// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

//import "OpenZeppelin/openzeppelin-contracts@4.0.0/contracts/access/Ownable.sol";

/\*\* @title A registry of dataset bids. \*/

contract DatasetBidRegistry{ //is Ownable {

/\*\* Organizations may issue bids for datasets.

\*

\* A bid has an expiry date, plus an amount which is the price willing to pay, and

\* a tag which serves for indexing (e.g. references to concepts in knowledge graphs or terminologies).

\*

\* The spec of the dataset is not stored in the smart contract, but instead

\* a hash to a claim file is included.

\* A claim file is just a JSON document with:

\* - Another IPFS hash to the dataset spec file.

\* - A digital signature of the hash (ideally associated to a certificate of the bidder).

\*

\* The dataset spec file is a JSON document with the complete description of the

\* desired dataset, which may be verbose and detailed.

\*

\* Claim files can be considered similar to the concept of a Ricardian contract, since the

\* bidder commits to pay the specified amount iif the specs are met.

\*/

uint256 Time1=block.timestamp;

struct Bid{

string expiry\_date; // Standard human readable date, ISO 8601 UTC

bytes32 hash\_spec; // IPFS address to the claim file of the dataset specs of the bid.

uint amount; // Amount in wei for the bid.

string tag; // Keyword

}

/\*\*

\* An offer is issued by a dataset provider in response to an (active) bid.

\*/

struct Offer{

address payable offerer; // address of the offerer. .

address bidder; // bidder and bid\_number identify the bid.

uint bid\_number;

bytes32 hash\_spec; // IPFS address to the claim file of the dataset specs of the offer.

uint value; // price proposed, in wei.

bool completed;

}

function IPFS\_Hash\_spec()public{

bytes32 hash\_spec= keccak256(abi.encodePacked(msg.sender));

emit Hash\_spec(hash\_spec, msg.sender, block.timestamp-Time1);

}

event Hash\_spec(bytes32 hash\_spec, address bidder, uint256 Latency);

// Notification for interested parties of new bids.

event BidRegistered(

address who,

uint position,

uint256 Latency

);

// Notification for interested parties of new offers.

event OfferRegistered(

address from,

address to,

uint bid\_number,

uint offer\_number,

uint value,

uint256 Latency

);

// The bids per bidder.

mapping (address => Bid[]) public bids;

// The offers received per bid. Associates a (bidder address, bid no) to a list of offers.

mapping (address => mapping (uint => Offer[])) public offers;

/\*\* @dev Register a temporary dataset bid.

\* @param date Expiration date of the bid.

\* @param hash\_spec Hash to the spec of the bid.

\* @param amount Amount to be paid for the dataset.

\*/

function register(string memory date,

bytes32 hash\_spec,

uint amount,

string memory tag) public {

bids[msg.sender].push(Bid(date, hash\_spec, amount, tag));

emit BidRegistered(msg.sender, bids[msg.sender].length-1, block.timestamp-Time1);

}

/\*\*

\* Gets info of a particular bid.

\* @param bidder The address of the bidder.

\* @param pos The position in the list of bids by the bidder (included in BidRegistered events).

\*/

function bidinfo(address bidder, uint pos) public view

returns (string memory expiry\_date, bytes32 hash\_spec, uint amount, string memory tag, uint256 Latency){

Bid memory b = bids[bidder][pos];

return (b.expiry\_date, b.hash\_spec, b.amount, b.tag, block.timestamp-Time1);

}

/\*\*

\* An offer is a response to an (active) bid for a dataset.

\*

\* Expiry dates of bids are considered informative and not checked inside the contract,

\* it is expected that clients do the check.

\*/

function offer(address payable offerer, address bidder, uint bidno, bytes32 hash\_spec, uint price) public {

offers[bidder][bidno].push(Offer(offerer, bidder, bidno, hash\_spec, price, false));

emit OfferRegistered(offerer, bidder, bidno, offers[bidder][bidno].length - 1, price, block.timestamp-Time1);

}

/\*\*

\* The bidder accepts and finalize an offer.

\*

\* Note bidder is the transaction sender.

\*/

function finalize(uint bidno, uint offerno) public payable{

assert(msg.value==offers[msg.sender][bidno][offerno].value);

assert(msg.sender==offers[msg.sender][bidno][offerno].bidder);

offers[msg.sender][bidno][offerno].completed = true;

// Transfer the value.

offers[msg.sender][bidno][offerno].offerer.transfer(msg.value);

emit PaymentCompleted(block.timestamp-Time1);

}

event PaymentCompleted(uint256 Latency);

}