/\*\*

   \* @title ContractName

   \* @dev ContractDescription

   \* @custom:dev-run-script file\_path

   \* @custom:dev-run-script NatSpec tag

   \*/

// SPDX-License-Identifier: MIT

pragma solidity 0.8.18;

contract ProofOfStake {

    uint public totalSupply;

    mapping(address => uint) public balances;

    mapping(address => uint) public stakes;

    address[] public stakers;

    event Staked(address indexed staker, uint amount);

    event Unstaked(address indexed staker, uint amount);

    event RewardPaid(address indexed staker, uint amount);

    constructor(uint \_initialSupply) {

        totalSupply = \_initialSupply;

        balances[msg.sender] = \_initialSupply;

    }

    function stake(uint \_amount) external {

        require(\_amount > 0, "Amount must be greater than zero");

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

        balances[msg.sender] -= \_amount;

        stakes[msg.sender] += \_amount;

        stakers.push(msg.sender);

        emit Staked(msg.sender, \_amount);

    }

    function unstake(uint \_amount) external {

        require(\_amount > 0, "Amount must be greater than zero");

        require(\_amount <= stakes[msg.sender], "Insufficient stake amount");

        balances[msg.sender] += \_amount;

        stakes[msg.sender] -= \_amount;

        emit Unstaked(msg.sender, \_amount);

    }

    function reward() external {

        uint totalReward = calculateReward();

        require(totalReward > 0, "No reward available");

        for (uint i = 0; i < stakers.length; i++) {

            address staker = stakers[i];

            uint stakeAmount = stakes[staker];

            uint stakerReward = (totalReward \* stakeAmount) / totalSupply;

            balances[staker] += stakerReward;

            emit RewardPaid(staker, stakerReward);

        }

    }

    function calculateReward() private view returns (uint) {

        // Here you can implement your own logic to calculate the reward

        // based on factors like total staked amount, time, or other parameters.

        // For simplicity, we'll return a fixed value in this example.

        return 100; // Fixed reward for demonstration purposes

    }

}