VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY

THE INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING



**D-FUNDRAISER APPLICATION BUILT ON ETHEREUM BLOCKCHAIN**

By

Ho Minh Huy

A thesis submitted to the School of Computer Science and Engineering

in partial fulfillment of the requirements for the degree of

Bachelor of Computer Science

Ho Chi Minh city, Vietnam

2018

**CHARITY FUNDRAISING APPLICATION BUILT**

**ON ETHEREUM BLOCKCHAIN**

APPROVED BY:

Tran Thanh Tung, Ph.D.

THESIS COMMITTEE

**ACKNOWLEDGMENTS**

I would like to express my very great appreciation to Dr. Tran Thanh Tung for his valuable and constructive suggestions and guidance during the planning and development of this research work. His constant encouragement and support helped me to achieve my goal.

Table of Contents

[LIST OF TABLES 5](#_Toc512869996)

**[LIST OF FIGURES](#_Toc512869997)** [6](#_Toc512869997)

[ABSTRACT 7](#_Toc512869998)

# LIST OF TABLES

# **LIST OF FIGURES**

# ABSTRACT

Blockchain is a recent technology which provides a alternative type of database, structured as a chain of blocks of data, which is managed autonomously using a peer-to-peer network and distributed timestamping server. By design, a blockchain is unchanged. At the time researching this study, there are 3 major blockchain protocols such as Bitcoin, Ethereum and Ripple Consensus Network.

Current approaches of making payment through applications, especially on websites, is that payers have to trust and provide their private information to the middleman which is a bank or a payment service to valid transactions from their accounts and transfer their money to the final destinations. It is too risky. Not only in making payment but also in funding, the fundraiser owner does not directly receive funds from donators. Both sender and receiver are able to keep track of their transactions. But, it is not the way they expected.

This research approach is to apply and create D-Fundraiser which is a decentralized application running on Ethereum private blockchain. This application is used for fund raising and has its own customized Tokens for exchanging and funding between owners and fundraisers. Besides, it is also required HTML including CSS, Javascript, Jquery, Bootstrap and Web3js for the front-end. The local server is run using Express module in Nodejs and a distributed database, which is MongoDB, to store fundraisers contents.

# **CHAPTER I**

## INTRODUCTION

The current fundraising web application is very centralized. Kickstarter, one of the largest fundraising platform, is a conspicuous example. Product teams can go to <https://www.kickstarter.com/> to create a project, set a goal and start collecting money from others who interested in their ideas. Kickstarter is essentially a third party that sits between product teams and supporters. If the project successfully funded, the project team expects Kickstarter to give them the money. On the other hand, supporters want their money to go to the project if it was funded or to get a refund when it hasn’t reach its goals. Therefore, both of them need to trust Kickstarter to handle their money correctly.

Not only Kickstarter, Most of the current web applications are now running in a centralized way. In the regular web application architecture, there is a Front End client, and a Backend server. The Front End is written using HTML, CSS, JavaScript, and the backend is written in a framework like Rails, Node, or Django. The Front and Backend interact with each other by sending JSON messages over HTTP. There is typically a hosting service involved where Backend is running like Amazon Web Services (AWS). The traditional client server architecture is so common that it's rarely revisited. There are some of the issues with the traditional client server architecture.

The first issue is that a server is running on a Centralized Hosting Service who you outsourced your hosting needs to. As more and more people around the world use the same popular hosting service, it becomes a bigger single point of failure for the entire internet. To take down all the sites that are running their Backend on this popular hosting service, all hackers have to do is disrupt that provider's infrastructure in order to take down potentially half the Internet. This isn't a very resilient architecture for the web. It's also an increasing amount of pressure on a singular organization, to expect them to build and maintain all infrastructures for the entire world. Centralized servers are also easy to censor and control. Taking down a web application is as simple as removing it from the centralized server. This can happen at the state level where government agencies request hosting providers to take down certain websites they may not like or agree with.

When users increasingly rely on centralized services, they create honeypots of data for attackers. There used to happen time and time again with various hacks. One of the conspicuous was Equifax hack. Equifax is a centralized credit reporting service and they were responsible for maintaining millions of users’ personal credit information including their social security numbers. Millions of people put their trust in this one organization to keep their data safe from getting into the wrong hands. As a potential attacker, it was very lucrative to hit just one database and get access to millions of people's sensitive data, which could be used to open credit cards, for example.

Another issue with the traditional client server architecture is that of data integrity. Along with Backend server, the database powering the application is also likely running on the same hosting service. Assuming the web’s owner trust the hosting provider a hundred percent this isn't an issue. But how could everything be guaranteed? What could stop unauthorized employees at the hosting service from making changes to the data in the database? And what if the database contained people's sensitive information like their bank account balances? That's an awful lot of trust to put into any single organization with the most valuable part of an application, especially someone’s personal data. In fact, data is so valuable that many companies are built on the very premise of selling your data. They do this by creating a centralized website for users to interact with by either posting photos or entering search words. They record users’ interactions to a database, and then sell these data to advertisers for lots of money without users’ knowledge or consent. Think about that. All the data that is generated by everyone in the world is monetized by only a handful of companies.

Protecting from data failure is another issue. Unless organizations are making the extra effort to create backups and replicas of their websites’ database, there is no guarantee of surviving catastrophic failures. Everyone running a database with their application has to take on the responsibility of backing it up as well. There is no standardized process for creating these backups and some organizations may not even create backups. Some may even mishandle the backups and place them in a vulnerable place for attackers to take advantage of.

With blockchain, developers can handle these issues. A blockchain is a decentralized database which managed by a peer-to-peer network. Everything in the blockchain is publicly readable and redundantly copied to various nodes of the network ensuring that the data is never lost. A blockchain cannot be controlled by any individual since taking blockchain down would not be as straight forward. To censor a blockchain application, people would have to take down all the networks nodes simultaneously. Everything is. Even if a certain number of nodes go down, the database can still continue to work.

The Ethereum is the next generation of blockchain which currently has over 25,000 nodes spread around the world. Ethereum refers to a protocol suite that defines a platform for developing decentralized applications. Decentralized application (Dapp) architecture also has a Front End client and a Backend server. The Front End is written using the same HTML, CSS, JavaScript, but for the Backend developers use a blockchain. The Front and Backend would still be interacting with each other using JSON messages. This architecture is a bit oversimplified, but the point is that the end user will not be able to tell whether they are interacting with a Dapp or any other regular web app. The change would be invisible and in the background. A blockchain data attacks are not as trivial as getting access to a single database. With decentralized applications, users can regain control of their data and decide for themselves whether to monetize it by choosing to sell to advertisers or not, and they get to keep the money, not some other company.

The main goal of this research is to build a Decentralized Charity Fundraising Application called D-Fundraiser which is a decentralized web application that allows everyone to raise their fund online in a university. All the fundraisers are donated with a customized cryptocurrency named Fund Token. Technically, this application is a combination of smart contracts deployed to Ethereum blockchain testnet- a simulation of real Ethereum blockchain which is managed by a peer-to-peer network – to make it be decentralized, immutable, indelible and transparent.

# CHAPTER 2

## BACKGROUND

Developing applications on the blockchain is certainly something that is new and cutting edge –in fact, called Decentralized Applications (Dapps) is a new word to describe the type of applications that are powered by the blockchain. To have a well prepared, there are some basic knowledge that developer should clearly understand.

### 2.1 Blockchain technology

Blockchain technology is amazingly trendy nowadays. This technique was authentically described in 1991 by a group of researcher and its purpose was to “timestamp” digital documents in order to prevent them from being tampered or put on a date that was earlier than which they were written. But until 2009, Satoshi Nakamoto adapted this technology to run the digital cryptocurrency called Bitcoin, Blockchain has become widespread.

Hash function and Merkle Tree are two techniques that must be known before going into details with blockchain.

#### 2.1.1 Hash Function

00111100...10101

INPUT

e1703fa96e70xa....

OUTPUT

HASH

This is a one-way function. It takes an input, usually a sequence of bits which could be any digital data, and produces a fixed size values,such as an alphanumeric string, as output. However, if just only a single bit of the input is changed, the output would be completely different. Moreover, there is noway to figure out the input if there is only the output.

#### Merkle Tree

Merkle Tree is a data structure where each layer is a combination of hashes.

1

2

12

3

4

34

R

1234

To construct the Merkle tree, each data is hashed and then coupled by hashing their hashes together. For example, from the figure......, [1], [2], [3] and [4] are hashes of data. They are then coupled and produce the output of two new hashes [12] and [34]. To form the Root of the tree, [12] and [34] are hashed together to produce the Top hash [R]. The root will be a representation of this data structure and it is impossible to figure out each individual data from the root.

Into details, a blockchain is a distributed computing architecture. Instead of using a central entity to manage the chain, blockchain uses a peer-to-peer network. Anyone is allowed to join and became a node of the network. When someone runs as a full-node, he or she gets the full copy of the blockchain. In this way, every network node executes and records the same transactions, which keep the blockchain in consensus across the whole network. Individual user interactions with blockchain are called transactions. At a time, only one block can be added and every block contains a mathematical proof that verifies that it follows in sequence from the previous block. The common format of a block consists is illustrated in figure…..

#### A block structure

BLOCK

HEADER

Previous block hash

Technical data

Timestamp

Merkel Root

Difficulty target

Nonce

Transaction count

Block content

**Header**: The block header is hashed twice to create the fingerprint which is referred to in the next block.

* *Technical data*: Included a Magic ID, a version number ( to specify which set of protocol rules a block conforms to), the size of a block.
* *Merkel Root*: Distills all the transactions in the block into a single hash (the root of transactions Merkel Tree).
* *Timestamp*: Approximate timestamp of when a block was created. Use to figure out mining difficulty re-target i.e if the network is making blocks too quickly or too slowly
* *Previous block hash*: 2x SHA256 has of previous block header (excluding magic ID & block size). This is the link that creates the chain of blocks.
* *Difficulty target*: Related to mining and how hard it is successfully mine the block.
* *Nonce*: A random number. One of the things that miner can change when mining to create different hashes, while search for a suitable hash.

**Transaction count**: How many transactions are in the block.

**Block content**: Holding a list of transactions.

To create a new block and link it to a chain, blockchain network has its workers called miners. A miner start constructing a candidate block by gathering the transactions in the “*transaction pool*”-a pool of new or unconfirmed transactions. This candidate block arrangement is same to a normal block structure. Next, he or she spends resources (dedicate hardware and electricity) to compute the hash of the block’s header. If the output is fulfill the *Difficulty target,* new block is created and linked to the chain*.* Otherwise, that miner has to change the *Nonce* until an appropriated output is found. In this way, miner is “mining” for a new block. Since the computational process of solving for the solution to create a new block is costly and time-consuming, the output is literally called a Proof-of-work.

After a mining is done, that new block is sent to every nodes on the network. Each node then verifies the block to make sure it is valid. All the node in this network create consensus. They make agreement about which blocks are valid and which are not by following those criterion:

* Block header hash is appropriated to the block target.
* Block size is within acceptable limits.
* Block timestamp is less than a *T* time in the future.
* All transactions within the blocks are valid (also have a checklist on their own).

After successfully mining a new block, miner get a total of reward for mining block and transactions fees.

### Cryptographic Proof of identity

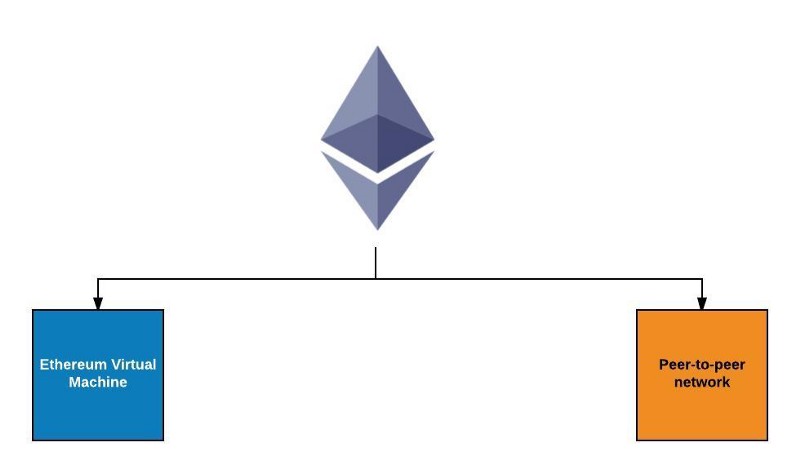
Cryptographic Proof of identity in simply means proving one’s identity without revealing it. It use the cryptography techniques to generate a digital signature. This will make a person become anonymous because he or she does not need to provide personal information but still can prove the privileged possession of a property.

Here is what a cryptographic system can do. It generate something called a public key-private key pair. They are a set of long characters that are mathematically connected. A public key is public like a username, and the private key is a secret like a password. Once a property of a person is encrypted with his or her public-private key pair by the cryptographic algorithm, only that person can prove the ownership of that property. Since, it would take 1000’s of years to break the cryptographic algorithm because of the computational limits nowadays. Quantum computers in the future might

### Ethereum

A next-generation blockchain that had the ambitions to implement a general, fully trustless smart contract platform called Ethereum, which is a combination of Blockchain, Cryptography and Smart Contract technologies.

In 2014, Vitalik Buterin, Gavin Wood and Jeffrey Wilcke became the founders of Ethereum. It comprises a Ethereum Virtual Machine and a peer-to-peer network protocol. The Ethereum blockchain database is maintained and update by many nodes connected to the network. Each and every node of the networks runs the Ethereum Virtual Machine and executes the same instructions. For this reason, Ethereum can be described as a “World Computer”.



Ethereum Virtual Machine

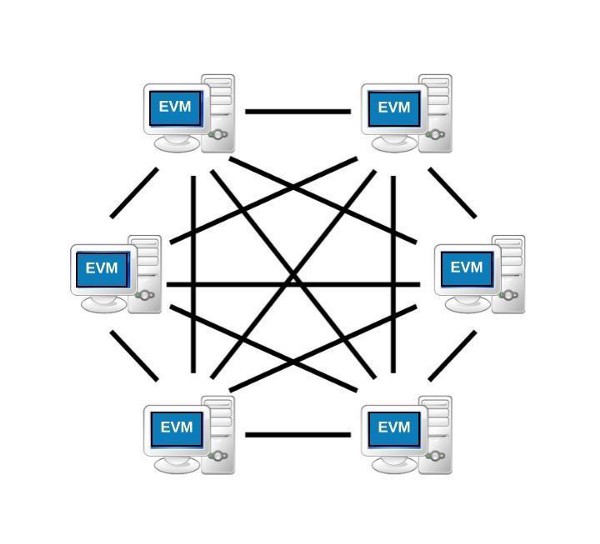
Peer-to-peer Network

#### Ethereum Virtual Machine

As has been said, Ethereum is a programmable blockchain. Instead of providing users a set of predefined operations (like Bitcoins did with its transactions), Ethereum furnish them a friendly language running on a “Virtual Machine”-called an Ethereum Virtual Machine (EVM). At the heart of Ethereum, Ethereum Virtual Machine (EVM) can execute code of arbitrary algorithmic complexity, which makes it to be “Turing complete” – a system that can understand a set of instructions and execute them in some logical order, just like a computer does. Using Solidity as its programming languages, Ethereum serves as a platform for various types of decentralized blockchain applications, including but not limited to cryptocurrencies.

#### Peer-to-Peer Network Protocol

The fundamental network consists of nodes (or computers) connected in a decentralized, peer to peer network. Each node runs an EVM, and process the same instructions to make sure that consensus across the network is achieved on any particular transaction.



#### Ether

Ether is the cryptocurrency of the Ethereum network which can be used in trading with other supported cryptocurrencies. In the other hand, it can be seen as the fuel, which is needed to run a transaction on Ethereum blockchain. Ether is also a reward to miners each time they succeed mining a new block on Ethereum blockchain. Developers who intend to use Ethereum as a platform to build apps need Ether, as it is used to pay for computation within the EVM. Shortly, users who want to interact with Ethereum blockchain applications must use Ether.

Ether is to the Ethereum network, what bitcoin is to the Bitcoin Blockchain network. Hence, users should become Ethereum miners, or trade with other currencies using centralized or trustless services in order to gain some Ethers.

The base unit of Ether is called Wei.

#### Ethereum Accounts

Comparing to Bitcoins, one of the first and biggest blockchain application which is purely a list of transactions, Ethereum’s basic unit is the account. There are two type of accounts:

|  |  |
| --- | --- |
| **Externally Owned Accounts (EOAs)** | **Contract Accounts** |
| Controlled by an External party or person  Accessed through private keys  Contain Ether Balance  Can send transactions as well as ‘trigger’ contract accounts | Have code that executes when being triggered  Also contain Ether Balance  Can trigger other contract accounts  Live on the Ethereum Blockchain |

If a user wants to participate in the Ethereum ecosystem, he or she should have a user account and the keys to operate that account. These types of accounts are called Externally Owned Accounts (EOAs) in Ethereum. Once an Externally Owned Account (EOA) is created, transactions can be made from that account to the others as well as to other types of account – Contract Accounts - on the Ethereum.

A Contract Account contains code to execute some designed functions, and this code is put on the Ethereum Blockchain. When a smart contract is triggered – by an Externally Owned Account or by another contract account – the code inside is executed by the EVM on each participating nde.

#### Smart Contract

The born of Smart Contracts is one of the recent developments in this field. They are very popular nowadays. The term “smart contract” was first used by Nick Szabo in 1997. He wanted to use a distributed ledger to store contracts.

Nowadays, smart contracts are similar to contracts in real world. The only difference is that they are completely digital. Two parties can agree on a set of rules, create and code a contract. As soon as the criterion/rules are met, it automatically authorizes the validation between two parties. Smart contracts not only store conditions but also data. However, people should admit the limitations of the smart contract. Some systems need human support, computers cannot handle everything. In the real world implementing smart contracts is not easy. Smart contracts once published cannot be altered, silly mistakes can be costly.

There are steps which a smart contract is created in Ethereum:

1. A smart contract is coded in Ethereum programming language (such as Solidity) following some conditions (e.g. “If A is true, then do B”) and then deployed on Ethereum Blockchain by an EOA (using some Wei as fees).
2. Once deployed, the smart contract gets a public key address, which can be used to reach the contract and trigger its code execution. This address is the contract account of that smart contract.
3. A deployed smart contract cannot be changed, even by the EOA that created it.
4. The EOA who deployed a smart contract becomes the owner of that Contract Account.

The bottom line is that Smart Contracts are code that is deployed on the Ethereum blockchain, and this code runs on every single node connected to the Ethereum network.

#### Solidity

Solidity is a contract-oriented, high-level language for implementing smart contracts. It was influenced by C++. Python and JavaScript and is designed to target the Ethereum Virtual Machine (EVM)

Solidity is statically typed, supports inheritance, libraries and complex user-defined types among other features.

#### The workflow of Ethereum

Ethereum has the same Hashing data and Proof-of-work mechanism as a blockchain, but its hashing function is keccak256, sometimes (erroneously) called sha3. On Ethereum, a user could trigger a transaction from his or her EOA. A transaction is a validated packet that contain:

. The address of the recipient.

.A signature that proves the possession of the sender’s account.

.Value field – The amount of ether to transfer from the sender to the recipient. It can be empty, in some cases.

.An optional data field, which could be an arbitrary message or function call to a contract. As an example, accidental insurance smart contract would require to input the proportion of damage in order to release the respective compensation.

.Transactions on ethereum also need something called “gas” to run. Gas is Ethereum’s metering scheme and it accounts for bandwidth used, cost of data storage and cost of computation on the Ethereum blockchain. Every computational operation in the EVM consumes gas and different computations consumes different amount of gas. Those additional attributes must be generated before triggering new transactions:

-startgas: the quantity of “gas” that a transaction is willing to consume to cover its user of the EVM’s computation and any storage bandwidth used. It is really difficult to define exactly how much is the startgas. Therefore, there are certain APIs pretending the transaction was actually being included in the transaction, and then returning the estimated amount that would have been charged if that pretend operation was real.

- gasprice: mentioned by the transaction sender. This is the amount of ether that a sender is willing to pay per unit gas.

- gas\_rem: If a transaction execution is done and consuming less gas then its specified limit, the transaction sender receives a refund of *gas\_rem\*gasprice*.

Once a transaction is sent, the destination can be another EOA or a Contract Account. The transaction to an EOA is simply an Ether transfer, which means ether balances from both sender and receiver will be adjusted. On the other hand, destination of the transaction is a Contract Account, the smart contract’s code will be executed automatically. In some cases, a Contract Account needs to run a function from another Smart Contract (or Contract Account), it will send a message to that Contract Account which contains:

+ the address of the contract sending the message

+ the address of recipient contract

+ the amount of ether to transfer alongside the message

+ an optional data field

+ a startgas value

The message results in the recipient contract running its code.

EOA

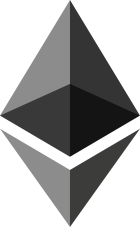
sender

EOA

receiver

Contract

Account



Update blockchain

Update blockchain

Run code

Transaction

message

Ethereum

Another

Contract Account

Transaction

After a transaction is done, the “state” of an account is updated in the Ethereum blockchain. At the first time an account (either EOA or Contract Account) is created, it is at the genesis state. When a transactions on a Ethereum is successfully mined by a miner (a blockchain worker who groups transactions – which included many updates to the ”state” of accounts- into block and compete with one another for his or her block to be the next one to be added to Ethereum blockchain), the current state of the recipient accounts is updated in the Ethereum blockchain by the whole nodes on the network.

Miners are rewarded the amount of Ethers, which is equal to *startgas\*gasprice* of a trasaction. At the start of a transaction execution, this amount of ethers is removed from the transaction sender’s account to ensure the miner receives the fee even if the sender account is bankrupted midway during execution. If there is gas refund *gas\_rem* to the sender, miner of that transaction receives a refund of *(startgas – gas\_rem)\* gasprice.*

#### Ethereum Decentrailizaed Application

At their core, DApps are software programs that use the power of blockchain technology and smart contracts to achieve application decentralization. This means that control over any DApp cannot be exercised by any single entity. For example, on a decentralized Facebook or Youtube like platform, once a post or video is made it would be censorship resistant, as the blockchain itself is immutable. Smart contracts are self-executing contracts that are designed to enforce an agreement created between two parties. Smart contracts can be used to create whole ecosystems of value exchange within a DApp, making the use case of DApps even more extensive.

The problem Dapps are trying to address is that being control over one’s data. When users use the Facebook application, they are liking, commenting and generally interacting with others on the platform. When users do this, however, they are giving data to Facebook. In turn, Facebook is storing that data and selling it to advertisers so that it can tailor ads to suit users’ preferences. For many, it is an uncomfortable feeling to have their data treated in this manner. Ideally, data should remain oneself. DApps give users back control over their data, by ensuring that no single entity is in charge of any user’s data.

For an application to be truly decentralized it usually must meet the following requirements:

Open Source: The application’s code base must be widely available so as to allow for public scrutiny.

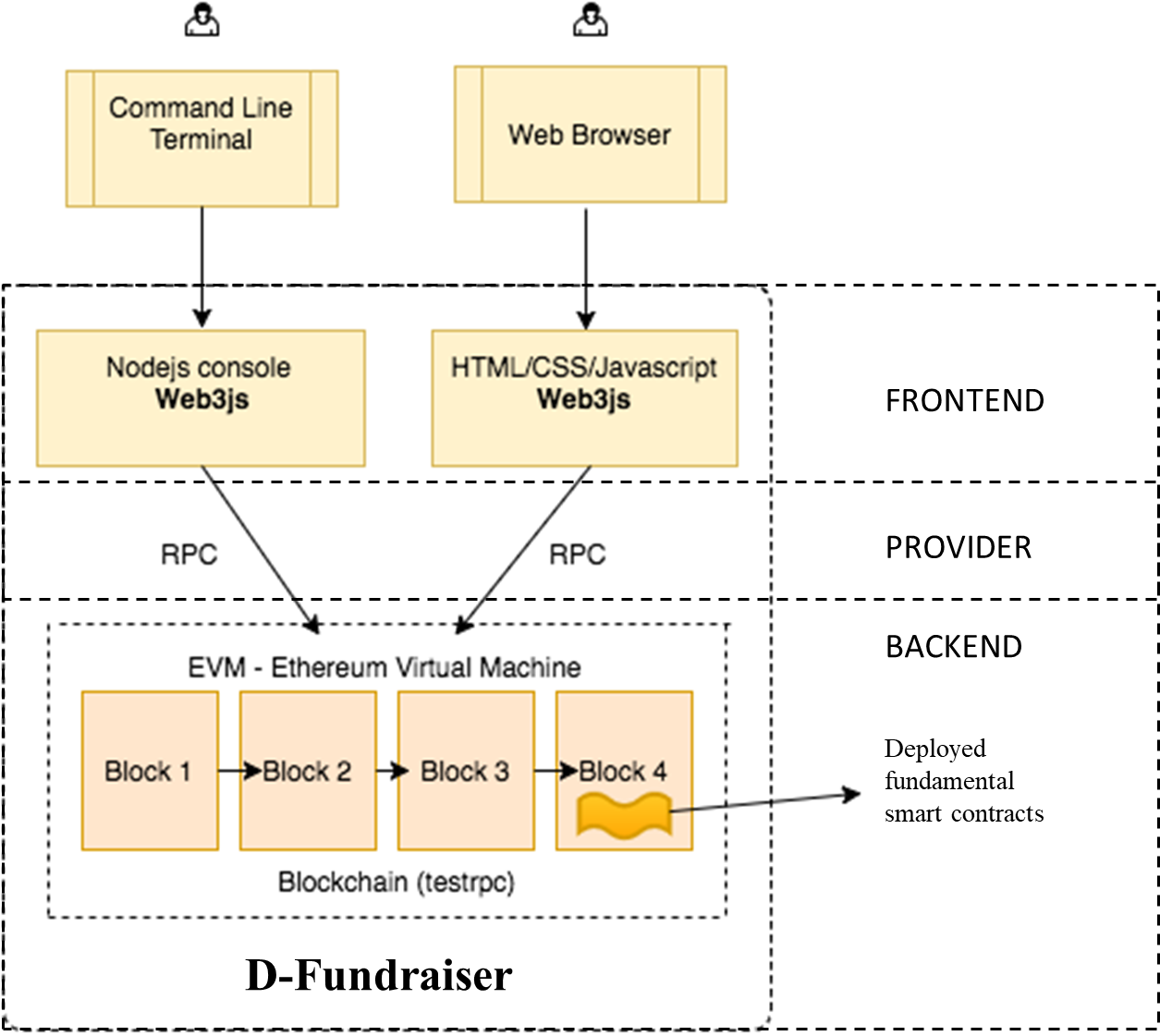
Decentralized: The application’s data must be cryptographically stored on a public and decentralized blockchain.

Incentive: The application must utilize tokens/digital assets to reward its network supporters.

Protocol: The application must generate tokens using a cryptographic consensus algorithm to demonstrate proof of value (e.g. proof of work or proof of stake).

The Ethereum blockchain is the most well-developed network for facilitating the creation of DApps. Using its very own programming language, Solidity, developers can build DApps on just about anything. The DApps that have been produced so far speaks volumes to where this part of the ecosystem can take us.

##### Core Components of Ethereum Dapps

****

###### BACKEND

+Smart Contracts act as the back-end logic and storage. A contract is written in Solidity, a smart contract language, and is a collection of code and data that resides at a specific address on the Ethereum blockchain. It’s very similar to a class in Object Oriented Programming, where it includes functions and state variables. Smart Contracts, along with the Blockchain, are the basis of all Decentralized Applications. They are, like Blockchain, immutable and distributed, which means upgrading them will be a pain if they are already on the Ethereum Network. Fortunately, here are some ways to do that.

+The Ethereum Virtual Machine(EVM) handles the internal state and computation of the entire Ethereum Network. Think of the EVM as this massive decentralized computer that contains “addresses” that are capable of executing code, changing data, and interacting with each other.

###### FRONTEND

Front end of the Decentralized applications are typically created as single page or desktop applications using Javascript/HTML/CSS. It is used to handle with users inputs and display responses from the Backend. Since there are multiple Javascript frameworks, developers are free to use any one or more JS frameworks of their choice.

Web3.js is a Javascript API that allows clients to interact with the Blockchain, including making transactions and calls to smart contracts. This API abstracts the communication with Ethereum Clients, allowing developers to focus on the content of their application. Developers must have a web3 instance imbedded in their browser to do so.

###### RPC

RPC is a connection between a local computer and Ethereum Blockchain. This connection can be archived easily by becoming a full-node of the network or just a light one. For most users, a light client or Light client protocol is the best choice. The purpose of the light client protocol is to allow users in low-capacity environments (embedded smart property environments, smartphones, browser extensions, some desktops, etc) to maintain a high-security assurance about the current state of some particular part of the Ethereum state or verify the execution of a transaction. Although full security is only possible for a full node, the light client protocol allows light nodes processing about 1KB of data per 2 minutes to receive data from the network about the parts of the state that are of concern to them, and be sure that the data is correct provided that the majority of miners are correctly following the protocol, and perhaps even only provided that at least one honest verifying full node exists. This means that it doesn’t store all of the blockchain data, and depends on asking the network for the data it needs every time.

# CHAPTER III

## METHODOLOGY

As applied the theory in the previous chapter 2, this chapter deals with the description of technologies, design and technical are used in the author's thesis implementation.

### Method Overview

D-fundraiser is a fundraising web application programmed using Ethereum framework. It has backend codes (which are smart contracts) run on a Ethereum blockchain managed by a peer-to-peer network and frontend code working as user interfaces written in HTML (HyperText Markup Language) and JavaScript that can make calls to its backend. Before an Ethereum decentralized application is public and run on the Ethereum network, it is programmed and tested in a “local” environment which contains a local peer-to-peer network and a private blockchain customized by developers.

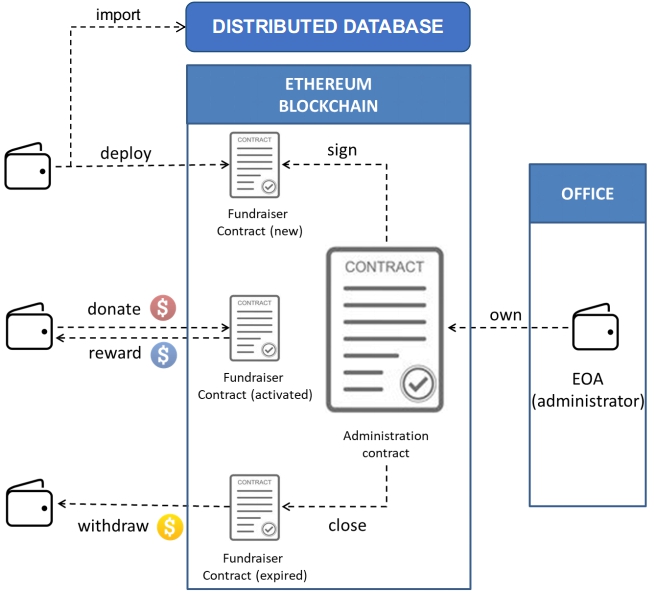
### Main methodology

Looking in detail, a blockchain application layers is illustrated by figure….

|  |
| --- |
| **APPLICATION**  D-Fundraiser WebApplication |
| **BLOCKCHAIN LAYER**  Ethereum |
| **DATABASE LAYER**  MongoDB |

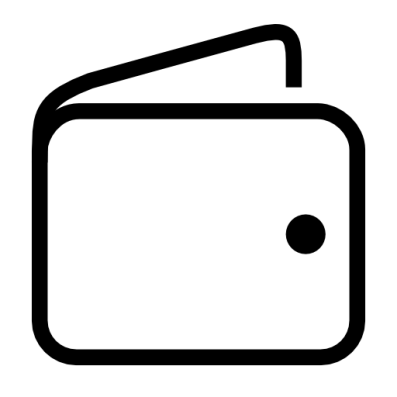
On the top, of course, is the Application layer supporting the business process. The Blockchain layer provides decentralized, immutable, indelible and transparent attributes to the application, which is the public ledger in our cases. Blockchain applications ,the bottom line, is a database storing data in a decentralized way. But, why is a Dabatabse layer needed here? While the state of blockchain is already a key-value database, it is rather primitive. The amount of stored data is very limited and slow responsive. For recent applications, this is not enough. Hence, when developing applications on Ethereum, the problem of data storage is very actual.

#### SYSTEM WORKFLOW DESIGN



**PRIVATE BLOCKCHAIN**

As it is running on Ethereum, a user must have an Externally Owned Account (EOA) in order to join the Ethereum Network. It is also called a wallet (public-private key pair). A wallet has an address (A public wallet address) which identifies an individual in the network. This is also help to protect the privacy of a person. Ether and another customized cryptocurrencies such as tokens are “stored” in this wallet. The customized digital currency is called Token, which acts as an coin in real life. Ether is a cryptocurrency in Ethereum, which is also the core coin. Token cannotwork as a core coin, but does have a coin’s attributes and functions (tradeable and payable). In this D-fundraiser system, users directly donate token to the Fundraiser contract account without any third-party organization in the middle. Donators are anonymous. With this system, users do not have to provide any personal information. Moreover, nobody has permission to control an individual’s assets.



To start a new fundraiser, a user or an organization is required to fill in all the necessary information. Of course, the owner’s information is stored in the decentralized database and publicly displayed on the front-end because donators have to clearly know and understand what they are spending money for. Fundraisers owners and organizations must not be anonymous. When a new fundraiser is generated, it must be signed by the Administrator Contract Account to active its token receiving function. Otherwise, it would be not able to be donated. A Funraiser smart contract is created and deployed to the Ethereum blockchain each time a user want to generate a new fundraiser. This creation is announced to the Administration Smart Contract to ask for it approval. Fundraiser smart contract consists of start and end date, a goal, a total donation, a detailed list of donators and the sign of the Administrator Contract Account. With these attributes and some related functions, the Fundraiser smart contract automatically open for donating when start date is reached and stops receiving funding token when it is expired or its goal is reached. Moreover, the contract only supports Fund Token. Ethers will be rejected if they are sending to a fundraiser Contract Account. Once a fundraiser contract account is done, it will be removed from the browsing fundraiser site. But, users is able to view finished fundraisers by visiting the audit sites.

When a user comes into this system, he or she can browse and donate any activated fundraisers. For this system, only Fundraiser Contracts which are signed by the administration contract can be donated on-line, so when a person generate a new Fundraiser and deploy it to the Ethereum blockchain, he or she must wait for the approval from the Administration contract first. When a fundraiser is successfully deployed and signed, it cannot be edited or tampered with.

The process is anonymously fair and transparent. For example, a donator want to donate 6 Fund Tokens to an activated fundraiser, she use her public-private key pair to sign a transaction from her wallet to that fundraiser contract account. If the transaction is successfully mined, 6 Fund Tokens will be deposited from her wallet and the fundraiser’s total donation will be increased by 6 Happy tokens.She is rewarded some Reward tokens for her first time donating. No personal information is required. The owners can withdraw all the funds if their fundraisers are expired or the goals are reached. If there is a transaction triggering a withdraw function sent to an expired fundraiser contract account, the total Happy tokens from a Contract Account will be send to its owner wallet. Note that a withdraw function is only called by the owner.

In addition, there is an organization to ensure the validation of a fundraiser: an EOA managed by Fundraiser Office. Quite different from other system, the administrator of D-Fundraiser application is a person who uses the EOA, which is the owner’s account of the Administration Contract Account, to sign a fundraiser contract account and manage D-fundraiser GUI. The administrator has to read through all the new fundraisers to check the validation of each and sign them by making transaction to Administration Contract Account in order to active them. The administrator also needs to keep the GUI clean and clear. Every closed fundraiser must be wiped out from the GUI. What is a closed fundraiser? A fundraiser is closed if only it is withdrawn by the owner and its owner has received the fund in cash. Those fundraisers are only used for auditing purposes. Therefore, they will be closed by the administrator.

#### SYSTEM COMPONENTS

There are three fundamental components in this system: Front-end, Back-end and a core database.

##### Front-end

The GUI is programmed as a website. Users are able to “talk” to the Smart Contracts on Ethereum by interacting with the website. The website is divided into pages:

##### Back-end

*Cryptocurrencies, Fundraiser and Administrator* smart contracts are used for performing some vital methods and computation.

###### Token contracts

*Ethereum Request for Comments 20, or ERC20, is an Ethereum Improvement Proposal introduced by Fabian Vogelsteller in late 2015. It’s a standard by which many popular Ethereum smart contracts abide. It effectively allows smart contracts to act very similarly to a conventional cryptocurrency like Bitcoin, or Ethereum itself. In saying this, a token hosted on the Ethereum blockchain can be sent, received, checked of its total supply, and checked for the amount that is available on an individual address. This is analogous to sending and receiving Ether or Bitcoin from a wallet, knowing the total amount of coins in circulation, and knowing a particular wallet’s balance of a coin. A smart contract that follows this standard is called an ERC20 token.*

In this system, there are 3 ERC20 tokens:



+ Fund Token: it can be bought by sending Ether to the Administrator Contract account or directly exchanged from cash to token by contacting the office.

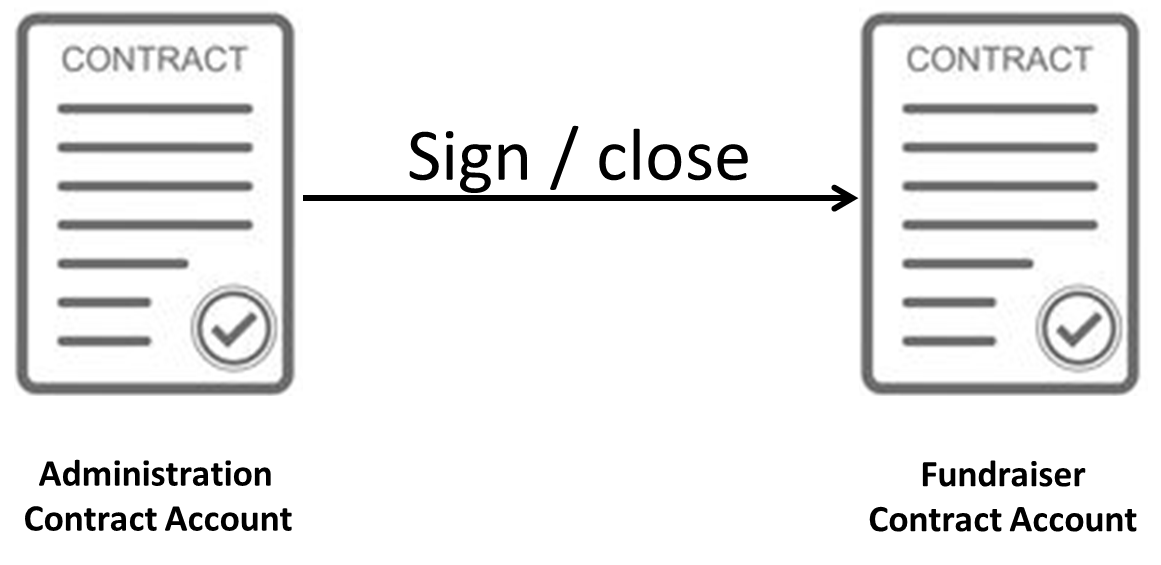


+ Happy token: this token represents the fund in a fundraise, which is the amount of token that the owner could withdraw at the end of a fundraiser.



**+** Reward token: the token that is rewarded to a donator. Donators are only rewarded at their first time donating to a fundraiser.

###### Administration and Fundraiser contracts



|  |  |
| --- | --- |
| **ADMINISTRATION SMART CNTRACT** | **FUNDRAISER SMART CONTRACT** |
| This contract account is own by an administrator’s EOA managed by the Charity Office  The owner of administration contract account can be only changed by the current administrator. This means that only the current administrator along with the EOA could promote a new administrator with an EOA to own this Contract Account | This smart contract stores back-end codes to execute fundraiser functions. Each contract has an owner. Only the owner of a fundraiser can withdraw its fund. |

##### Core Database

User’s data in D-Fundraiser application is divided into two group: Static and Dynamic. Dynamic data is data records involving in some vital system computations. Hence, they are stored in smart contracts as states and deployed to the Ethereum blockchain. They can be updated by making transactions to their Contract Account. Static data is data that does not change after being recorded. It is a fixed data set. The contents of a fundraiser such as the title, the owner story and information, etc are not necessary to be kept on the blockchain. Therefore, a core database is included in order to store and responsively query these records to display them to the front-end.

The requirements to the ideal decentralized database that would be suitable for the decentralized applications needs. MongoDB is a distributed database at its core, so high availability, horizontal scaling, and geographic distribution are built in and easy to use. It is free and open-source. MongoDB lets developers work faster, deploy easier and scale bigger. Modern applications are more networked social and interactive than ever. This is driving new requirements to support big data fast feature development and flexible deployment strategies. Database is expected to store more and more data, and being accessed at higher and higher rates .if database runs on a single server, it will soon reach a scaling limit. MongoDB scales by adding more servers so that developers can add more capacity whenever they want. Agile development and continuous deployment have accelerated roadmaps. If a data model is not flexible, it can slow development down. MongoDB increases productivity modeling data as documents is simpler and allows schemas to evolve effortlessly without duplication. Cloud computing is changing how people deploy applications both inside and outside the firewall. If a database needs complex Hardware, it's not ready for the cloud and can keep the organization stuck in expensive data centers. MongoDB was designed to work with commodity servers and elastic virtualized environments, saving money and headaches. MongoDB, it's scalable agile and cloud ready.

# CHAPTER IV

## IMPLEMENTATION

After all research about technologies and technical that required for building the final product as mine thesis. This chapter is the most important in all five chapters because it covers all techniques how to code the system and release it.

PREPARATION

### Environment settings

To build he final product, developers need the environment to implement.

#### Operating system

Ubuntu is an open source software operating system. This implementation is running on Ubuntu version 16.04 64-bit.

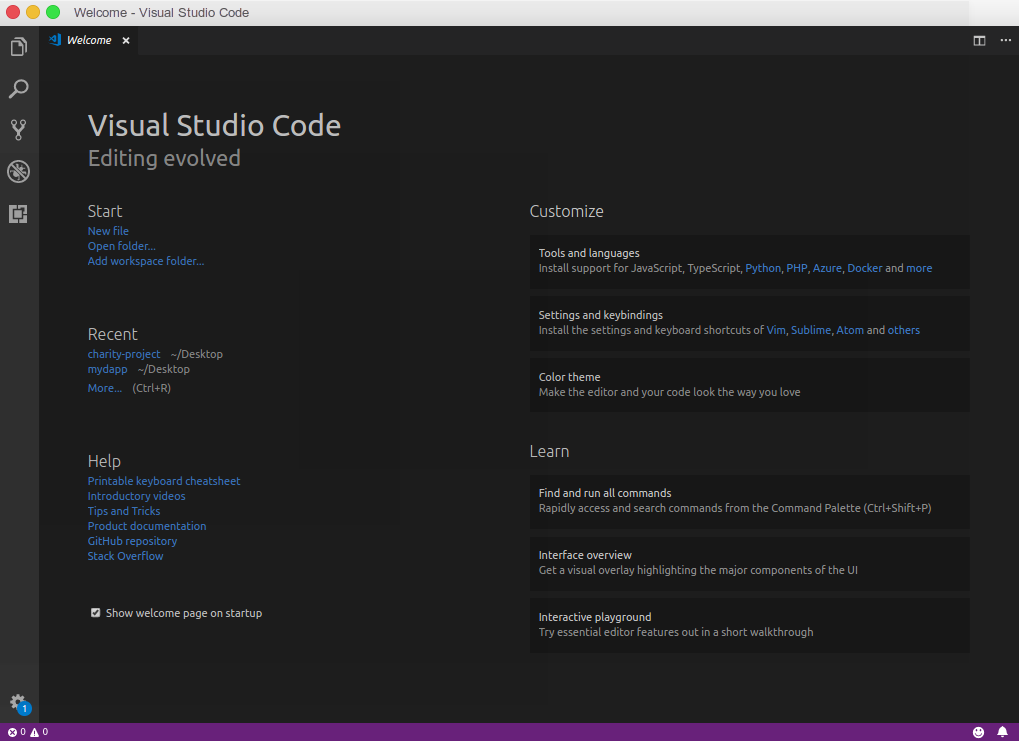
##### Terminal

The traditional Unix environment is a CLI (command line interface), where you type commands to tell the computer what to do. That is faster and more powerful, but requires finding out what the commands are. The shortcut to run Terminal in Ubuntu is Ctrl + Alt + T



#### Code Editor

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Ubuntu. It comes with built-in support for HTML, JavaScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity).

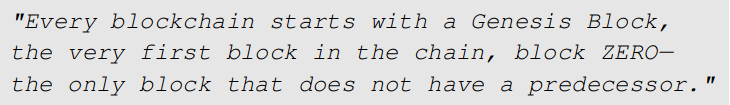


#### Private blockchain and testnet

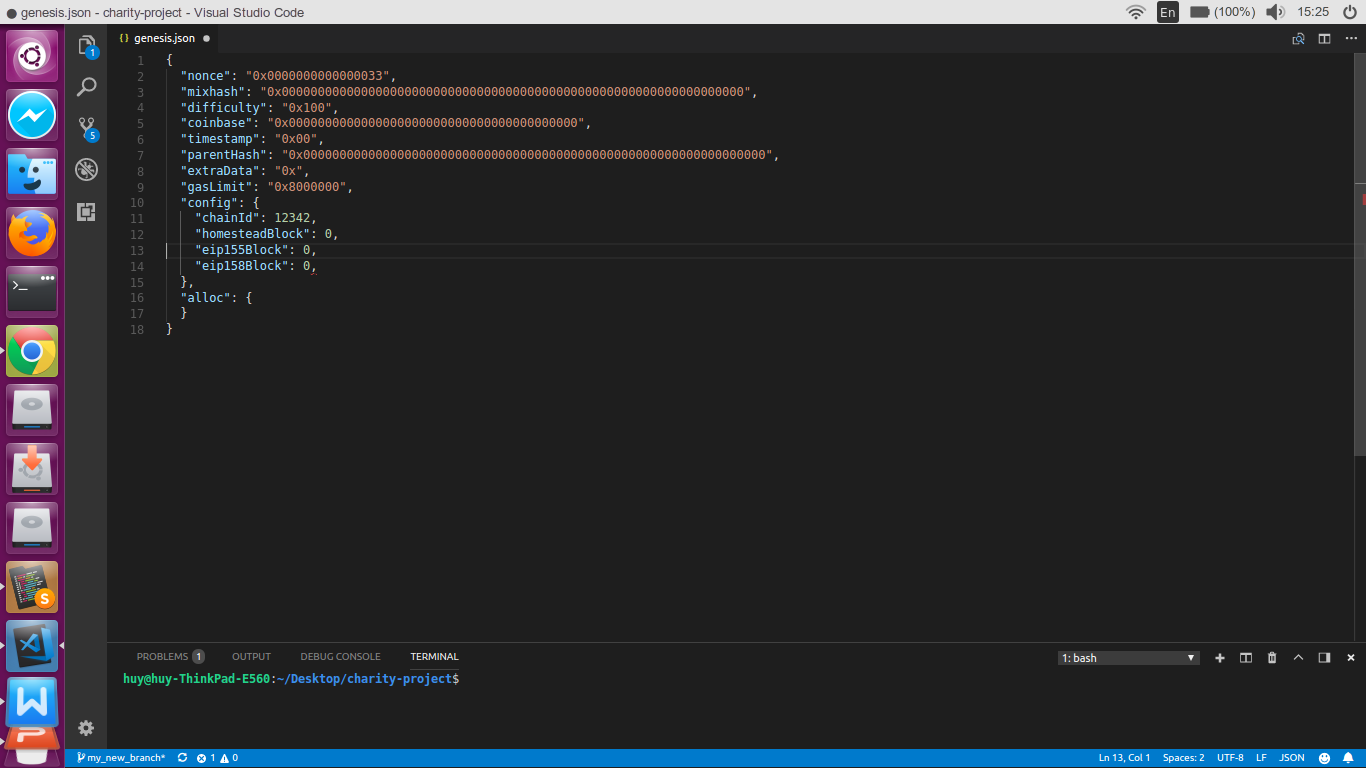
Ethereum allows developers to create their own “private” blockchain network, a sort of a developing-staging version. Its functionality is exactly the same to the main Ethereum Network. The differences are that any transactions and smart contracts deployed on this network can only be seen and accessed by the nodes that are connected to this private network. Moreover, users can setup the characteristics of the private blockchain as they like, from the chainID , difficulty of a block or even the initial Ethers of an account in this network.

##### Genesis block

One of Blockchain application is a digital ledger, where transactions are recorded chronologically and publicly in blocks.



To have a private blockchain, the first thing is to create a genesis block. The genesis block configurations are stored in a .json file. The file is named as “genesis.json” or whatever you want to. The genesis file is showed in the figure......



###### Mixhash, nonce

mixhash and nonce work together to decide whether the block was mined appropriately. The reason why there are both mixhash and nonce is that if a hacker counterfeits blocks with a false nonce, it is still computationally expensive for other peers (or nodes) in the network to recognize that the nonce was faked. The purpose of Mixhash calculation is to prove the nonce which is not as costly to produce. A node on the network can use the mixhash to check if a block is valid or not without recalculating the nonce. The default value of these attributes is not vital so they can be random values in order to prevent other peers from accidentally connect to a chain by having the same exact genesis.json file.

###### Difficulty

This decides how challenged it is to mine a block in a blockchain. Ethereum’s mainet is still uses Proof Of Work minng algorithms. The hexadecimal value “0x100” is equivalent to 256 in decimal, which is meanss there is a 1/256 chance first attempt at mining a block of a miner succeeds. How fast miners can mine a block depends on how fast their computer could produce on average 256 hash computations. Not every block will have this value of difficulty. The hardware potential power is different for each node. Hence, the difficulty value is algorithmically calculated in the Ethereum client (miner) side for a subsequent block.

###### Coinbase

The 160-bit address to which all rewards (in Ether) collected from the successful mining of this block have been transferred. This can be anything in the Genesis Block since the value is set by the setting of the Miner when a new Block is created.

###### Timestamp

The output of the Unix time() function when the block was created.

###### parentHash

The Keccak 256-bit hash of the previous block’s header. The purpose is to make the genesis block have the same format as any other block. Therefore, it is assigned a value. Genesis block is block number 0 and has no parent block.

###### extraData

An optional free, but max. 32-byte long space to conserve smart things for ethernity.

###### gasLimit

The maximum number of computations any block on that chain can support. High in our case to avoid being limited by this threshold during tests.

###### Config

Config is a struct. These struct values are essential since they should match the configuration information of other nodes that you want to interact with.

chainId: identity which chain you are on. It can be any integer starting from 0. This is an id list of some existing chain:

0: Olympic, Ethereum public pre-release testnet

1: Frontier, Homestead, Metropolis, the Ethereum public main network

1: Classic, the (un)forked public Ethereum Classic main network, chain ID 61

1: Expanse, an alternative Ethereum implementation, chain ID 2

2: Morden, the public Ethereum testnet, now Ethereum Classic testnet

3: Ropsten, the public cross-client Ethereum testnet

4: Rinkeby, the public Geth PoA testnet

8: Ubiq, the public Gubiq main network with flux difficulty chain ID 8

42: Kovan, the public Parity PoA testnet

77: Sokol, the public POA Network testnet

99: Core, the public POA Network main network

7762959: Musicoin, the music blockchain

Any other number different from those id could indicate that your connected to a local development test network.

homesteadBlock

when set to 0, means you will be using the Homestead release of Ethereum. This is expected, and the mainnet gensis configuration also has this set to 0.

eip155Block

EIP155(Ethereum Improvement Proposal 155) is a proposal which was accepted to help prevent replay attacks.

eip158Block

EIP158 was accepted to change how Ethereum clients deal with empty accounts. This new protocol began treating them as nonexistent, saving space on the blockchain.

###### Alloc

Allows defining a list of pre-filled wallets. That’s an Ethereum specific functionality to handle the “Ether pre-sale” period

##### Ethereum private neworkt account

Ethereum private blockchain is manage by a local peer-to-peer network. To join this network, users must have an **Externally Owned Accounts. An external account must have a public key (or address) and private key.**

##### Geth

Geth is a command line interface (CLI) tool that communicate with the Ethereum Network. Geth helps to build a bridge from a node computer to the rest of other Ethereum nodes in the network. If a block is mined by another node, Geth program will pick it and pass on the new information onto GPU or CPU to update the blockchain. Geth allows people to

* Mine ether
* Transfer amount of ether between addresses
* Create smart contracts and send transactions to them
* Explore block history and more.

It provides a console run in JavaScript environment which has all of the main features to “play” with Ethereum private blockchain.

Geth can be downloaded from <https://geth.ethereum.org/downloads/> and installed following the instruction. Once having installed, developers can technically connect to the main (or publc) Ethereum blockchain network and run as a full node. For this research, Geth is used for creating a private blockchain (local network blockchain). It enables developers to setup a private (or testnet) Ethereum blockchain. They can build smart contracts, make transactions and even distributed applications without needing real ether. Fake ether can also be created and pre-assigned to an EOA on this private network.

###### Run a private blockchain using Geth command

In Terminal console, run this command

geth --datadir "data directory" init “directory/genesis.json”

The initial step is to let Geth know the “data directory” -Data directory for the databases and keystore(EOA, public-private keypair) - and the directory of the genesis.json file.

On another hand, init - Bootstrap and initialize a new genesis block - helps to create the genesis block within all the configurations in genesis file of our private blockchain.

The next step is to create a private (local) network that manages the private blockchain ( peer-to-peer network)

To start the first node

geth --datadir "datadir" --port 30301 --nodiscover --networkid 12342 --rpc --rpcport 8001 --rpccorsdomain "\*" --rpcapi="db,eth,net,web3,personal" --ipcpath "datadir/geth.ipc"

--port 30301: Network listening port 30301.

--nodiscover: Disables the peer discovery mechanism (manual peer addition).

--networkid value: Network identifier (which is the chainId in your genesis file)

--rpc: Enable the HTTP-RPC server.

--rpcport 8001: HTTP-RPC server listening port (default: 8001)

--rpccorsdomain”\*”: Comma separated list of domains from which to accept cross origin requests (browser enforced).

--rpcapi =”db,eth,net,web3.personal”: API's offered over the HTTP-RPC interface

--ipcpath: Filename for IPC socket/pipe within the datadir (explicit paths escape it)

To access the Geth JavaScript ipc console of this node.

geth attach datadir/geth.ipc

We started the node with the ipc console, so that we can grab the enode url for instance

> admin.nodeInfo.enode

enode://8c544b4a07da02a9ee024def6f3ba24b2747272b64e16ec5dd6b17b55992f8980b77938155169d9d33807e501729ecb42f5c0a61018898c32799ced152e9f0d7@9[127.0.0.1]:30301

To generate subsequent nodes on another computers, we run the same command line with different data directory, port and rpc port. For example:

geth --datadir "datadir2" --port 30302 --nodiscover --networkid 12342 --rpc --rpcport 8002 --rpccorsdomain "\*" --rpcapi="db,eth,net,web3,personal" --ipcpath "datadir2/geth.ipc"

If you want to connect this instance to the previously started node you can add it as a peer from the console with

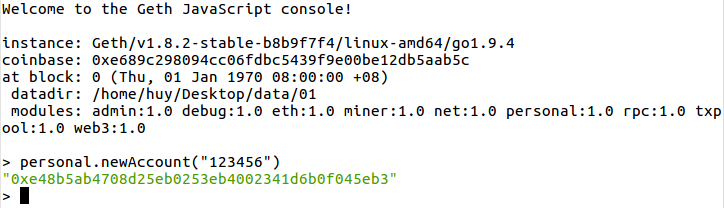
> admin.addPeer(enodeUrlOfFirstInstance)

###### Register with Geth

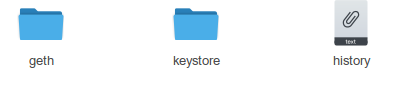
New EOA account

After start a node using GetH, an account must be created in order to mine or deploy a smart contract to the private blockchain. In geth ipc console

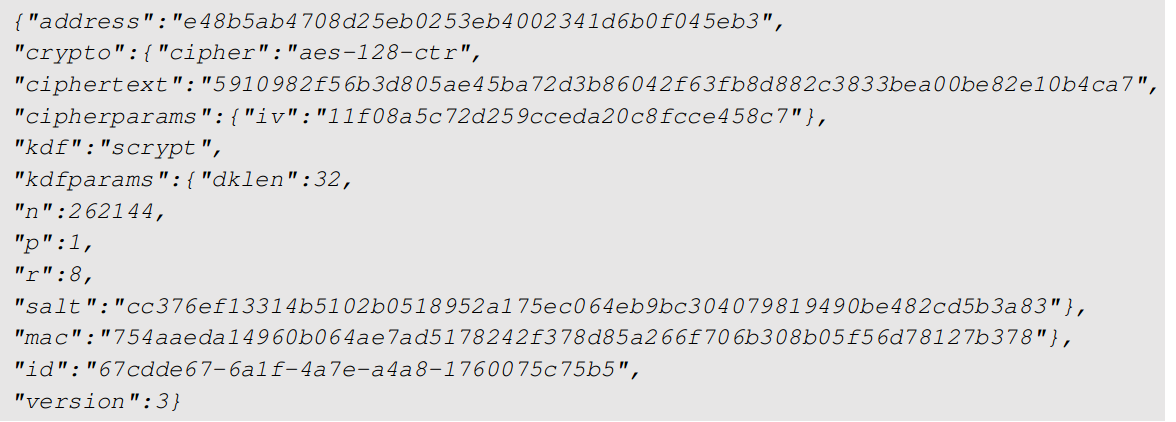
> personal.newAccount("your password") - create a new EOA with your password. Keep in mind that you cannot get the password back if you forgot it !



As can be seen from figure....., the return of newAccount() functon is a public key of your EOA. Go to your node data director, there is a folder called keystore. It is where your *A UTC--{year}-{month}--{account}* encrypted account file is stored.



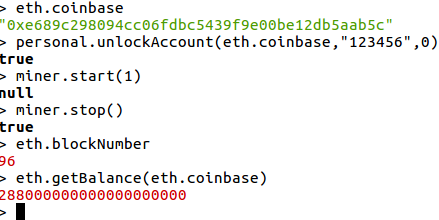
Here is the encrypted account file format



The contents of the file also includes an encrypted version of your private key. The Ethereum account file is locally stored offline on a user hard disk. EOA owners must keep this file safe and do not let anybody know the contents of this file. A backup or copy of this file should be created and stored somewhere else.

Mining some ether

In private blockchain is not complicated compare to Main Ethereum Blockchain since we can modify the difficulty level to be extremely low in the genesis file. In geth console



The figure.....shows that eth.coinbase is the account used to mine or make transaction. It will be infinitely unlock using personal.unlockAccount(account=”0xe689c298094cc06fdbc5439f9e00be12db5aab5c”, password=”123456”, duration=0)

miner.start(1) - Start the CPU mining process using the given number of threads.

miner.stop() - Stop the CPU mining operation

eth.blockNumber - return the number of recent mined block

eth.getBalance(eth.coinbase) - get balance of the coinbase account in wei.

##### Ganache

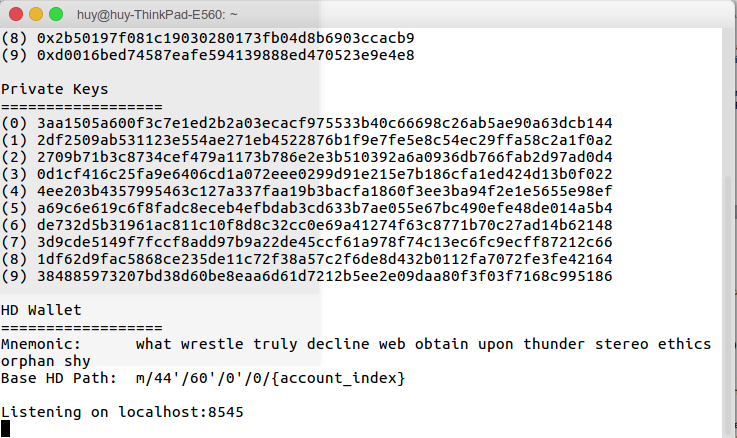
Ganache is also a CLI tool, but much simple than Geth. It creates a virtual Ethereum blockchain and generates some fake accounts that will be use during development. Its default RPC port is 8545.

Ganache CLI uses ethereumjs to simulate full client behavior and make developing Ethereum applications faster, easier, and safer. It also includes all popular RPC functions and features (like events) and can be run to make development a breeze. Ganache simulates the automatical mining process. Hence, no command or Account Creation is needed.

ganache-cli is written in Javascript and distributed as a Node package via npm. Make sure Node.js version greater than v6.11.5 is installed.

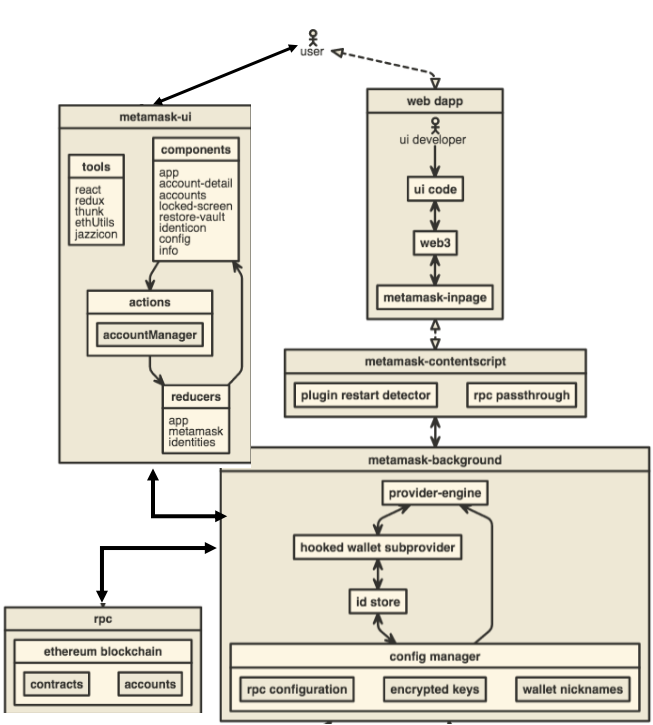
In terminal, type this command to run ganache-cli

ganache-cli -u 0 - run Ethereum private blockchain and-u (unlock) the account with index 0.



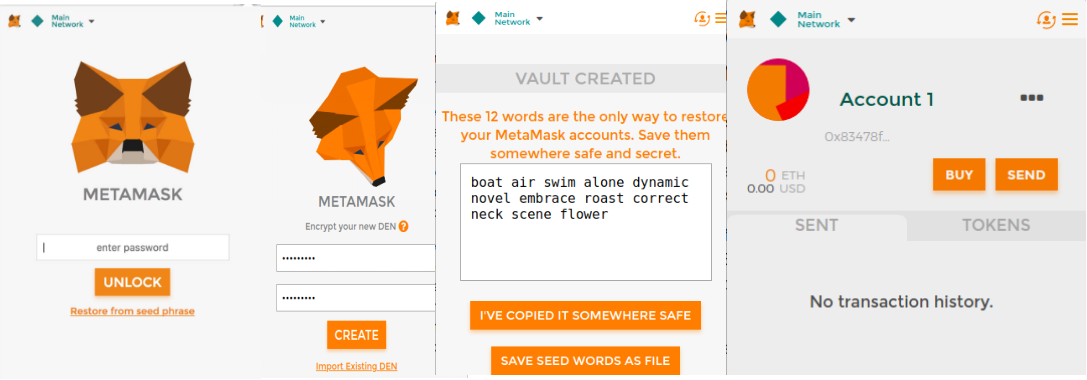
##### Metamask Browser Extension

###### The workflow



Metamask is the standard Ethereum web3 API. It is an addon of our web browsers such as FIrefox or Chrome. There for, installing Metamask is quite simple. With metamask, every user can interact with Ethereum blockchain (Main net, test net or even private blockchain) without becoming a full node. A light node user is not required to install Geth or Ganache to use Ethereum Dapps.To playing with Ethereum Dapps, users now only need to install Metamask addon on their web browsers, custom the RPC url of it and register account.

###### Functionality



MetaMask injects web3 object and convenience Web3.js library into the javascript context. It helps user to handle authorization and Dapp likely will not need to call sendRawTransaction - which requires user to manually valid a transaction using their private key. Any time a user makes a call that requires a private key to sign something (sendTransaction, sign), MetaMask will automatically prompt the user for permission, and then forward the signed request on to the blockchain (or return it to you, if it was a call to sign).

Just listen for a response, and when the blockchain RPC has received the transaction and broadcast it, you'll get a callback.The user does not have the full blockchain on their machine, so data lookups can be a little slow. Metamask is one of the Ethereum Wallet.

Metamask can be simply understood as an Ethereum wallet. It helps to protect your private key and sign a contract whenever you allow it to do. MetaMask provides a platform for managing Ethereum (or "ETH") accounts, and allowing ordinary websites to interact with the Ethereum blockchain, while keeping the user in control over what transactions they approve, through our website located at https://metamask.io/ and browser plugin (the "Site") — which includes text, images, audio, code and other materials (collectively, the “Content”) and all of the features, and services provided. The Site, and any other features, tools, materials, or other services offered from time to time by MetaMask are referred to here as the “Service.”

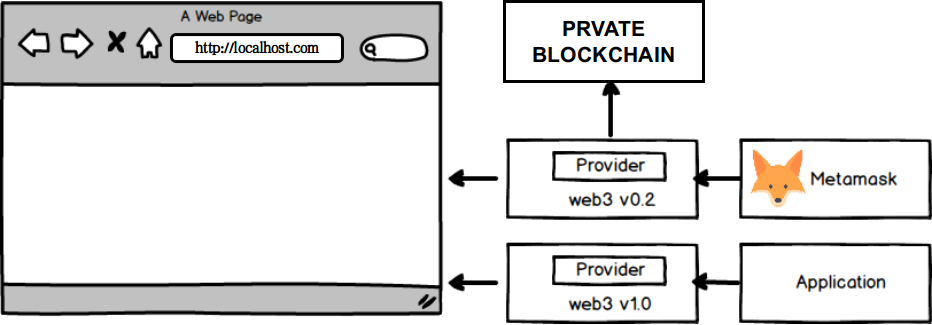
After Metamask is installed through Chrome Extension, it is able to use by clicking the “fox head” icon on the chrome toolbar



Users are now asked to enter a password, which protects their key by encrypting it. This means users are more secure. Choose a strong, easy to remember password. Do not share this password with anyone. Keep it in mind !

Users are now shown 12 words. This is a fancy private key which creates an infinite amount of private keys underneath it. Users need this to access their Ether. No one can recover it if they lose it, their computer dies, reinstall Chrome, uninstall the MetaMask extension, or reinstall OS.A better way is to store a second copy of these 12-word in a different physical location.

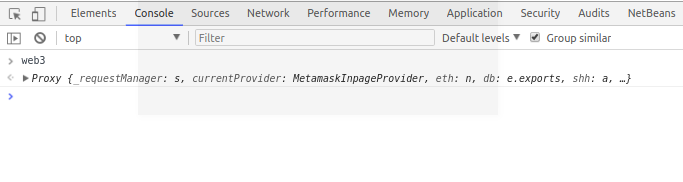
#### Web3 and web3 provider endpoint



Web3 is a JavaScript API that wraps RPC calls to help us interact with a blockchain in an easy way. It can be easily installed via npm ( Nodejs Package management).

In Web3 library, there is a component called provider. Provider is a bridge between our application and blockchain network. There are 2 web3 bridges : one for the Front End and the other is used by Metamask. Users can check by visit any website on that browser

After installing Metamask addon on the browser, it embed the web3 library in the browser and open the Develop Tools (for Chrome, Shift+Ctrl + I). In the Console tab, type web3 to see the result

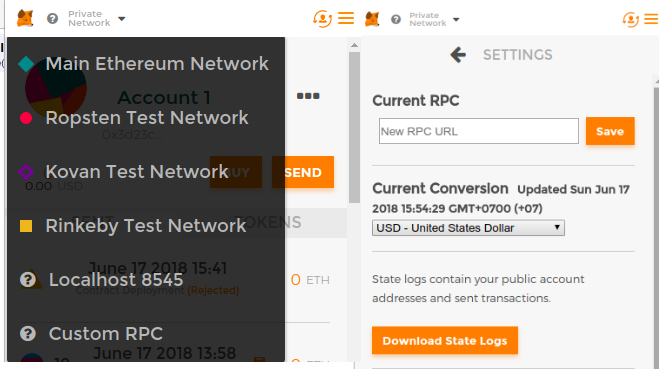


For the front end, Web3 should be manually installed using npm

npm install --save [web3@1.0.0-beta.34](mailto:web3@1.0.0-beta.34)

To make an app work on Ethereum, developers can implement the web3 object provided by the web3.js library. It communicates to a local node through RPC calls. web3.js works with any Ethereum node, which exposes an RPC layer.

##### Connect metamask to private blockchain network



As we using Ganache (or even Geth) as a node running on our network, Metamask must have its provider connect to the private blockchain through a node. Therefore, in the network list of Metamask (on the left corner of the GUI), we choose Localhost 8545 since the Ganache endpoint rpc is ***localhost:8545.*** If you want to connect to another endpoint, go to Custom RPC.

##### Adding web3

IT requires to create a web3 instance and set a provider. If there is already a web3 provider in the web browser (for example, Metamask), use it. Otherwise, a new web3 object must be created

if (typeof web3 !== 'undefined') {

web3 = new Web3(web3.currentProvider)

} else {

// set the provider you want from Web3.providers

web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"))

}

By Geth or Ganache default setup, the Web3 provider endpoint should be "http://127.0.0.1:8545" or "http://localhost:8545". The 8545 is the default RPC port of a node . It can be the RPC port of any running nodes in the private blockchain network. After that developers can use the API of the web3 object.

##### Using callbacks

Web3 API is designed to work with a local RPC node, all its functions use synchronous HTTP requests by default. But in calling smart contract function, developers have to make an asynchronous request and you can pass an optional callback as the last parameter to most functions. Here is the example:

contract.getDonators(function(error, result) {

if (!error) {

/\* do something with the result

}

else{

console.log(error);

});

#### Web application environment and tools

##### HTML, DIV and CSS

DIV is an element of html, div + css is a page layout approach, this page layout is

different from the traditional table layout approach, and achieves the w3c separation of

content and presentation.

DIV element is used within the document for the bulk of the content structure and

background elements. DIV start and end tags are used for everything between the block[25],

in which the characteristics of the elements contained in the DIV tag attribute to control, or

through the use of style sheets to format the block to be controlled.

Relative to the conventional table, using DIV + CSS technology web, the website will

be friendlier to search engine. It also separate website content and style, so that the

adjustment of the page and style has become more convenient.

##### JAVASCRIPT

JavaScript is an object oriented web page programming language used by web

application programmer.

Based on the html, using Javascript can develop dynamic interactive Web pages.

The emergence of Javascript enables web pages and users to achieve a real-time, dynamic,

interactive relationship, so that pages contain more active elements and more exciting content.

##### NODEJS

###### Nodejs and Node package management (npm)

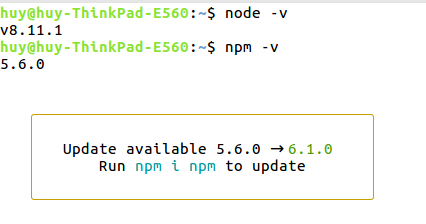
Node.js is a software platform which helps to build asynchronous and event-driven network

applications. It contains built-in HTTP server libraries which allow developers to

create their own web server and build highly scalable web applications on top of it. Node.js is the first program to be downloaded since its provides the platform on which D-fundraiser web application is built. It can be downloaded from the website:

<https://nodejs.org/en/download/>

After the download and installation, both nodejs and npm are now running on your computer. Their version can be checked by issuing commands from the terminal as illustrated in figure



###### Express

Express is a node module which provides a minimal and flexible framework for Node.js web applications. It works on top of the core node modules without hiding any of the features of Node.js. In addition, it provides robust and clean functions to add to the node modules so the development of Node.js application using Express is far easier than using the native node modules. The use of Express framework on top of Node.js helps to maintain clarity of the code. It also makes module integration easy to handle, and provides a solution structure for applications. Express is installed using the npm package manager issuing command.

$ sudo npm install express”

An Express server is made up of three building blocks: router, routes, and middleware. A web server’s core functionality depends on its excellent routing methods. In a client-server communication, a client requests some resources from the server, the server locates the resources and responds by sending the resources to the client. This is the core functionality of a web server and it requires excellent routing methods to serve the request. Express makes this tedious job really easy by allowing developers to create routes in simple structure. A route in Express is a combination of a HTTP verb and a path. The HTTP verb is generally one of the four HTTP methods: GET, POST, PUT and DELETE, and the path is the location of the resource (URI). A basic route in Express is created as below:

app.METHOD (PATH, HANDLER) where:

• app is an instance of express

• METHOD is an HTTP request method

• PATH is a route path (URI)

• HANDLER is the function which executes when the route is matched.

Middleware in Express are the functions that have the pattern function (req, res, next). The req is the incoming request from the client, res is the response from the server, and next is the callback function. Therefore, middleware functions execute any code inside it, handle request and response objects, end request-response cycle, and call the next middleware function. A current middleware function must always call the next middleware function, even in the case of incomplete request-response cycle to 16 avoid request hanging. A simple Express web server containing all three building blocks is shown in figure .

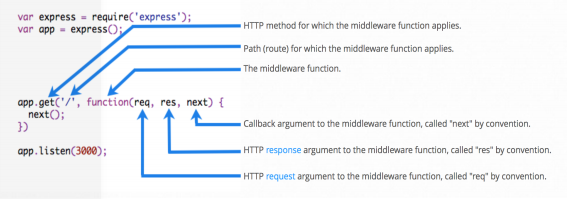


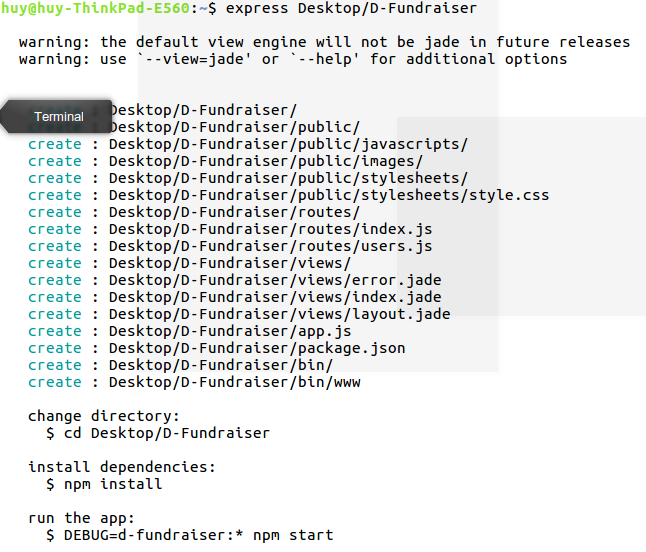
Figure 6 illustrates the three building blocks of an Express application: router, route, and middleware. The first two lines uses Node.js require() method to load express module in the application by creating the app object. The third line is a simple router with a route to ‘/’ location and a middleware function. The application listens to port 3000 for any request.

Express also provides a simple application generator tool for providing structure to our Node.js application. It can be installed from the CLI by issuing command.

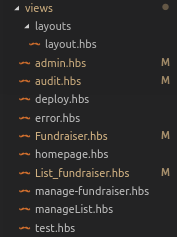
$ sudo npm install express-generator

It also provides the options for creating template engine to write HTML codes. To start new project and place its folder on Desktop, using this command Terminal

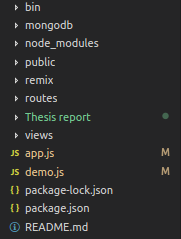
$ express Desktop/D-fundraiser



The application generator, by default, creates jade templates for the views, but this research using **Handlebars**.



The application structure created by the Express generator has separate directory for routes, views, and public-rendering files. However, the application structure is just one of many ways to structure an Express application. It can be easily modified during the application development process to meet the requirements of the application.



#### MongoDB

MongoDB is an open-source, non-relational, document database. It deviates from the need of creating Object Relational Mapping (ORM), for rapid application development. Unlike the relational databases, it does not contain columns and rows. However, the concept of rows still exits in MongoDB but it is called a document. A document is a set of fields and it can contain complex data such as lists, arrays, or an entire document. Each document contains an ID field which can be used as a primary key for a query operation. A set of documents is called a collection and MongoDB holds a set of collections. The format in which the MongoDB stores the data is called BSON, which stands for binary JSON. Since JSON is the JavaScript way of storing data, MongoDB works perfectly with the applications built with JavaScript stack.

Fundraiser stored as MongoDB document is illustrated in figure.........

{ \_id: 5b25d7558bcd1a0fd0efe4ff,

title: 'Raise for me', // Title

content: '<p>[Description]</p>', // Description

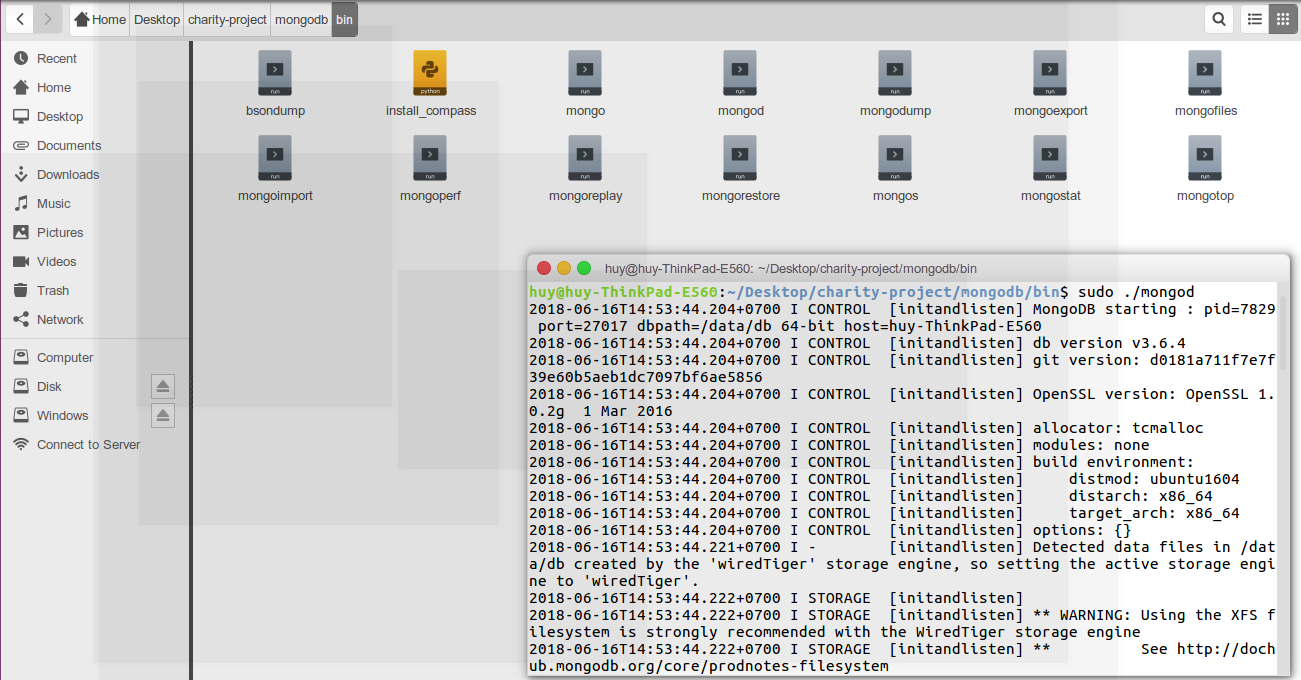
poster: '<image link>', // Image link

address: '0xdb2a26c095aa186fc941f6a875bb792c6b3f58f6', // Fundraiser Contract Address.

display: true } // Allow to display on the website or not.

MongoDB can be download from <https://www.mongodb.com/download-center?jmp=nav>. To run mongoDB, navigate to its original **folder/bin** and start Terminal with command

**$ sudo ./mongod**



The default mongodb url is "***mongodb://localhost:27017/***"

##### Monk

A tiny layer that provides simple yet substantial usability improvements for MongoDB usage within Node.JS. Monk is installed using the command

**npm install --save monk**

###### Syntax

Connect from project to mongoDB mydb database and funds collection



Here is how the we get the date (mongoDB documents) from that collection:



+ update: update the value of the key “display” of document that satisfies the specified query criteria in one atomic operation.

+ findOne: Returns one document that satisfies the specified query criteria. If multiple documents satisfy the query, this method returns the first document according to the natural order which reflects the order of documents on the disk. In capped collections, natural order is the same as insertion order. If no document satisfies the query, the method returns null.

+ find({}): Selects all documents in a collection and return them.

###### Features

Well-designed API signatures

Easy connections / configuration

Command buffering. You can start querying right away

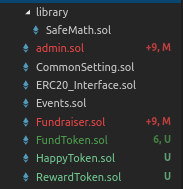
Promises built-in for all queries. Easy interoperability with modules

Auto-casting of \_id in queries

Allows to set global options or collection-level options for queries.

### Smart contract

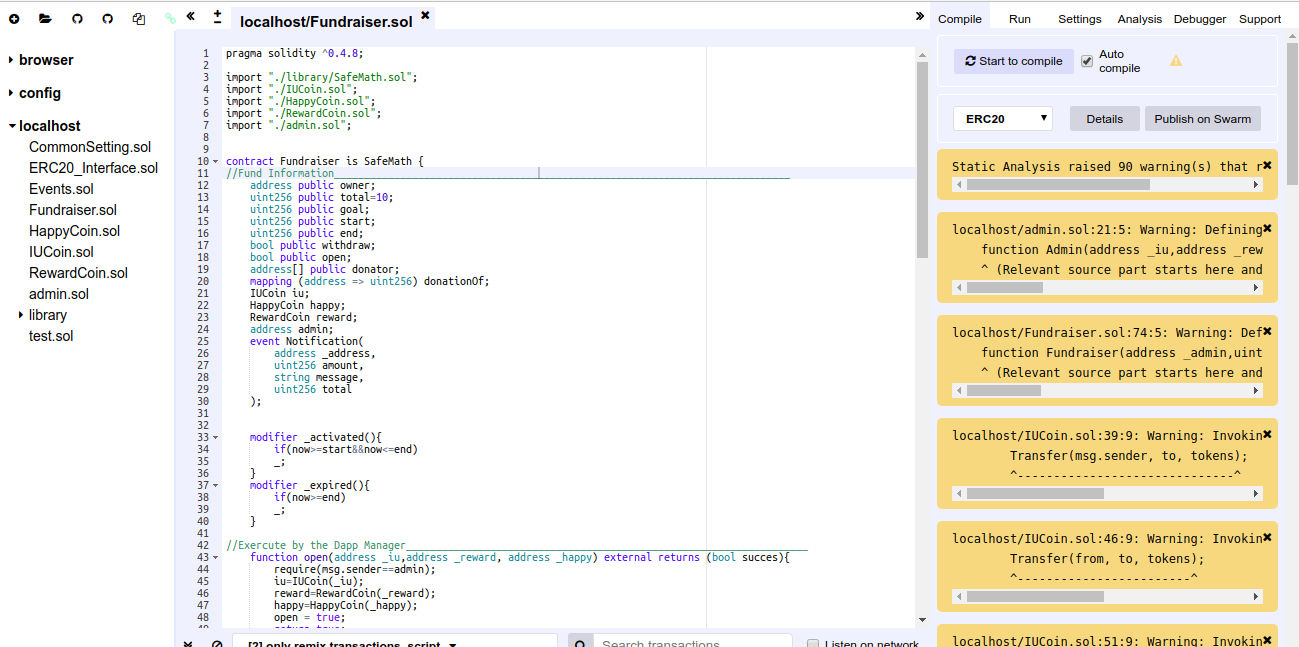
Smart contracts are written in Solidity, a contract-oriented, high-level programming language.



#### Remix IDE

Remix is a browser-based compiler and IDE that enables users to build Ethereum contracts with Solidity language and to debug transactions. Visit <https://remix.ethereum.org> to start coding a new Smart Contract. Remix consists of many modules and in this repository you will find the Remix IDE (aka. Browser-Solidity).

##### Connect to localhost



In Terminal, run command

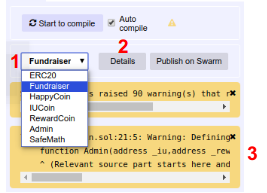
remixd -s Desktop/charity-project/remix - This navigates to the smart contract folder on your computer.

Interact with your file system from Remix. Click connect and find shared folder in the Remix file explorer (under localhost). Connection will start a session between https://remix.ethereum.org/#optimize=false&version=soljson-v0.4.24+commit.e67f0147.js and your local file system ws://127.0.0.1:65520. Hence, make sure your system is secured enough (port 65520 neither opened nor forwarded).



##### Compile Smart Contract

Remix IDE supports auto compiling. Click the  to compile your contract or you can let the IDE do it each time you choose a contract by ticking .



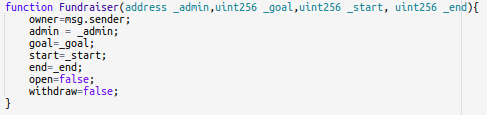
In figure....... Illustrates

1. List of compiled smart contract. You can choose whichever contract to look into its detail
2. Details: show a contract in detail.
3. Show warning or errors (if there is any).

In , there are 3 important fields we must paid attention to if we want to deploy a contract to the blockchain. In the WEB3DEPLOY



###### #1 Define input variable (constructor’s parameters)



###### #2 A new contract instance with all its methods and events defined in json format.

This JSON format definition of the smart contract is also called the contract’s Application Binary Interface or ABI. Application Binary Interface or ABI definition of a smart contract is the list of the contract’s functions and arguments in JSON format.

If an account wants to use a smart contract’s function , it uses the smart contract’s ABI to create a bytecode which along with a smart contract’s address on the network helps anyone access the functions of that particular smart contract.

###### #3 The owner of the contract is set and the bytecode of the contract to be deployed & the cost of deploying this contract to the Ethereum blockchain

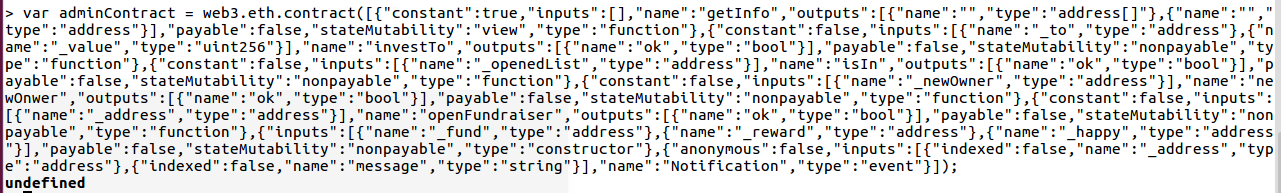
Bytecode you get when the source code in your smart contract .sol file is compiled.

##### Deploy smart contract to private blockchain.

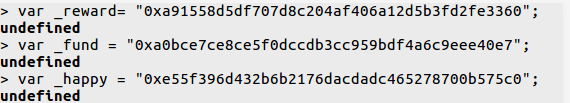
###### Using Geth

After having unlocked EOA, users are allowed to deployed smart contracts to private blockchain

Firstly, copy the ABI code and paste to the Geth console



Then, define all the initial variable



Finally, copy the bytecode along with gas that remix has already defined it for you:

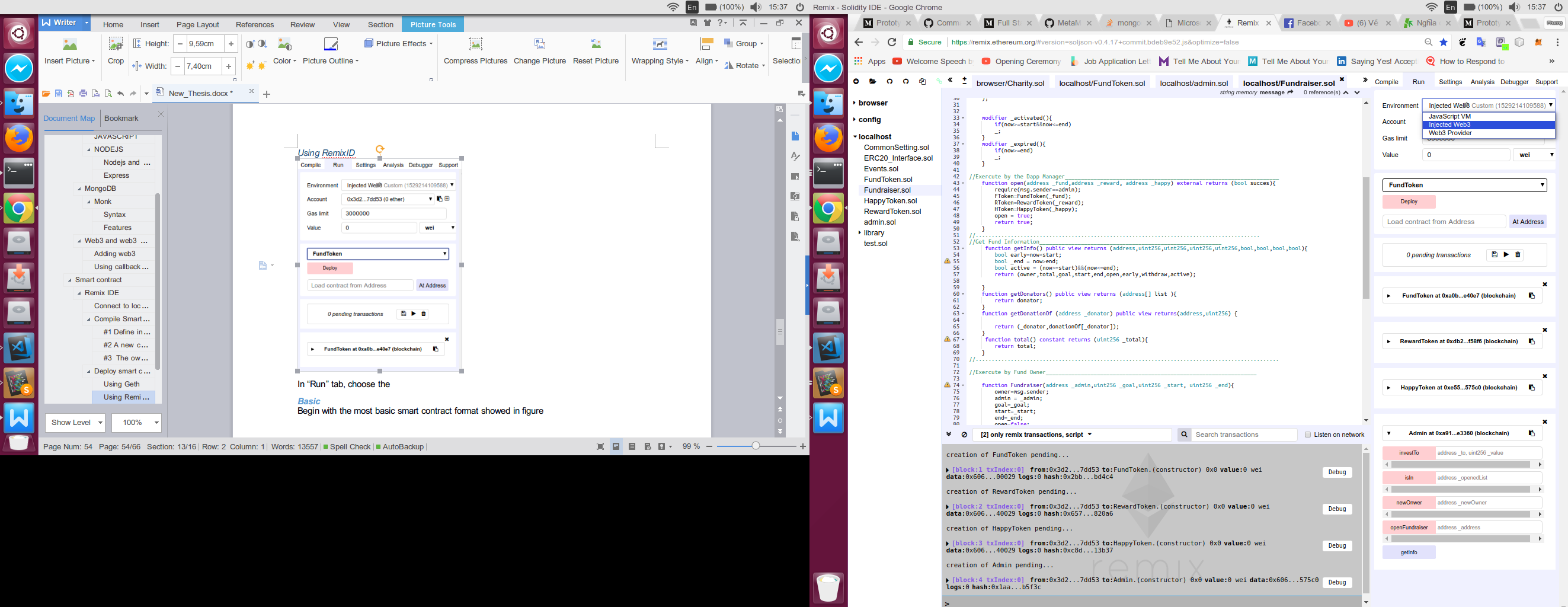


Start mining and wait until your contract is mined. The Contract Account will be returned after a while

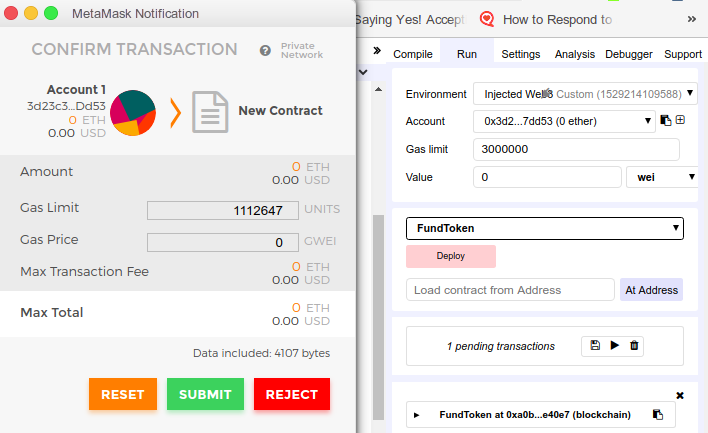


The Contract Account (address) is “0xcdf041a044feee0cc72eb52b8d7b2e4d42ab9262”

###### Using RemixID with Metamask Account



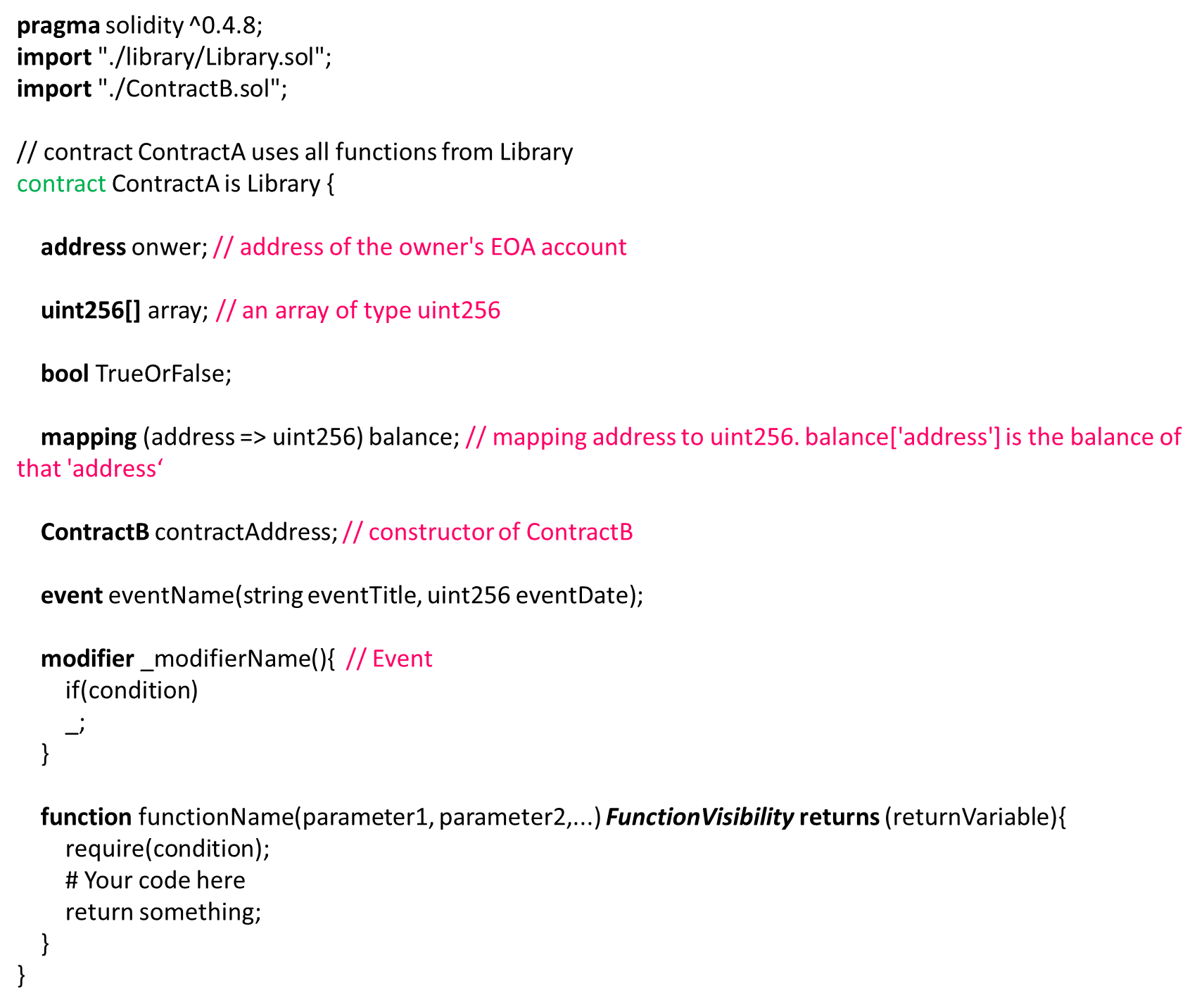
Connect Metamask provider to private blockchain. In “Run” tab, choose the Environment as Injected Web. Now, every transaction is made using Metamask.



In the figure... The FundToken contract will be deployed to the blockchain. As user click the Deploy button, Metamask asks for Submit transaction. After submitting , the transaction is mined and that contract is deployed to the private blockchain.

#### Basic

Begin with the most basic smart contract format showed in figure



The first line simply tells that the source code is written for Solidity version 0.4.8 or anything newer that does not break functionality. This is to ensure that the contract does not suddenly behave differently with a new compiler version. Contracts in Solidity are similar to classes in object-oriented languages. Each contract can contain declarations of State Variables, Functions, Function Modifiers, Events. Furthermore, contracts can inherit from other contracts.

##### Import

Contracts and libraries can be imported using **import** *source code directory.* A library provides build-in functions.

##### Functions

Functions are the executable units of code within a contract.

###### Function Visibility Specifiers

**public**: visible externally and internally (creates a getter function for storage/state variables)

**private:** only visible in the current contract

**external**: only visible externally (only for functions) - i.e. can only be message-called (via this.func)

**internal:** only visible internally

##### State Variables

State variables are values which are permanently stored in contract storage.

###### Data type

Like other static type languages such as C#, Solidity has Value types and Reference types

Value Types: Value Type variables are always passed by value. These variables are copied when they are used as function arguments or in assignment. These are basic data types provided by Solidity

*Boolean*: this is used to store true or false value.

*Integer*: Solidity support signed (***int***) and unsigned (***uint***) integers. In addition to better optimize memory Solidity allows specifing the size of the integer data types, such as ***uint8, uint16, …, uint256*** and ***int8, int16, …, int256***.

*Address*: Address data type is used to store an Ethereum address. It can contain the address of a EOA or a Contract Account. Ethereum address is a 20 byte value.

*Function:* Function Types are the types of functions

Reference Types: These are the complex data types that cannot be copied. Their memory or data location is passed around.

*String*: Strings are the dynamic sized array (see below for Arrays). This type is used to store an arbitrary length of string.

*Array*: Arrays can be fixed size(array[k] with size ‘k’) or dynamic (array[]) at the compile time. Solidity support multi-dimensional array and this multi-dimensional array is declared as array[][].

*Struct*: Structs are custom defined types that can group several variables. It let us define complex data types or user defined data types in solidity

*Mapping*: Mapping data types are the dynamic array of key value pair. Mappings are similar to hash table or dictionary in C# or other programming languages

##### Function Modifiers

Function modifiers can be used to amend the semantics of functions in a declarative way.

##### Events

Events are convenience interfaces with the EVM logging facilities.

#### ERC20 token

ERC20 token has the following method-related interfaced functions:

*totalSupply()*- Get the total token supply.

*balanceOf(address \_owner) constant returns (uint256 balance)* - Get the account balance of another account with address \_owner.

*transfer(address \_to, uint256 \_value) returns (bool success)* - Send \_value amount of tokens to address \_to.

*transferFrom(address \_from, address \_to, uint256 \_value) returns (bool success)* - Send \_value amount of tokens from address \_from to address \_to.

*approve(address \_spender, uint256 \_value) returns (bool success)*- Allow \_spender to withdraw from your account, multiple times, up to the \_value amount. If this function is called again it overwrites the current allowance with \_value.

*allowance(address \_owner, address \_spender) constant returns (uint256 remaining)* - Returns the amount which \_spender is still allowed to withdraw from \_owner.

Events format:

*Transfer(address indexed \_from, address indexed \_to, uint256 \_value)* - Triggered when tokens are transferred.

*Approval(address indexed \_owner, address indexed \_spender, uint256 \_value)* - Triggered whenever approve(address \_spender, uint256 \_value) is called.

Token contracts of D-Fundraiser system inherit the ERC20 Token Interfaced functions and implement them. Additionally, there are also some configurations and additions to make it compatible with the system.

###### Fund Token:

It is used for donating and sharing between donators.

*useToken(address \_from,uint256 \_amount) public returns(bool ok) –*  to use the \_amount of Fund Tokens of the EOA’s address “\_from” for donating and remove the corresponding amount from balance[\_from].

*investToken(address \_to, uint256 \_amount) public returns(bool ok) –* to invest the \_amount of Fund Tokens to the EOA’s address “\_to”.

###### Happy Token

Represent the funds of a fundraiser

*investToken*(address \_to,uint256 \_amount) public returns(bool ok) *–* to invest the \_amount of Happy Tokens to the EOA’s address “\_to”.

*useToken*(address \_from, address \_admin,uint256 \_amount) public returns(bool ok) – to move the \_amount of Happy Tokens from address ‘\_from’ to the administrator EOA after the fundraiser is done and the Happy Tokens had been exchanged to cash.

###### Reward Token

Reward token is represented how often a student participated in University fundraising activities. The more tokens have a student, the more activities he or she participated in, the higher the bonus point is. Reward token can also give a student special offers in some cases while studying in the University. Hence, they cannot be traded or sold. All related functions (*transfer,* *transferFrom, approve*, *allowance)* are removed from the contract.

In addition, a rewarding(address \_to, uint256 \_amount) public returns (bool ok) is created to reward the \_amount of Reward Token to address ”\_to”.

#### Fundraiser

Fundraiser contract is deployed to Ethereum Blockchain each time a person or an organization want to generate new fundraiser. The fundraiser’s owner is only allowed to withdraw all the tokens when a fundraiser expired.

##### States

|  |  |  |
| --- | --- | --- |
| address | owner | the address of the fundraiser owner (EOA public key). |
| uint256 | total | The total donation amount |
| goal | The fundraiser goal |
| start | Start date (unix tmestamp format) |
| end | End date (unix tmestamp format) |
| bool | isWithdrawn | True if the fundraiser has been withdrawn. Otherwise, False. |
| isSigned | True if the fundraiser has been signed. Otherwise, False. |
| address[] | donator | List of addresses of donators. |
| mapping(address => uint256) | donationof | The amount of Donation of a donator. |
| fundtoken | Ftoken | Fund Token contract account |
| happytoken | Htoken | Happy Token contract account |
| rewardtoken | Rtoken | Reward Token contract account |

##### Modifiers

\_activated() – function with this modifier only runs if the fundraiser is in its service life (after start date and before end date).

\_expired() – function wwith this modifier only runs if the fundraiser is expired.

##### Functions

###### Executed by Administration Contract Account

signed(address \_fund, address \_happy, address \_reward) external returns (bool signed) – Sign a fundraiser contract account.

###### Executed by Fundraiser Owner Account

Fundraiser(address \_admin,uint256 \_goal,uint256 \_start, uint256 \_end) – constructor of the Fundraiser Contract; use initialize a fundraiser.

ownerWidthdraw() \_expired returns(bool res) – Widthdraw fund.

###### Executed by Donator Owner Account

donate (uint256 \_value) \_activated returns(bool res) – donate using charity tokens

###### View functions

getInfo() public view returns (address,uint256,uint256,uint256,uint256,bool,bool,bool,bool) – Return the owner address, total fund, the goal, the start date, the end date, isSigned, isEarly, isWithdraw, isActive.

getDonators() public view returns (address[] list ) – return the list of donators of a fundraiser

getDonationOf (address \_donator) public view returns(address,uint256) – return the address and the donation of a donator.

total() constant returns (uint256 \_total) – return the total fund.

##### Events

Notification(address \_address, uint256 amount, string message, uint256 total) – triggered each time function *donation()* is executed.

#### Administration

##### States

|  |  |  |
| --- | --- | --- |
| Address | owner | the address of the fundraiser owner (EOA public key). |
| address[] | FundraiserList | List of addresses of signed fundraisers. |
| fundtoken | Ftoken | Fund Token contract account |
| happytoken | Htoken | Happy Token contract account |
| rewardtoken | Rtoken | Reward Token contract account |

##### Modifiers

\_owner() – function with this modifier is only executed by the contract owner.

##### Functions

Administration(address \_fund,address \_reward,address \_happy ) – the constructor of the Administration contract.

signFundraiser(address \_address) \_owner public returns (bool ok) – sign a fundraiser by its address.

investTo (address \_to, uint256 \_amount) \_owner public returns (bool ok) – invest the \_amount of Charity token to EOA address “\_to” .

###### View functions

getInfo() public view returns(address[],address ) – return the list of signed Fundraisers.

#### Interacting with deployed smart contracts

To call a function of a smart contract on Ethereum private blockchain, we need 2 things: Its ABI and its address(Contract Account).

Firstly, we create a contract instance: contractInstance = contractABI.at(contractAddress)

To call function abc() of that contract instance, using this syntax: contractInstance.abc().

### IMPLEMENTATION AND RESULT

#### Setup

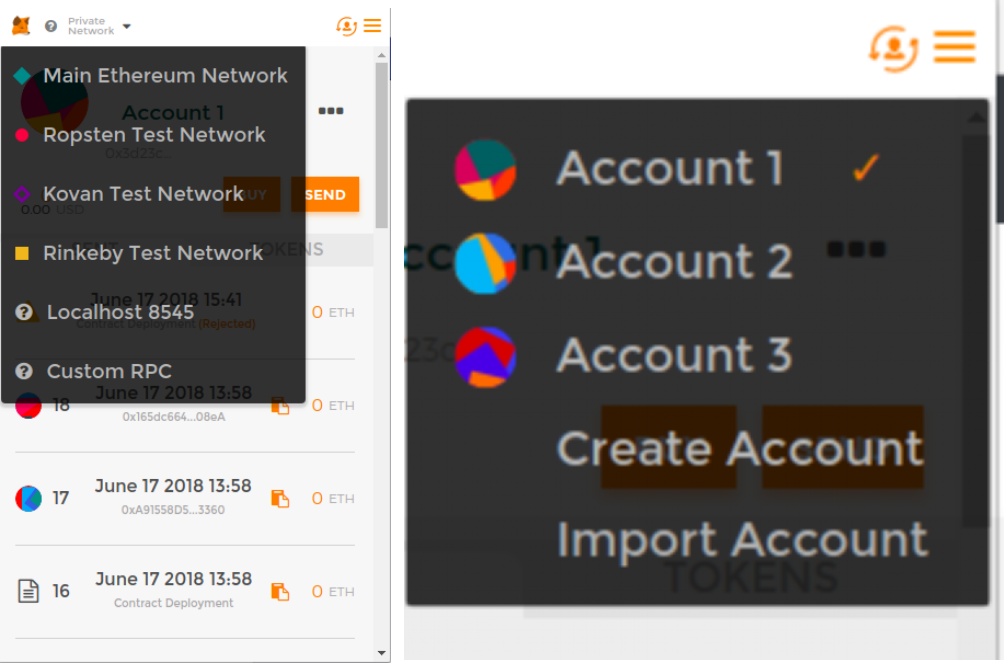
##### Start Ganache

Ganache is used instead of Geth because it is faster and easier. We want our transactions to be mined as quick as possible in order to test D-Fundraiser workflows. Ganache is run on **localhost:8545**



##### Setup Metamask

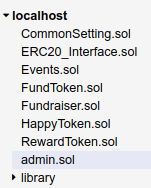
In Metamask, select Localhost 8545 to connect it to Ganache private blockchain.



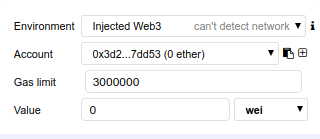
Click  to switch between accounts.

##### Deploy contracts

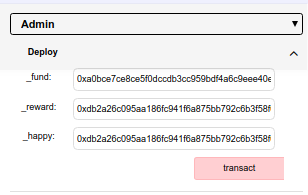
Connect Remix to contracts folder in local host.



Choose Injected Web3 for the Environment and the administrator account. In this test, **The Administrator Account** is “0x3d23c35f3cc70318c51e4b925501cc2f4e87dd53”



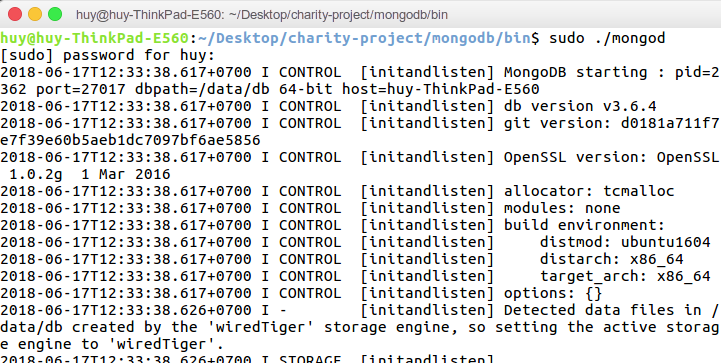
There are 3 contracts we must deploy before deploying the Aministration contract: FundToken, RewardToken and HappyToken. We then take their address and use them to deploy the Aministration contract.



The table below show the addresses of Contract Account

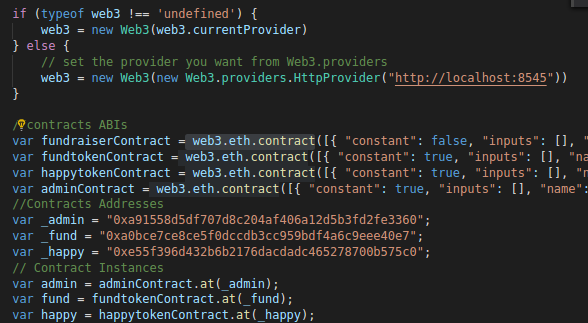
|  |  |
| --- | --- |
| FundToken | 0xa0bce7ce8ce5f0dccdb3cc959bdf4a6c9eee40e7 |
| RewardToken | 0xdb2a26c095aa186fc941f6a875bb792c6b3f58f6 |
| HappyToken | 0xe55f396d432b6b2176dacdadc465278700b575c0 |
| Admin | 0xa91558d5df707d8c204af406a12d5b3fd2fe3360 |

##### Start mongoDB

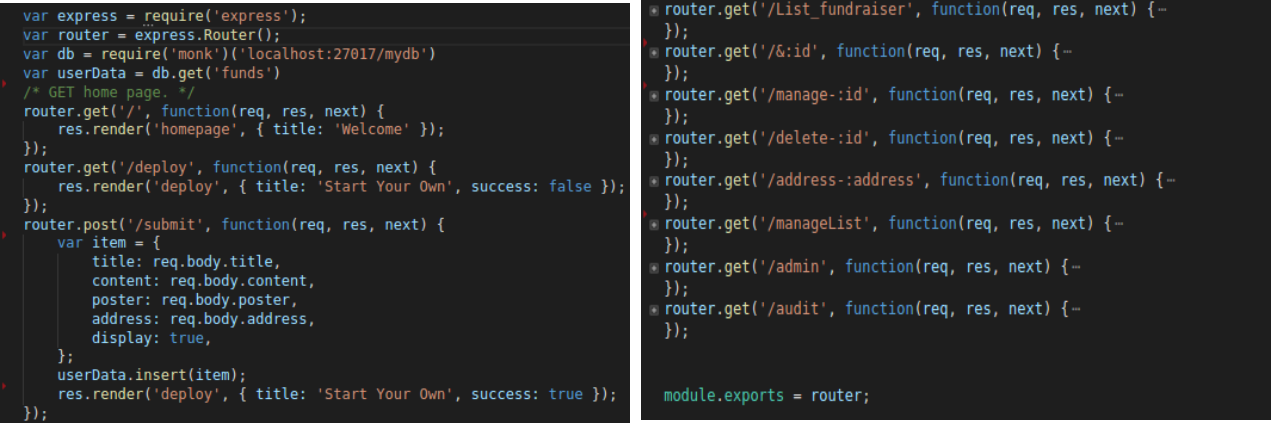
In mongoDB origin folder/bin, start Terminal. Use **$sudo ./mongod** command to start MongoDB database server.

##### Setup communication

There mus be a Web3 provider to help us talk to the private blockchain. We put ABIs and Addresses of contracts that we want to interact with in a .js javascript file or in the code of a script in HTML.



##### Routing and database query



A index.js file is programmed in order to navigate users each time they send request to the local web-server.

To connect application to mongoDB, we require the module ‘monk’.

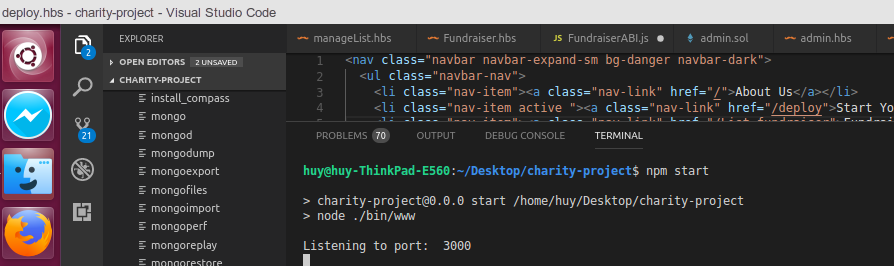
var db = require('monk')('localhost:27017/mydb')

##### Account and role

|  |  |
| --- | --- |
| Administrator  (Account 1) | 0x3d23c35F3CC70318c51E4b925501cc2f4E87Dd53 |
| Fundraiser owner  (Account 2) | 0xCcFC7FaFf1d00cC0C63928ba42F8052af98a9573 |
| Donator  (Account 3) | 0x9fbEA6924AAF34B5413632CCAE001d1278830505 |

##### START PROJECT

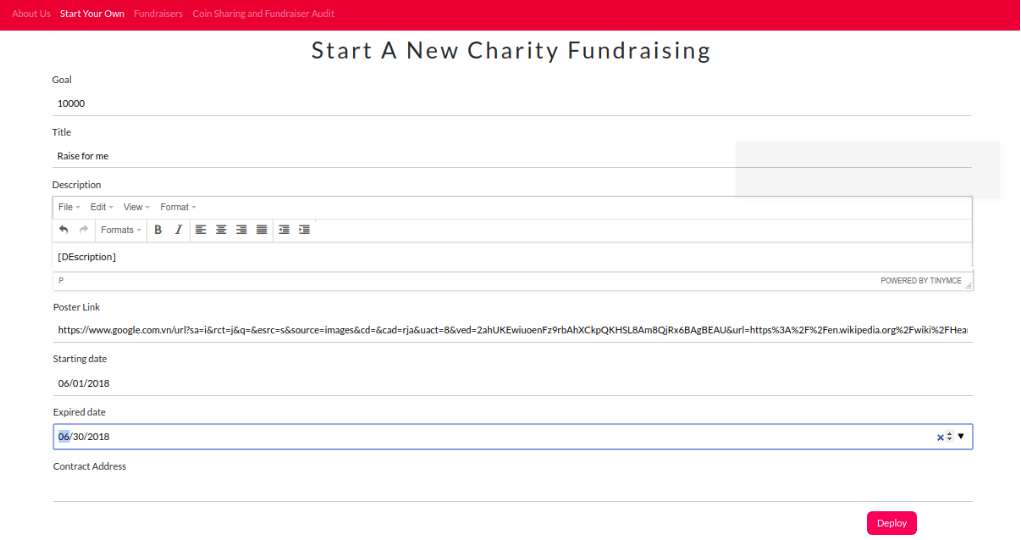
In Visual Studio Code, start project by command **npm start** in tab **Terminal**



The D-Fundraiser web application is running on **http://localhost:3000/**

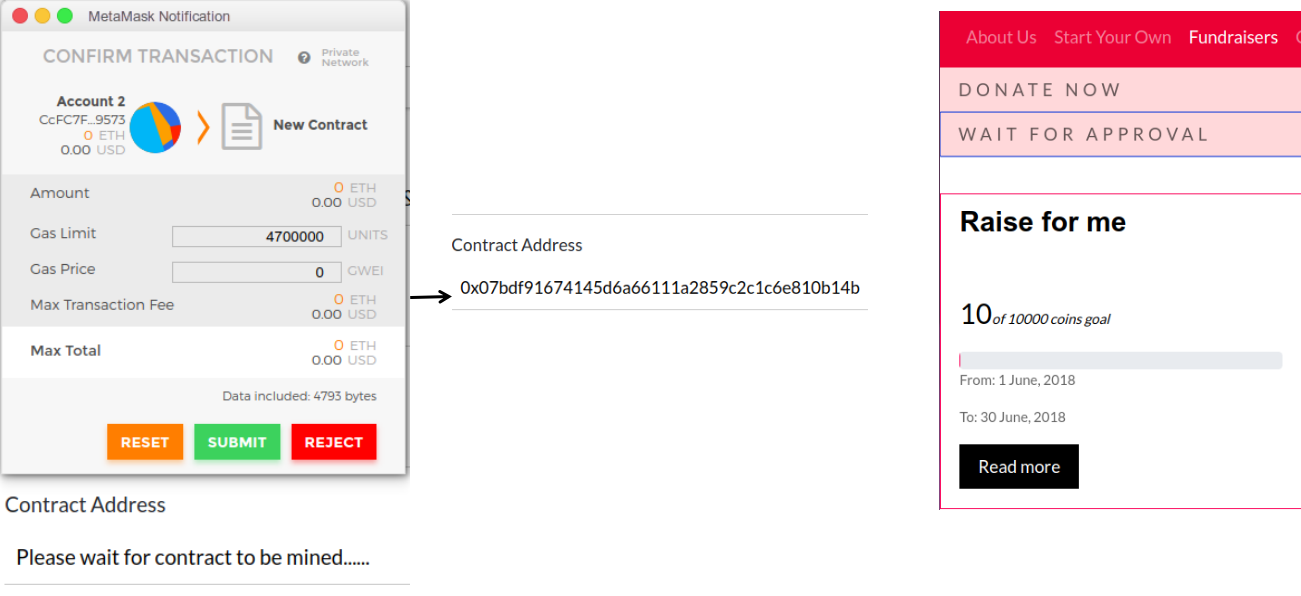
#### Testing Workflow

First, use the Owner Account to generate a new fundraiser. Go to <http://localhost:3000/deploy>

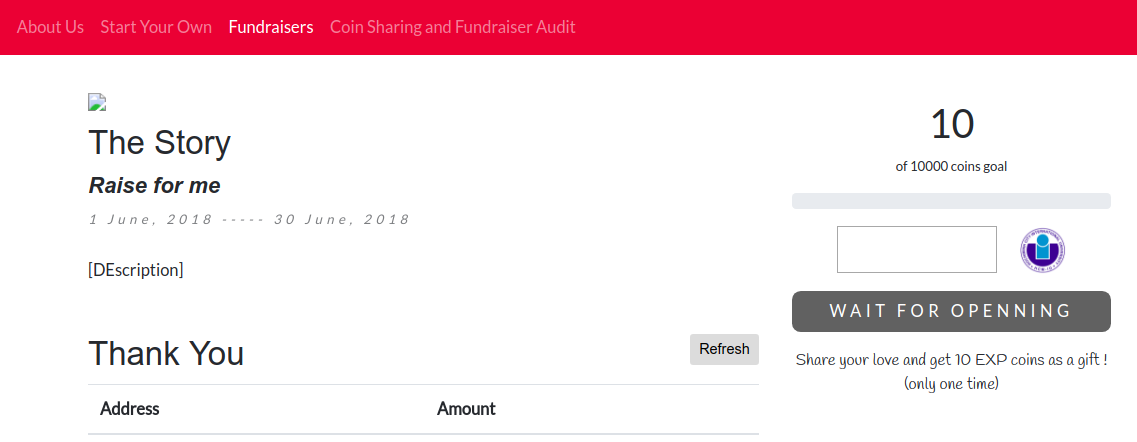


Click button to deploy a new Fundraiser contract to the blockchain.

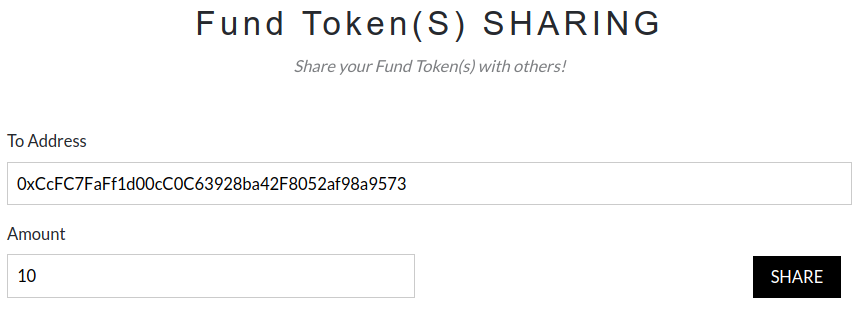
Wait for a while until it returns the Fundraiser address (Contract Account). Click Register button to push fundraiser to the website. Go to <http://localhost:3000/List_fundraiser> to see the fundraiser is displayed in a waiting list.



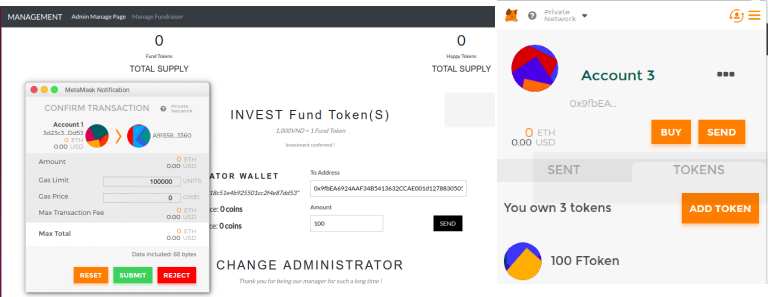
Here is the Fundraiser GUI.



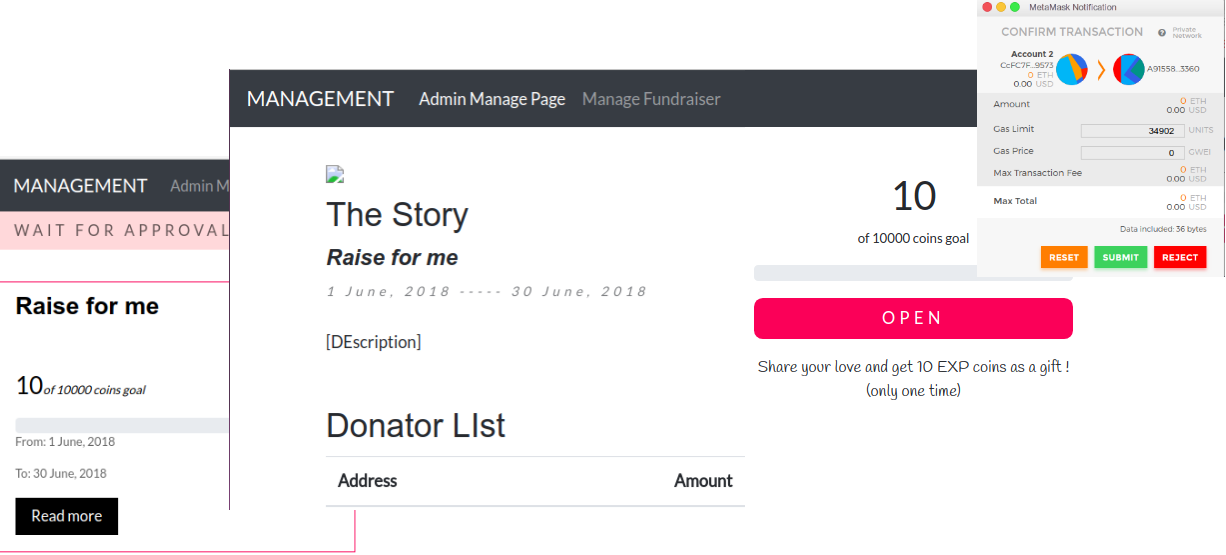
Sharing Token can be done by visiting <http://localhost:3000/audit>



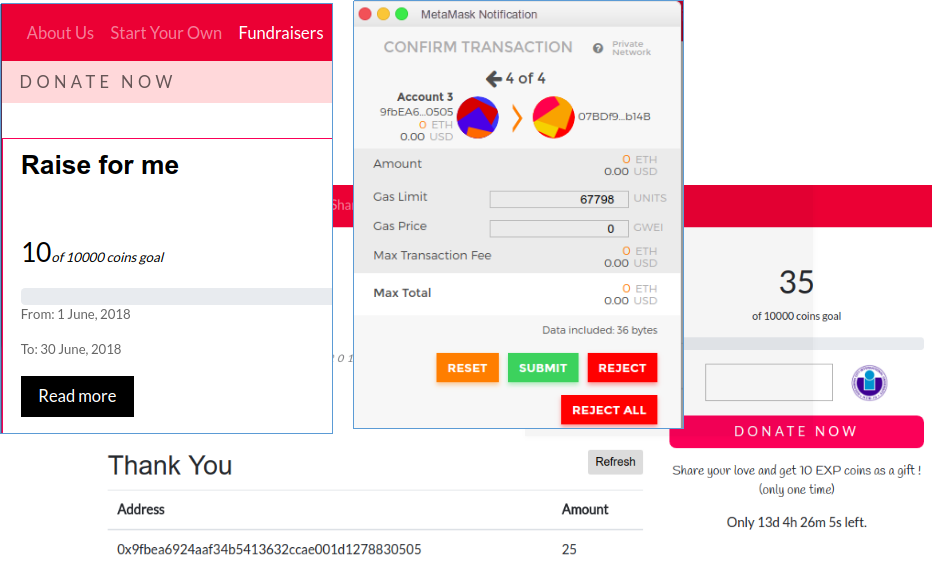
Switch to administrator account in Metamask. Go to <http://localhost:3000/admin> to Invest 100 FundTokens to Donator account. There should be a transaction from administraction account to the FundToken Contract Account in order to call its **investToken()** function. Donator possesses 100 Fundtoken in account.



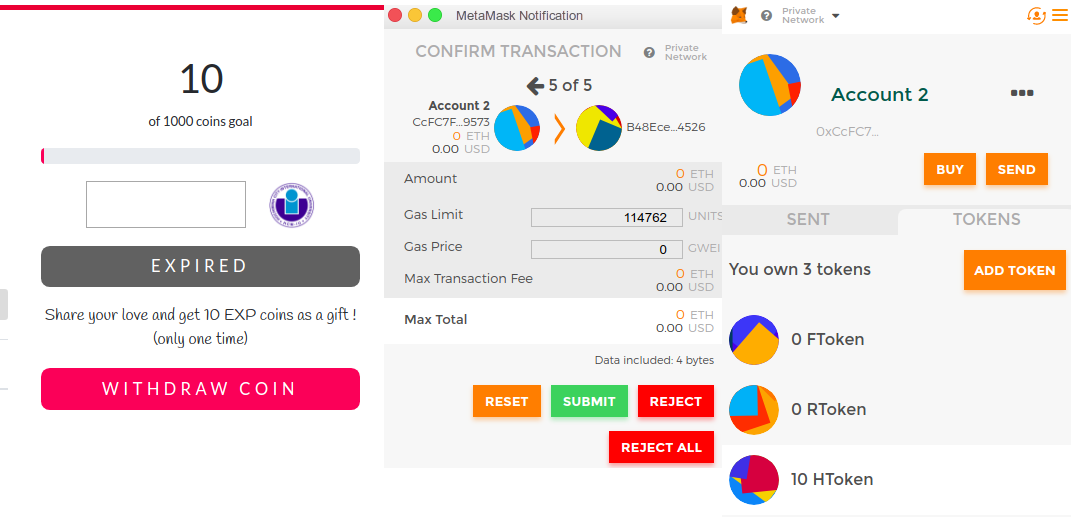
Administrator opens a fundraiser by visiting <http://localhost:3000/manageList> and check for waiting fundraisers. To open a fundraiser for donating, administrator has to make a transaction to Admin Contract Account and call the **signFundraiser()** function



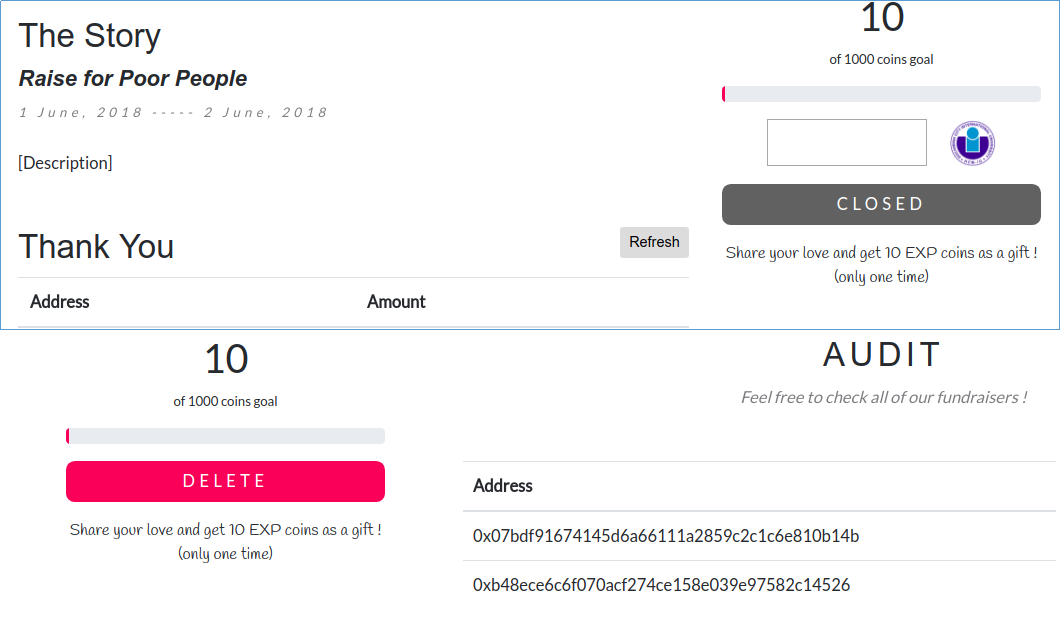
Donator can donate singed fundraiser by input the amount token and click . A transaction is sent from donator account to fundraiser contract account calling **donate()** function. The FundToken amount in Donator account will appropriately decrease.



With an expired fundraiser, the owner can withdraw the fund if want to by clicking button to generate a transaction from The Owner Account to the Fundraiser Contract Account . All the HToken will be sent to the owner account and the fundraiser will be closed.



Expired fundraiser can be delete from GUI, but user still can fiind it from <http://localhost:3000/audit> because all the signed fundraiser addresses is stored in the Administration Contract Account on the blockchain. Hence, they will not be lost under any circumstances.



# CHAPTER VI

## CONCLUSION

## 1. Summary

### The benefits of Ethereum mechanism

Web applications programmed not only on Etherem but also on Blockchain platform are very promising. Computing across the entire Ethereum network is not done to make everything more efficient. Reversely, this process makes computation on Ethereum far slower and more costly than on a computer. Every node runs the EVM in order to keep the consensus across the blockchain. This decentralized consensus provides Ethereum extreme levels of fault tolerance, no service interruption, and makes data stored on the blockchain forever unchangeable.

Ethereum, also a platform, is suited for applications that automate direct interaction between peers or facilitate coordinated group action across a network such as applications for coordinating peer-to-peer marketplaces, or the automation of complex financial contracts. While Bitcoin allows individuals to exchange cash without involving any intermediary, Ethereum could do more. It helps financial interactions or exchanges any complexity being carried out automatically and reliably using code running on Ethereum. Furthermore, any systems, which trust, security and permanence are important such as asset-registries, voting, governance, etc could be massively impacted by the Ethereum platform.

Not only Ethereum platform but also blockchain technology will start a new era of decentralized, **Immutable and indelible, Transparent but private web applications.**

#### Decentralized

The purpose of creating smart contracts is to rebuild a community exchange system that does not require a third-party such as bank, middle organizations, etc. Programmers can write a smart contract which contains conditions and put into a blockchain to make them completely distributed. This technique could remove the role of the middlemen or a third-party, which means no one is in control of others.

Are smart contracts trustful? Because they are stored on a blockchain they inherit some sophisticated properties. Smart contracts are immutable and distributed. Once a smart contract is created, it can never be changed. Hence, no one can go behind the back and edit the code of a smart contract. The output of a smart contract in a blockchain is validated by everyone on the network, which means nobody could force one smart contract to release a wanted output because other nodes on the network will spot this attempt and mark it as invalid. This ability is related to smart contract distributed property. Tampering with smart contracts becomes almost impossible. Smart contracts can be applied to many different things: on crowdfunding, automatic payments, processing claims of insurances, delivery payments and so on.

#### Immutable and indelible

Once transactions are committed to the blockchain, there is no going back to edit or delete it. The hash of a block in a blockchain can be comparable to a fingerprint. It identifies a block and all of its contents. Just as a fingerprint, a hash is always unique. Once a block is created, its hash is calculated. If there is any change inside the block, it will be easily detected because the hash of that block is also changed. Therefore, it no longer is the same block. In a blockchain, if any block is tampered, the hash of that block will be recalculated. In turn, that will make its adjacent block and all the following blocks invalid since they no longer store a valid hash of the previous one. Hence, changing a single block will cause all the hashes of other blocks to be recalculated in order to make the blockchain valid again.

Computers these days are so sophisticated that can produce hundreds of thousands of hashes per second. The recalculation of all hashes of other blocks as a consequence of tampering with a block can be done easily. Now, it is time to check for the Proof-of-work. A blockchain has a mechanism to automatically adjust the Difficulty Target in a block in order to remain the time required to calculate the appropriated block header hash. The valid proof-of-work is based on the correct header hash within a time period. For example, in Bitcoin blockchain, it takes at least 10 minutes to mine a new block. A Bitcoin mining software running on a node computer would automatically detect the performance of the hardware and adjust the Difficulty Target of the block. So that, the miner of that node could not mine a new block less than 10 minutes. Even if the miner successfully mine the block less than the time, that block is finally broadcast to the whole network for checking validity. If the Proof-of-work is cheated in mining process, it could not reach the Consensus across the whole network because there is no proof that the miner has worked on that block for a sufficient time. So if a person want to tamper with block N in a blockchain requiring 5 minutes to solve for the proof-of-work, he or she has to recalculate the hash of ( N-1) previous blocks for at least T = ( N-1)\*5 minutes, take control of more than 50% number of nodes on the network. By doing that, the tampered block could have a little chance to be validated. Besides, this is the parts where a dishonest miner can be found out. If there is any invalidation, the block will be unaccepted and the miner would have wasted his time and computing power. Messing with blockchain seems to be impossible!

#### Transparent but private.

All transactions on the blockchain are traceable, back to the very first genesis block. As a shared ledger, it is open everyone to view or audit. But how privacy and transparency can effectively coexist. The transaction can only show the sender and receiver public key, the data which is hashed and related financial information. There is nothing personal here. Therefore, it is total private but still transparent.

### conclusion

All this isn't to say that blockchain applications are the silver bullet to all the webs problems, since they do come with their own limitations. But blockchain applications do provide a new level of baseline guarantees that aren't found in other popular systems, like superior fault tolerance and trustless execution. In this context, trustless execution means not having to trust any person or organization to execute code as your wrote it. People can use cryptography or math to provide a guarantee of the execution of their code.

## Future work:

The blockchain technology should not stop at financial applications. Developers are working on new solution adapted blockchain technology in various fields that help to improve human life such as tracking the products’ origins in Supply Chain and Management or converting real properties and money into digital asset.

Furthermore, some experts predict that the large enterprises will build their own personal blockchain networks. According to the report of the enterprise project, “Suppliers and customers cannot and would not join the private blockchain for every one of their business partners. And, the long-term future of the blockchain relies on the ability of companies in order to conduct private business over a public.”

# LIST OF REFERENCES

<https://blockgeeks.com/guides/what-is-hashing/> - what is hahsing, hash function

<https://en.wikipedia.org/wiki/Merkle_tree> - merkel tree

<https://www.coindesk.com/information/what-is-blockchain-technology/> - what is blockchain technology

<https://en.wikipedia.org/wiki/Ethereum> - Ethereum and smart contract

<https://github.com/MetaMask/metamask-extension> - Metamask workflow

<https://theethereum.wiki/w/index.php/ERC20_Token_Standard> - ERC20 token standard

<https://hackernoon.com/heres-how-i-built-a-private-blockchain-network-and-you-can-too-62ca7db556c0> - how to build a blockchain network

<https://nodejs.org/en/> - Nodejs

<https://safenetforum.org/t/where-do-decentralized-applications-store-their-data/13616> - Dapp database.

<https://github.com/ethereum/wiki/wiki/JavaScript-API> - Ethereum web3 JavaScript API

<https://medium.com/@mvmurthy/full-stack-hello-world-voting-ethereum-dapp-tutorial-part-1-40d2d0d807c2> - full stack of DAPP development