// SPDX-License-Identifier: MIT

pragma solidity ^0.8.20; // Using a common, stable version

// A secure Student Registry contract

contract StudentRegistry {

// THE BLUEPRINT: We've added an 'owner' field to store the student's unique address.

struct Student {

string name;

uint256 rollno;

string class;

address owner; // The unique Ethereum address of the student

}

Student[] private students;

// --- OUR FUNCTIONS ---

// 1. ADD A STUDENT (Create)

// The student provides their info, and we automatically record their address.

function addStudent(string memory \_name, uint256 \_rollno, string memory \_class) public {

// We use 'msg.sender' to get the address of the person calling the function.

// This links the student record to their Ethereum account forever.

students.push(Student(\_name, \_rollno, \_class, msg.sender));

}

// 2. GET TOTAL STUDENTS (Read)

function getTotalStudents() public view returns (uint256) {

return students.length;

}

// 3. GET A STUDENT BY ID (Read)

// We now also return the owner's address.

function getStudentById(uint256 \_id) public view returns (string memory, uint256, string memory, address) {

require(\_id < students.length, "Student does not exist");

Student memory s = students[\_id]; // a bit cleaner way to write it

return (s.name, s.rollno, s.class, s.owner);

}

// 4. UPDATE A STUDENT'S INFO (Update) - NOW WITH SECURITY!

function updateStudent(uint256 \_id, string memory \_newName, string memory \_newClass) public {

require(\_id < students.length, "Student does not exist");

// \*\*\* THE SECURITY CHECK! \*\*\*

// We check if the person calling this function (msg.sender) is the same

// person who created the record (students[\_id].owner).

require(students[\_id].owner == msg.sender, "You are not the owner of this record!");

students[\_id].name = \_newName;

students[\_id].class = \_newClass;

}

// 5. DELETE A STUDENT (Delete) - NOW WITH SECURITY!

function deleteStudent(uint256 \_id) public {

require(\_id < students.length, "Student does not exist");

// \*\*\* THE SECURITY CHECK! \*\*\*

require(students[\_id].owner == msg.sender, "You are not the owner of this record!");

// Efficient deletion by moving the last element to the deleted spot

students[\_id] = students[students.length - 1];

students.pop();

}

// 6. A simple fallback function to receive Ether

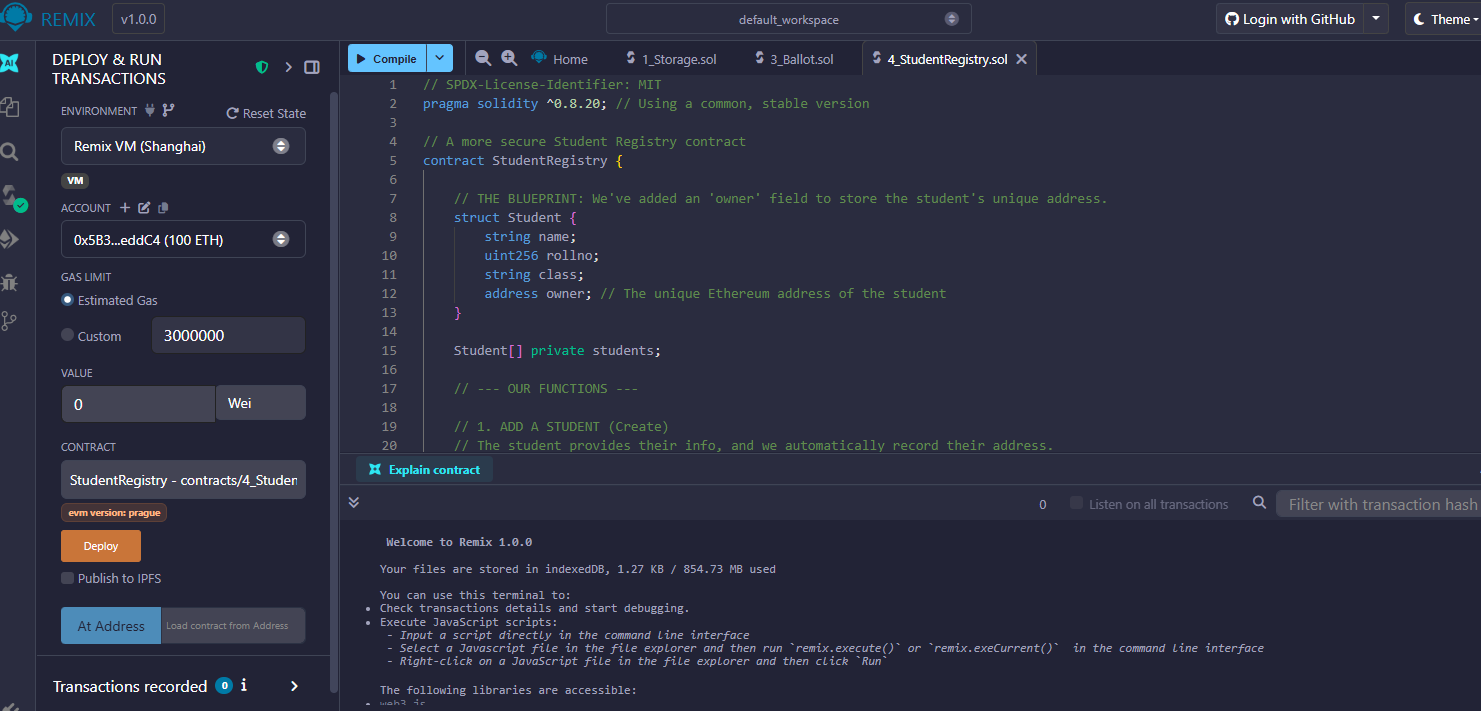
// This is required by your practical's instructions.

fallback() external payable {

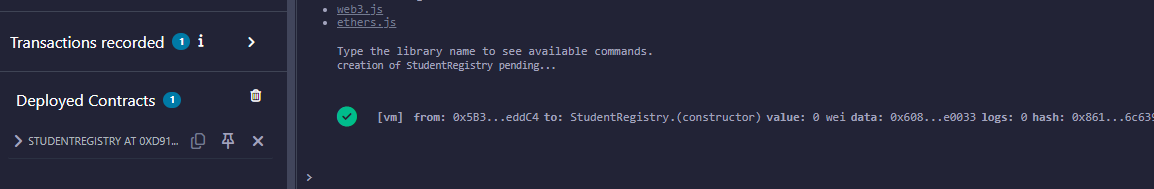
// This function can be left empty. It just allows the contract to receive Ether.

}

}

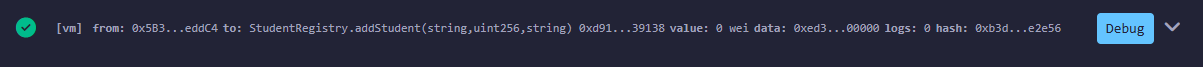


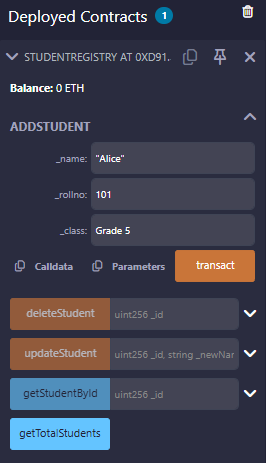
Deploying the contract



Testing Transaction

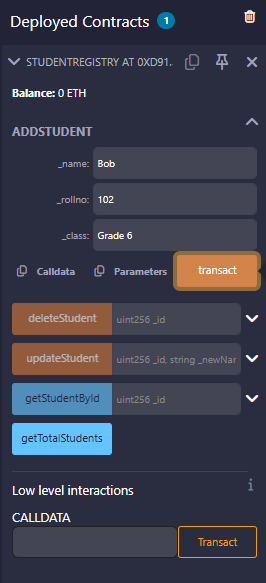
Transaction 1: Alice Registers



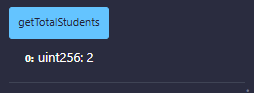


Transaction 2: Bob Registers



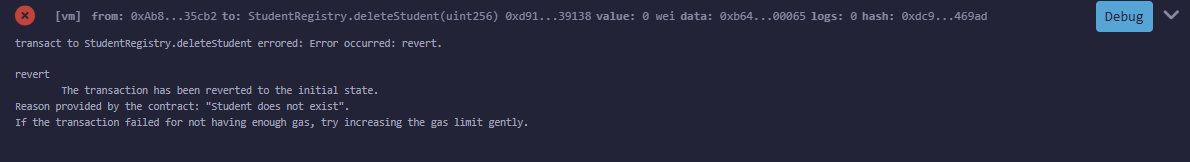


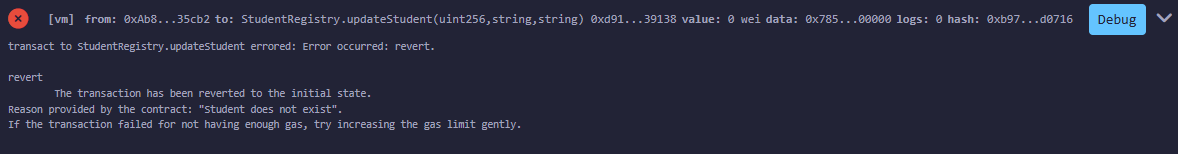
Transaction 3: Verify Data and Total Count

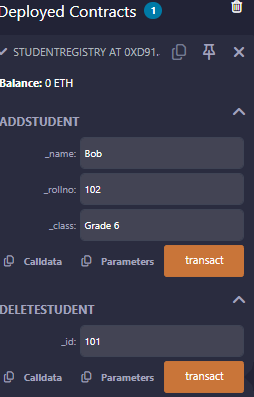


Step 5: Security Test - Unauthorized Update (FAIL)

Bob tries to delete or update Alice account details





Security Test - Authorized Update (SUCCESS)

Bob Updates and deletes his own account

