStudentDetails(
        uint256 _Id
    ) public view returns (Student memory) {
        require(
            studentRecords[_Id].Id != 0,
            "No student with the given ID found"
        );
        return studentRecords[_Id];
    }

    function isStudentEligibleForScholarship(
        uint256 _Id
    ) public view returns (bool) {
        require(
            studentRecords[_Id].Id != 0,
            "No student with the given ID found"
        );
        Student memory s = getStudentDetails(_Id);
        if (s.Percentage >= 80) {
            return true;
        } else {
            return false;
        }
    }

    //

    function createScholarship(
        uint256 _StudentId,
        uint256 _CollegeId
    ) public onlyGovernment {
        require(
            studentRecords[_StudentId].Id != 0,
            "No student with the given ID found"
        );

        require(
            collegeRecords[_CollegeId].Id != 0,
            "No college with the given ID found"
        );
        require(
            scholarshipRecords[_StudentId].StudentId == 0,
            "Scholarship with the given ID already exists"
        );
        require(
            isStudentEligibleForScholarship(_StudentId),
            "Student is not eligible for a scholarship"
        );

        College memory c = collegeRecords[_CollegeId];
        uint min_amount = 2000000000000000000;
        uint scholarship_amount = 0;

        if (c.Fees < min_amount) {
            scholarship_amount = c.Fees;
        } else {
            scholarship_amount = min_amount;
        }

        Scholarship memory new_scholarship = Scholarship({
            Id: scholarships.length + 1,
            StudentId: _StudentId,
            CollegeId: _CollegeId,
            ScholarshipName: "Merit Scholarship",
            Amount: scholarship_amount,
            To: payable(c.Address),
            Status: "Pending"
        });

        scholarshipRecords[_StudentId] = new_scholarship;
        scholarships.push(new_scholarship);
    }

    function disburseScholarship(
        uint256 _StudentId
    ) public payable onlyGovernment {
        Scholarship
            storage scholarship_of_registered_stdduent = scholarshipRecords[
                _StudentId
            ];
        require(
            scholarship_of_registered_stdduent.StudentId != 0,
            "Scholarship for the given student not found"
        );
        require(
            keccak256(
                abi.encodePacked(scholarship_of_registered_stduent.Status)
            ) == keccak256(abi.encodePacked("Pending")),
            "Scholarship is not pending"
        );

        scholarship_of_registered_stduent.Status = "Awarded";
        scholarship_of_registered_stduent.To.transfer(
            scholarship_of_registered_stduent.Amount
        );

        StudentFeesPaid storage getCollege = college_student_fees_paid[
            scholarship_of_registered_stduent.CollegeId
        ];

        getCollege.studentFeesPaid[
            scholarship_of_registered_stduent.StudentId
        ] = scholarship_of_registered_stduent.Amount;
    }

    function getScholarshipStatus(
        uint256 _StudentId
    ) public view returns (string memory) {
        require(
            scholarshipRecords[_StudentId].Id != 0,
            "Scholarship with the given ID does not exists"
        );

        Scholarship memory scholarship = scholarshipRecords[_StudentId];
        return scholarship.Status;
    }

    function updateScholarshipStatusToCancel(
        uint256 _StudentId
    ) public onlyGovernment {
        require(
            scholarshipRecords[_StudentId].Id != 0,
            "Scholarship with the given ID does not exists"
        );
        Scholarship storage _scholarship = scholarshipRecords[_StudentId];
        _scholarship.Status = "Cancel";
    }

    function updateScholarshipStatusToPaid(
        uint256 _StudentId
    ) public onlyGovernment {
        require(
            scholarshipRecords[_StudentId].Id != 0,
            "Scholarship with the given ID does not exists"
        );
        Scholarship storage _scholarship = scholarshipRecords[_StudentId];
        _scholarship.Status = "Paid";
    }

    function updateScholarshipStatusToFailed(
        uint256 _StudentId
    ) public onlyGovernment {
        require(
            scholarshipRecords[_StudentId].Id != 0,
            "Scholarship with the given ID does not exists"
        );
        Scholarship storage _scholarship = scholarshipRecords[_StudentId];
        _scholarship.Status = "Failed";
    }

    function updateScholarshipStatusToActive(
        uint256 _StudentId
    ) public onlyGovernment {
        require(
            scholarshipRecords[_StudentId].Id != 0,
            "Scholarship with the given ID does not exists"
        );
        Scholarship storage _scholarship = scholarshipRecords[_StudentId];
        _scholarship.Status = "Active";
    }
}
```

Output:


DEPLOY & RUN TRANSACTIONS
Deployed Contracts
IDENTITYMANAGEMENT AT OXD!
Balance: 0 ETH
addStudentRe... uint256 _id, string _firstNa
createScholar... uint256 _StudentId, uint2!
disburseSchol... uint256 _StudentId
enrollCollege uint256 _id, uint256 _fees,
updateScholar... uint256 _StudentId
updateScholar... uint256 _StudentId
updateScholar... uint256 _StudentId
updateScholar... uint256 _StudentId
getCollegeDet... uint256 _Id
getScholarshi... uint256 _StudentId
getStudentDe... uint256 _Id
government
isStudentEligi... uint256 _Id
Low level interactions
Home
innovative.sol
listen on all transactions
Search with transaction hash or addre...

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Identity
    address publi

    struct Studer
        uint256 I
        string FirstName;
        string LastName;
        uint8 Percentage;
        bool IsEligibleForScholarship;
        address Address;
    }

    struct College {
        uint256 Id;
        uint Fees;
        string Name;
        string Location;
        address Address;
    }

    struct Scholarship {
        uint256 Id;
        uint256 StudentId;
        uint256 CollegeId;
        string ScholarshipName;
        uint256 Amount;
        address payable To;
        string Status;
    }

    mapping(uint256 => Student) internal studentRecords;
```

```
string internal FirstName

innovative.sol 8:8
```



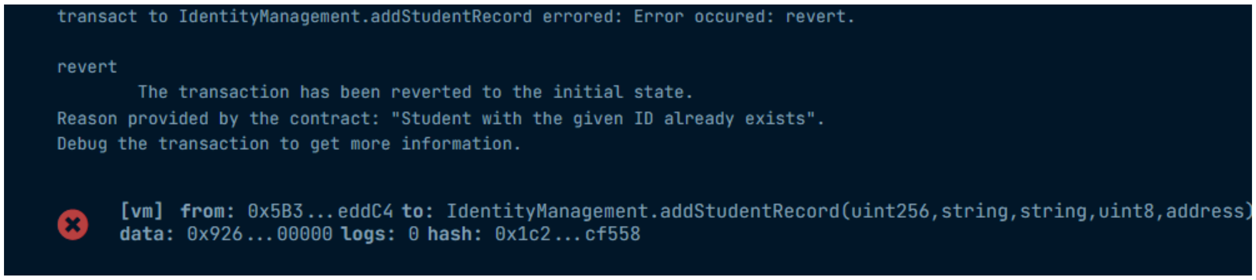
_id: 1

_firstName: Raj

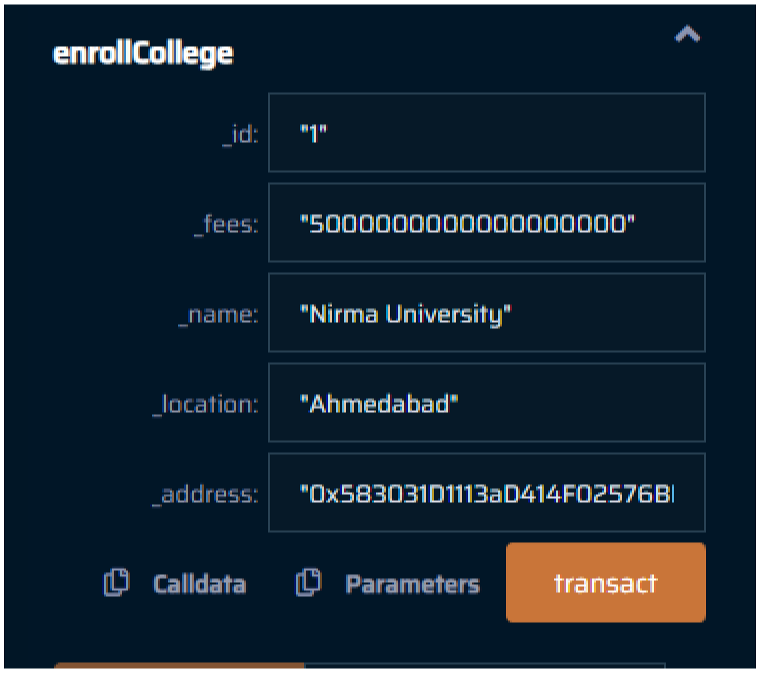
_lastName: Patel

_percentage: 99

_address: 0xdD870fA1b7C4700F2BD7f44



```
transact to IdentityManagement.addStudentRecord errored: Error occured: revert.

revert
        The transaction has been reverted to the initial state.
Reason provided by the contract: "Student with the given ID already exists".
Debug the transaction to get more information.

      [vm] from: 0x5B3...eddC4 to: IdentityManagement.addStudentRecord(uint256,string,string,uint8,address)
      data: 0x926...00000 logs: 0 hash: 0x1c2...cf558
```


enrollCollege

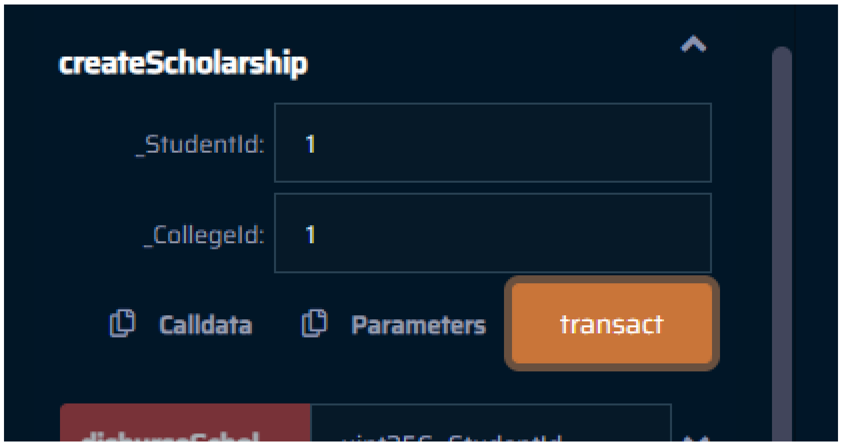
_id: "1"

_fees: "5000000000000000000"

_name: "Nirma University"

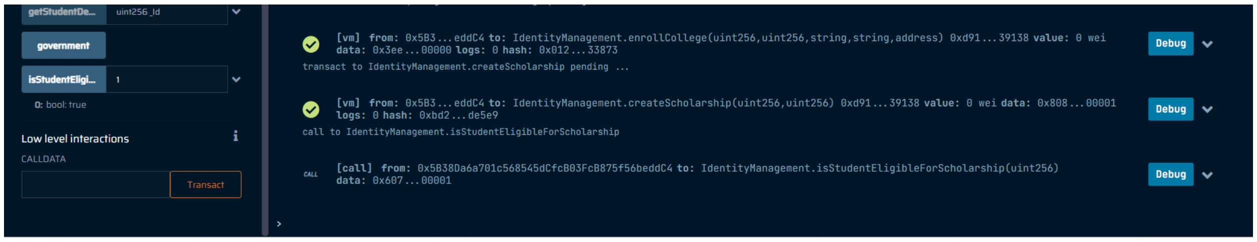
_location: "Ahmedabad"

_address: "0x583031D1113aD414F02576BI


createScholarship

_StudentId: 1

_CollegeId: 1


getStudentDe... uint256 _Id

government

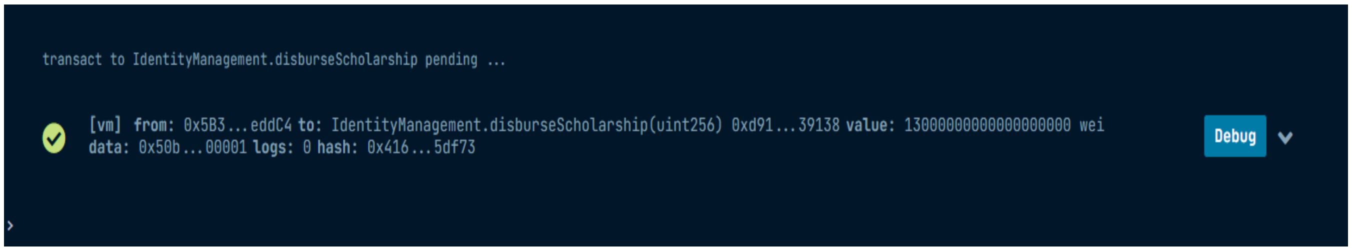
isStudentEligi... 1

0: bool: true

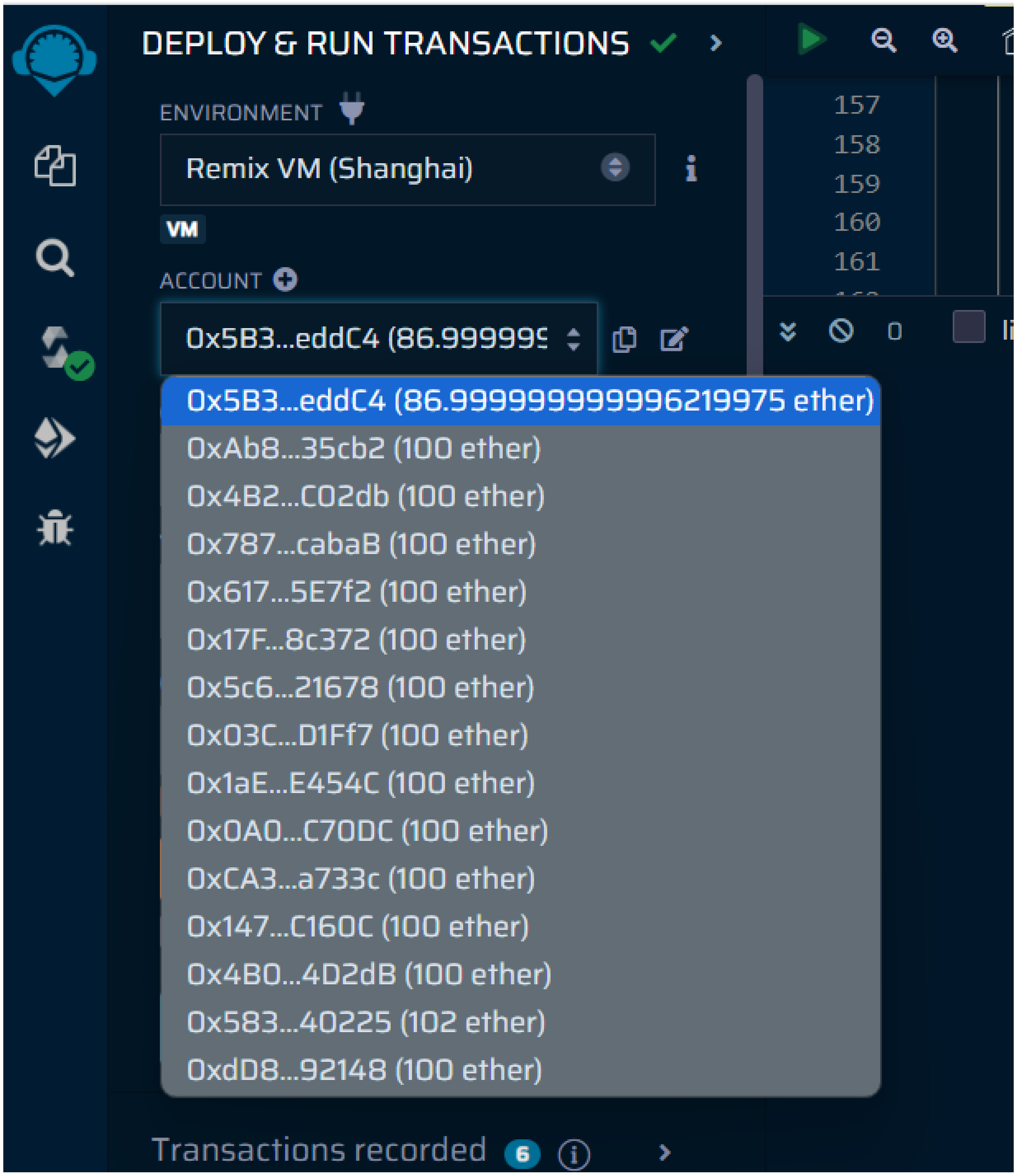
Low level interactions

CALLDATA

```
[vm] from: 0x5B3...eddC4 to: IdentityManagement.enrollCollege(uint256,uint256,string,string,address) 0xd91...39138 value: 0 wei
data: 0x3ee...00000 logs: 0 hash: 0x012...33873
transact to IdentityManagement.createScholarship pending ...

[vm] from: 0x5B3...eddC4 to: IdentityManagement.createScholarship(uint256,uint256) 0xd91...39138 value: 0 wei data: 0x808...00001
logs: 0 hash: 0xbd2...de5e9
call to IdentityManagement.isStudentEligibleForScholarship

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: IdentityManagement.isStudentEligibleForScholarship(uint256)
data: 0x607...00001
```



```
transact to IdentityManagement.disburseScholarship pending ...

[vm] from: 0x5B3...eddC4 to: IdentityManagement.disburseScholarship(uint256) 0xd91...39138 value: 13000000000000000000 wei
data: 0x50b...00001 logs: 0 hash: 0x416...5df73
```


DEPLOY & RUN TRANSACTIONS

ENVIRONMENT

Remix VM (Shanghai)

VM

ACCOUNT

0x5B3...eddC4 (86.99999

0x5B3...eddC4 (86.999999999996219975 ether)
0xAb8...35cb2 (100 ether)
0x4B2...C02db (100 ether)
0x787...cabaB (100 ether)
0x617...5E7f2 (100 ether)
0x17F...8c372 (100 ether)
0x5c6...21678 (100 ether)
0x03C...D1Ff7 (100 ether)
0x1aE...E454C (100 ether)
0x0A0...C70DC (100 ether)
0xCA3...a733c (100 ether)
0x147...C160C (100 ether)
0x4B0...4D2dB (100 ether)
0x583...40225 (102 ether)
0xdD8...92148 (100 ether)

Transactions recorded 6