. 3 Write a smart contract on a test network, for Bank account of a customer for following operations:

· Deposit money

· Withdraw Money

· Show balance

// SPDX-License-Identifier: MIT

pragma solidity  >=0.4.22 <0.7.0;

contract banking{

  mapping(address=>uint) public userAccount;

  mapping(address=>bool) public userExists;

  function createAcc() public payable returns(string memory){

      require(userExists[msg.sender]==false,'Account Already Created');

      if(msg.value==0){

          userAccount[msg.sender]=0;

          userExists[msg.sender]=true;

          return 'account created';

      }

      require(userExists[msg.sender]==false,'account already created');

      userAccount[msg.sender] = msg.value;

      userExists[msg.sender] = true;

      return 'account created';

  }

  function deposit() public payable returns(string memory){

      require(userExists[msg.sender]==true, 'Account is not created');

      require(msg.value>0, 'Value for deposit is Zero');

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

      return 'Deposited Succesfully';

  }

  function withdraw(uint amount) public payable returns(string memory){

      require(userAccount[msg.sender]>amount, 'insufficeint balance in Bank account');

      require(userExists[msg.sender]==true, 'Account is not created');

      require(amount>0, 'Enter non-zero value for withdrawal');

      userAccount[msg.sender]=userAccount[msg.sender]-amount;

      msg.sender.transfer(amount);

      return 'withdrawal Succesful';

  }

  function TransferAmount(address payable userAddress, uint amount) public returns(string memory){

      require(userAccount[msg.sender]>amount, 'insufficeint balance in Bank account');

      require(userExists[msg.sender]==true, 'Account is not created');

      require(userExists[userAddress]==true, 'to Transfer account does not exists in bank accounts ');

      require(amount>0, 'Enter non-zero value for sending');

      userAccount[msg.sender]=userAccount[msg.sender]-amount;

      userAccount[userAddress]=userAccount[userAddress]+amount;

      return 'transfer succesfully';

  }

  function sendAmount(address payable toAddress , uint256 amount) public payable returns(string memory){

      require(amount>0, 'Enter non-zero value for withdrawal');

      require(userExists[msg.sender]==true, 'Account is not created');

      require(userAccount[msg.sender]>amount, 'insufficeint balance in Bank account');

      userAccount[msg.sender]=userAccount[msg.sender]-amount;

      toAddress.transfer(amount);

      return 'transfer success';

  }

  function userAccountBalance() public view returns(uint){

      return userAccount[msg.sender];

  }

  function accountExist() public view returns(bool){

      return userExists[msg.sender];

  }

}

Write a program in solidity to create Student data. Use the following constructs:

Structures

Arrays

Fallback

Deploy this as smart contract on Ethereum and Observe the transaction fee and Gas values.

// Solidity program to implement

// the above approach

pragma solidity >= 0.7.0<0.8.0;

// Build the Contract

contract MarksManagmtSys

{

    // Create a structure for

    // student details

    struct Student

    {

        int ID;

        string fName;

        string lName;

        int marks;

    }

    address owner;

    int public stdCount = 0;

    mapping(int => Student) public stdRecords;

    modifier onlyOwner

    {

        require(owner == msg.sender);

        \_;

    }

    constructor()

    {

        owner=msg.sender;

    }

    // Create a function to add

    // the new records

    function addNewRecords(int \_ID,

                        string memory \_fName,

                        string memory \_lName,

                        int \_marks) public onlyOwner

    {

        // Increase the count by 1

        stdCount = stdCount + 1;

        // Fetch the student details

        // with the help of stdCount

        stdRecords[stdCount] = Student(\_ID, \_fName,

                                    \_lName, \_marks);

    }

    // Create a function to add bonus marks

    function bonusMarks(int \_bonus) public onlyOwner

    {

        stdRecords[stdCount].marks =

                    stdRecords[stdCount].marks + \_bonus;

    }

}