



Study of Decentralized applications in Ethereum Block Chain and Smart Contracts

Name: Krishna Chaitanya Kancharla
ICS 690-01: Blockchain Technology
Spring 2022
Submitted: April. 19, 2021
Instructor: Dr. Calcaterra, Craig

1. Introduction:

Blockchain is just a distributed ledger made up of blocks. Each block is chronologically linked to the previous block. Each block in the chain has a set of transactions, each person on the network is validated, and the entire process is open, transparent, and irrevocable. Extensive applications of blockchain exist in the cryptocurrency domain. Every operation on the block chain network is transparent to every member of the network. The irreversible nature of the currency exchange leaves a marginal room for fraudulent activity.

An important aspect of blockchain technology is that it is designed to store information in such a way that it is impossible to change, delete, or add information without being discovered by other users. This will determine the origin and authenticity of the transaction and thereby increase overall transparency and trust when it is linked to a particular product. One of the features of blockchain is that the blockchain network can accept its own rules on the network without a third party. Because transactions and ledgers are encrypted, blockchain technology offers greater security than the banking model, and its instantaneous transmission over the Internet eliminates the banks' two to three-day clearing process and the costs of transferring money from one account to another. The term "blockchain" derives from the "blocks" of verified and unchanged transactions and how they converge chronologically to form a chain. Hence the term "blockchain."

2. Ethereum Smart Contracts:

Smart contract is nothing but a simple program that can be written in Solidity, Vyper, Rust so on. Smart contracts contain code written in functions and data which are stored in blockchain at a specific address. When conditions are met, smart contracts are deployed to the network and executes automatically, and transactions are made over network but are not controlled by any single user. Once deployed users can then interact with the smart contract by providing transactions which are executed as functions that are defined in the contract via a code.

Bitcoin is designed for virtual currency scenarios and is not universal since it is not Turing-Complete. As such, many other blockchain-based systems have emerged and have allowed for a variety of applications in the form of "smart contracts" on the blockchain. Ethereum first realized the full suite of blockchain and smart contracts [1]. Once a smart contract is deployed on the blockchain, it will be executed according to predefined terms, and no one can change it. Smart Contract in Ethereum is written as a stock-based low-level byte code, operated by Ethereum Virtual Machine (EVM) and is called EVM code. These codes are stored in blockchain in a binary format specific to Ethereum. Smart contracts can also be written in high-level languages such as Solidity [2] and then compiled into EVM code.

The core of Ethereum is the Ethereum Virtual Machine, which can execute code with arbitrary algorithmic complexity. In computer parlance: Ethereum is Turing complete. Developers can create contracts and a variety of DApps on the Ethereum virtual machine based on existing programming languages (such as Java and Python). Smart contracts can handle a variety of operational logic, make full use of Ethereum's blockchain capabilities, make the blockchain more scalable, and inspire Ethereum to develop into the most comprehensive blockchain development platform.

3. CryptoZombies Learnings:

Since I am new to Blockchain and have no knowledge on how blockchain and solidity code works as a part of learning I started working on blockchain courses on cryptozombies.io. As I progress through the course work Solidity Path module helped me a lot on how smart contracts works.

Git commits for the six modules completed on building ZombieFactory game.

<https://github.com/krishna-github-metrostate/cryptozombies-lesson-code>

- **VersionPragma:** Every contract should always start with version; this is used to protect future compiler issues when changes are made.
E.g:
- **State Variables and Integers:** These variables are saved on contract storage, meaning they are written to the blockchain.
uint256 – a 256-bit unsigned integer.
- **Struct:** can be used to create complicated datatype, with various properties.
- **Array:** Collection of data points. Can be a fixed or dynamic, same as state variables these can be stores in contracts. Public array or private array.
- **Function Declaration:** By value or by Reference, by value compiler creates a copy of parameters and pass it to the function whereas by reference, parameters are referenced. Public (anyone can call it) or private (suggested - default) functions.
- **Events:** To communicate to blockchain to app front-end and can take actions.
- **Addresses:** Unique identifier for each account. Blockchain is made up of accounts. Each address is associated to a specific account.
- **Import:** if sometime codes can get long when we write struc, functions and event to make them manageable we can split the file into multiple and use import function to call into the other contracts.
- **Storage and Memory:** Storage is permanent whereas Memory is temporary storage.

Started with a simple zombie and increased the complexity by building multiple zombies in two different smart contracts. To make it secure, additional security features are added by making functions public and private so only a specific individual can access the game. Openzeppelin is also used for added security. Random number functions are generated to build more zombies and features for other players to join. Standardizes the process by using ERC271 tokens using the default libraries.

4. Deploying smart contract with Truffle and Ganache CLI:

4.1 Truffle: Used to create solidity templates to deploy. It's a testing framework which gives the developers the ability to spin up a smart contract with click of a button.

4.2 Ganache-CLI: It's a simulation environment where full client behavior can be created resulting in faster, easier, and safe Ethereum based applications. It will allow us to create our own private Ethereum blockchain. It gives all the abilities to perform actions same as main blockchain but without any cost. Many developers as their testing ground.

4.3 Prerequisites:

Operating Systems

Software: npm. npm must be installed from the root folder of the application.

```
$ sudo apt-get install nodejs
$ sudo apt-get install npm
$ sudo apt-get install npm
$ npm -v (to check the version)
or
#npm install
```

4.4 Installing Truffle and Ganache CLI:

First, we have to make sure atleast V5.0 of npm has been installed then run `npm install -g ganache-cli truffle` (this is installed globally so we need not to be worried about installation location)

4.5 Run Ganache CLI:

We can now start to develop our own smart contracts. To run locally we need to create a simulated environment

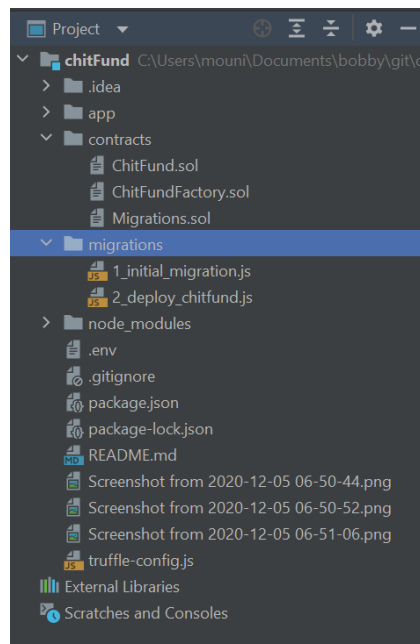
```
$ ganache-cli
```

After running the above command ganache will create 10 accounts with private keys with test ethers (100). Since ganache is all in memory if we restart everything will be refreshed back to normal.

4.6 Build a Chit fund smart Contract:

/contracts: Directory path for smart contracts. Here is where we will save out 3 chit fund solidity contract files.

/migrations: used to deploy the smart contract in the contracts folder.

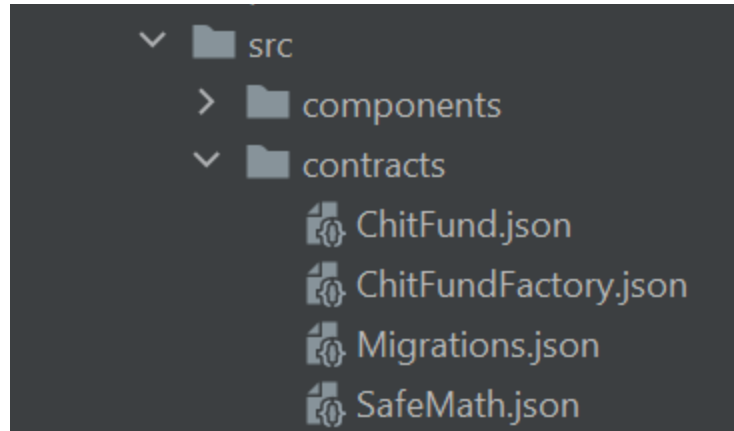


/node_modules: here is where we test codes for smart contracts that we built.

/truffle-config.js: no need to touch it, it's just a configuration file.

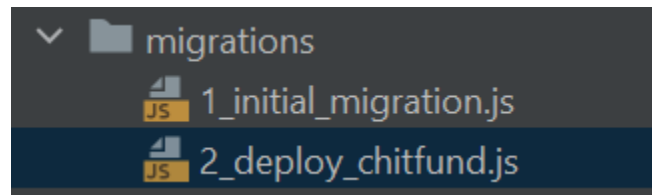
4.7 Compiling the Contracts:

Contracts can be compiled with ease using `$ truffle compile` when we execute this command what it does at the backend is it converts the original code into blockchain bytecode. Once compiling is completed without issues there will be json files created under `app/src/contracts`



4.8 Deploying the Contracts:

Under the migrations directory create a json file “2_deploy_chitfun.js” and enter below code.



```
const ChitFund = artifacts.require("ChitFund");
const ChitFundFactory = artifacts.require("ChitFundFactory");

module.exports = function (deployer) {
  deployer.deploy(ChitFund, "KrishnaChitFund", 1, 3, 3);
  deployer.deploy(ChitFundFactory);
};
```

Modify the `truffle-config.js` to include the local host address (which connects to ganache) and run `$ truffle migrate`. Once you run it and navigate to `ganache-cli` we can see logs running in the terminal.

5. Installing IntelliJ IDEA:

For our app to run it mandatory to have install nodejs and npm. To overcome additional installation, we used IntelliJ IDEA which comes with default nodejs. Clone the latest application code from github into your local machine and go to the root directory which is `/app` and install npm using `npm install`. You can use `npm start` to run the the app, if error are observed before installing npm we should trigger `npm i --package-lock-only`, `npm audit fix` and `npm cache clean --force`

6. Smart Contract Design for Chit fund:

A smart contract is defined as a self-executing contract which includes the agreement between Member and underwriter being directly written into lines of code. The code and the agreements together make the distributed, decentralized blockchain network. The code helps in controlling the execution, and transactions can be trackable and irreversible.

We have 3 different smart contracts to run the chitfund app `ChitFund.sol`, `ChitFundFactory.sol` and `Migrations.sol`. For our project we are using version `pragma solidity >=0.40.21 <0.7.0`

`ChitFunFactory.sol` is the heart of the code designed in solidity version `pragma solidity ^0.6.8` which contains the import of `ChitFund.sol` smart contract. State variables and integers are designed and permanently stored in this contract which are written into blockchain. Function `createFund` with state variables `uint256` amount, installments and participants are defined in this smart contract with a requirement of `>0` and fund are deployed using this smart contract.

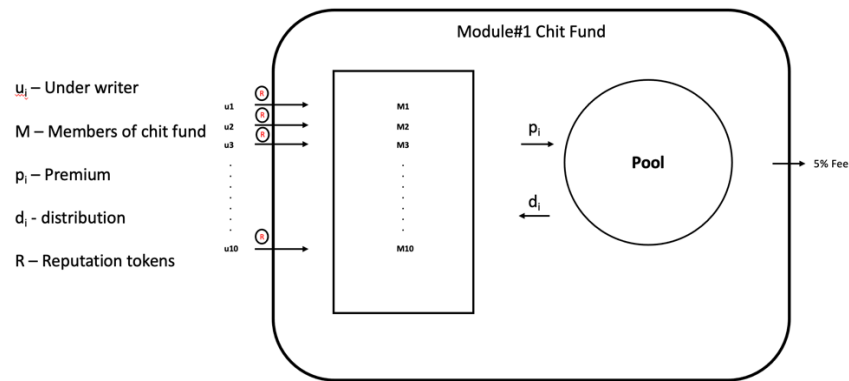
In `ChitFund.sol` used preinstalled `openzeppelin` to call `SafeMath.sol` which is used to wrap over arithmetic operations. And contract `ChitFund` is created using state variables `uint256` with public function `fundName`, `jackpot`, `numOfInstallments`, `noOfInvestors`, `manager` and `installmentAmount`. All the critical parameters/variables such as `noOf InvestorsJoined`, `fundBalance`, `currentRound` are designed. Structure of the investor is also designed to get information about `hasWonRound`, `hasJoined` and `isReadTOinvest`.

All the testing parameters are also represented in this code using function `contribute()` that checks if the member has already Invested or contributed or already paid all the installments. Function `bidForJackpot()` is create that tell or define who is the jackpot winner and finally `releaseFund()` function is created to release the fund to the winner.

7. Design of Chit fund in Block Chain

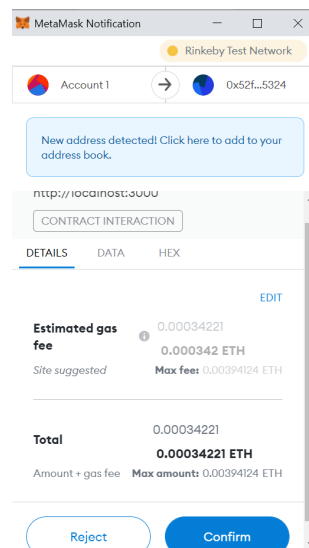
7.1 How Chit fund in Block Chain works:

- As a start a new chitfund is created is Ethereum block chain.
- Once the creation is completed users are asked to join the fund.
- To allow users into the chitfund underwriter chooses a member and to underwrite and grant entry.
 - Underwriter has reputation token, where they can use those reputation token to underwrite users for each cycle of the fund.
- Users run through the chit fund process as expected, funds and fees paid round completed.
- Member are dropped either due to fraud activities or insufficient funds, resulting underwriter take over the process and takes the distribution or vice versa where member takes the distribution and defaults, resulting in loss for the member underwriter.



7.2 Who can join DAO: Members with a valid wallet ID and user credentials will be able to join.

7.3 MetaMask: MetaMask is designed to access Ethereum based applications. It gives a secure login, token wallet and token exchange that is needed to manage DAO. It generates passwords and keys so that user has complete access over the account using the passphrase.



7.4 Rinkeby Test Ether:

Test Ethereum: To test the decentralized application we used Rinkeby Faucet tool for testnet Ether (ETH) before go live. Rinkeby is used as its completely free and easy to use.

Testnet Tokens: These are test currencies which will be used to test/troubleshoot our DAO.

How it works: For every 24 hours without any authentication, we can request 0.1 Rinkeby ETH, but to increase the amount of ETH to 0.5 Rinkeby ETH we can login to Alchemy which is also free to use/login.


7.5 Infura:

For developers like infura allows developers like us to take the testing application and scale it to production with easy access to Ethereum.

.env: To initiate the app .env file must be configured with user specific mnemonic passphrase from the metamask where test Ether is stored.

```
MNEMONIC="horse goat mouse"  
RINKEBY_INFURA="https://rinkeby.infura.io/v3/6244d4354814c56f87159"
```

Name of the chit fund can be configured using the javascript file 2_deploy_chitfund.js file.



```
const ChitFund = artifacts.require("ChitFund");  
const ChitFundFactory = artifacts.require("ChitFundFactory");  
  
module.exports = function (deployer) {  
  deployer.deploy(ChitFund, "KrishnaChitFund", 1, 3, 3);  
  deployer.deploy(ChitFundFactory);  
}
```

8. Deployment of smart contracts in Chit fund Project:

Truffle [5] framework is used to for compiling and deploying the smart contracts. Designing the DApp and developing it becomes easier with Truffle framework. Truffle is a development tool for Ethereum solidity language. Solidity language is used for writing smart contracts and Atom is used as editor which is an open-source text editor. JavaScript library Web3.js is used to interact smart contracts with blockchain. In this project truffle default builder is used for importing the compiled contract artifacts automatically [6].

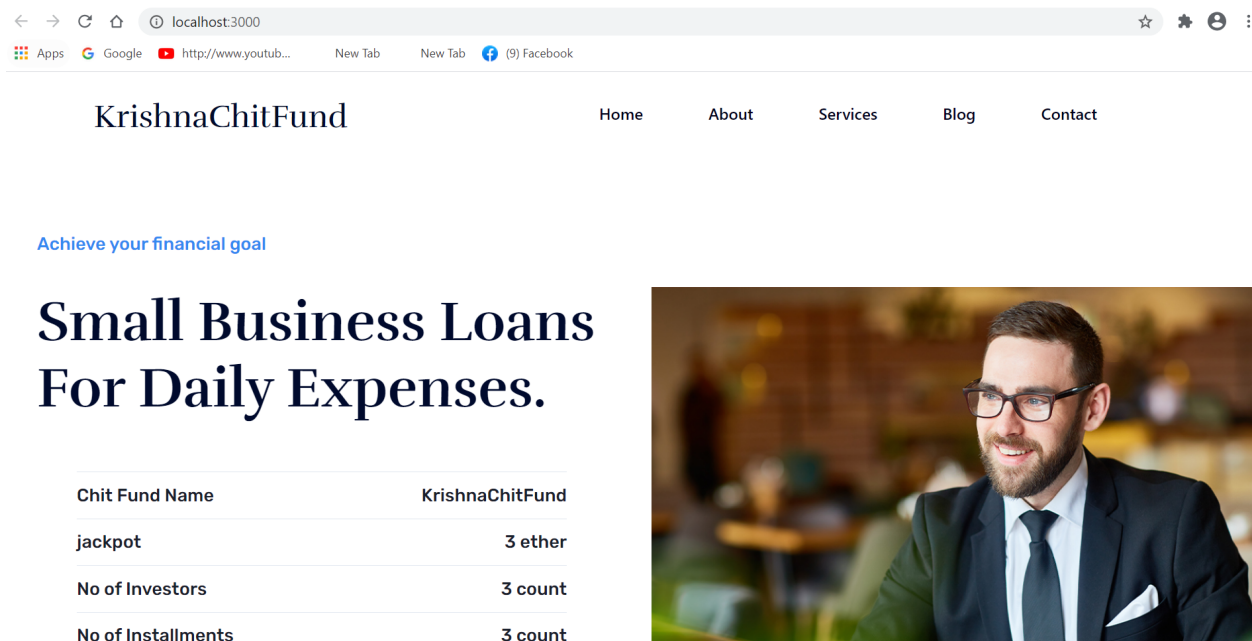
Decentralized applications work basically with the contracts and the webpages. Through webpages one can interact with the contracts. Webpage in our project developed using reactjs the interaction logic with webpage developed by using JavaScript code and connection is established. Write or read blockchain data and methods to interact logic is written in smart contract. Once the smart contract is deployed, with the help of webpage buttons one can interact with contracts. Firstly, with the help of truffle box create a project and create a solidity file. Write a logic in smart contract and logic to interact in JavaScript. Once the coding part is completed, Smart contract is ready to deploy to interact with blockchain.

Run the Chit fund app:

```
C:\Users\mouni>cd C:\Users\mouni\Documents\bobby\git\chitFund\app

C:\Users\mouni\Documents\bobby\git\chitFund\app>
C:\Users\mouni\Documents\bobby\git\chitFund\app>npm start

> app@0.1.0 start
> react-scripts start
```



9. Module 3 Writing contracts for the underwriter to underwrite members: (In Progress)

As a start a simple assumption is made i.e., 1:1 an underwriter will under a member.

Event 1: Event triggers when a member wants to join a chit fund to check if the member has any pre-build reputation tokens. If NO event 2. If YES join the chit fund.

Event 2: If a member has no preexisting reputation tokens member is in queue for the underwrite to choose and under write. Meaning underwriter will provide reputation tokens to join the fund.

Event 3: Underwriter verifies the userID for authentication and any fraud activates, if No event 4 if Yes event 5

Event 4: Underwriter underwrites for the member which will enable him to join the chit fund.

Event 5: Underwriter rejects the member request to join the chit fund.

InsuranceHelper Code Triggers when Underwriter wants to make decision on Approve or Reject:

When event 4 is trigger below event is called to check the history of the members.

Event 4.1: Checks if the members who is requesting to underwrite has history to make better decision on approving or rejecting the underwriting.

```
contract InsuranceHelper {

    struct ChitFund {
        string name;
    }

    ChitFund[] public chitFunds;

    mapping (uint => address) public chitFundToMember;
    mapping (address => uint) memberChitFundCount;

    function getChitFundsByMember(address _member) external view
returns(uint[] memory) {
        uint[] memory result = new
uint[] (memberChitFundCount[_member]);
        uint counter = 0;
        for (uint i = 0; i < member.length; i++) {
            if (chitFundToMember[i] == _member) {
                result[counter] = i;
                counter++;
            }
        }
        return result;
    }
}
```

10. Conclusion:

Through this course work I have learned and understood on how Block Chain works and DApps can be built by make better design considerations. Applying CryptoZombies.io learnings I can now code/design smart contracts using solidity. Also, I was able to pull the existing code on Chit fund DApp and was able to execute it using Metamask wallet, Rinkeby to collect text ether and truffle to compile and deploy the smart contract on Ethereum test block chain. As next steps I will be working to see how I can use these learning to implement Block chain in Retail Supply Chain Industry.

Reference

1. G. Wood. Ethereum: A Secure Decentralised Generalised Transaction Ledger. Yellow Paper. Accessed: Jun. 28, 2019. [Online]. Available: <https://ethereum.github.io/yellowpaper/paper.pdf>.
2. [26] G. Wood. Ethereum: A Secure Decentralised Generalised Transaction Ledger. Yellow Paper. Accessed: Jun. 28, 2019. [Online]. Available: <https://ethereum.github.io/yellowpaper/paper.pdf>.
3. S. Nakamoto. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. [Online]. Available: <https://bitcoin.org/bitcoin.pdf>
4. Johan Quist and Peter Magnusson, “Blockchain technology in food supply chains”, Karlstads University, Serial number: 1, July 2018.
5. TruffleSuite. Accessed: Jun.28,2019.[Online].Available:<https://www.trufflesuite.com/truffle>.
6. <https://github.com/trufflesuite/truffle-default-builder>
7. <https://rinkebyfaucet.com/>
8. <https://codebrahma.com/brief-intro-smart-contracts-endless-possibilities/>