The title of the report: Learnings on Ethereum - Decentralized Applications

Kiran Kumar Guntumadugu
ICS 690 – Blockchain Technology
Spring 2022
Due: Apr. 19, 2022
Instructor: Dr. Craig Calcaterra

Contents

Introduction 3
Background 3
CryptoZombies 4
ChitFund DApp 5
Conclusion 5
References 6
Appendix 6
    Cryptozombies 6
    Coding key takeaways 6
    ChitFund DApp 7
    ChitFund DApp Screenshots 7

Introduction

This paper is the outcome from the blockchain technology course in spring 2022. The content in this paper is based on the cryptozombies interactive learning, the ChitFund DApp project and the lecture classes. I would like to present my learning on blockchain technology and contributions made from myside.

Background

In the current era we have the centralized systems that are controlled, governed by a single central unit. Some of the limitations on the centralized systems are anonymity, single point of failure, vertical scalability. The decentralized systems concept evolved to overcome these limitations. Blockchain stores the data in distributed blocks, where each block is linked to previous block with a hash value using cryptography and forms a chain. The data cannot be tampered and ensures anonymity, if the previous block is modified the cryptography hash values are changed and breaks the blockchain.

Bitcoin is one of the popular blockchain used for digital currency exchange. Whereas Ethereum blockchain is developed to execute programmable code on a decentralized virtual machine. The code executed on Ethereum are called smart contracts. Ethereum infrastructure also enables to build decentralized applications with anonymity, transparency, middleman charges and scalability. Ethereum uses ether (ETH) as the digital currency unit.

Ethereum platform is also used for digital art and collectibles that are tokenized and termed as non-fungible tokens. There are several DApps that are built for decentralized finance systems, gaming, and technology. I would like to share more the gaming DApp that I have learned using cryptozombies learning tool. The chitfund DApp is taken as the project for the blockchain course, and I was able to successfully run the application in my local machine with few contributions which I will discuss later.

Openzeppelin is a library of secure and community-vetted smart contracts that can be used in the DApps. The library consists of tradable tokens based on ERC20 and ERC721 standards, role-based permissions, reusable solidity contracts can use in your own DApps. Openzeppelin defender helps to scale automatically and monitor the DApp.

CryptoZombies

Cryptozombies is an interactive learning tool to build a decentralized gaming application. Cryptozombies have courses on Solidity, smart contracts, blockchain infrastructure and deployment, Kubernetes and Golang.

I have completed the first 6 lessons and would like to share how each lesson will help to build a decentralized application.

At first, I have started with the basics of Solidity programming and created a smart contract in Ethereum. I have created a level 1 zombie. In the next step, to increase the complexity of the game worked on a multi-player that has two smart contracts. Zombiefactory contract creates a new zombie and zombiefeeding contract feeds a cyrptokitty and generated a new DNA hash value. The two contracts are integrated to talk with each other. This step will create a new cryptozombie when a zombie if fed with a cryptokitty.

To make progress in the game, cryptozombie army if built and levels are added as the game progresses. Security feature is embedded to make sure only the owner of the game must access to their game and no one else can change the levels or feed kitty. I have also used openzeepelin library smart contract ownable to add the security feature.

In the next lesson, further enhanced the game to generate random numbers dynamically for cryptozombie creation and added payable functions to withdraw money from owner and added fees for changing the cryptozombie levels.

Then, I have added a contract to create the ERC721 standard tokens using Solidity libraries. Updated the security features to prevent the uint overflows and underflows using safemath library. Also, embedded the comments in the code using Ethereum Natural Language Specification Format (NATSPEC) standard.

At last, added the front-end coding for the smart contracts using web3.js. JavaScript is used as an intermediate layer between the smart contracts and front end. Event listeners are added to the JavaScript to listen smart contract events and show users real-time updates in the UI.

GitHub code and the cryptozombies learning shareables that are completed by me are provided under cryptozombies section in Appendix.

I have added key coding takeaways that caught my attention during the cryptozombies learnings under coding key takeaways section in Appendix.

ChitFund DApp

ChitFund DApp is a group project idea for the Maghribi-Foundation from the blockchain class in spring 2022. The first objective of the project is to identify and select if there is any reusable code that can be used as a baseline code. I have performed research and found couple of GitHub repositories that has chitfund project code. I have voted to the chitfund DApp that was selected by the class.

As a next step, I have cloned the chitfund DApp into my local machine. Metamask wallet and Infura project is required if we need to deploy the application in a network chain. So, I have setup the Metamask wallet, Infura project and executed the DApp using node v16.13. Informed to the class that the application works with node v16.13. I have understood the code and identified the member limitations in the chitfund DApp. Updated the number of members and contributions in deploy_chitfund.js script to overcome the exception error in local. I have setup Infura project, Metamask and created my own version of the chitfund DApp in local by following the guidelines.

Apart from these contributions, I have worked on the Insurance smart contract which has two compilation issues by referring to [5]. The approach considered to develop the insurance contract is below:

- Call in an event when a new member joined to verify the if the member has underwriter or not.
    - If Yes - then approve and have the member joined
    - If No - invoke the underwriter function and approve or decline the member
- How to identify if the member has underwriting or not.
- Create separate functions to verify the underwriters, assign an underwriter and decision making

I was able to successfully run the chitfund DApp in local machine by setting up Infura, Metamask, node, truffle and modified the deploy chitfund with my own version name.

GitHub code, slack conversations and Insurance contract links are provided under ChitFund DApp section in Appendix.

Included the screenshots to highlight what has completed from myside under ChitFund DApp Screenshots section in Appendix.

Conclusion

There are several decentralized applications in the market which are not fully evolved or in initial phase. I have gained better understanding the key design factors like governance, anonymity, security to consider in building DApps. Able to code smart contracts using solidity, understand how Infura, Metamask, truffle integrations are required for deploying the DApp by working on cryptozombies and chitfund DApps. I would like to continue developing the chitfund DApp for my own interest and add the governance aspect to the DApp.

References

[1] IEEE Xplore Full-Text PDF:

[2] #1 Solidity Tutorial & Ethereum Blockchain Programming Course | CryptoZombies

[3] samarth30/chitFund: chitFund - built using smart contracts , web3 , reactjs (github.com)

[4] OpenZeppelin/openzeppelin-contracts: OpenZeppelin Contracts is a library for secure smart contract development. (github.com)

[5] ProvidentOne/contracts: Solidity contracts that run Provident One insurance fund (github.com)

[6] 11 Best Blockchain Quotes: Know this Amazing Technology (techpikk.com)

Appendix

Cryptozombies:

- GitHub Code:
  - go2kiran/cryptozombies-lesson-code: cryptozomebie lesson code (github.com)
  - go2kiran/cryptoZombiesDApp: DApp for cryptoZombies Game (github.com)
- CryptoZombies learning shareable links:
  - https://share.cryptozombies.io/en/lesson/1/share/Kumar?id=Y3p8MTk4MjUw
  - https://share.cryptozombies.io/en/lesson/2/share/Kumar?id=Y3p8MTk4MjUw
  - https://share.cryptozombies.io/en/lesson/3/share/Kumar?id=Y3p8MTk4MjUw
  - https://share.cryptozombies.io/en/lesson/4/share/Kiran?id=WyJjenwxOTgyNTAiLDIsMTRd
  - https://share.cryptozombies.io/en/lesson/5/share/H4XF13LD_MORRIS_%F0%9F%92%AF%F0%9F%92%AF%F0%9F%98%8E%F0%9F%92%AF%F0%9F%92%AF?id=Y3p8MTk4MjUw
  - https://share.cryptozombies.io/en/lesson/6/share/The_Phantom_of_Web3?id=Y3p8MTk4MjUw

Coding key takeaways:

- Solidity reserves 256 bits of storage regardless of the uint size. But exception in structs, where uint16 or uint32 etc. can be defined to save unwanted storage.
- msg.sender is a global function, which gives the address of the user.
- Ethereum uses keccak256 built in SHA3 hash function and maps an input into a random 256-bit hexadecimal number. keccak256 expects a single parameter of type bytes, A slight change in the input will cause a substantial change in the hash.
- Types of modifiers:
  - Visibility modifier – control the function calling mechanism
    - private, external, public, internal
  - State modifier – used for defining the DApp interaction with blockchain

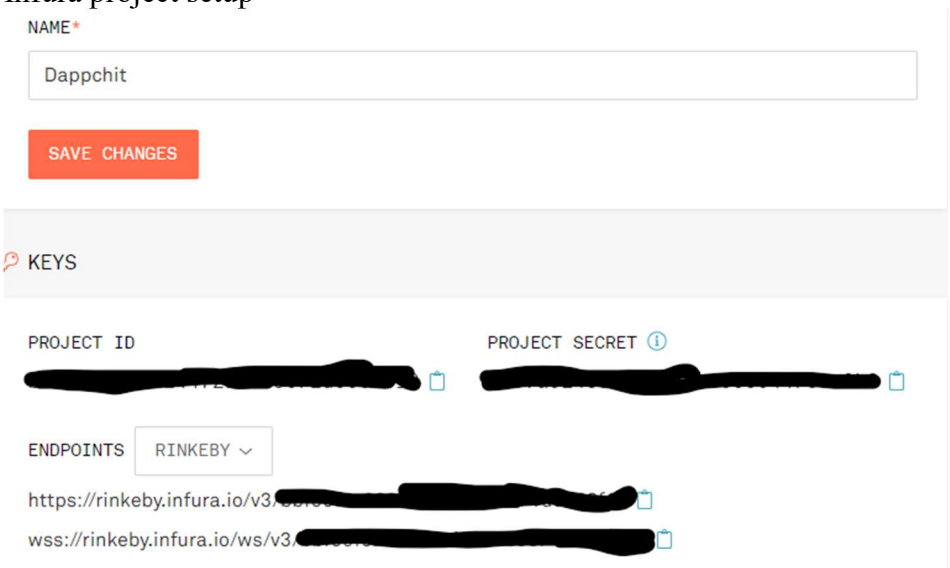
        - view - only returns the data
        - pure - does not access the app, just gives the output
    - Custom modifier – user defined modifier with custom logic
    - Payable modifier – used to send money from one address to another
- Every action in blockchain must pay Gas to avoid infinite loop in the blockchain and make sure there is a minimum gas paid for every action. View actions are free or very minimal gas cost.
- Once a DApp is deployed in the blockchain, it is immutable and cannot change. We must inform the users to use a newer version if there is any bug in the code.
- Solidity libraries will have reusable smart contracts which can be accessed in building decentralized applications.

ChitFund DApp

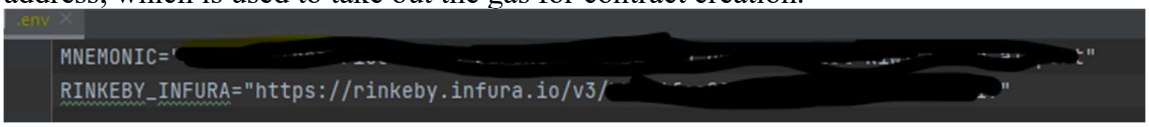
- GitHub Code: chitFund/Insurance.sol at kiranv1 · Maghribi-Foundation/chitFund (github.com)
- Slack Chat:
    - https://ics690blockch-32t4074.slack.com/archives/C031796HCGY/p1643593887095339
    - https://ics690blockch-32t4074.slack.com/archives/C031796HCGY/p1643638769053649?thread_ts=1643638114.605259&cid=C031796HCGY
    - https://ics690blockch-32t4074.slack.com/archives/C031796HCGY/p1643912435123349?thread_ts=1643899143.458999&cid=C031796HCGY
- Sample chit fund projects - Google Docs

ChitFund DApp Screenshots

- Infura project setup



- Added an env file to establish the network chain to deploy the DApp and Metamask address, which is used to take out the gas for contract creation.



- Compiled the DApp using truffle

```
C:\Users\chand\OneDrive\Desktop\MS_Spring2022\ICS690\GITDAPP\chitFund>npx truffle compile

Compiling your contracts...
===========================
> Compiling @openzeppelin\contracts\access\AccessControl.sol
> Compiling @openzeppelin\contracts\math\SafeMath.sol
> Compiling @openzeppelin\contracts\presets\ERC20PresetMinterPauser.sol
> Compiling @openzeppelin\contracts\token\ERC20\ERC20.sol
> Compiling @openzeppelin\contracts\token\ERC20\ERC20Burnable.sol
> Compiling @openzeppelin\contracts\token\ERC20\ERC20Pausable.sol
> Compiling @openzeppelin\contracts\token\ERC20\IERC20.sol
> Compiling @openzeppelin\contracts\utils\Address.sol
> Compiling @openzeppelin\contracts\utils\Context.sol
> Compiling @openzeppelin\contracts\utils\EnumerableSet.sol
> Compiling @openzeppelin\contracts\utils\Pausable.sol
> Compiling .\contracts\Capitalization.sol
> Compiling .\contracts\ChitFund.sol
> Compiling .\contracts\ChitFundFactory.sol
> Compiling .\contracts\Migrations.sol
> Compiling .\contracts\RepToken.sol
> Artifacts written to C:\Users\chand\OneDrive\Desktop\MS_Spring2022\ICS690\GITDAPP\chitFund\app\src\contracts
> Compiled successfully using:
   - solc: 0.6.8+commit.0bbfe453.Emscripten.clang
```

- Deployed the DApp using truffle into rinkeby network

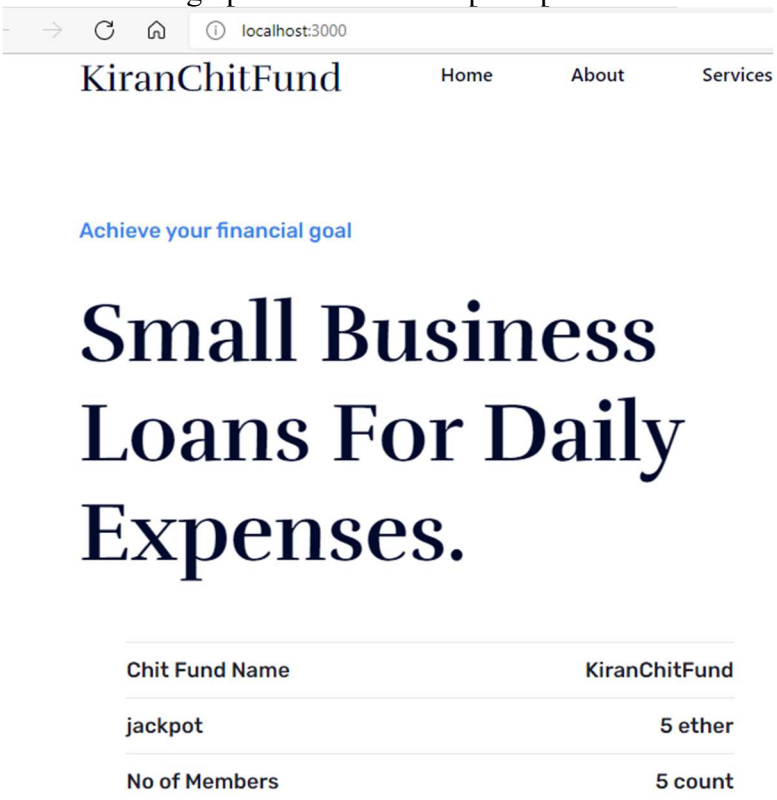
```
   Deploying 'ChitFund'
   --------------------
   > transaction hash:    0xc85875676a66747a43d603dbccb9fa4af7b45915da2936211bad4dfcfa297c36
   > Blocks: 1            Seconds: 12
   > contract address:    0xCdAee16646687F6D47899CFC1E5618f036AFDFB6
   > block number:        10531806
   > block timestamp:     1650376620
   > account:             0x42e742aF4E37Fe3947FD5d9425EA5F42091FD4E3
   > balance:             1.452478194709886959
   > gas used:            1917818 (0x1d437a)
   > gas price:           2.500000022 gwei
   > value sent:          0 ETH
   > total cost:          0.004794545042191996 ETH


   Deploying 'ChitFundFactory'
   ---------------------------
   > transaction hash:    0x50d54a831f135fa8b86a34aa19fd145d5399d27f177fade08a96430bbd5271ee
   > Blocks: 1            Seconds: 12
   > contract address:    0x7F35F8C183Ae51717Bf6E68008b3DCb71138D33f
   > block number:        10531807
   > block timestamp:     1650376635
   > account:             0x42e742aF4E37Fe3947FD5d9425EA5F42091FD4E3
   > balance:             1.446897462160776513
   > gas used:            2232293 (0x220fe5)
   > gas price:           2.500000022 gwei
   > value sent:          0 ETH
   > total cost:          0.005580732549110446 ETH

   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:     0.019968830171888283 ETH

Summary
=======
> Total deployments:   4
> Final cost:          0.020383057675367794 ETH
```

- After executing npm start the UI is spun up in local machine



- Metamask activity shows the gas transferred for contract creation and ether transferred to join the chitfund.