NAME : PAREENITA A.SHIRSATH PRN : 221101062 ROLL.NO : 57 B.E.A.I.&.D.S.

**BLOCK CHAIN EXPERIMENT NO : 2**

CODE :

1. **STUDENT**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract StudentRegistry {

    struct Student {

        string name;

        uint age;

        string course;

        bool exists;

    }

    mapping(address => Student) private students;

    // Event for student registration

    event StudentRegistered(address indexed studentAddress, string name, uint age, string course);

    // Event for course update

    event CourseUpdated(address indexed studentAddress, string newCourse);

    // Register new student

    function registerStudent(string memory \_name, uint \_age, string memory \_course) public {

        require(!students[msg.sender].exists, "Already registered");

        require(bytes(\_name).length > 0, "Name cannot be empty");

        require(bytes(\_course).length > 0, "Course cannot be empty");

        require(\_age > 0 && \_age < 120, "Invalid age");

        students[msg.sender] = Student(\_name, \_age, \_course, true);

        emit StudentRegistered(msg.sender, \_name, \_age, \_course);

    }

    // Update course

    function updateCourse(string memory \_course) public {

        require(students[msg.sender].exists, "Not registered");

        require(bytes(\_course).length > 0, "Course cannot be empty");

        students[msg.sender].course = \_course;

        emit CourseUpdated(msg.sender, \_course);

    }

    // Get student info

    function getStudent() public view returns (string memory, uint, string memory) {

        require(students[msg.sender].exists, "Not registered");

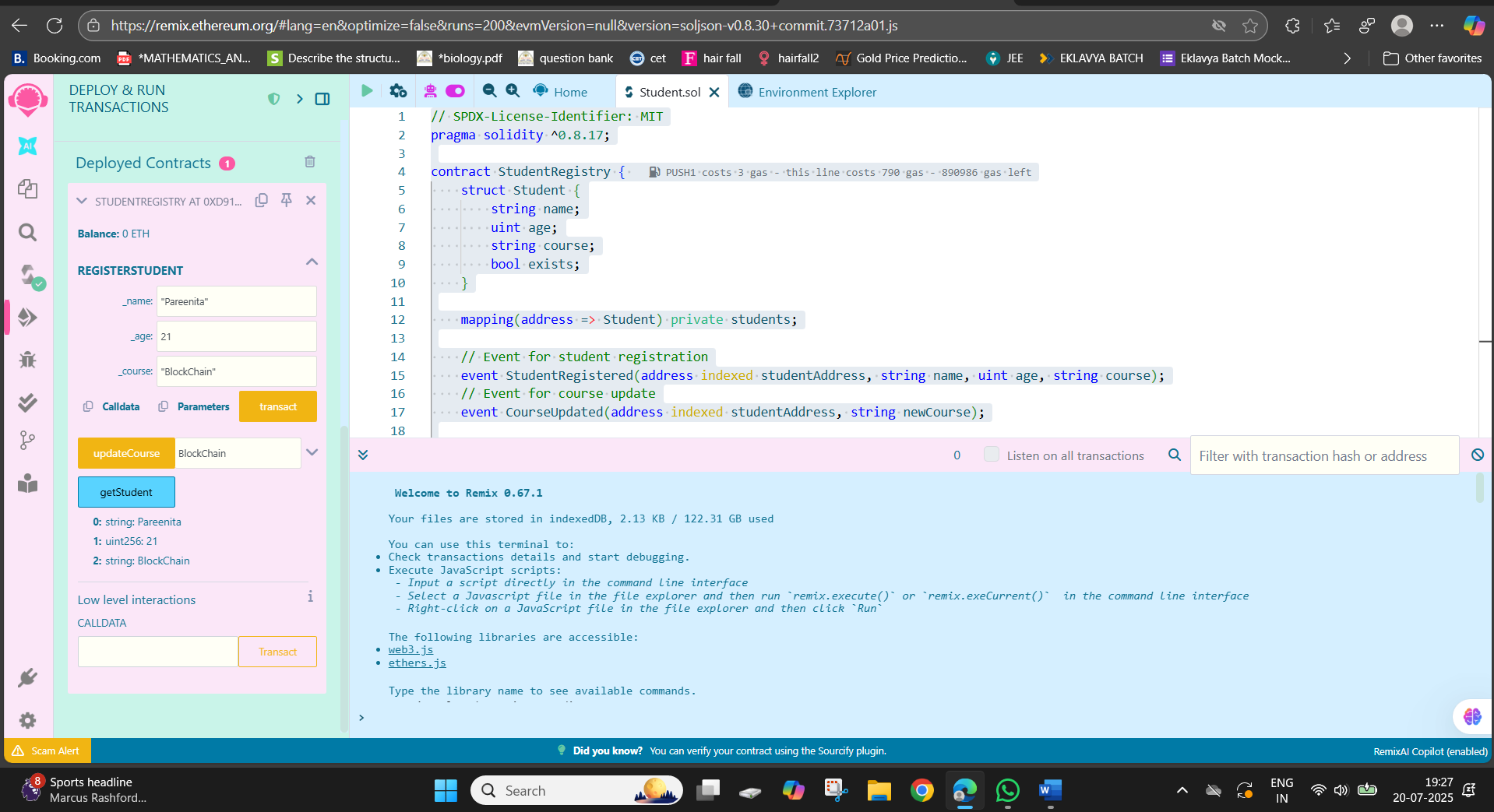
        Student memory s = students[msg.sender];

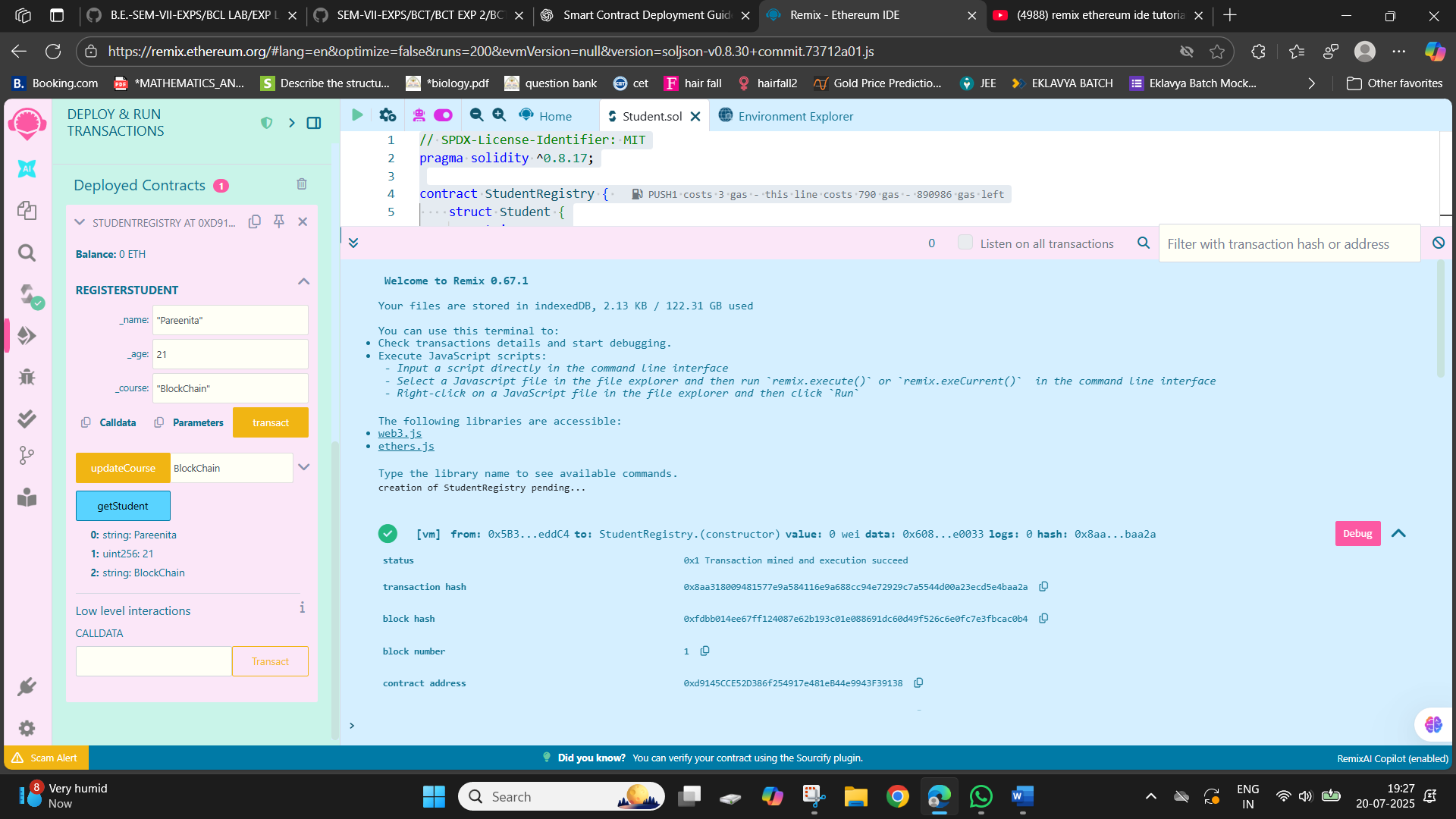
        return (s.name, s.age, s.course);

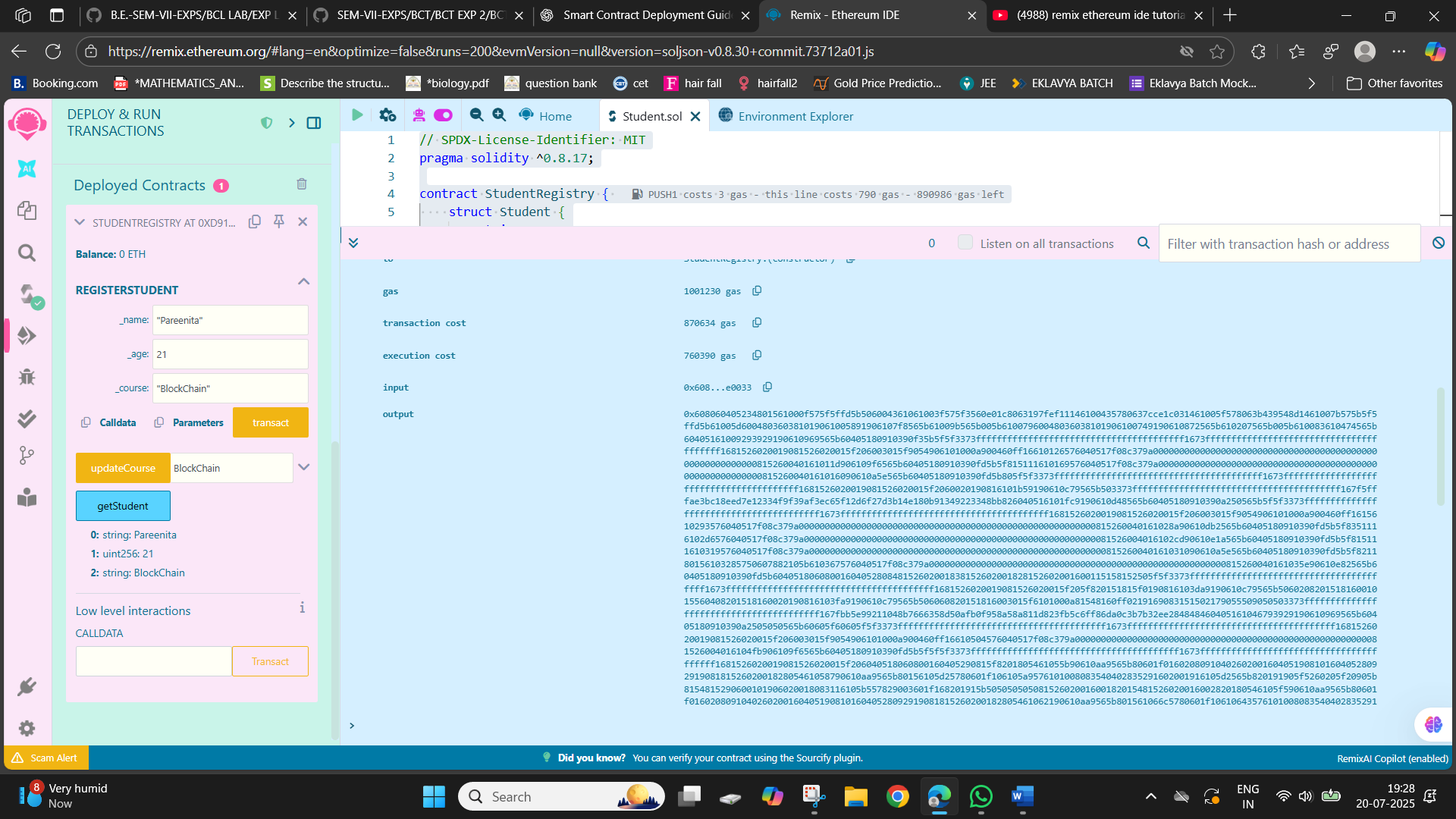
    }

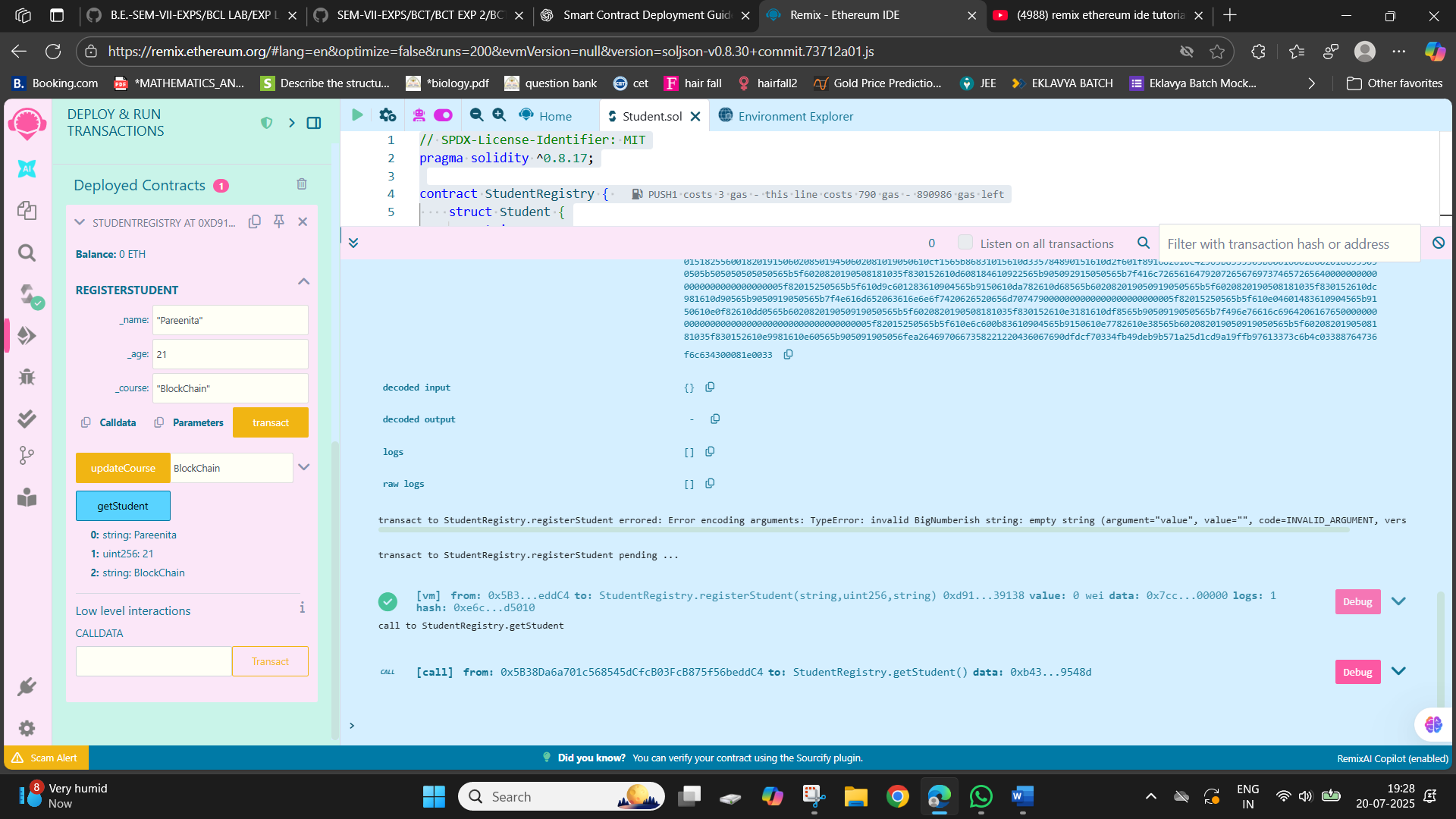
}

OUTPUT :









1. **BANK**

CODE :

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract Bank {

    mapping(address => uint) private balances;

    mapping(address => bool) private accountExists;

    // Events

    event AccountCreated(address indexed user);

    event DepositMade(address indexed user, uint amount);

    event WithdrawalMade(address indexed user, uint amount);

    // Create account

    function createAccount() public {

        require(!accountExists[msg.sender], "Account already exists");

        accountExists[msg.sender] = true;

        balances[msg.sender] = 0;

        emit AccountCreated(msg.sender);

    }

    // Deposit ether

    function deposit() public payable {

        require(accountExists[msg.sender], "Account does not exist");

        require(msg.value > 0, "Must send some ether");

        balances[msg.sender] += msg.value;

        emit DepositMade(msg.sender, msg.value);

    }

    // Withdraw ether

    function withdraw(uint \_amount) public {

        require(accountExists[msg.sender], "Account does not exist");

        require(\_amount <= balances[msg.sender], "Insufficient balance");

        balances[msg.sender] -= \_amount;

        payable(msg.sender).transfer(\_amount);

        emit WithdrawalMade(msg.sender, \_amount);

    }

    // Get balance

    function getBalance() public view returns (uint) {

        require(accountExists[msg.sender], "Account does not exist");

        return balances[msg.sender];

    }

}

OUTPUT :

