// SPDX-License-Identifier: MIT

pragma solidity ^0.8.21;

contract InsuranceContract {

address public insurer;

constructor() {

insurer = msg.sender; // contract deployer is the insurer

}

modifier onlyInsurer() {

require(msg.sender == insurer, "Only the insurer can perform this action");

\_;

}

enum PolicyStatus { Inactive, Active, Expired, ClaimSubmitted, ClaimApproved, ClaimPaid }

struct Policy {

address policyholder;

uint256 premium;

uint256 coverage;

uint256 duration; // in seconds

uint256 startTime;

PolicyStatus status;

}

struct Claim {

uint256 policyId;

uint256 claimAmount;

string reason;

bool approved;

bool paid;

}

uint256 public policyCounter = 0;

uint256 public claimCounter = 0;

mapping(uint256 => Policy) public policies;

mapping(uint256 => Claim) public claims;

event PolicyIssued(uint256 indexed policyId, address indexed policyholder);

event PremiumPaid(uint256 indexed policyId, uint256 amount);

event ClaimSubmitted(uint256 indexed claimId, uint256 indexed policyId, uint256 amount, string reason);

event ClaimApproved(uint256 indexed claimId);

event ClaimPaid(uint256 indexed claimId, address indexed policyholder, uint256 amount);

// Issue new policy

function issuePolicy(address \_policyholder, uint256 \_premium, uint256 \_coverage, uint256 \_duration) external onlyInsurer {

require(\_policyholder != address(0), "Invalid address");

policyCounter++;

policies[policyCounter] = Policy({

policyholder: \_policyholder,

premium: \_premium,

coverage: \_coverage,

duration: \_duration,

startTime: 0,

status: PolicyStatus.Inactive

});

emit PolicyIssued(policyCounter, \_policyholder);

}

// Pay premium

function payPremium(uint256 \_policyId) external payable {

Policy storage policy = policies[\_policyId];

require(msg.sender == policy.policyholder, "Only policyholder can pay");

require(policy.status == PolicyStatus.Inactive, "Premium already paid or invalid status");

require(msg.value == policy.premium, "Incorrect premium amount");

policy.startTime = block.timestamp;

policy.status = PolicyStatus.Active;

emit PremiumPaid(\_policyId, msg.value);

}

// Submit claim

function submitClaim(uint256 \_policyId, uint256 \_amount, string calldata \_reason) external {

Policy storage policy = policies[\_policyId];

require(msg.sender == policy.policyholder, "Only policyholder can submit claims");

require(policy.status == PolicyStatus.Active, "Policy not active");

require(block.timestamp <= policy.startTime + policy.duration, "Policy expired");

require(\_amount <= policy.coverage, "Claim exceeds coverage");

claimCounter++;

claims[claimCounter] = Claim({

policyId: \_policyId,

claimAmount: \_amount,

reason: \_reason,

approved: false,

paid: false

});

policy.status = PolicyStatus.ClaimSubmitted;

emit ClaimSubmitted(claimCounter, \_policyId, \_amount, \_reason);

}

// Approve claim

function approveClaim(uint256 \_claimId) external onlyInsurer {

Claim storage claim = claims[\_claimId];

Policy storage policy = policies[claim.policyId];

require(!claim.approved, "Already approved");

require(policy.status == PolicyStatus.ClaimSubmitted, "Claim not submitted");

claim.approved = true;

policy.status = PolicyStatus.ClaimApproved;

emit ClaimApproved(\_claimId);

}

// Pay claim

function payClaim(uint256 \_claimId) external onlyInsurer {

Claim storage claim = claims[\_claimId];

Policy storage policy = policies[claim.policyId];

require(claim.approved, "Claim not approved");

require(!claim.paid, "Already paid");

require(address(this).balance >= claim.claimAmount, "Insufficient contract balance");

claim.paid = true;

policy.status = PolicyStatus.ClaimPaid;

payable(policy.policyholder).transfer(claim.claimAmount);

emit ClaimPaid(\_claimId, policy.policyholder, claim.claimAmount);

}

// Helper function to fund the contract

function fundContract() external payable onlyInsurer {}

// Check contract balance

function getContractBalance() external view returns (uint256) {

return address(this).balance;

}

}