Administering Land Titles using Blockchain: Documentation

Team: Tripple

- Tathagato Roy (*2019111020*)
- Snehal Kumar (*2019101003*)
- Rutvij Menavlikar (*2019111032*)

Scope Of Project

- This project has the potential to replace the whole system for Land Title Management and transfer. The decentralisation of data makes the system resilient against any disaster or data tampering. A public ledger makes the system more transparent. Digitisation makes this system much faster than the current one. And the nature of blockchain, specifically inability to delete data makes the stored data more authentic and harder to tamper with.
- Adding multiple ownerships to a land, while a paperwork disaster is much easier to implement with blockchain. Also lease and rental contracts can be included in this system.
- Bringing in a bank as as entity in the system, home loans and EMI payments can be automated easily. Also, approval of a home loan, comparing loan agreements offerred by several banks, etc. can all be made much faster and easier with blockchain.
- Another scope is to convert the land ownership into NFTs, and trade them in open market, like stock market. Transferring ownership and transaction security is easily handled by blockchains.

Structure

Citizen

A citizen is a land owner or buyer.
For a citizen, we store:

- A unique id assigned by the system
- Their digital address
- Their name
- Identity proof number
- Status (Verfied or Unverified)

Validator

A validator authenticates transactions being made in the system.
For a validator, we store:

- A unique id assigned by the system
- Their digital address
- Their name
- Government id proof
- Status (Authorised or Unauthorised)

Ownership

An ownership stores all the past and current ownerships of properties
For an ownership we store:

- A unique id assigned by the system
- Id of the property
- Purchase cost
- Seller Id
- Buyer Id
- Buyer name
- ownership status (whether the buyer still owns it, or whether they sold it)
- Proof of Ownership

Property

Property is the land, owned by a citizen.
For a property, we store:

- A unique id assigned by the system
- Physical address of the property
- Id of the current owner.
- Proof of purchase/ownership by the current owner.
- Ownership Id
- Mapping from previous ownership Ids to Ownerships.

Transaction

Sale of property or registering property are transactions which need to be endored by Validators.

For a transaction, we store:

- Transaction type (Registering or sale)
- Property Id
- Seller Id
- Buyer Id
- A unique id assigned by the system
- Transaction value
- Transaction fee
- Transaction proof
- Number of validators who approved the transaction.
- Boolean checking whether it has been endorsed.
- Boolean checking whether it has been confirmed (by Buyer).
- Mapping from validator Ids to validators.

Implementation Logic

The methods in the smart Contract are,

start()

This initiates the system. To be called by the governing authority.

registerValidator()

Called by the accounts wanting to become validators.

- **Input**: Validator name and Validator government id proof
- **Checks**: Whether validator has registered previosuly
- **Emits**: `ValidatorValidationRequest`

registerCitizen()

Called by the accounts wanting to register as citizens to register, purchase or sell property.

- **Input**: Citizen name and citizen id proof
- **Checks**: Whether citizen has registered previosuly
- **Emits**: `CitizenValidationRequested`

approveValidator()

Only can be performed by governing authority. Called to authorise request for making an account a validator.

- **Input**: Validator Id
- **Checks**: Whether the caller is governing authority, Validator Id is valid and whether the Validator with the passed Validator Id has been validated.
- **Emits**: `ValidatorApproved`

endorseCitizen()

Only can be performed by validators. Called to register a citizen.

- **Input**: Citizen Id
- **Checks**: Whether the caller is an authorised validator, citizen Id is valid and whether the citizen with the passed citizen Id has been registered.
- **Emits**: `CitizenValidated`

registerProperty()

Only can be performed by a registered citizen. Called to register their property, which needs to be endorsed by atleast *k* validators.

- **Input**: Property address, Ownership proof
- **Checks**: Whether the caller is a registered citizen and whether sufficient fees have been paid or not.
- **Emits**: `RegisterPropertyRequest`

SellProperty()

Only can be performed by owner of the property. Called to sell their property, which needs to be confirmed by the buyer and endorsed by atleast *k* validators.

- **Input**: Property Id, Buyer Id, Transaction proof, Transaction price
- **Checks**: Whether the caller is the owner of the property, whether the property Id is valid, whether the buyer and seller are registered citizens and whether sufficient fees have been paid or not.
- **Emits**: `SellPropertyRequest`

ConfirmBuy()

Only can be performed by the buyer of a property. Called to to confirm the purchase, which needs to be endorsed by atleast *k* validators.

- **Input**: Transaction Id
- **Checks**: Whether the caller is the buyer of the property, whether transaction Id is valid, whether transaction has been confirmed previously, whether the transaction has been endorsed previously and whether the caller is a registered citizen.
- **Emits**: `BuyPropertyRequest`

endorseTransaction()

Only can be performed by validators. Called to endorse transactions.

- **Input**: Transaction Id
- **Checks**: Whether the caller is an authorised validator, whether transaction Id is valid, whether transaction has been confirmed previously and whether the transaction has been endorsed previously.
- **Emits**: `BuyPropertyRequest`

showAllPendingTransactions()

Returns array of all unendorsed and unconfirmed transactions.

- **Output**: An array of all unendorsed and unconfirmed transactions.

showPastOwnership()

Returns the ownership history of a property

- **Input**: Property Id
- **Output**: An array of all previous and current ownerships of the property.

How to Run?

In your terminal go to the submission directory `13/` and run the following command:

```
truffle develop
```

Then serially implement the following command:

```
// Initialisation
compile
migrate
let accounts = await web3.eth.getAccounts();
let instance = await LandRegistry.deployed();

// Set Government account:
instance.start({from: accounts[0]});

// Register Validators:
instance.registerValidator("Rutvij","Aadhar card",{from: accounts[1]});
instance.registerValidator("Snehal","Driver License",{from: accounts[2]});
instance.registerValidator("Tathagato","Pan Card",{from: accounts[3]});
// Duplicate registeration (invalid)
instance.registerValidator("Jack","valid id",{from: accounts[3]});
// Validator with invalid id (valid, but will not be approved)
instance.registerValidator("Mark","Invalid id",{from: accounts[4]});

// Approve Validators:
instance.approveValidator(1,{from: accounts[0]});
instance.approveValidator(2,{from: accounts[0]});
instance.approveValidator(3,{from: accounts[0]});
// Duplicate Approval
instance.approveValidator(3,{from: accounts[0]});
// Approval not coming from owner (invalid)
instance.approveValidator(4,{from: accounts[1]});
// Approving invalid id (invalid)
```

```
instance.approveValidator(5,{from: accounts[0]});

// Register Citizens:
instance.registerCitizen("Jay","Passport",{from: accounts[5]});
instance.registerCitizen("Ganesh","Driver License",{from: accounts[6]});
// Validator registering as citizen (invalid)
instance.registerCitizen("Tathagato","Pan Card",{from: accounts[3]});

// Endorse Citizens:
instance.endorseCitizen(1000001,{from: accounts[1]});
instance.endorseCitizen(1000002,{from: accounts[2]});

// Register Properties:
web3.eth.getBalance(accounts[5])
instance.registerProperty("address1","Electricity bill",{from: accounts[5],value: 25});
web3.eth.getBalance(accounts[5])
instance.registerProperty("address2","Electricity bill",{from: accounts[5],value: 25});
web3.eth.getBalance(accounts[5])
web3.eth.getBalance(accounts[6])
instance.registerProperty("address3","Electricity bill",{from: accounts[6],value: 30});
web3.eth.getBalance(accounts[6])
// Unendorsed citizen (invalid)
web3.eth.getBalance(accounts[7])
instance.registerProperty("address4","Electricity bill",{from: accounts[7],value: 25});
web3.eth.getBalance(accounts[7])
// Insufficient Fees (invalid)
web3.eth.getBalance(accounts[6])
instance.registerProperty("address5","Electricity bill",{from: accounts[6],value: 20});
web3.eth.getBalance(accounts[6])

// Endorse Registration Transaction:
instance.endorseTransaction(1,{from: accounts[1]});
instance.endorseTransaction(2,{from: accounts[1]});
instance.endorseTransaction(2,{from: accounts[2]});
instance.endorseTransaction(3,{from: accounts[2]});
instance.endorseTransaction(3,{from: accounts[3]});
instance.endorseTransaction(1,{from: accounts[3]});
// Endorsement by unregistered validator (invalid)
instance.endorseTransaction(1,{from: accounts[4]});
// Endorsing already endorsed transaction (invalid)
instance.endorseTransaction(1,{from: accounts[2]});

// Sell Property:
instance.SellProperty(1,1000002,"valid proof",10000,{from: accounts[5],value: 25});
instance.SellProperty(3,1000001,"valid proof",30000,{from: accounts[6],value: 30});
// Sale not by owner
instance.SellProperty(2,1000002,"valid proof",30000,{from: accounts[6],value: 30});

// Confirm Purchase:
instance.ConfirmBuy(4,{from: accounts[6]});
instance.ConfirmBuy(5,{from: accounts[5]});

// Endorse Sale Transaction:
instance.endorseTransaction(4,{from: accounts[2]});
instance.endorseTransaction(5,{from: accounts[2]});
instance.endorseTransaction(4,{from: accounts[3]});
instance.endorseTransaction(5,{from: accounts[1]});
```

```
// Get property history:
instance.showPastOwnership(1);
instance.showPastOwnership(2);
instance.showPastOwnership(3);
```

Or, check `test/test.txt` for the commands.
And the comments between these commands indicate the desired output you should see.