Week-03

Develop a program for the following using block chain concepts.

A. Create multiple nodes and manage using socket pgm

B. Manage block chain to maintain patient data across hospital(where dr access) access through once node. Whr. A new patient visit and hospital , create genesis block.

C.when dr write , do blood test. Patient goes to diagnostic center, they access patient data through block chain.(Dr created alrdy). Then they do blood test , that data you add in the next block. So dr.will get incentive .update his balance of e-cash. Update tx in ledger .

D.Then the patient goes back to dr to show blood test report. That time dr access the data created by diagnostic center, so now diagnostic center get the incentive in terms of e-cash . After seeing report, dr write prescription. For this new block to be added in block chain.

E. Then the patient goes to pharmacy , they access the prescription from block chain, that time dr get incentive. Then patient buy medicines.

All the people mentioned above will access by logging in to system. Once node can be used for authentication by storing username,pwd.

Part -2

This lab will focus on one of the most important properties – **Ethereum smart contract**.
In order to compile and successfully deploy smart contract, there are three things we need to fully understand:
Solidity (high level contract language),
Remix IDE (Integrated Development Environment) and
smart contract itself.

- ➢ First of all, we need to understand the differences between a **paper contract** and a **smart contract** and the reason why smart contracts become increasingly popular and important in recent years.
  - A contract, by definition, is a written or spoken (mostly written) law-enforced agreement containing the rights and duties of the parties.
    - Because most of business contracts are complicated and tricky, the parties need to hire professional agents or lawyers for protecting their own rights.
    - However, if we hire those professionals every time we sign contracts, it is going to be extremely costly and inefficient.
  - **Smart contracts** perfectly solve this by working on 'If-Then' principle and also as escrow services.

  - All participants need to put their money, ownership right or other tradable assets into smart contracts before any successful transaction.
  - As long as all participating parties meet the requirement, smart contracts will simultaneously distribute stored assets to recipients and the distribution process will be witnessed and verified by the nodes on Ethereum network.

- **There are a couple of languages we can use to program smart contract.**
  Solidity, an object-oriented and high-level language, is by far the most famous and well maintained one.
  - We can use Solidity to create various smart contracts which can be used in different scenarios, including voting, blind auctions and safe remote purchase.
  - In this lab, we will discuss the semantics and syntax of Solidity with specific explanation, examples and practices.
  - If you want to find more information about Solidity, please check its official website at: **solidity.readthedocs.io.**

- After deciding the coding language, we need to pick an appropriate compiler.
  - Among various compilers like **Visual Code Studio**, we will use Remix IDE in this and following labs because it can be directly accessed from browser where we can test, debug and deploy smart contracts without any installation. Remix can be reached at its official website: **http://remix.ethereum.org/.**

2. Smart Contract with Different Blockchain Networks

2.1 Introduction

In this section, we are going to deploy a simple smart contract and interact with different blockchain nodes. But before doing these, we need to activate some plugins in order to make Remix more manageable. We need to click ‘Plugin Manager’ at the left side of Remix interface and make sure that ‘Deploy & Run Transactions’ and ‘Solidity Complier’ are in ‘Active Modules’(we don’t need to manage other plugins at this moment, just keep them at default settings).

2.2 First Smart Contract

By clicking ‘File Explorers’ at the left side of Remix interface, we can see the list of smart contracts (or empty list) under ‘Browser’. Clicking ‘+’ next to ‘Browser’, we will start to compile our first contract by entering its name ‘MyFirstContract.sol’ (all solidity files need to add ‘.sol’ in the end of the file name).
Before compiling smart contracts, we need to choose the right version of compiler. Full list of Solidity versions displays under ‘Compiler’ in ‘Solidity Compiler’ tab. We are going to switch to any version starting with ‘0.6.x’ (we mainly use 0.6.10+commit.00c0fcaf in this and the following labs). Because Solidity made some breaking changes in every big version update, if we use version 0.7.x, 0.5.x, 0.4.x or even lower versions, Remix might show some errors on our codes. The sample of our first smart contract ‘MyFirstContract’ is in ‘LAB 2 Assignment.txt’. You need to copy and paste this contract into

Solidity file which is created earlier. The lab also provides the tutorial video 'My First Contract.mov'. You need to check if the deployment process of your contract is similar with the video.

2.3 JavaScript VM, Injected Web3 and Web3 Provider

In 'Deploy & Run Transaction', we are able to choose three different environments: JavaScript VM, Injected Web3 and Web3 Provider. When we use Injected Web3 (before using it, please make sure that you have followed the instructions in LAB 1 and installed MetaMask), Remix will automatically connect to Ethereum wallet - MetaMask. It means that any cost, including transaction fee and gas, will reduce Ether in MetaMask account.
Although Injected Web3 is the most similar environment to the real-world transaction, this environment does not suit for developers like us because we need to wait every time we deploy a smart contract. To save time for testing and debugging, we want an environment where we can get the result right after clicking 'Deploy' button. JavaScript VM (Virtual Machine) is a relatively better execution environment that simulates blockchain in memory of the browser (be careful that reloading or closing website might default Remix to its initialized settings). When we deploy a smart contract in JavaScript VM, the environment will immediately give us a feedback.

Another fast way to deploy smart contract is via Web3 Provider, which represents the external application for blockchain node. In this and following labs, we will use Ganache which can be downloaded at **https://www.trufflesuite.com/ganache** (you may also use other external blockchain nodes). By clicking 'QUICKSTART ETHEREUM' at its initialized interface, we will see ten Ethereum testing accounts with 100 testing ether each. If we switch the environment to Web3 Provider in Remix, Remix will pop out a 'External node request'. We do not need to make change about the settings except for 'Web3 Provider Endpoint'. The last four number (8545) of endpoint should be replaced by the last four numbers (7545) of 'RPC SERVER' in Ganache (new 'Web3 Provider Endpoint' becomes http://localhost:7545). As Remix connects to Ganache, the list of accounts in Remix will automatically switches to the list of ten accounts in Ganache; consequently, any transaction in Remix will immediately be shown in Ganache with details. Ganache not only shows what is going on behind the transaction, but also creates a private network which is important if we want to create a Java Script or HTML application with an actual user interface to connect with.

3. Solidity Language Description

3.1 Introduction

In this section, we are going to discuss the structure of different smart contracts and the type of variables, units and expressions. Some of them are similar with other programming language, like Java and Python, some of them are totally different. By the end of this lab, you should be able to compile an integrated smart contract. In order to have a better understanding about the smart contract and its properties, this lab will not only provide detailed explanations about each property,but also offer some practices in each following subsection.

3.2 Variables: Integers, Booleans, Address, Balance and String

Solidity has detailed explanations about Integers, Booleans, Address, Balance and Strings. You can find them in following websites:

https://solidity.readthedocs.io/en/v0.6.10/types.html#integers
https://solidity.readthedocs.io/en/v0.6.10/types.html#booleans
https://solidity.readthedocs.io/en/v0.6.10/types.html#address
https://solidity.readthedocs.io/en/v0.6.10/types.html#members-of-addresses
https://solidity.readthedocs.io/en/v0.6.10/types.html#bytes-and-strings-as-arrays

Here are some points we need to pay attention to. First of all, all variables in Solidity are initialized by default. For example, (u)int = 0, bool = false and string = ‘’. Unlike other programming languages, strings in Solidity are special arrays. A string does not have a length or index-access. Because using strings costs huge amount of gas, Solidity has very few functions to do any string manipulation, which means that we should avoid using strings as much as we can.

Then, as we can see in contract ‘MyFirstContract’, ‘public’ state variable automatically has a getter function with the name of the variable. If we run the contract and click that ‘public’ variable, the system will show the value stored in this variable.

3.3 Address and Global Msg-Object

The following websites give specific interpretation about the members of address types and the properties of block and transaction. In address types, you need to focus on how to get the balance of an address and how to transfer ether from one account to another.
Remix has special variables and properties for provid information about the blockchain, functions and transaction. In this subsection, you need to understand how to use ’msg-object’, especially ‘msg.sender’ and ‘msg.value’.

https://solidity.readthedocs.io/en/v0.6.10/units-and-global-variables.html#members-of-address-types

https://solidity.readthedocs.io/en/v0.6.10/units-and-global-variables.html#block-and-transaction properties