pragma solidity ^0.8.0;

contract DwitterManage {

//upvote, report, delete functionalities

//include display function

struct Dweet {

uint id;

string content;

address author;

uint timestamp;

uint upvotes;

uint reports;

string hashtag;

bool deleted;

}

struct User{

uint id;

address pkey;

string firstname;

string lastname;

string userName;

string bio;

uint following;

uint followers;

}

uint public dweet\_count=0;

uint public user\_count=0;

Dweet[] public dweets;

User[] public users;

uint256 MAX\_INT = 2\*\*256 - 1;

event upvoted(uint id, address upvoter);

event downvoted(uint id, address downvoter);

event reported(uint id, address reporter);

event deleted(uint id, uint timestamp);

event followed(uint followers, uint following);

modifier onlyAuthor(uint id) {

require(msg.sender == dweetToAuthor[id]);

\_;

}

mapping(address => uint) public addressToId;

mapping(string => uint) public userNameToId;

mapping(uint => address) public dweetToAuthor; //maps dweet's id to author's address

mapping(address => uint) public dweetCountAuthor; //stores number of dweets by individual author

mapping(address => bool) public accountCheck; //only one account per public key

mapping(string => bool) public userNameCheck; //check if username already taken

mapping(uint => string) public idToUsername;

mapping(address => uint[]) public upvotesList;

mapping(address => uint[]) public reportsList;

event NewUserAdd( string userName, address pkey);

event NewDweetAdd(string content, string hashtag, uint timestamp);

modifier accountAlreadyExists(){

require(accountCheck[msg.sender] == false);

\_;

}

modifier userNameAlreadyExists(string memory \_userName){

require(userNameCheck[\_userName] == false);

\_;

}

modifier userExists(){

require(accountCheck[msg.sender] == true);

\_;

}

modifier userNameExists(string memory \_userName){

require(userNameCheck[\_userName] == true);

\_;

}

function registerNewUser(string memory \_firstName, string memory \_lastName, string memory \_userName, string memory \_bio) public accountAlreadyExists userNameAlreadyExists(\_userName){

users.push(User(user\_count, msg.sender, \_firstName, \_lastName, \_userName, \_bio,0,0));

accountCheck[msg.sender] = true;

userNameCheck[\_userName] = true;

userNameToId[\_userName]= user\_count;

addressToId[msg.sender]= user\_count;

idToUsername[user\_count] = \_userName;

user\_count++;

emit NewUserAdd(\_userName, msg.sender);

}

function addNewDweet(string memory \_content, string memory \_hashtag) userExists public{

dweets.push(Dweet(dweet\_count, \_content, msg.sender, block.timestamp, 0, 0, \_hashtag, false));

dweetToAuthor[dweet\_count] = msg.sender;

dweetCountAuthor[msg.sender]++;

dweet\_count++;

emit NewDweetAdd(\_content, \_hashtag, block.timestamp);

}

function getUpvotesList(address user) view public returns(uint[] memory ids) {

return upvotesList[user];

}

function getReportsList(address user) view public returns(uint[] memory ids) {

return reportsList[user];

}

function upvoteDweet(uint id) userExists public {

dweets[id].upvotes++;

upvotesList[msg.sender].push(id);

emit upvoted(id, msg.sender);

}

function downvoteDweet(uint id) userExists public {

dweets[id].upvotes--;

uint[] memory temp = getUpvotesList(msg.sender);

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

if(temp[i] == id){

upvotesList[msg.sender][i] = MAX\_INT;

}

}

emit downvoted(id, msg.sender);

}

function reportDweet(uint id) userExists public

dweets[id].reports++;

reportsList[msg.sender].push(id);

emit reported(id, msg.sender);

if(dweets[id].reports > 10){

dweets[id].deleted = true;

}

}

function deleteDweet(uint id) onlyAuthor(id) public {

dweets[id].deleted = true;

emit deleted(id, block.timestamp);

}

function search(string memory \_userName) userNameExists(\_userName) view public returns(string memory \_firstname, string memory \_lastName, address pkey, string memory \_bio, uint followers, uint following){

uint id1= userNameToId[\_userName];

return(users[id1].firstname, users[id1].lastname, users[id1].pkey, users[id1].bio, users[id1].followers, users[id1].following);

}

function followUser(uint id) userExists public {

address toBeFollowed1= users[id].pkey;

require(msg.sender != toBeFollowed1);

users[id].followers++;

uint id1 = addressToId[msg.sender];

users[id1].following++;

emit followed(users[id].followers, users[id1].following);

}

}