Use Case: Property transfer

Problem: Currently, we use the central authority to transfer the property ownership. This makes it time consuming and attracts a lot of extra expenses too with an additional burden of document management. Also, since the system is centralized, there is always a possibility of fraudulent.

Possible Solution: One technology that immediately comes to mind is the blockchain and the elegant way in which it uses distributed ledgers. If we use blockchain to record land transfers, we can have an immutable history of every property transaction that can be viewed by everyone and tampered by no one

This brings us to a point that how the government will play its role when people shall try to transfer the property on their own? Here govt can initiate this service as a part of egovernance & cut down delays

Some common example of land registry on blockchain can be seen here: http://bitlandglobal.com/ .

So, here is the problem which you must solve. You need to build a smart contract which is capable of handling property transfer with below cases

Actions to be performed:

• Insert some dummy properties to replicate the real world. These properties shall have basic characteristics e.g. address, location, floors & etc. Make sure you create demo entries from a single owner. This single owner shall assign the properties to you. If you try to replicate it in real world, generally the government shall be signing the document which says that now you're the owner of this property. So here in this case, think that you're building this solution for a government. Government shall be assigning the properties after verifying your original documents. Hence, make sure that you generate the dummy properties by a single owner & this shall be distributing it to multiple people (addresses) Hint: Take motivation from coin example that we discussed in the class

• Any address can send the property from his account to other owner. Check the ownership of property before transferring it to the other owner

• Try to include as many general parameters as possible to keep the smart contract almost replicating the real-world transaction Example: Name of the owner, Name of the new owner, property cost, location etc. (Be creative here)

• After the successful transfer of the property, you need to be sure that the old owner is not able to send the property again (encountering double spending)

• Finally, the new owner must be able to send the freshly received property to any owner(address), he/she wishes to

Note: Each of the above shared points should be checked. Create a bare minimum UI so that all these can be validated from the UI itself.

Project Work for Property Transfer

As we all know that the property is something that takes very long-time process to transfer to one person to another. The government or other companies whose job is on transferring the property need to verify many things to take place for the transferring. They need to verify the current real owner who is going to sell the property.

We are making this use case for mimicking the real-world property transfer.

Pre-requisite of this use case is that:

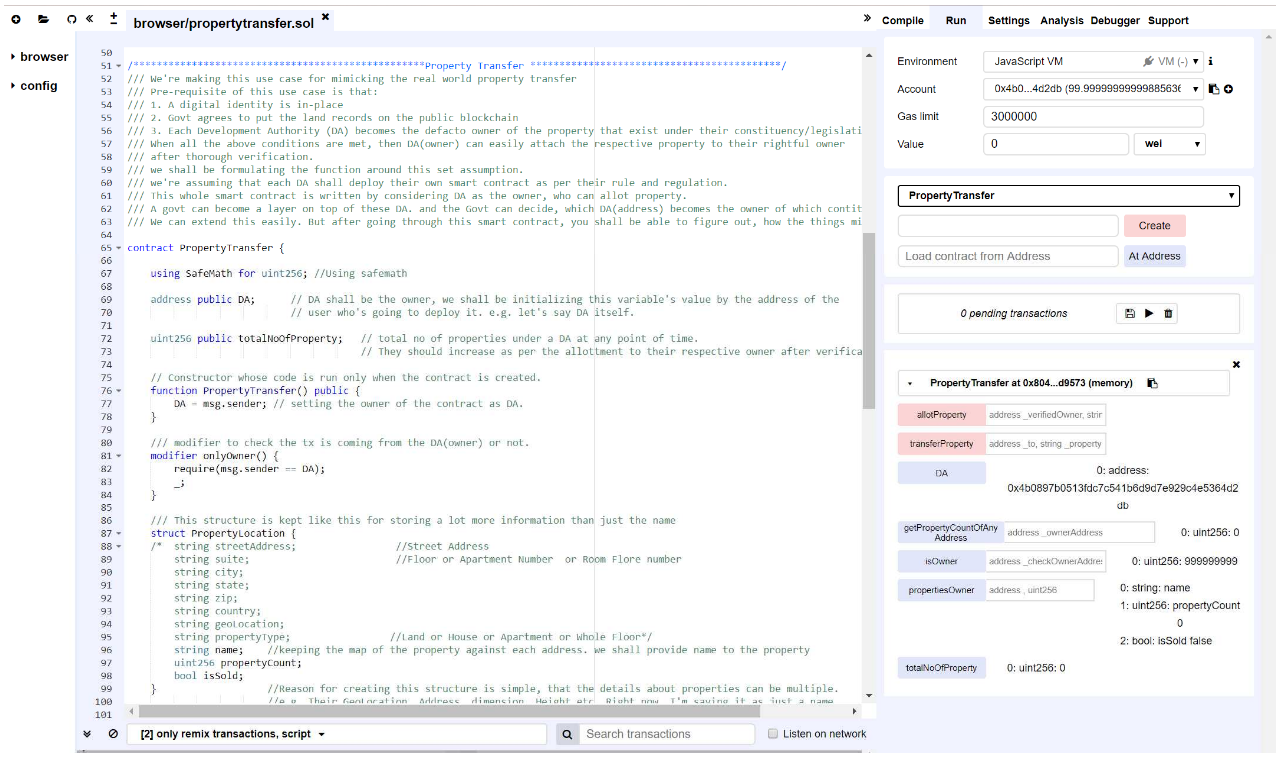
1. A digital identity is in-place
2. Govt agrees to put the land records on the public blockchain
3. Each Development Authority (DA) becomes the defacto owner of the property that exist under their constituency/legislative body

When all the above conditions are met, then DA (owner) can easily attach the respective property to their rightful owner after thorough verification. We shall be formulating the function around this set assumption. We are assuming that each DA shall deploy their own smart contract as per their rule and regulation.

This whole smart contract is written by considering DA as the owner, who can allot property. A government can become a layer on top of these DA. and the government can decide, which DA (address) becomes the owner of which constituency.

We can extend this easily. But after going through this smart contract, we shall be able to figure out, how the things will work.

Project Code in Remix browser



```
/**************************************************Property Transfer *******************************************/
/// We're making this use case for mimicking the real world property transfer
/// Pre-requisite of this use case is that:
/// 1. A digital identity is in-place
/// 2. Govt agrees to put the land records on the public blockchain
/// 3. Each Development Authority (DA) becomes the defacto owner of the property that exist under their constituency/legislati
/// When all the above conditions are met, then DA(owner) can easily attach the respective property to their rightful owner
/// after thorough verification.
/// we shall be formulating the function around this set assumption.
/// we're assuming that each DA shall deploy their own smart contract as per their rule and regulation.
/// This whole smart contract is written by considering DA as the owner, who can allot property.
/// A govt can become a layer on top of these DA. and the Govt can decide, which DA(address) becomes the owner of which contit
/// We can extend this easily. But after going through this smart contract, you shall be able to figure out, how the things mi

contract PropertyTransfer {

    using SafeMath for uint256; //Using safemath

    address public DA;      // DA shall be the owner, we shall be initializing this variable's value by the address of the
                            // user who's going to deploy it. e.g. let's say DA itself.

    uint256 public totalNoOfProperty;   // total no of properties under a DA at any point of time.
                                        // They should increase as per the allottment to their respective owner after verifica

    // Constructor whose code is run only when the contract is created.
    function PropertyTransfer() public {
        DA = msg.sender; // setting the owner of the contract as DA.
    }

    /// modifier to check the tx is coming from the DA(owner) or not.
    modifier onlyOwner() {
        require(msg.sender == DA);
        _;
    }

    /// This structure is kept like this for storing a lot more information than just the name
    struct PropertyLocation {
    /*  string streetAddress;                   //Street Address
        string suite;                           //Floor or Apartment Number  or Room Flore number
        string city;
        string state;
        string zip;
        string country;
        string geoLocation;
        string propertyType;                   //Land or House or Apartment or Whole Floor*/
        string name;    //keeping the map of the property against each address. we shall provide name to the property
        uint256 propertyCount;
        bool isSold;
    }                   //Reason for creating this structure is simple, that the details about properties can be multiple.
                        //e.g. Their GeoLocation, Address, dimension, Height etc. Right now, I'm saving it as just a name
```

[2] only remix transactions, script

Search transactions

Listen on network

Environment: JavaScript VM — VM (-)

Account: 0x4b0...4d2db (99.9999999999885636

Gas limit: 3000000

Value: 0 wei

PropertyTransfer

Create

Load contract from Address — At Address

0 pending transactions

PropertyTransfer at 0x804...d9573 (memory)

allotProperty — address _verifiedOwner, strin

transferProperty — address _to, string _property

DA — 0: address: 0x4b0897b0513fdc7c541b6d9d7e929c4e5364d2db

getPropertyCountOfAnyAddress — address _ownerAddress — 0: uint256: 0

isOwner — address _checkOwnerAddres — 0: uint256: 999999999

propertiesOwner — address , uint256 — 0: string: name 1: uint256: propertyCount 0 2: bool: isSold false

totalNoOfProperty — 0: uint256: 0

Property Transfer Smart Contract Code

```
pragma solidity ^0.4.18;

/*************************************SafeMath****************
*******************************************
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
library SafeMath {

  /**
  * @dev Multiplies two numbers, throws on overflow.
  */
  function mul(uint256 a, uint256 b) internal pure returns
(uint256) {
    if (a == 0) {
      return 0;
    }
    uint256 c = a * b;
    assert(c / a == b);
    return c;
  }

  /**
  * @dev Integer division of two numbers, truncating the quotient.
  */
  function div(uint256 a, uint256 b) internal pure returns
(uint256) {
    /// assert(b > 0); // Solidity automatically throws when
dividing by 0

    uint256 c = a / b;

    /// assert(a == b * c + a % b); // There is no case in which
this doesn't hold

    return c;

  }


 /**

  * @dev Substracts two numbers, throws on overflow (i.e. if
subtrahend is greater than minuend).

  */

  function sub(uint256 a, uint256 b) internal pure returns
(uint256) {

    assert(b <= a);

    return a - b;

  }


  /**

  * @dev Adds two numbers, throws on overflow.

  */

  function add(uint256 a, uint256 b) internal pure returns
(uint256) {

    uint256 c = a + b;

    assert(c >= a);

    return c;

  }

}

/***********************************************SafeMath Ending
***************************************************/
```

```
/************************************************Property
Transfer *****************************************/

contract PropertyTransfer {


     using SafeMath for uint256;   //Using safemath


    address public DA;        // DA shall be the owner, we shall
be initializing this variable's value by the address of the

                                   // user who's going to deploy
it. e.g. let's say DA itself.


    uint256 public totalNoOfProperty;   // total no of properties
under a DA at any point of time.

                                                  // They should
increase as per the allottment to their respective owner after
verification


    // Constructor whose code is run only when the contract is
created.

    function PropertyTransfer() public {

        DA = msg.sender; // setting the owner of the contract as
DA.

    }


    /// modifier to check the tx is coming from the DA(owner) or
not.

    modifier onlyOwner() {

        require(msg.sender == DA);

        _;

    }
    /// This structure is kept like this for storing a lot more
information than just the name

    struct PropertyLocation {

     /*   string streetAddress;                //Street Address

          string suite;                        //Floor or
Apartment Number  or Room Flore number

          string city;

          string state;

          string zip;

          string country;

          string geoLocation;

          string propertyType;                //Land or House or
Apartment or Whole Floor*/

        string name;     //keeping the map of the property against
each address. we shall provide name to the property

        uint256 propertyCount;

        bool isSold;

    }                        //Reason for creating this structure
is simple, that the details about properties can be multiple.

                             //e.g. Their GeoLocation, Address,
dimension, Height etc. Right now, I'm saying it as just a name


    mapping(address => mapping(uint256=>PropertyLocation)) public
propertiesOwner; // we shall have the properties mapped against
each address


                             // by its name and it's individual
count.

    mapping(address => uint256) individualCountOfPropertyPerOwner;
               // how many property does a particular person hold
    event PropertyAlloted(address indexed _verifiedOwner, uint256
indexed  _totalNoOfPropertyCurrently, string _nameOfProperty,
string _msg);


    event PropertyTransferred(address indexed _from, address
indexed _to, string _propertyName, string _msg);


    /// This shall give us the exact property count which any
address own at any point of time

    function getPropertyCountOfAnyAddress(address _ownerAddress)
public constant returns (uint256){

        uint count=0;

        for(uint i =0;
i<individualCountOfPropertyPerOwner[_ownerAddress];i++){

            if(propertiesOwner[_ownerAddress][i].isSold != true)

            count++;

        }

        return count;

    }


    /// this function shall be called by DA only after
verification

    function allotProperty(address _verifiedOwner, string
_propertyName) public onlyOwner {


propertiesOwner[_verifiedOwner][individualCountOfPropertyPerOwner[
_verifiedOwner]++].name = _propertyName;

        totalNoOfProperty++;


PropertyAlloted(_verifiedOwner,individualCountOfPropertyPerOwner[_
verifiedOwner], _propertyName, "property allotted successfully");

    }
    /// check whether the owner have the said property or not. if
yes, return the index

    function isOwner(address _checkOwnerAddress, string
_propertyName) public constant returns (uint){

        uint i ;

        bool flag ;

        for(i=0 ;
i<individualCountOfPropertyPerOwner[_checkOwnerAddress]; i++){

            if(propertiesOwner[_checkOwnerAddress][i].isSold ==
true){

                break;

            }

         flag =
stringsEqual(propertiesOwner[_checkOwnerAddress][i].name,_property
Name);

            if(flag == true){

                break;

            }

        }

        if(flag == true){

            return i;

        }

        else {

            return 999999999;// We're expecting that no individual
shall be owning this much properties

        }

    }


    /// functionality to check the equality of two strings in
Solidity

    function stringsEqual(string a1, string a2) private constant
returns (bool) {
        if(sha3(a1) == sha3(a2))
            return true;
        else
            return false;
    }

    /// transfer the property to the new owner
    /// todo : change from to msg.sender

    function transferProperty (address _to, string _propertyName) 
public returns (bool ,  uint )
    {
        uint256 checkOwner = isOwner(msg.sender, _propertyName);
        bool flag;

        if(checkOwner != 999999999 && 
propertiesOwner[msg.sender][checkOwner].isSold == false){
            /// step 1 . remove the property from the current 
owner and decrase the counter.
            /// step 2 . assign the property to the new owner and 
increase the counter
            propertiesOwner[msg.sender][checkOwner].isSold = true;
            propertiesOwner[msg.sender][checkOwner].name = 
"Sold";// really nice finding. we can't put empty string

propertiesOwner[_to][individualCountOfPropertyPerOwner[_to]++].nam
e = _propertyName;
            flag = true;
            PropertyTransferred(msg.sender , _to, _propertyName, 
"Owner has been changed." );
        }
        else {
            flag = false;
            PropertyTransferred(msg.sender , _to, _propertyName,
"Owner doesn't own the property." );
        }
        return (flag, checkOwner);
    }
}
```