Land-Registry-Implementation

Dhanush GA

March 11, 2019

Contents

1 Implementation

The particular application has been developed using Blockchain technologies provided by Ethereum. The core of this application is handled by smart contracts. All the programming logic required to register users, lands and handle land transactions are written in smart contracts using the programming language called Solidity. This language is similar to javascript but helps us prevent critical errors like memory leaks, type errors etc. These smart contracts are compiled and then run on the Ethereum Virtual Machine (EVM). This Virtual Machine provides the resources for our smart contract to execute and its speciality is that it runs on the Ethereum Blockchain. Each state of a program is recorded as a transaction in the blockchain.

To test this application, we run it in our local network using tools like Ganache and Truffle. Ganache simulates a local blockchain in our test environment by hosting a local blockchain and providing a few accounts to perform transactions. Truffle is used to compile the written smart contract and migrate (deploy) the contract to a running instance of a blockchain network. In our test network, we provide the address as that of the localhost (//127.0.0.1) and the port as 8545 (we can set the port value manually). After setting up all this to deploy our contract in a blockchain network, we could interact with our smart contract using 'truffle console' command.

This command starts a console in which we could interact with our deployed contract and call the functions linked with our contract.

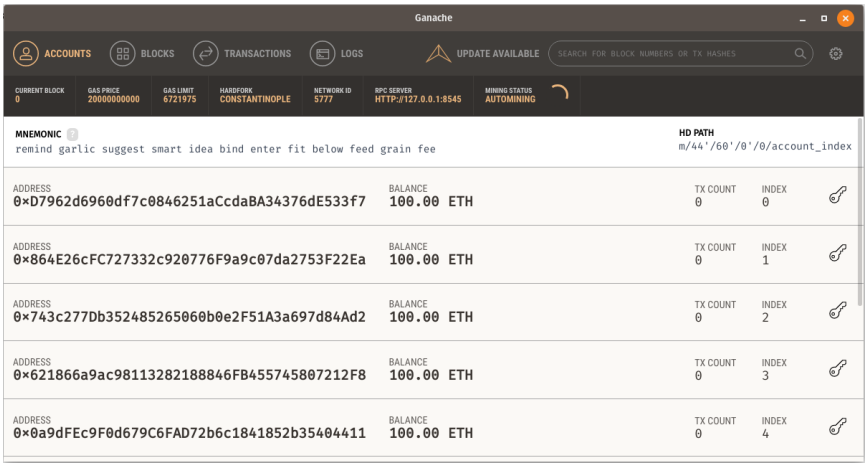

Figure 1: Ganache User Interface

1.1 User Registration

To register an user, we invoke registerUser() method. This function call receives user details and converts into a data structure holding user details. This data is then permanently stored in our blockchain.

Figure 2: registerUser() method demonstration

1.2 Land Registration

Similar to user registration, we invoke registerLand() method to resgister a land. This method can only be used by privileged users (admin and regsis-

trar). This data is converted to a data structure holding land details and then permanently stored in our blockchain.

Figure 3: registerLand() method demonstration

1.3 Land Transaction

Land Transaction is carried out in this system only when the land is allowed to be sold. So, we include a method called chLandStatus() to allow the land owner to sell the land.

If land is on sale, then invoking transactLand() method transfers the land from the land owner to the person invoking the method. Balance is deducted from the buyer's account and added to the seller's account. The land again changes to owned state.

$\alpha \to \beta$

Figure 4: Land stored in Blockchain

Figure 5: chLandStatus() method demonstration

```
truffle(development)> app.transactLand(1001, 20, {from: acc[2]})
{ tx: '0x3a8ad755b272ea7b2ee99310ab1b42ba7ce4f7afc1d7daba374190a6a8f3a62a',
  receipt:
   { transactionHash: '0x3a8ad755b272ea7b2ee99310ab1b42ba7ce4f7afc1d7daba374190a6a8f3a62a',
     transactionIndex: 0,
     blockHash: '0x35c242eff719ef082cb2d6defa107a0d47c3bf1765db961703c144d9652245c0',
     blockNumber: 16,
     from: '0x743c277db352485265060b0e2f51a3a697d84ad2',
     to: '0x255ff0cce5e50c5a483bb063c755650463a3d946',
     gasUsed: 100082,
     cumulativeGasUsed: 100082,
     contractAddress: null,
     logs: [],
     status: true,
     logsBloom: '0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000',
     v: '0x1b',
     r: '0xb46738fe5529459411b23cf5b0cfb00f1df28908d3eb84cf4ae6e84f1e58d242',
     s: '0x41b9a0e9c1a9d1a9914023c03aa6c0f844d54a4c8302e617f0b804f7c992d555',
     rawLogs: [] },
  logs: [] }
```

Figure 6: transactLand() method demonstration

```
truffle(development)> app.Lands(1001)
Result {
  '0': <BN: 1>,
  '1': <BN: 3e9>,
  '2': 'Person 2',
  '3': '0x743c277Db352485265060b0e2F51A3a697d84Ad2',
  '4': '0xD7962d6960df7c0846251aCcdaBA34376dE533f7',
  '5': <BN: 0>,
  '6': '0x0000000000000000000000000000000000000000',
  landId: <BN: 1>,
  landTag: <BN: 3e9>,
  landOwnerName: 'Person 2',
  landOwnerAddress: '0x743c277Db352485265060b0e2F51A3a697d84Ad2',
  landOwnerValidator: '0xD7962d6960df7c0846251aCcdaBA34376dE533f7',
  landStatus: <BN: 0>,
  specifyLandTo: '0x0000000000000000000000000000000000000000' }
```

Figure 7: Land details after transaction

```
truffle(development)> app.Users(acc[1])
Result {
  '0': <BN: 1>,
  '1': 'Person 1',
  '2': '0x864E26cFC727332c920776F9a9c07da2753F22Ea',
  '3': <BN: 78>,
  '4': <BN: 2>,
  userId: <BN: 1>,
  userName: 'Person 1',
  userAddress: '0x864E26cFC727332c920776F9a9c07da2753F22Ea',
  userBalance: <BN: 78>,
  userPrivilege: <BN: 2> }
truffle(development)> app.Users(acc[2])
Result {
  '0': <BN: 2>,
  '1': 'Person 2',
  '2': '0x743c277Db352485265060b0e2F51A3a697d84Ad2',
  '3': <BN: 50>,
  '4': <BN: 2>,
  userId: <BN: 2>,
  userName: 'Person 2',
  userAddress: '0x743c277Db352485265060b0e2F51A3a697d84Ad2',
  userBalance: <BN: 50>,
  userPrivilege: <BN: 2> }
```

Figure 8: Updated user account details