// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract BlueHostingServerContract {

    address payable public companyAddress = payable(0x970a5303E58D896f670A1940c82b2B34D4FDdD8e);

    uint public storagePricePerGB = 20;

    uint public commissionLevel1 = 60;

    uint public commissionLevel2 = 20;

    struct Client {

        bool isRegistered;

        uint256 storageSize;

        uint256 commissionToPay;

    }

    struct Miner {

        bool isRegistered;

        uint256 storageCapacity;

        uint256 level;

    }

    mapping(address => Client) public clients;

    mapping(address => Miner) public miners;

    event ClientRegistered(address indexed client);

    event MinerRegistered(address indexed miner);

    event DataStored(address indexed client, address indexed miner, uint256 storageSize);

    modifier onlyClient() {

        require(clients[msg.sender].isRegistered, "You are not registered as a client");

        \_;

    }

    modifier onlyMiner() {

        require(miners[msg.sender].isRegistered, "You are not registered as a miner");

        \_;

    }

    function registerAsClient() external {

        require(!clients[msg.sender].isRegistered, "Already registered as a client");

        clients[msg.sender].isRegistered = true;

        emit ClientRegistered(msg.sender);

    }

    function registerAsMiner(uint256 \_storageCapacity, uint256 \_level) external {

        require(!miners[msg.sender].isRegistered, "Already registered as a miner");

        miners[msg.sender].isRegistered = true;

        miners[msg.sender].storageCapacity = \_storageCapacity;

        miners[msg.sender].level = \_level;

        emit MinerRegistered(msg.sender);

    }

    function uploadData(uint256 \_storageSize) external onlyClient {

        require(minersAvailableStorage() >= \_storageSize, "Not enough storage capacity available");

        uint256 commission = (\_storageSize \* storagePricePerGB) / 100;

        clients[msg.sender].storageSize = \_storageSize;

        clients[msg.sender].commissionToPay = commission;

        address currentMiner = findAvailableMiner(\_storageSize);

        storeData(msg.sender, currentMiner);

    }

    function minersAvailableStorage() public view returns (uint256 availableStorage) {

        availableStorage = 0;

        for (uint256 i = 0; i < minersCount(); i++) {

            address minerAddress = getMinerAtIndex(i);

            availableStorage += miners[minerAddress].storageCapacity;

        }

    }

    function minersCount() public view returns (uint256 count) {

        count = 0;

        for (uint256 i = 0; i < minersCount(); i++) {

            address minerAddress = getMinerAtIndex(i);

            if (miners[minerAddress].isRegistered) {

                count++;

            }

        }

    }

    function getMinerAtIndex(uint256 index) public view returns (address minerAddress) {

        minerAddress = address(uint160(uint(keccak256(abi.encodePacked(blockhash(block.number), index)))) & 0x00FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF);

    }

    function findAvailableMiner(uint256 \_storageSize) internal view returns (address availableMiner) {

        for (uint256 i = 0; i < minersCount(); i++) {

            address minerAddress = getMinerAtIndex(i);

            if (miners[minerAddress].isRegistered && miners[minerAddress].storageCapacity >= \_storageSize) {

                return minerAddress;

            }

        }

    }

    function storeData(address \_client, address \_miner) internal onlyMiner {

        emit DataStored(\_client, \_miner, clients[\_client].storageSize);

        // Pay commission to miners

        uint256 commission = clients[\_client].commissionToPay;

        uint256 commissionToPayLevel1 = (commission \* commissionLevel1) / 100;

        uint256 commissionToPayLevel2 = (commission \* commissionLevel2) / 100;

        uint256 commissionToPayCompany = commission - commissionToPayLevel1 - commissionToPayLevel2;

        miners[\_miner].storageCapacity -= clients[\_client].storageSize;

        clients[\_client].isRegistered = false;

        clients[\_client].storageSize = 0;

        clients[\_client].commissionToPay = 0;

        payable(\_miner).transfer(commissionToPayLevel1);

        payable(findAvailableMiner(clients[\_client].storageSize)).transfer(commissionToPayLevel2);

        companyAddress.transfer(commissionToPayCompany);

    }

}

File Upload

Softwaere need Go lang

GO lang IPFS