// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

//我希望在这个合约中能实现所有的电子商务的步骤

//在合约中使用类似于状态机的东西

contract Commerce {

uint public auctionEnd;

uint public value;

address payable public seller;

address payable public buyer;

address payable public transport;

enum State { Created,Locked,confirm,Transporting,Return,Release,Inactive }

State public state;

modifier condition(bool \_condition) {

require(\_condition);

\_;

}

modifier onlyBuyer() {

require(

msg.sender == buyer,

"Only buyer can call this."

);

\_;

}

modifier onlySeller() {

require(

msg.sender == seller,

"Only seller can call this."

);

\_;

}

modifier inState(State \_state) {

require(

state == \_state,

"Invaild state."

);

\_;

}

event Aborted();

event PurchaseConfirmed(string goods);

event Transports(string name);

event ItemReceived();

event SellerRefunded();

event Canceled();

event Evaluated(uint trans,uint quality,uint general,string comment);

//确保“msg.value”是一个偶数，

//如果它是一个奇数，则它将被截断

//通过乘法检查它不是奇数。

constructor() payable {

seller = payable(msg.sender);

value = msg.value/2;

require((2\*value) == msg.value,"Value has to be even.");

}

//这一结构体能用来再次说明商品的名称以及价格

struct Product {

string name; //商品名称

uint price; //商品价格

}

//平台发布声明

Product[] public products;

//这个函数的作用是声明商品的价格与名称

function declare(string memory name,uint price)public onlySeller inState(State.Created) returns(uint){

products.push(Product(name,price));

return products.length;

}

///终止购买并回收以太币

///在合约锁定前只能由卖家调用

function abort() public onlySeller inState(State.Created)

{

emit Aborted();

state = State.Inactive;

seller.transfer(address(this).balance);

}

///买家确认购买

///交易需包含2\*value个以太币

///以太币会被锁定，直到confirmReceived被调用

//买家需要输入对应的商品名称

function confirmPurchase(string memory goods) public inState(State.Created) condition(msg.value == (2\*value)) payable

{

emit PurchaseConfirmed(goods);

buyer = payable(msg.sender);

state = State.Locked;

}

//这里仅能为卖家调用，用于认证买家购买的信息，并不在阶段上加以限制，意为可不入流程。

function ecrecovery(bytes32 hash, bytes memory sig)onlySeller public returns (address)

{

bytes32 r;

bytes32 s;

uint8 v;

/\*

if (sig.length != 65) {

return 0;

}

\*/

assembly {

r := mload(add(sig, 32))

s := mload(add(sig, 64))

v := and(mload(add(sig, 65)), 255)

}

// https://github.com/ethereum/go-ethereum/issues/2053

if (v < 27) {

v += 27;

}

/\*

if (v != 27 && v != 28) {

return 0;

}

\*/

/\* prefix might be needed for geth only

\* https://github.com/ethereum/go-ethereum/issues/3731

\*/

// bytes memory prefix = "\x19Ethereum Signed Message:\n32";

// hash = sha3(prefix, hash);

return ecrecover(hash, v, r, s);

}

//引入第三方“物流”，用以监督交易并且完成运输工作

//这一过程必须要有，才能完成阶段的转换

//物流方需要输入自己的名称

function logistics(string memory name,uint \_biddingTime) public inState(State.Locked)

{

transport = payable(msg.sender);

emit Transports(name);

auctionEnd = block.timestamp + \_biddingTime;

state = State.Transporting;

}

//这里仅为买家调用，买家可验证物流信息，不入流程，即也可不验证

function ecrecovery2(bytes32 hash, bytes memory sig) onlyBuyer public returns (address)

{

bytes32 r;

bytes32 s;

uint8 v;

/\*

if (sig.length != 65) {

return 0;

}

\*/

assembly {

r := mload(add(sig, 32))

s := mload(add(sig, 64))

v := and(mload(add(sig, 65)), 255)

}

// https://github.com/ethereum/go-ethereum/issues/2053

if (v < 27) {

v += 27;

}

/\*

if (v != 27 && v != 28) {

return 0;

}

\*/

/\* prefix might be needed for geth only

\* https://github.com/ethereum/go-ethereum/issues/3731

\*/

// bytes memory prefix = "\x19Ethereum Signed Message:\n32";

// hash = sha3(prefix, hash);

return ecrecover(hash, v, r, s);

}

//确认你（买家）已经收到商品

//这会释放被锁定的以太币

function confirmReceived() public onlyBuyer inState(State.Transporting)

{

require(block.timestamp<=auctionEnd,"time is over,commerce is over.");

emit ItemReceived();

state = State.Release;

buyer.transfer(value);

}

//这个函数是用于在规定时间内收不到货或者需要退货时所使用的的，有时限

function Returned() public onlyBuyer inState(State.Transporting)

{

require(block.timestamp>=auctionEnd,"time is over,commerce is over.");

emit Canceled();

state = State.Return;

//buyer.transfer(2\*value);

}

//在正常交易的情况下，只需向卖家付款，所付款项中包含有物流费

function refundSeller() public onlySeller inState(State.Release)

{

emit SellerRefunded();

state = State.Inactive;

seller.transfer(3\*value);

}

//在买家发起退款的情况下，经卖家确认进行退款

//退款时，先对卖家进行退款，gas的花费由买家付

function Seller\_Returned() public onlySeller inState(State.Return)

{

state = State.Inactive;

seller.transfer(2\*value);

buyer.transfer(2\*value);

}

//在最后引入一个评价与打分的机制，

//方便之后的顾客进行参考

function Comment(uint trans,uint quality,uint general,string memory comment) public onlyBuyer inState(State.Inactive)

{

emit Evaluated(trans,quality,general,comment);

}

}