// Tracking contract

// contract to allow supply chain stakeholders to track delivery of goods and automatically execute payment in tokens

pragma solidity ^0.4.24;

contract Tracking {

address admin;

uint[] contractLocation; // array containing latitude & longitude

uint contractLeadTime; // in seconds

uint contractPayment; // in tokens

mapping (string => Delivery) deliveries;

mapping (address => uint) balances;

struct Delivery {

string item;

uint quantity;

uint[] locationData;

uint timeStamp;

address sender;

}

// events to display messages after execution of certain transactions

event Success(string \_message, string trackingNo, uint[] \_locationData, uint \_timeStamp, address \_sender);

event Payment(string \_message, address \_from, address \_to, uint \_amount);

event Failure(string \_message);

// initial token supply upon contract deployment

function Tracking(uint \_initialTokenSupply) {

admin = msg.sender;

balances[admin] = \_initialTokenSupply; // Initially all tokens are held by admin

}

// send tokens from one account to another

function sendToken(address \_from, address \_to, uint \_amount)returns (bool success) {

if (balances[\_from] < \_amount) {

Failure('Insufficient funds for payment');

return false;

}

balances[\_from] -= \_amount;

balances[\_to] += \_amount;

Payment('Payment sent', \_from, \_to, \_amount);

return true;

}

// show balance of an account

function getBalance(address \_account) constant returns (uint \_balance) {

return balances[\_account];

}

// set contract parameters for next leg of delivery (can only be done by admin)

function setContractParameters(uint[] \_location, uint \_leadTime, uint \_payment) onlyAdmin returns (bool success) {

contractLocation = \_location; // set next location that will receive delivery

contractLeadTime = \_leadTime; // set acceptable lead time for next leg of delivery

contractPayment = \_payment; // set payment amount for completing next leg of delivery

return true;

}

// input details of delivery to send

function sendDelivery(string trackingNo, string \_item, uint \_quantity, uint[] \_locationData) returns (bool success) {

deliveries[trackingNo].item = \_item;

deliveries[trackingNo].quantity = \_quantity;

deliveries[trackingNo].locationData = \_locationData;

deliveries[trackingNo].timeStamp = block.timestamp;

deliveries[trackingNo].sender = msg.sender;

Success('Item delivered', trackingNo, \_locationData,block.timestamp, msg.sender);

return true;

}

// input details of received delivery

function receiveDelivery (string trackingNo, string \_item, uint \_quantity, uint[] \_locationData) returns (bool success) {

// check that item and quantity received match item and quantity delivered

if (sha3(deliveries[trackingNo].item) == sha3(\_item) && deliveries[trackingNo].quantity == \_quantity) {

Success('Item received', trackingNo, \_locationData, block.timestamp, msg.sender);

// execute payment if item received on time and location correct

if (block.timestamp <= deliveries[trackingNo].timeStamp + contractLeadTime && \_locationData[0] == contractLocation[0] && \_locationData[1] == contractLocation[1]) {

//sendToken(deliveries[trackingNo].sender, admin, contractPayment);

sendToken(msg.sender, admin, contractPayment);

}

else {

Failure('Payment not triggered as criteria not met');

}

return true;

}

else {

Failure('Error in item/quantity');

return false;

}

}

// remove details of delivery from database (can only be done by admin)

function deleteDelivery(string trackingNo) onlyAdmin returns (bool success) {

delete deliveries[trackingNo];

return true;

}

// display details of delivery

function checkDelivery(string trackingNo) constant returns (string, uint, uint[], uint, address) {

return (deliveries[trackingNo].item, deliveries[trackingNo].quantity, deliveries[trackingNo].locationData, deliveries[trackingNo].timeStamp, deliveries[trackingNo].sender);

}

// allow only admin to execute specific function using this modifier

modifier onlyAdmin() {

if (msg.sender != admin) throw;

\_;

}

}

// Provenance Contract

// Contract that allow supply chain stakeholders to check the provenance of goods

pragma solidity ^0.4.24;

contract Provenance {

address CAuthority;

struct Producer {

string name;

string city;

uint contactno;

bool authorized;

}

struct Product {

address producer;

uint[] location; // array containing latitude & longitude

uint timeStamp;

}

mapping (address => Producer) producers;

mapping (string => Product) products;

function Provenance() {

CAuthority = msg.sender;

}

// Add producer (only CAuthority can add)

function Add\_Producer(string \_name, string \_city, uint \_contactno) returns (bool success) {

// Avoide overwrite of existing entries and ensure name is not null

if (bytes(producers[msg.sender].name).length == 0 && bytes(\_name).length != 0) {

producers[msg.sender].name = \_name;

producers[msg.sender].contactno = \_contactno;

producers[msg.sender].city = \_city;

producers[msg.sender].authorized = false;

return true;

}

else {

return false; // Entry already exists in database or a null name is entered

}

}

// Authorize producer (only CAuthority authorize)

function Authorize\_Producer(address \_producer) onlyCAuthority returns (bool success) {

producers [\_producer].authorized = true;

return true;

}

// Add product details (producer can add this)

function Add\_Product(string serial\_no, uint[] \_location) returns (bool success) {

// ensure no duplicate or null serial number

if (products[serial\_no].producer == 0X0 && bytes(serial\_no).length != 0) {

products [serial\_no].producer = msg.sender;

products[serial\_no].location = \_location;

products[serial\_no].timeStamp = block.timestamp;

return true;

}

else {

return false; // already used serial number or null entry

}

}

// Get producer details

function Find\_Producer(address \_producer) constant returns (string, uint, string, bool){

return (producers[\_producer].name, producers[\_producer].contactno, producers[\_producer].city, producers[\_producer].authorized);

}

// Get product details

function Find\_Product(string serial\_no) constant returns (address, uint[], uint) {

return (products[serial\_no].producer,

products [serial\_no] .location, products[serial\_no] .timeStamp);

}

// Remove producer (only CAuthority can remove)

function Remove\_Producer(address \_producer) onlyCAuthority returns (bool success) {

delete producers [\_producer];

return true;

}

// Remove product (only CAuthority can remove)

function Remove\_Product(string serial\_no) onlyCAuthority returns (bool success) {

delete products [serial\_no];

return true;

}

// Allow only CAuthority to execute specific functions using this modifier

modifier onlyCAuthority() {

if (msg.sender != CAuthority) throw;

\_;

}

}

**Tracking Contract modified 10 April 2019**

pragma solidity >=0.4.22 <0.6.0;

contract TrackingContract {

address admin;

uint[] contractLocation; // array containing latitude & longitude

uint contractLeadTime; // in seconds

uint contractPayment; // in tokens

mapping (string => Delivery) deliveries;

mapping (address => uint) balances;

struct Delivery {

string item;

uint quantity;

uint[] locationData;

uint timeStamp;

address sender;

}

// events to display messages after execution of certain transactions

event Success(string \_message, string trackingNo, uint[] \_locationData, uint \_timeStamp, address \_sender);

event Payment(string \_message, address \_from, address \_to, uint \_amount);

event Failure(string \_message);

// initial token supply upon contract deployment

function Tracking(uint \_initialTokenSupply) public {

admin = msg.sender;

balances[admin] = \_initialTokenSupply; // Initially all tokens are held by admin

}

// send tokens from one account to another

function sendToken(address \_from, address \_to, uint \_amount) public returns (bool success) {

if (balances[\_from] < \_amount) {

emit Failure('Insufficient funds for payment');

return false;

}

balances[\_from] -= \_amount;

balances[\_to] += \_amount;

emit Payment('Payment sent', \_from, \_to, \_amount);

return true;

}

// show balance of an account

function getBalance(address \_account) public returns (uint \_balance) {

return balances[\_account];

}

// set contract parameters for next leg of delivery (can only be done by admin)

function setContractParameters(uint[] memory \_location, uint \_leadTime, uint \_payment) onlyAdmin public returns (bool success) {

contractLocation = \_location; // set next location that will receive delivery

contractLeadTime = \_leadTime; // set acceptable lead time for next leg of delivery

contractPayment = \_payment; // set payment amount for completing next leg of delivery

return true;

}

// input details of delivery to send

function sendDelivery(string memory trackingNo, string memory \_item, uint \_quantity, uint[] memory \_locationData) public returns (bool success) {

deliveries[trackingNo].item = \_item;

deliveries[trackingNo].quantity = \_quantity;

deliveries[trackingNo].locationData = \_locationData;

deliveries[trackingNo].timeStamp = block.timestamp;

deliveries[trackingNo].sender = msg.sender;

emit Success('Item delivered', trackingNo, \_locationData,block.timestamp, msg.sender);

return true;

}

// input details of received delivery

function receiveDelivery (string memory trackingNo, string memory \_item, uint \_quantity, uint[] memory \_locationData) public returns (bool success) {

// check that item and quantity received match item and quantity delivered

if (keccak256(abi.encodePacked(deliveries[trackingNo].item)) == keccak256(abi.encodePacked(\_item)) && deliveries[trackingNo].quantity == \_quantity) {

emit Success('Item received', trackingNo, \_locationData, block.timestamp, msg.sender);

// execute payment if item received on time and location correct

if (block.timestamp <= deliveries[trackingNo].timeStamp + contractLeadTime && \_locationData[0] == contractLocation[0] && \_locationData[1] == contractLocation[1]) {

//sendToken(deliveries[trackingNo].sender, admin, contractPayment);

sendToken(msg.sender, admin, contractPayment);

}

else {

emit Failure('Payment not triggered as criteria not met');

}

return true;

}

else {

emit Failure('Error in item/quantity');

return false;

}

}

// remove details of delivery from database (can only be done by admin)

function deleteDelivery(string memory trackingNo) onlyAdmin public returns (bool success) {

delete deliveries[trackingNo];

return true;

}

// display details of delivery

function checkDelivery(string memory trackingNo) public returns (string memory, uint, uint[] memory, uint, address) {

return (deliveries[trackingNo].item, deliveries[trackingNo].quantity, deliveries[trackingNo].locationData, deliveries[trackingNo].timeStamp, deliveries[trackingNo].sender);

}

// allow only admin to execute specific function using this modifier

modifier onlyAdmin() {

if (msg.sender != admin) revert();

\_;

}

}