CROWDSOURCE FUNDING SOLUTION BASED ON BLOCKCHAIN TOKENS

A Project

Presented to the faculty of the Department of Computer Science

California State University, Sacramento

Submitted in partial satisfaction of
the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

by

Sumukha Shegakula Nagaraj

FALL
2018

CROWDSOURCE FUNDING SOLUTION BASED ON BLOCKCHAIN

TOKENS

A Project

by

Sumukha Shegakula Nagaraj

Approved by:

_____, Committee Chair
Dr. Jinsong Ouyang, Ph.D.

_____, Second Reader
Dr. Haiquan Chen, Ph.D.

_____
Date

Student: Sumukha Shegakula Nagaraj

I certify that this student has met the requirements for format contained in the University format manual, and that this project is suitable for shelving in the Library and credit is to be awarded for the project.

_____, Graduate Coordinator _____
Dr. Jinsong Ouyang, Ph.D. Date

Department of Computer Science

Abstract

of

CROWDSOURCE FUNDING SOLUTION BASED ON BLOCKCHAIN

TOKENS

by

Sumukha Shegakula Nagaraj

In this generation, when an entrepreneur wants to bring his ideas to life he will start searching for funds. There are people around the globe who have small amounts of money that they can invest in ideas that might grow bigger or solve problems faced by them in real world, but for an entrepreneur it's a very challenging task to reach out these investors. To solve this problem there are crowdfunding sites such a Kickstarter and Indiegogo which act as a platform for these entrepreneurs to pitch their ideas to investors. These trusted third-party sites act as escrow for the investments to help the investors from being scammed.

Our application replaces intermediaries such as Kickstarter to provide a platform for entrepreneurs to raise funds and for investors to invest in projects that interest them. The entrepreneurs can raise funds by issuing tokens to investors investing in their project. These tokens are immutable and impossible to forge. There are smart contracts built according to the investors and Entrepreneurs to mediate the risk taken by the investors. Smart contracts are deployed on blockchain, they contain steps and policies that are executed according to the agreement and pay incentives to the investor as the idea is built and sold to public.

This application is built using Remix IDE, Visual studio code editor, employing languages such as Javascript and solidity. It uses Ethereum blockchain network for storing and processing smart contract transactions.

_____, Committee Chair
Dr. Jinsong Ouyang

_____
Date

DEDICATION

To my family

Who instilled courage in me to believe anything I dream is possible to achieve

To my friends

Who helped me to accomplish my dreams

"Inspiring passion in family and friends has more

enduring value than just staying alive for them"

Finally to my almighty God

ACKNOWLEDGEMENTS

This project has helped me widen my technical expertise and would not have been possible without the assistance of my mentors and guides. It has been an excellent learning experience building this application. I would like to thank my project guide Dr. Jinsong Ouyang for giving me this opportunity to build this application and enhance my application developing skills required for gaining a competitive edge over other candidates in the job market. Dr. Jinsong Ouyang has been supportive and motivating to drive me towards completion of this project. All his valuable inputs has helped me to navigate through the project building phase and improve my approach and technologies. I would like to acknowledge and appreciate my second reader Dr. Haiquan Chen for his efforts in reviewing the project report and for giving valuable feedback.

I also would like to thank Jay Bibodi for his valuable suggestions for this application. Finally, I would like to acknowledge my sister's support and motivation which helped me to keep my focus intact towards achieving my goal. There are innumerable friends who helped me during this project, I am grateful to them.

TABLE OF CONTENTS

Page

LIST OF TABLES

Tables Page

LIST OF FIGURES

Figures Page

INTRODUCTION

Crowdfunding is a decentralized application based on Ethereum blockchain platform that allows users to invest money to the campaigns that interest them. By using blockchain, we can make sure that the investors engage in low-risk support of new ventures and venture creators can gain more supporters globally making it easy for them to raise large amount of funds in minimal time. Especially in blockchain world at present, there are lot of projects created by individuals or small-distributed teams that want to raise funds by issuing tokens to the investors. Crowdfunding platform simplifies the whole idea of raising capital with help of global public that might be interested in the campaign for an incentive that is profitable to the investor.

1.1 Traditional fundraising problem and solution

Traditionally, banks and venture capital funds are the main way to fill the gap in funding chain. A startup founder would approach a bank or a venture capitalist with his project pitch for funding and if they are interested in the project then the bank or venture capitalist will fund it for some returns, such as equity in case of venture capitalist or loan interest amount in case of banks. However, this way of raising funds has limitations associated with it. This process of fundraising requires huge amount of time, money and valuable resources that project creators from developing countries or remote places do not have access. If we consider bank loan as the solution for funding a project then the bank might become a bottleneck in the project as a bank needs concrete proof of how the project generates revenue and also it requires the founder to provide a collateral for the amount loaned.

Crowdfunding is the solution to the issues with traditional approach of fund raising. In crowdfunding, a person or a team with an idea to solve a problem can raise capital from a huge number of individuals who are interested in funding the venture. Crowdfunding provides a platform to anyone who has an idea to pitch in front of investors ready with money to invest. The major benefits of crowdfunding are:

- Access to large number of accredited investors who can see and interact with the campaign.
- Get a top-level view of traction, addressable market and value proposition of the idea.
- Presenting the concept to many investors helps the start-up founder to validate and refine his offerings.
- The best thing about online crowdfunding is its ability to centralize and streamline the campaign creator's fundraising efforts by building a single, comprehensive profile that targets to all the potential investors eliminating the need to pursue each one of them individually.

1.2 Purpose

Few established crowdfunding platforms such as Kickstarter and Indiegogo have revolutionized the start-up world with the flexibility and efficiency in raising funds. Blockchain based crowdfunding might be the next step in evolution of fundraising platforms assisting start up founders in the journey of building their dream idea.

The major issues with these established crowdfunding platforms are that they are centralized bodies controlled by a corporation charging high fees and influencing campaigns. Blockchain based crowdfunding platform can help this process by decentralizing the funding model from the likes of Kickstarter and other companies.

Blockchain's distributed ledger helps in getting rid of the centralized intermediaries such as Kickstarter and Indiegogo that take huge amounts of money from a campaign as a maintenance fee. Blockchain crowdfunding is a purer form of crowdfunding as it removes any intermediaries between the backers and the start-up [1].

Crowdfunding dapp allows creators to post their campaigns and then soliciting funds from a community of interested people. Once the funding is successful, it returns the backers with campaign specific tokens or if the campaign fails, it returns the backer with his investment. All these multiple transactions are accounted for and kept track of by blockchain, immutable distributed ledger, and thus it is impossible to forge. Blockchain also gets rid of the influence and manipulation done by the centralized crowdfunding platforms that have more than required access to the campaigns running in their platform.

BACKGROUND

This chapter aims to provide information on technologies and concepts used in Crowdfunding dapp. Crowdfunding dapp is based on the Ethereum blockchain. Several aspects make Ethereum a revolutionary blockchain that enables developers to program it to work according to different use cases. Ethereum has various components and mechanisms, which make it programmable. It is essential to understand Ethereum before we get any further into our dapp.

2.1 Bitcoin and the Blockchain

Bitcoin garnered enormous popularity in recent years and has become a household name. However, most of the people are not aware of the underlying technology blockchain that powers bitcoin [2]. Bitcoin is a decentralized digital currency not regulated by any central bank or a single administration, it can be sent from a user-to-user on the peer-to-peer bitcoin network without the need for intermediaries [3]. Transactions (movement of bitcoin from one account to another account) are verified by the nodes connected in the network through cryptography and recorded into a public distributed ledger called blockchain. Bitcoin was created by an anonymous person or a group of people going by the name Satoshi Nakamoto and released in the year 2009 as an open source software [4]. Bitcoins are rewarded to the nodes connected in the network when the perform a process called mining. Bitcoins are now accepted in various places for exchange of local currency, goods or services.

Bitcoin is the forerunner to Ethereum, which is a blockchain that has more functionality than Bitcoin and is the backbone of this project. Not just Bitcoin and Ethereum every crypto

currency is powered by Blockchain technology hence it is important for us to explore in detail how Blockchain works. First lets get an idea about the transaction system in Bitcoin which gives us a base for understanding the transaction system in Ethereum, followed by description of blockchain that is core to both Bitcoin and Ethereum.

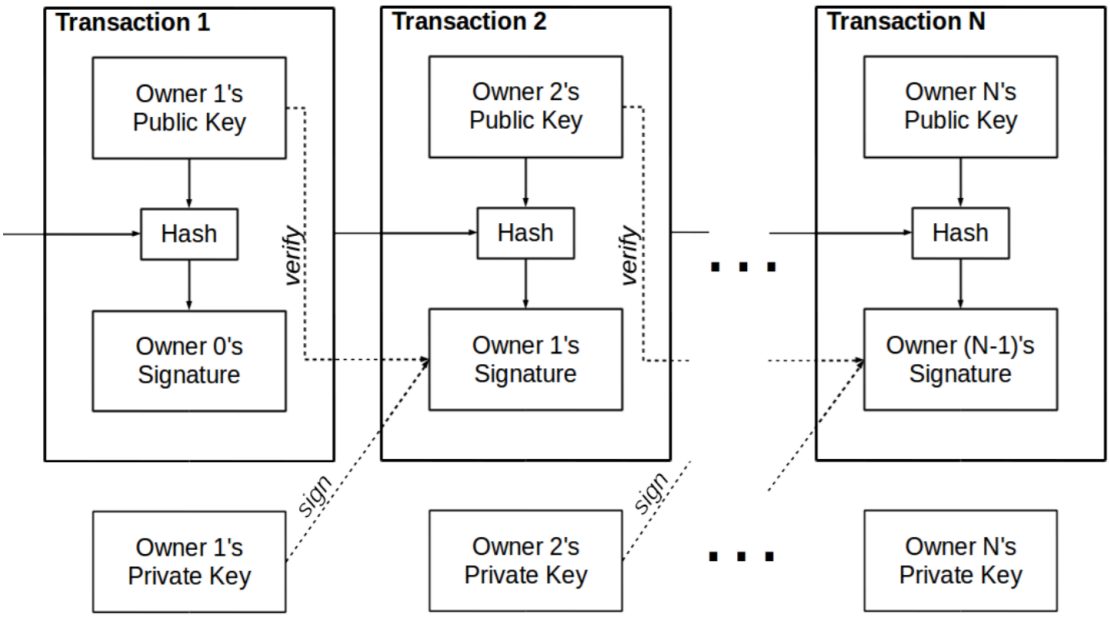

Figure 1 Bitcoin transaction chain of ownership

In the bitcoin white paper written by Satoshi Nakamoto and released in year 2008, an electronic coin is defined as a chain of digital signatures [5]. Ownership of a coin is transferred when the owner of a coin initiates a new transaction by digitally signing a hash of both owner's public key and the previous transaction that is added to the end of the coin. After signing, a new transaction is added to the chain as shown in Bitcoin transaction chain of ownership in Figure 1 bitcoin transaction chain of ownership. Transactions can include multiple inputs, all of them have to be signed individually, as well as multiple outputs. The

main reason behind this design is the coins do not have to be handled individually, similar to using the smallest unit of currency such as pennies, instead can be combined and split when needed in transactions.

This transaction chain system makes it possible for the recipient of the coin to validate the chain of coin ownership. However, there raises a new problem where the recipient cannot prove that the coin has been already spent which is commonly known as double spending problem. Previous currency systems similar to bitcoin had run into the same double spending problem [6]. The major breakthrough achieved by bitcoin was the development of a new mechanism that enables it to work without relying on a trusted third party to validate the transactions as most of the crypto currencies that preceded it. The new mechanism devised by bitcoin was Blockchain, a decentralized digital public ledger managed by a peer-to-peer network to maintain consensus on the current state of the system. With the consensus built into the system it makes it impossible to spend the same coin twice because everyone on the network, via the block mining algorithm they run, will agree on the same sequence of transactions that determine the current state of the coin ownership.

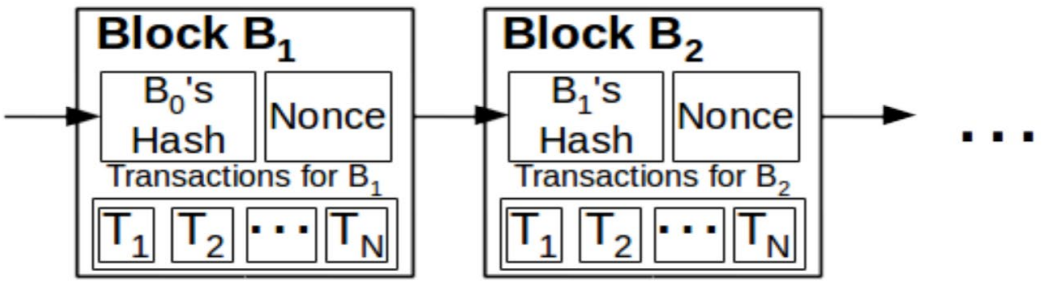

Figure 2 A simple blockchain structure

The blockchain is called so because at regular intervals of time a “block” is concatenated to the existing chain of blocks, with block essentially being a sequence of most recent transactions. The average time a new block is created in case of bitcoin blockchain is 10 minutes, which means that the state of the blockchain changes very often by adding a new block to the blockchain. If we replay all the transactions from the beginning to end then we can get the current state of blockchain.

A “proof-of-work” algorithm run by “miners” in the blockchain peer-to-peer network makes decision regarding which transaction and order of transactions to be included in a block. Miners receive transactions from Bitcoin users and use those transactions to “mine” a block. Miner who successfully mines a block will be rewarded with bitcoin as an incentive for spending his resources such as computing power and time. To mine a block, a miner takes a part of previous block’s hash, all the new transactions that are received in sequence and an integer called a nonce as shown in Figure 2 a simple blockchain structure. If the value of the hash is below a certain threshold (that determines the difficulty of mining which results in average block time) then the block is successfully mined else the miner increases nonce and recomputes hash to see if it is below the threshold. The first miner who is able to compute the hash is rewarded, then he will broadcast the result to other miners in the network who will validate it and run the transactions in the block to get the current state of blockchain. This process repeats and new transactions that arrive are persisted in the blockchain. Although this algorithm is essentially just busy work at its core, it makes the system secure because the proof-of-work aspect means that an attacker cannot just spawn an army of virtual machines to take control of the blockchain. He must have to

take control of more than half of the computational power present in the network to control the state of the blockchain. In addition to this, it serves to establish a digital crypto currency that does not require a third party to oversee the validation of the transaction by using the busy work to bring about a consensus on the system's state.

2.2 Ethereum

Ethereum is an open-source public distributed computing platform and operating system based on blockchain technology first used by Bitcoin. Ethereum extends the usefulness on Blockchain well beyond cryptocurrencies by making the blockchain programmable according to developer's needs. It was proposed by a cryptocurrency researcher Vitalik Buterin in his whitepaper published in the year 2013, where he states the intention of Ethereum is to provide, "a blockchain with a built-in fully fledged Turing-complete programming language that can be used to create "contracts" that are used to encode arbitrary state transition functions" [7]. These features make it the apt choice for building truly decentralized applications similar to this project and many other decentralized applications (dapps).

Although Ethereum blockchain is much more advanced and intricate, it is still based on the same principles as Bitcoin's. Ethereum similar to Bitcoin also uses a proof-of-work algorithm run by a peer to peer distributed network to find consensus on the current state of the system, with the miners being rewarded in Ether(crypto currency used by Ethereum network). Network gets transactions from users distributed across the globe and the proof of work algorithm at regular intervals determines a sequence of those transactions to be included in the next block in the blockchain. Every new block added to the chain

determines the state of the system. The block creation time in case of Ethereum averages around 14 seconds while that of Bitcoin averages around 10 minutes, both operate on the same set of core principles of blockchain.

The major difference in Ethereum is the complexity of both, the state stored by the blockchain and how the transactions can alter the state of the blockchain. Ethereum's state mainly consists of objects called accounts located by 20-byte address [7]. There are two types of accounts, externally owned accounts and contract accounts. Externally owned accounts or EOAs can be accessed by private keys similar to Bitcoin and have a field to store the current ether balance of that account. Contract accounts or CAs similar to EOAs have additional fields to store the contract code and storage. As CAs are a part of contracts, their interaction with other accounts and how they access or modify their storage are controlled by the owning contract. The contract storage is a key value store of data persistent among transactions.

Unlike externally owned accounts, contract accounts cannot initiate transactions on their own. A transaction must have recipient address, a signature identifying the sending account and amount of Ether transferred from sender to recipient. These information is enough to move value from one account to another account either in case of CA or EOA. However, there exists an optional data field used to specify a public function name present in the contract code of the recipient CA. This data field provides a powerful way for a transaction to induce state changes in the blockchain. Even though contracts cannot create transactions they can send "messages" to other contracts. Messages are similar to transactions but it takes place between two contracts, by a CA triggering a function in its contract code, rather

than EAO and CA. Similar to transactions they do contain a message sender, a message recipient, amount of ether to transfer and an optional data field to call a function in the recipient's contract code.

The actual state changes are done by the Ethereum virtual machine (EVM) upon receiving a transaction, by running the low-level contract bytecode. A fascinating concept called gas is used to by EVM to operate; it can be thought of as fuel purchased to execute a transaction. Bitcoin also allows scripting for complicated transactions but it is not turing-complete like Ethereum. Ethereum solved a major problem that Bitcoin had by finding a solution to transactions that involve loops. In case a transaction consisted of an infinite loop then calculating the state would be impossible, computational time would be massive, that can create problems for the blockchain. Ethereum's fairly simple solution to overcome this problem was to have a cost for each transaction that it executed. For every transaction a user creates the amount of gas required for the transaction to be processed is purchased from the ether balance in the sender's account at an arbitrary price, typically the market price at that moment (the amount of gas sent directly affects the transaction processing time as the miner running the Ethereum blockchain get that gas amount is a incentive for mining so higher amount create higher incentive). Every bytecode operation done in the EVM costs a certain integer amount of gas, operations such as modify or add to contract storage are the most expensive because all those changes are persisted on the blockchain forever. The gas starts depleting when a transaction starts executing and stops if it completes or runs out of gas. If the transaction completes and some gas is left behind then

it is refunded back to the transaction sender. However, if it terminates because of gas amount depleted to zero then no ether is returned and the transaction fails.

High-level smart contract languages used to write smart contracts, that compile to bytecode executed by the EVM, and make writing a smart contract simple. Solidity, a smart contract language, chosen for this project due to the support available in developer community, object oriented features and syntax similar to JavaScript [8]. Writing a contract in solidity is similar to writing a class in other object oriented programming languages with functions, member variables and interacting with other contracts. The most important feature of a smart contract is their permanence and the immutability of the smart contract code. The access to a function in a contract can be restricted by using modifiers that make it impossible for a developer to call that function if his address does not have the privileges. Even if he adds code to permit his address to call the function, he will not be able to change the contract code once deployed. This might sound like a burden but there are certain important considerations why this was done. Smart contracts are digital version of written contracts, hence as written contracts cannot be modified after signed, Ethereum contract code is like a digital contract specifying, with the logical certainty of code, the exact behavior of the contract account, which can be seen as a kind of autonomous entity on the blockchain. This is required for building applications that are based on completely trustless model where even the developer cannot manipulate once it is deployed, especially when the application involves money. An initial Ethereum dapp that took advantage of the complete trustless model was “The DAO” (decentralized autonomous organization). It was all built using smart contracts that users could invest in, make proposal, cast votes on the

proposal based on the investments, get rewarded based on the weight of the investments when a proposal succeeds and lot more [9]. It was a wildly successful app raising $150 million from user investments, but it was hacked in mid 2016 and lost $50 million of the investments due to a flaw in the code [10]. So no matter how well intentioned a smart contract is, if it is badly coded then it is quite similar to poorly written hand contract and can be exploited by hackers.

Ethereum was the perfect choice for this project “Crowdfunding dapp” that creates an environment for people to raise money for their dream project. With the smart contracts present, the application can store and manage user account data, store metadata and users can interact with the smart contracts using transactions created by project’s client application. Ethereum makes the smart contracts always available, accessible anywhere and restrict control to people who are only authorized to use them. It can achieve this because of the underlying technology “blockchain” first used in Bitcoin.

CROWDFUNDING DAPP REQUIRNMENTS

3.1 Overview

The crowdfunding dapp's purpose is to create an ecosystem where it will be effortless for everyone to start or support a campaign aiming to create new products or services. Crowdfunding dapp provides a crowdsourcing platform to raise capital by giving long-term incentive sharing mechanism on the "internet of value".

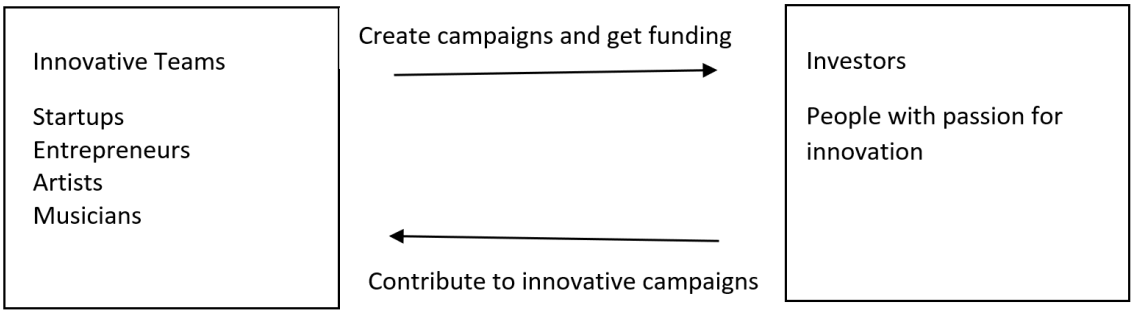

Figure 3 Crowdfunding dapp platform ideology

As seen in Figure 3 Crowdfunding dapp platform ideology by using crowdfunding dapp platform teams or individuals such as startup founders, business project leaders, music artists, movie directors etc. will be able to raise funds by issuing ERC20 type tokens on the blockchain in a very simple way. The campaign funding is governed by smart contracts deployed on Ethereum. The metadata of a campaign like campaign image, campaign video, campaign description etc. is not stored on the blockchain. Throughout this report, we use two actors "creator" and "investor" who use that platform for their needs, if we are not differentiating the roles we use the term "user" to represent both of them. We use the term "creator(s)" to address a person or a team like a startup team, an artist, a director or an

entrepreneur who use the platform to create a campaign and raise funds. We use the term "investor(s)" to address an individual who uses the platform to search for interesting projects and invest in those projects to earn project specific tokens. The terminology used is to resemble a business usecase scenario where an entrepreneur tries to raise funds for building his project or a venture.

To create a campaign, a creator has to have a clear picture about what he wants to build and how the investors who comes to the platform are benefitted by his product or service. If the creator himself is not sure about his product/service then he cannot expect his campaign to succeed. Once he has finalized his proposal, sure about the timelines on his deliverables, has proper resources to build the product and has done some market research then the chances of his campaign being funded will be high. All campaigns are timed for raising funds and this time is decided by the campaign creator so as to give maximum flexibility to the creator.

For creating a campaign, a campaign name, campaign creator's name, single line description, detailed description, a banner image and a campaign video is required. Then he has to decide how many tokens he would like to give away and the price, name and symbol of each token. He also needs to give his Ethereum wallet address to receive the funds once his campaign is successful. There is no guarantee that the campaign will be successful, it entirely depends on the creator's research about how much required is the product/service in market and how can he attract investor traction for his campaign.

An investor who lands on the crowdfunding dapp platform can browse through all the campaigns, search or sort the campaigns based on his interests and invest ether to support

his favorite campaign. The investor needs to have sufficient amount in his wallet to invest and pay for the transaction (gas) to be mined. Once the project is successful, he can claim the campaign specific tokens depending upon the criteria specified by the creator. If the campaign fails, he can claim his refund of ether that he had invested in that specific campaign. The campaign specific token are the incentives for an investor to invest in a campaign that he feels might be successful and the value of the token might increase in future.

The ideology behind UI is to keep it as simple as possible for both the investor and creator to use the platform and achieve their goals. Below are the main core UI functionalities that are required for utilizing the platform productively:

- Provide an interface to list all the campaigns.
- Provide an interface for investing into a campaign.
- Provide an interface for creating a campaign.
- Provide an interface for listing all the campaigns by the creator to track his investors and claim the funds if it is successful.
- Provide an interface for listing all the campaigns invested by an investor and claim his refund or campaign specific tokens depending upon the result of the campaign.

3.2 Campaign creation and management

Users must be able to create a campaign and track the investors who have invested ether in their campaign. User lands on the “Register campaign” page from landing page and fills in all the details such as campaign name, creator name, description, banner image, banner

video and detailed description about the campaign. Then he gets to the token creation page where he gives all the details of the campaign specific token such as the token name, token symbol, token price per ether invested and number of tokens he wants to give. Then the contract creation is triggered and values provided are used to create the campaign on the Ethereum blockchain.

The campaigns are designed to end according to the time specified by the creator in days. In the campaign creators page only campaigns created by him are displayed. He can get a list of all the investors for each campaign with the investment amount and time using campaign creator dashboard. In the same dashboard he can claim the funds if the campaign is successful.

3.3 Investing Ether into campaigns

Users, after getting into a campaign description page can read about the campaign and invest Ether into the campaign. They will go through a series of reviews to confirm the amount they want to invest and the particular account that they want to invest the amount. Once they have selected everything and successfully invested then an investor can track his investments through investor details page.

After a successful funding, an investor can claim his campaign specific tokens from the Investor dashboard and if the campaign is failure then he can claim the amount he had invested in that campaign. In addition, if he had invested multiple times then he can reclaim all his multiple investments.

DEVELOPMENT TOOLS

4.1 Solidity

Solidity is an object oriented high-level language used for writing smart contracts in the crowdfunding dapp. A statically typed language has features such as inheritance, libraries and complex user defined types. C++, python and Javascript, majorly influenced it. Solidity is compiled to bytecode that is executable by Ethereum virtual machine (EVM). Using solidity developers can write dapps that implement self-enforcing business logic contained in smart contracts, leaving an undeniable and permanent record of transactions [11].

4.2 Remix

Remix is an integrated development environment (IDE) used for developing smart contracts in solidity. Developing in remix assists developers to find bugs and debug code with ease. Remix supports three different kinds of environments to deploy and run the smart contract:

- Javascript VM: It creates a mock of blockchain environment so you can test your smart contract functionality.
- Injected Web3: This environment uses a browser plugin or a blockchain based browser such as Mist to connect to any Ethereum network (test or main).
- Web3 Provider: This environment connects to Ethereum node running at localhost and send the transactions to any network (test or main) as specified by the user [12].

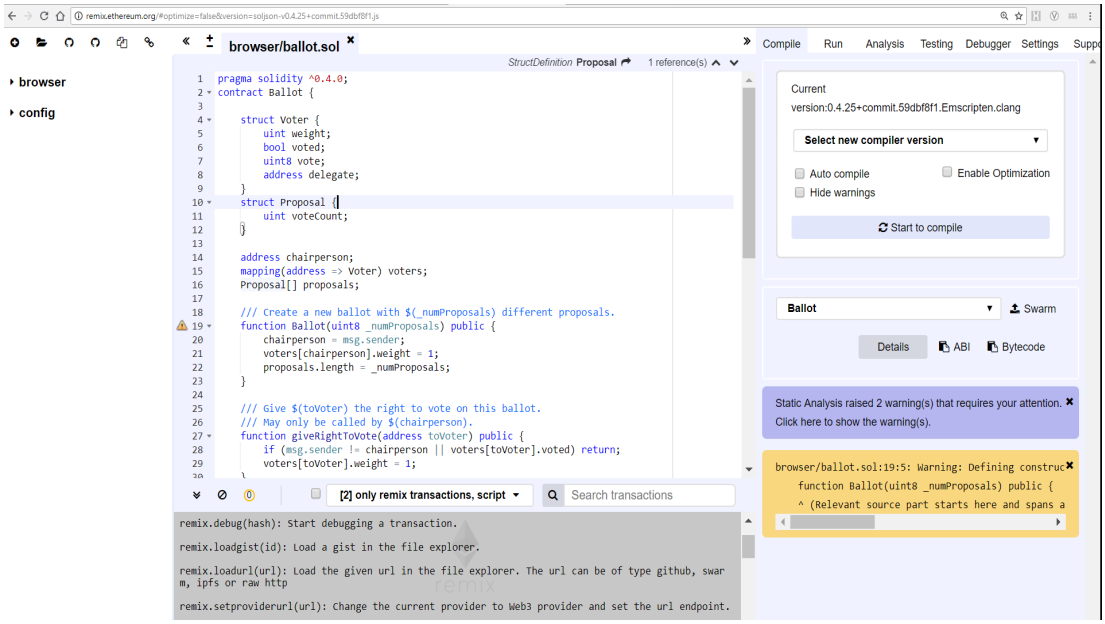

Figure 4 Remix IDE in chrome browser

After writing a smart contract, user can compile it to bytecode and send transactions to Ethereum using Remix as shown in the Figure 4 remix IDE in chrome browser.

4.3 Visual studio code

Visual studio code is a code editor developed by Microsoft. It is a feature rich editor that supports code highlighting, code debugging, intelligent code completion and code refactoring to build web applications. It supports wide range of programming languages and has a built in terminal to execute command line commands. As crowdfunding is a frontend, heavy application with lots of Javascript Visual studio code was a best choice as the editor to develop the application [13].

4.4 Metamask

Metamask is a chrome browser plugin that acts as a bridge between your browser and Ethereum blockchain by providing a secure identity vault, a user interface to manage multiple Ethereum wallets and sign blockchain transactions. It is one of the best ways to send transactions to Ethereum blockchain because it keeps a track of transaction execution and returns if any error occurs during mining or execution. It supports any ERC20 type token to be added to your wallet and trigger transaction on those ERC20 tokens. It is an Ethereum community open source project having more than million active users; hence, it is the most popular plugin to interact with blockchain.

4.5 Testnet

Test network (Testnet) is a copy of Ethereum blockchain identical in every way to main network except the fact that their Ether is worthless [14]. There are three types of testnets public, private and GanacheCLI. As names suggest, public testnet are available to everyone and connected to the internet, private testnet are similar to one's own blockchain and GanacheCLI is a simulation of Ethereum network on a single computer. For this project, we are using a public testnet called Ropsten.

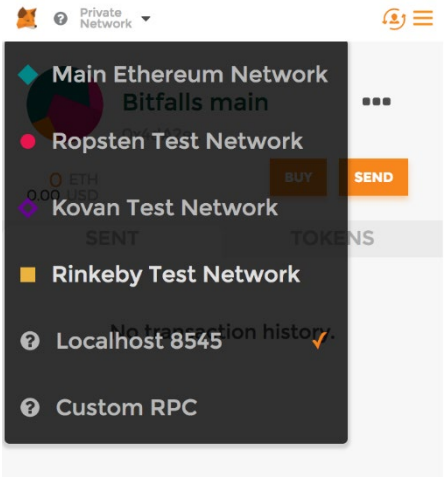

Figure 5 Testnets available in Metamask

Using Metamask we can connect to any Ethereum network as seen in the Figure 5 testnets available in Metamask. Of all the tree networks ropsten resembles to the mainnet the most [14].

4.6 IPFS

IPFS stands for interplanetary file system. It is a protocol and network used to store and share hypermedia in a distributed file system. It is an open source project maintained by a huge community of developers. The contents in IPFS are accessed in two ways, via FUSE (in case of Linux) and over HTTP even though IPFS wants to replace HTTP. IPFS can be seen as a BitTorrent swarm, exchanging objects within a single Git repository [15].

DESIGN

5.1 System architecture

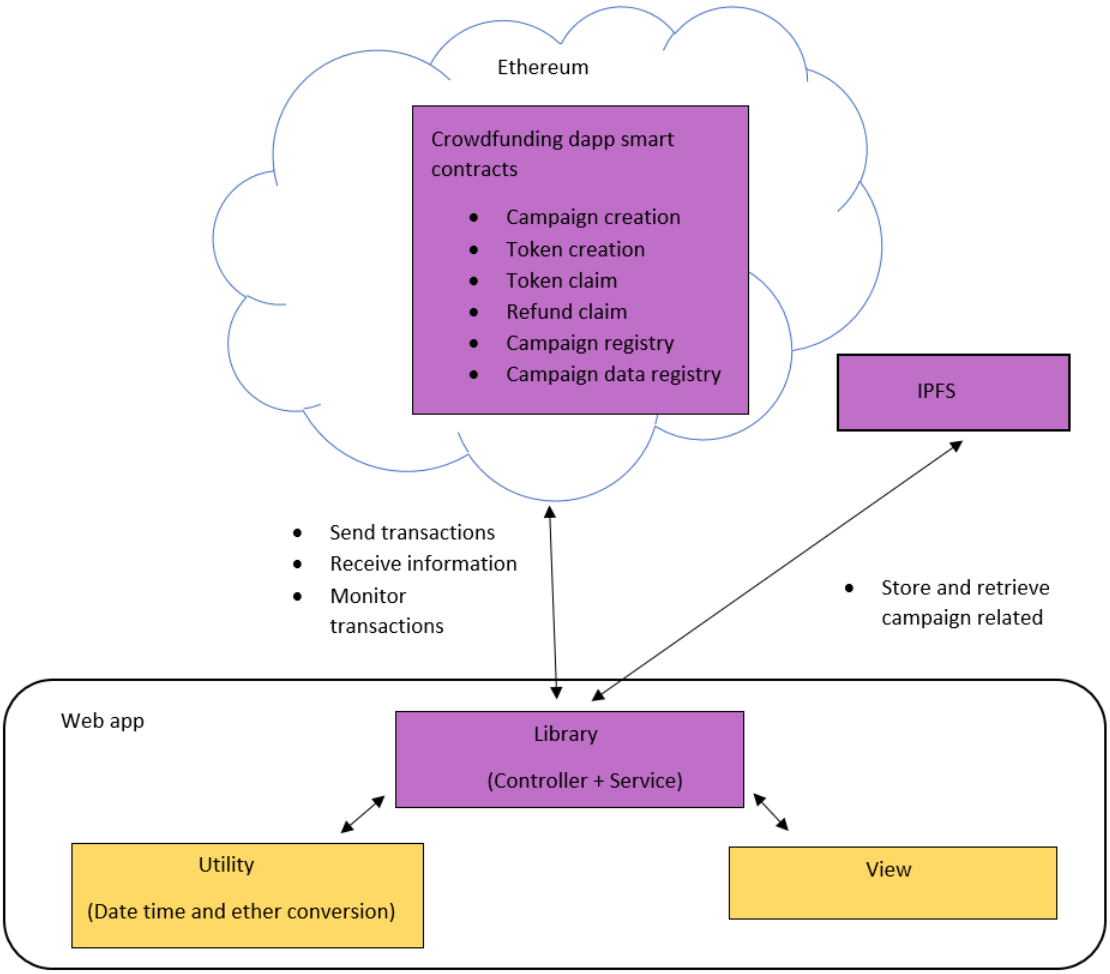

Figure 6 A basic crowdfunding dapp architecture

The basic architecture of the crowdfunding dapp is depicted in the Figure 6 a basic crowdfunding dapp architecture diagram, concentrating on high-level components. All interactions between a campaign creator (a person arriving in the platform to raise funds) and a campaign investor (a person arriving in the platform to invest ether) are mediated by

the smart contracts written for crowdfunding dapp deployed in Ethereum blockchain platform. For example if an investor, wants to invest certain amount of Ether in a particular campaign that interests him, a transaction is initiated and sent to Ethereum network with additional Ether to pay for the mining fees.

Creating a campaign has other meta-data associated with it such as the campaign description, detailed plan and concept about the campaign, images and videos to describe the concept, documents such as white paper and creator or team name. All these meta-data are stored in IPFS and fetched by the library component when the application loads to supply to the view component. The view component is responsible for building the HTML view displayed to the user.

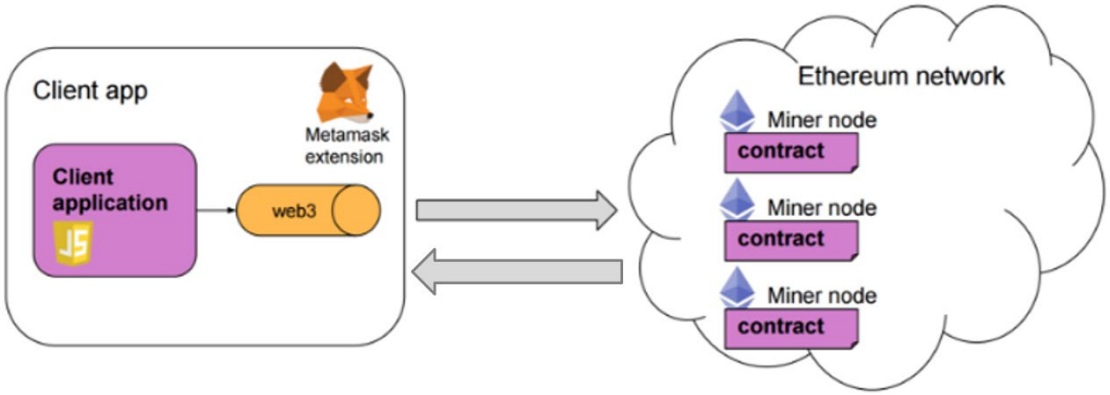

Figure 7 Library component using Metamask to communicate with Ethereum nodes

As shown in Figure 7 library component using Metamask to communicate with Ethereum nodes library component in the dapp uses web3 (Ethereum JavaScript API) to trigger transactions depending on the user interaction with the view component. Metamask acts as lightweight client that provides async functionality to library component for making calls

to Ethereum network using API's provided by web3. Metamask also tracks the transaction processing and updates the library component about the status of mining of that transaction.

A transaction needs to be signed before it can be sent to Ethereum blockchain. Metamask handles this by using the API's provided in web3 library. Once the transaction is signed Metamask will make a RPC call to send the transaction to Ethereum network where the smart contracts are deployed. Metamask polls Ethereum network to check the status of the transaction and update the Library component using a callback function present in library component.

The other network calls handled by library component is to the IPFS to store and retrieve campaign metadata. IPFS provides standard set of API's to interact with its file system. Library component makes web service calls to IPFS to store the metadata and receive a link to the stored data so in future it can just make a web call to retrieve that data.

The library component builds a data object with details received from IPFS and Ethereum network; it converts the data to a format suitable for view component to render in the browser. To perform this data conversion library uses the utility component that has many stateless helper functions. These helper functions take input in certain format such as big number and convert it to UI renderable strings. The view component becomes lightweight by moving all the data formatting logic to utility component. Library component mainly acts as the controller and service module as per the MVC architecture.

View component is comprised mostly of HTML templates rendered in the browser. It receives all the data needed from library component; then parses it to get the exact values

to be replaced in the dynamic HTML templates. It also takes care of the routing of pages and style sheets required to beautify the web page. Event handlers are in place to respond for any user actions such as button clicks and drop-down changes. Library component is tightly coupled with view component so any changes in the UI can be bubbled to library component for communicating it to blockchain.

5.2 User interface

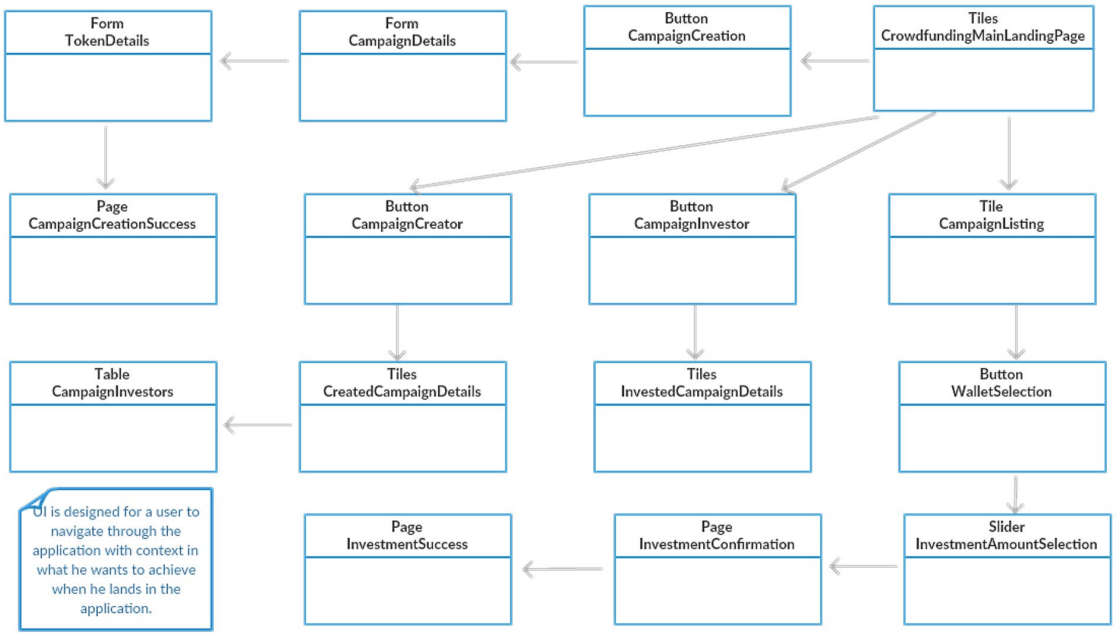

Figure 8 crowdfunding app user interface design

The user interface is designed in keeping mind the ease for a campaign creator to create a new campaign and an investor to invest in that campaign. The landing page of the application is the root of the user interface as seen in the Figure 8 crowdfunding app user interface design. It is made up of campaign tiles briefing about each campaign that is listed on the platform. User can sort the campaigns based on category each campaign belongs to.

Also user is provided with a search option to shortlist campaigns based on various search criterias.

When a user clicks on a campaign tile the campaign details page containing videos, images, documents and detailed descriptions of the campaign is displayed. A button to contribute to the campaign is displayed using which the user can select the wallet details from Metamask. Then the user is presented with a slider to adjust the amount of ether he wants to contribute to the campaign. After he fills in the amount he wants to contribute to a campaign then he navigates to confirmation page to see his final contribution details. A user transaction is triggered in Metamask which will send the transaction to Ethereum network. Once a confirmation about the cmapign is received the confirmation page redirects to a transaction investment success page to display a success message.

A campaign investor to track his investments can navigate to the investor details page from the landing page using the campaign investor button. He can find all the details about campaigns that he has invested in and if he has to claim the tokens or if he needs to reclaim his investemtn in case a campaign fails.

For creating a campaign the user can navigate to register campaign page from the landing page using the register a campaign button. After filling in the details about his campaign in the campaign creation form then he can navigate to the token details page to fill campaign specific token details. After entering all the details, he can register the campaign by clicking on register button which takes him to the progress page to track the campaign creation progress.

Once the campaign is created, to track the details about investors and campaign progress a campaign creator can use the campaign creator button in the landing page. It consists of tiles representing campaigns created by a logged in creator. When a tile is clicked it navigates to the investor details page showing all the investors of that campaign with their respective investment details and a button to claim the invested amount if campaign is successful.

5.3 Ethereum Smart contract design

The smart contracts written for crowdfunding dapp tie the whole platform together and make it work by controlling the flow of business logic. Even though they do not contain any of the bulk data, the business model they define and transactions that change the business logic flow are how this system of crowdfunding can work in a peer-to-peer manner, without central control.

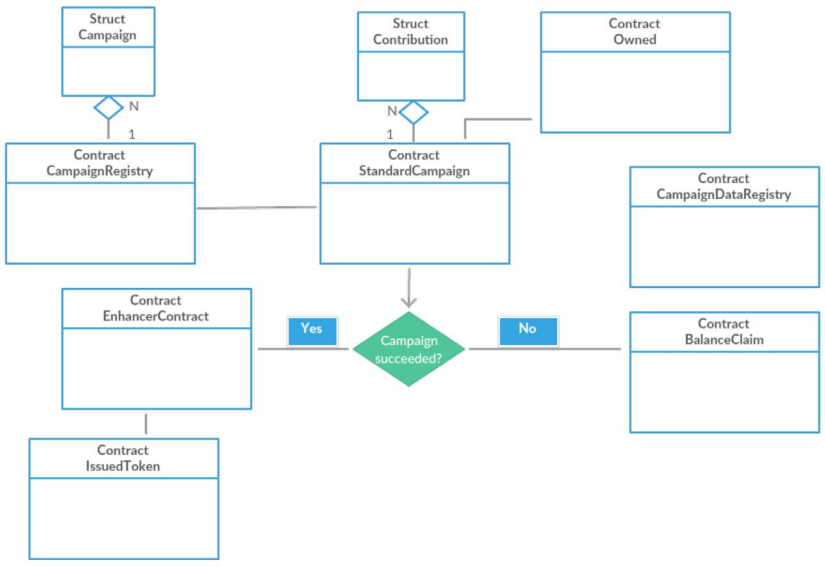

Figure 9 Crowdfunding dapp smart contract design

The layout of crowdfunding dapp smart contracts is shown in Figure 9 crowdfunding dapp smart contract design. The root of the platform is CampaignRegistry contract, which is created only once and contains all the campaigns reference that are created in the platform. When a new campaign contract is created it's contract address and contract creation timestamp is stored in the Campaign registry contract. The crowdfunding dapp platform fetches all the contract references, stored in a struct datatype, from Ethereum network when the app is loaded and builds the landing page with campaign tiles.

The CampaignDataRegistry contract contains information about IPFS hash, where a contracts metadata is present, so that when a campaign details page has to be built the crowdfunding dapp can call this contract and use the reference returned to fetch a campaign's images, videos and other details. It consists of a map that takes in a campaign's address and returns the corresponding IPFS hash.

IssuedToken contract contains details pertaining to a campaign specific token that is issued to the investor. It is associated with an enhancer contract that triggers the transfer of tokens to a particular investor if the campaign succeeds.

ModelEnhancer contract as the name suggests is an enhancer contract that contains details about each contribution to the campaign and calculates the number of campaign specific tokens to be issued in case the campaign succeeds.

StandardCampaign contract is the most important contract that handles the business logic flow of a campaign. It contains the contributor contribution details and stage information of a campaign such as success, operational and failure details. Every campaign when

created will have its own standard campaign contract that guides the execution of the campaign. Depending upon the result of funding standard campaign contract triggers modelenhancer contract to disperse camping specific tokens or creates a balance claim contract to transfer ether that has to be refunded to the investors. Incase of a campaign failure standard campaign contract triggers multiple claim balance contracts for each investor to get his investment back. Standard campaign uses a struct data type called contribution to record all the contributions made to a campaign. It refers to this datatype in case of a camping failure to create the balance claim contracts.

IMPLEMENTATION OF CROWDFUNDING DAPP

Crowdfunding dapp is divided into various activates a user can perform in the platform. Anyone who lands on the platform can browse through all the campaigns listed and explore more about each campaign. The application extensively uses Javascript libraries to build the user interface, handle user inputs and communicate with Ethereum network. The application is built on model view controller architecture so that there is clear distinction on each layer's responsibilities.

The landing page uses CampaignRegistry contract to get all the campaigns registered on the platform. Once it gets the campaigns then it accrues campaign related data from IPFS and builds the user interface of the landing page. Next section gets into the code level details of CampaignRegistry contract and user interface implementation.

6.1 Campaigns listing

A campaign struct datatype containing three data fields i.e. address of the campaign, abi(application binary interface gives a standard interface for calling particular function in a smart contract) and time the campaign is created are stored in the CampaignRegistry contract.

```
struct Campaign {
  uint256 registeredTime;
  address adr;
  address abi;
}

Campaign[] public campaigns;

mapping(address => uint) public ids;
//Get the address of campaign
function addressOfCampaign(uint256 campaignID) constant public returns (address addrCampaign) {
  return campaigns[campaignID].addr;
}
//Unix timepstamp of campaign creation
function registeredAtTimeStamp(uint256 campaignID) constant public returns (uint256 registeredTime) {
  return campaigns[campaignID].registeredTime;
}
//Number of campaigns registered
function numberOfCampaigns() constant public returns (uint256 countOfCampaigns) {
  return uint256(campaigns.length);
}
//Campaign registered event
event CampaignRegistered(address interface, address campaign, uint256 campaignID);
```

Figure 10 Campaign registry contract code

The solidity code in Figure 10 campaign registry contract code shows how the campaigns are stored and accessed. It consists of an array named campaigns of type campaign which is called by the dapp to get all the campaigns. It also consists of other auxiliary functions that are used while building individual campaign pages. To get the total number of campaigns created in the platform the web application calls the numberOfCampaigns() function. If a contract's function is declared public then it can be called by any Ethereum network user or by any contract with the contract's address. The registeredAtTimeStamp() function takes a campaignID which is a number as a input and gives the time at which a campaign is created. During the view construction these pieces of information is assembled by the library and fed to the view component which populates the HTML template rendered on the browser.

```
//get campaign details
const getCampaign = function (option, callback) {
  var campID = parseInt(option.campaignID, 10);
  var web3 = option.web3;
  var ipfs = option.ipfs;
  var network = option.network;
  var contracts = new Contracts(network, web3.currentProvider);

  //get campaign information from campaign registry
  contracts.CampaignRegistry.instance().campaigns(campID, function (crError, crResult) {
    //detect campaign registry error
    if (crError) {
      return callback(crError, null);
    }
    //campaign address
    var campaignAddress = crResult[0];
    //check for empty campaign address
    if (campaignAddress === emptyWeb3Address || campaignAddress === '0x') {
      return callback('Invalid campaign ID or address. No campaign is registered under that address..', null);
    }
```

Figure 11 Fetching campaign details client code

The view component calls getCampaigns() function by passing an array of campaign ids got by calling the numCampaigns() function in campaign registry contract. getCampaigns() iterates over those campaign ids ,which are simple array indexes, to get the corresponding campaign object having the campaign contract address as shown in the Figure 11 fetching campaign details client code. Using the campaign address retrieved from the campaign registry contract the library component creates an instance of Standard campaign contract to get the campaign specific details such as the expiry, funding goal, funding cap, owner and amount raised. Once the getCampaings() accrues all the data then it forwards a assembled campaign object to view component.

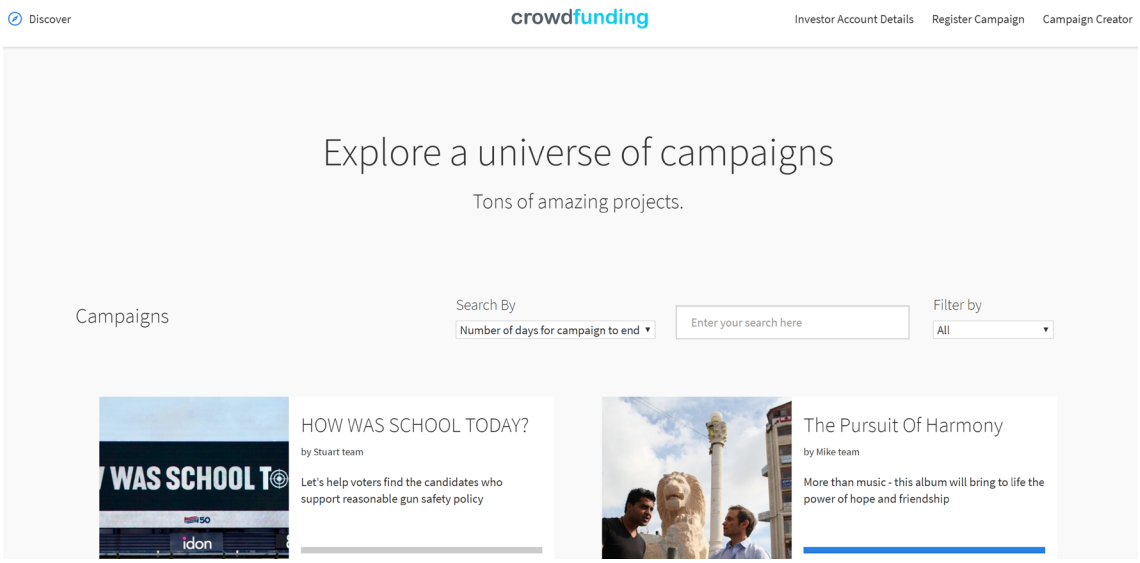

Figure 12 Campaigns listing page

The view component receives the assembled campaign object, parses it and builds the landing page listing all the campaign tiles as shown in Figure 12 campaigns listing page. The landing page with the help of campaign highlight medium template builds the campaign tiles.

The landing page has a filter by functionality and search functionality. Filter by is used to filter campaigns based on category they belong to such as design and tech, art or music. Search by is used to search for campaigns based on multiple criteria such as campaign name, expiry, number of days for campaign to end, campaign creator, percentage of amount raised or keyword.

```
el('#campaignViewSearch').addEventListener('keypress', (e) => {
  //detect enter pressed
  if (e.keyCode === 13) {
    //make a copy of campaigns object
    var campaignsToFilterForSearch = iterationCopy(getStoredCampaigns());
    var searchBy = el('#campaignviewSearchSelect').value;
    var searchString = el('#campaignViewSearch').value;

    //filter campaigns based on serach key
    var filteredCampaignsBySearchTerm = filterCampaignsOnSearch(campaignsToFilterForSearch, searchBy, searchString);
    if (typeof filteredCampaignsBySearchTerm != 'undefined' && filteredCampaignsBySearchTerm.length > 0) {
      //draw the campaigns
      drawCampaigns(filteredCampaignsBySearchTerm);
    }  else if(searchString === '') {
      drawCampaigns(getStoredCampaigns());
    }  else {
      el('#campaigns_list').innerHTML = `<h4>There are no campaigns that macth your search.</h4>`;
    }
  }
});
```

Figure 13 Search functionality event handling code

The user can enter the search term and hit enter for searching. Once the enter key is pressed the corresponding event handler is triggered that performs the search. Figure 13 search functionality event handling code is the event handler that triggers the actual search logic function to filter campaigns based on the search term.

```
//search campaigns by name
function filterByName(campaigns, searchString){
  var filteredCampaigns = [];
  //iterate through all the campaigns
  for(var campaignID = campaigns.length - 1; campaignID >= 0; campaignID--) {
    var campaign = campaigns[campaignID];
    if(typeof campaign !== 'undefined') {
      //match the campaign names
      if(campaign.name.toLowerCase().indexOf(searchString.toLowerCase()) !== -1) {
        filteredCampaigns.push(campaign);
      }
    }
  }
  return filteredCampaigns;
}
```

Figure 14 Search functionality code for searching by campaign name

The filterCampaignsOnSearch() calls the appropriate search logic based on the criteria selected by the user. Figure 14 search functionality code for searching by campaign name shows the code for searching campaigns based on the campaign name. It iterates through every campaign present in the campaigns object and pushes the one that matches the search criteria into a filteredCampaigns array, this filteredCampaigns array is returned to the filterCampaignsOnSearch function. If there are no results matching the search criteria it will display an error message.

6.2 Campaign details

The view component calls the library component to get all the details to build the campaign details page. The library component creates an instance of the standard campaign contract and fetches campaign specific details such as name, funding goal, expiry, owner, beneficiary and funding cap. After getting these campaign specific details then it uses the campaign data registry contract to get the IPFS hash using which it can fetch any meta-data related to the campaign.

```
//get the owner of campaign
contracts.Owned.factory.at(campaignAddress).owner(function (oError, oAddress) {
  //detect any error
  if (oError) {
    return callback(oError, null);
  }
  //set the owner address
  returnedCampaignObject.owner = oAddress;
  //get amount invested in campaign
  web3.eth.getBalance(campaignAddress, function (bError, bResult) {
    //detect any error
    if (bError) {
      return callback(bError, null);
    }
    //set amount invested in campaign
    returnedCampaignObject.balance = bResult;
```

Figure 15 Get the campaign owner and amount raised

The address of the owner and the amount raised by a campaign can be got by accessing the campaign's contract in the Ethereum network as shown in Figure 15 get the campaign owner and amount raised. Similarly, other details such as:

- Campaign token address
- Campaign enhancer address
- Contributions to a campaign
- Campaign Expiry

are fetched and a campaign object is populated. Then the library calls a mapping in campaign data registry contract by passing campaign's address as the input to the map which returns campaign's IPFS hash.

```
//get campaign related ipfs data
const getCampaignIPFSData = function (options, callback) {
  var ipfs = options.ipfs;
  //registered data got from contracts
  var registeredCampaignData = options.registeredCampaignData;
  //setup default campaign data object
  var campaignDataObject = {
    ipfsData: null,
    ipfsHash: '',
  };
  var ipfsHash = registeredCampaignData;
  //set campaign data ipfs hash
  campaignDataObject.ipfsHash = ipfsHash;
  //handle in ipfshash
  if (ipfsHash === '0x') {
    return callback(null, campaignDataObject);
  }
  //fetch data from IPFS
  ipfs.catJSON(ipfsHash, function (ipfsError, ipfsJsonResult) {
    //detech ipfs error
    if (ipfsError) {
      return callback(ipfsError, null);
    }
    // set IPFS data
    campaignDataObject.ipfsData = filterXSSObject(JSON.parse(ipfsJsonResult));
    // return data callback
    return callback(null, campaignDataObject);
  });
};
```

Figure 16 Client code to fetch meta-data from IPFS

After the IPFS hash is retrieved, library component calls catJSON library function as shown in Figure 16 client code to fetch meta-data from IPFS and passes the IPFS hash as an argument to get a JSON object containing meta-data related to the campaign. The main information contained in meta-data are:

- URL to the campaign
- category
- banner image
- banner video
- other images
- other videos
- description
- about campaign

This meta-data is stored as a JSON and got back as a JSON which is attached with the campaign object returned to the view component.

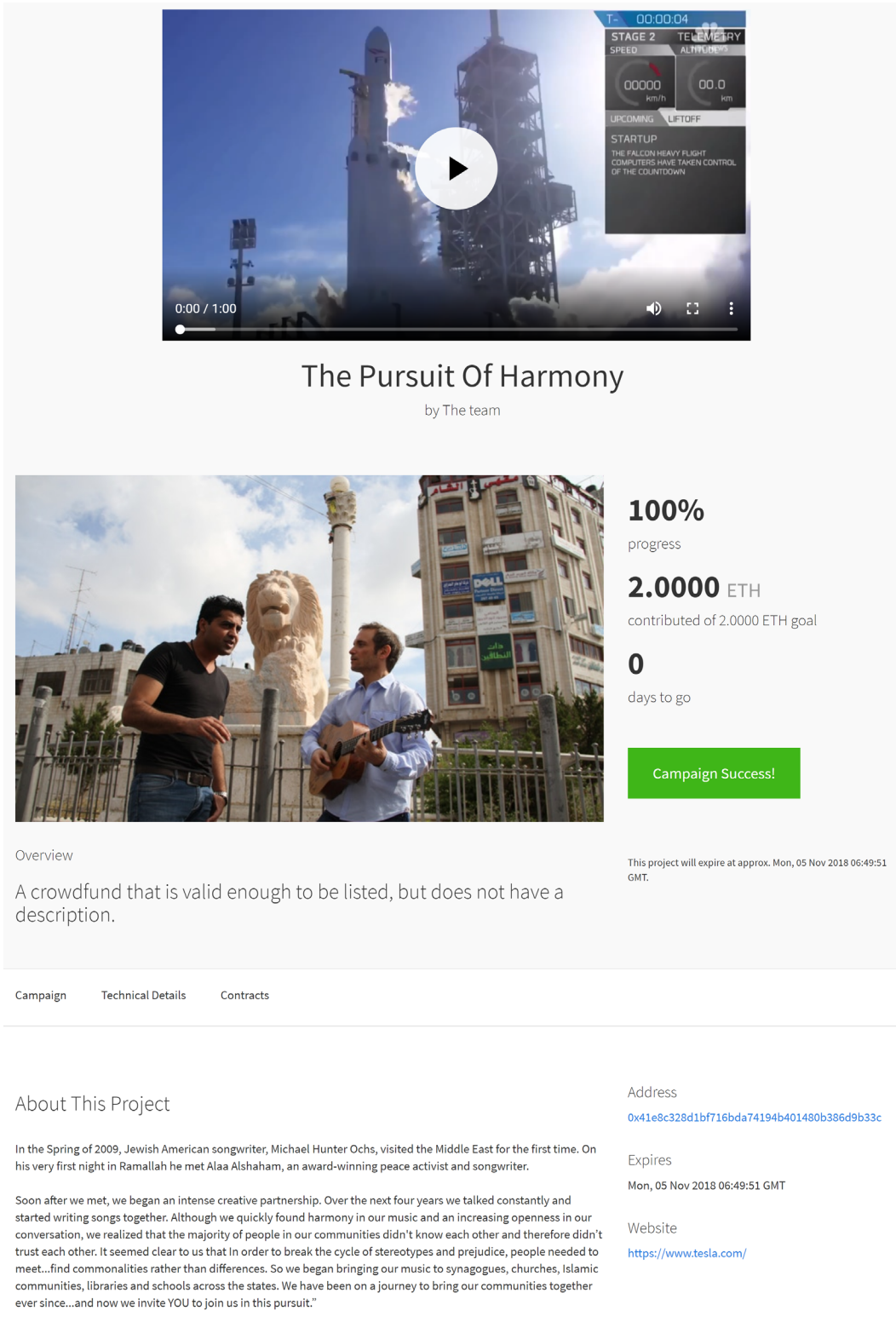

Figure 17 Camping details page

The view component with the help of the campaign overview template and campaign object got from the library component builds the campaign details page as shown in Figure 17 camping details page which gives a detailed overview of the campaign and a button to invest in the campaign.

6.3 Campaign investment

Ether can be invested in a campaign by an investor using “Back this project” button. This button is associated with an event handler which navigates the page to wallet selection page where a button is provided for the investor to confirm his usage of Metamask.

```
//display contribution review page
el('#campaignContributeAmount').addEventListener('change', e => handleCampaignContributeReview(campaignData));

//contirbute to campaign buton
el('#campaignContributeToCampaign').addEventListener('click', () => {
  //check if contributor has enoughbalance
  if (getAccountBalance().gte(web3.toWei('1', 'finney'))) {
    getRouter()(`/campaign/${campaignID}/contribute/form`);
    history.pushState({}, null, `/campaign/${campaignID}/contribute/form`);
  }
});

//handleCampaignContribution
el('#campaignContributeReviewButton').addEventListener('click', () => {
  attemptingContribution = false;
  if(handleCampaignContributeReview(campaignData)) {
    getRouter()(`/campaign/${campaignID}/contribute/review`);
    history.pushState({}, null, `/campaign/${campaignID}/contribute/review`);
  }
});
```

Figure 18 Client code to handle campaign contribution

After the confirmation of his usage of Metamask then the page navigates to contribution form to allow the investor to specify his contribution amount. His Metamask account balance is checked to see if he has sufficient funds to invest in that campaign as shown in the Figure 18 client code to handle campaign contribution. Also, the token price is fetched

to calculate the amount of tokens to be returned to the investor in case the campaign is successful.

```
//get the enhancer contract instance
var enhancer = contracts.Model1Enhancer.factory.at(campaignData.enhancer);
var pollTokenPrice = () => {
  enhancer.price((error, result) => {
    if (!error && result) {
      //ether value
      var etherValue = new BigNumber(web3.fromWei(result, 'ether'));
      //append the ether value
      el('#contributeTokenPrice').innerHTML = '';
      el('#contributeTokenPrice').appendChild(yo`<span>${etherValue.toString(10)}</span>`);
      el('#campaignContributeAmount').value = etherValue.toString(10);
      //Set the token price in environment.js for global access
      setTokenPrice(etherValue);
    }
  });
};
```

Figure 19 Client code to fetch the token price

The Figure 19 client code to fetch the token price shows how the client code creates an instance of enhancer contract and calls the price function to get the price of the token. Once the price is fetched then the slider to toggle the investment amount is configured. The investor can use the slider to increase or decrease the amount he wants to invest using this slider. Once the investor decides the amount he wants to invest then he clicks on review contribution button to get a summary about his contribution.

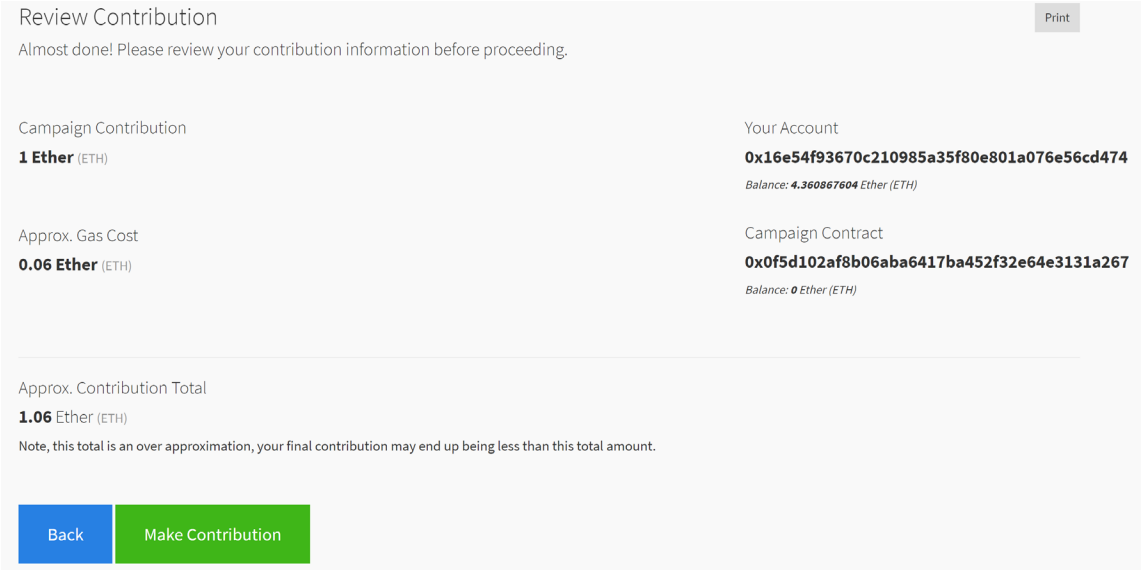

Figure 20 Review campaign contribution

Figure 20 review campaign contribution shows a sample review of contribution to a campaign. Once the investor verifies all the contribution details then he can make the contribution using Make Contribution button which triggers Metamask to send a campaign contribution transaction. Once the transaction is successful then the page is navigated to contribution receipt page confirming successful contribution.

6.4 Campaign Creation

For a user to create a campaign he can use the “Register Campaign” button. There are two forms that must be filled up by the campaign creator:

Campaign details form - This form has all the fields that are used to build the campaign details page and the campaign tile present in the landing page. Below listed are the fields present in the campaign details form:

- Name of the campaign

- Campaign creator name
- Description
- Category
- Link to the campaign page if any
- Button to upload introduction video
- Button to upload a banner image
- Button to upload additional videos
- Button to upload additional images
- Button to upload any other documents
- Text area to write about the campaign in detail

The campaign creator needs to fill in all the fields and upload the appropriate images, videos and documents. Then he can proceed to the next form where he is given a set of fields that pertain to campaign contract, model enhancer contract and token contract creation.

Token and campaign contract creation details form - All the details that are needed to create campaign related contracts are present in this form. Campaign creator needs to make decisions about how many tokens he wants to give to the investors and what is the price for each token. Also, he needs to decide on how many days he wants to run the campaign for. Below are the fields present in this form:

- Wallet address of the owner
- Wallet address of the beneficiary in case the campaign creator is not the owner

- Funding goal in Wei (Wei is the smallest unit of ether)
- Number of days to fund the campaign successfully
- Name of the token returned to the investor
- Token symbol
- Total number of tokens he wants to give to the investors
- Number of token per ether invested

Once all the details are filled the campaign creator clicks on “Register” button that triggers the handleCampaignRegister event handler that will initiate transactions to the Ethereum network.

```
//Campaign creation constructor
function StandardCampaign(string campaignName,
  uint256 campaignExpiry,
  uint256 campaignFundingGoal,
  uint256 campaignFundingCap,
  address campaignBeneficiary,
  address campaignOwner,
  address campaignEnhancer) public {

  name = campaignName;
  expiry = campaignExpiry;
  fundingGoal = campaignFundingGoal;
  fundingCap = campaignFundingCap;
  beneficiary = campaignBeneficiary;
  owner = campaignOwner;
  created = block.number;
  enhancer = Enhancer(campaignEnhancer);
}
```

Figure 21 Standard Campaign constructor contract code

HandleCampaignRegister uses the new function to create a new campaign. As seen in the Figure 21 standard campaign constructor contract code there are seven parameters needed to create a new campaign. HandleCampaignRegister fetches all the required values from both the forms and constructs a request that is sent to the Ethereum network using

Metamask. After the campaign contract is mined the campaign specific token is created according to the parameters provided by the campaign creator.

```
//token creation constructor
function IssuedToken(
  uint256 tokenFreezePeriod,
  uint256 tokenLastIssuance,
  address campaignOwner,
  string tokenName,
  string tokenSymbol) {

  freezePeriod = tokenFreezePeriod;
  owner = campaignOwner;
  lastIssuance = tokenLastIssuance;
  name = tokenName;
  symbol = tokenSymbol;
}
```

Figure 22 Token constructor contract code

Issued token consists of the token name and symbol as shown in Figure 22 token constructor contract code used by the wallets to represent the token in them. Token issuance is moderated by an enhancer contract that keeps track of contributors and the tokens that have to be issued to them depending upon the completion of funding. Enhancer contract has claim method that is used to claim the tokena upon successful funding of the campaign.

```
//claim if the campaign is successful
function claim()
  public
  atStage(Stages.CrowdfundSuccess)
  validTokenClaim() {
  //make the claimed flag true
  claimed[msg.sender] = true;
  //transfer the token
  if (!token.transfer(msg.sender, balances[msg.sender])) {
    throw;
  } else {
    //if no error trigger ClaimSuccess event
    ClaimSuccess(msg.sender);
  }
}

//number of tokens that have to be issued
function calculateTokenAmount(uint256 _value) public constant returns (uint256) {
  return _value / price;
}
```

Figure 23 Enhancer contract code to claim tokens

As seen in the Figure 23 enhancer contract code to claim tokens, the contract makes sure that all the required conditions such as if the campaign is successful and if the investor is owed any tokens before transferring the tokens. When a contribution is made to a campaign the enhancer contract notates the contribution and calls the calculateTokenAmount to calculate the number of tokens to be issued, depending upon the token price set by the campaign creator, and stores it in a balances array.

Once the campaign contract and associated token contract with enhancer contract are created then it is registered with campaign registry contract. As the name suggests a campaign registry contract is a registry of all the campaigns registered in the crowdfunding dapp platform.

```
//register a new campaign
function register(address campaignAddress)
  validRegistration(campaignAddress)
  public
  returns (uint256 campaignID) {
  //give the campaign a ID based on the campaigns array length
  campaignID = campaigns.length++;
  idsOfCampaign[campaignAddress] = campaignID;
  //create a campaign object and store it in campaigns array
  campaigns[campaignID] = Campaign({
      addr: campaignAddress,
      registered: now
  });
  //trigger the campaignRegistered event
  CampaignRegistered(campaignAddress, interfaceAddress, campaignID);
}
```

Figure 24 Campaign registry contract code to register a campaign

The register function in the campaign registry as shown in Figure 24 campaign registry contract code to register a campaign accepts the campaign address as a parameter and stores it in a campaign object array containing two fields i.e. the address of the campaign contract and time the contract was registered. The library component uses the campaigns array to build the campaigns object as discussed earlier.

The meta-data related to a campaign got from both the forms campaign details form and token and campaign contract creation details form must be stored. IPFS is a distributed file system to store and share data as described in the development tools section. IPFS provides a JavaScript library to store and retrieve data. All the meta-data collected from both forms are converted to a JSON as it will be easy to convert it to a JavaScript object when retrieved.

```
//fetch data from IPFS
ipfs.catJSON(ipfsHash, function (ipfsError, ipfsJsonResult) {
  //detech ipfs error
  if (ipfsError) {
    return callback(ipfsError, null);
  }
  // set IPFS data
  campaignDataObject.ipfsData = filterXSSObject(JSON.parse(ipfsJsonResult));
  // return data callback
  return callback(null, campaignDataObject);
});
```

Figure 25 Client code to retrieve meta-data from IPFS

When constructing the campaign object, the library component uses the catJSON function as shown in Figure 25 client code to retrieve meta-data from IPFS to get the meta-data JSON and add it to campaign object returned to view component.

The IPFS hash got after storing meta-data on IPFS is stored in a campaign data registry contract. When the contracts are retrieved from campaign registry, the library component uses the campaign address present in the details retrieved to get the IPFS hash from campaign registry. It then uses the IPFS hash to get the meta-data.

6.5 Campaign creator dashboard

For a campaign creator to track all his campaigns and get details about contributions made to a campaign he can use the campaign creator dashboard. The view component calls the getCampaign() method of library component which in turn calls the contributions array present in the campaign contract to fetch the contributions.

```
async function displayContribution(currentCampaignContributions) {
    //iterate through all the contrbutions
    for(var i = 0; i < currentCampaignContributions.length; i++ ) {
        var contributor = currentCampaignContributions[i];
        //get the trasnasction time
        var timestampPromise =  new Promise ((resolve, reject) => {
            web3.eth.getBlock(contributor[2], (err, blockDetails) => {
                resolve(new Date(blockDetails.timestamp * 1000));
            });
        });

        var timestampOfTransaction = await timestampPromise;
        //pass the details to the template
        el('#contibutors-list').appendChild(campaignContributorList({
            contributor: currentCampaignContributions[i],
            count: i+1,
            timestamp: timestampOfTransaction,
        }));
    }
}
```

Figure 26 Client code to populate a contribution object and pass it to the template

Library component returns the contribution array to the event handler which parses each contribution and fetches the block creation time in which the contribution transaction is present. Once it gets all the details needed by the page template the library component passes those details over to template as shown in Figure 26 client code to populate a contribution object and pass it to the template which is rendered to the user.

The dashboard also contains a claim button used by the campaign creator to claim the funds when the campaign is successful. The claim button triggers a transaction to the payoutToBeneficiary() function which moves the Ether from campaign contract to beneficiary's wallet.

6.6 Campaign investor dashboard

For a campaign investor to claim his token or to claim a refund if the campaign that he invested fails he must use the campaign investor dashboard. Once he logins through his Metamask all the campaigns that he invested is displayed with the token status as claimed or unclaimed and number of tokens that are owed to him. Using the claim button that is present with each campaign tile he can claim the tokens on successful funding of the campaign.

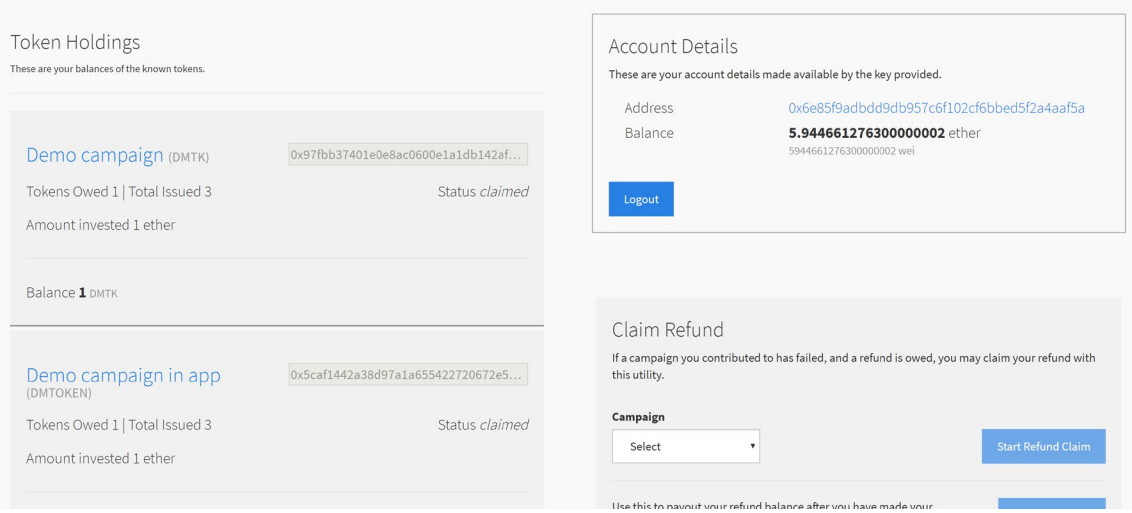

Figure 27 Campaign investor dashboard

It also shows the amount of ether that he had invested in that campaign as shown in the Figure 27 campaign investor dashboard. If the campaign is a failure, then he can start a refund claim using the small prepopulated form with a “Start Refund Claim” button.

Refund claim consists of two transactions:

- The first transaction creates a new claim contract to which the refund owed is sent to. This is done to prevent further reentrancy from a third-party account.

- Once the funds are transferred to the balance claim contract then the investor can use “Payout balance” button to get his ether back.

```
//balance claim constructor
function BalanceClaim(address ownerAddress) {
  //set the owner
  owner = ownerAddress;
}
//claim balance and suicide once it is done
function claimBalance() onlyowner public {
  //send the balance and self destruct itself
  selfdestruct(owner);
}
```

Figure 28 Balance claim contact code

Figure 28 balance claim contact code shows the self-destructing feature available in Solidity. It transfers all the Ether to the calling contract and destructs itself. We are making sure only the campaign investor has access to the claim function using onlyowner modifier which is the respective investor address.

EVALUATION

7.1 Crowdfunding dapp transactions and their costs

Creating a campaign, investing in a campaign and calling a function in a campaign to get some details every interaction with Ethereum network is a transaction. It costs money for a transaction to be performed in the Ethereum network. As described in the previous chapters, transactions in Ethereum use gas as a fuel to perform operations and pay miners. Gas is purchased using ether, cost of ether in US dollars for crowdfunding dapp transactions vary from fractions of a penny to over $27 at the current price of ether which is approximately $212. The most expensive transactions are those that involve creating contracts such as creating a new campaign. The data stored on blockchain is permanent which makes data storage as the most expensive operation and contract creation takes significant amount of storage depending upon the size of the contract code.

Transaction	**Ether cost**	**Gas**	**USD**
Token contract creation	0.129905	1299046	17.41
Enhancer contract creation	0.115869	1158689	14.44
Campaign contract creation	0.195444	1954439	21.23
Adding campaign to campaign registry	0.004382	43824	0.92
Adding campaign metadata to campaign data registry	0.008937	89373	1.89
Campaign contribution	0.019449	194492	3.04

Table 1 Crowdfunding transaction costs

Table 7.1 crowdfunding transaction costs shows a brief view of transaction costs in US dollars, gas and ether for a variety of transactions. We can see that even though some transactions do not directly create contracts, the code they are calling do indeed create and store contracts. From this table we can get to know that creating a campaign is not expensive compared to other crowdfunding sites which at present charging enormous fee for creating and handling the campaigns. There are ways to keep the costs low by deploying a contract once and referencing it for future creations.

7.2 Ethereum platform

The usage of Ethereum blockchain is rapidly increasing due to which the transaction mining times are exponentially increasing. Ethereum community is researching on ways to increase the scalability and have come up with two solutions:

- Sharding: According to this only a small percentage of nodes view and process every transaction allowing many transactions to be parallelly processed at same time. Sharding also isn't expected to reduce the native security of a blockchain [16].
- Layer 2 protocol: Layer 2 or data-link layer is used to process transactions off-chain and interact with the underlaying blockchain to enter and exit from layer 2 system.

Both solutions are debated over in the Ethereum community. Depending upon the pros and cons of implementing it, Ethereum community is planning to launch the solutions incrementally so that it will not cause any major harm to the current infrastructure.

CONCLUSION

As the world is moving towards Web 3.0 and decentralized systems to solve their daily problems, it is important to test and build new alternative architectures that show us the ideology to provide innovative solutions. With the existing solutions in the crowdfunding world created and handled by intermediary corporations that have a say on various parameters of a campaign, the alternative solution based on peer-to-peer network handling the campaign transactions seems ripe. This project explores ways to remove intermediaries in a crowdfunding business use case. This was done with the help of smart contracts, written for the crowdfunding dapp application deployed in Ethereum blockchain, that guide the execution of a transaction. This interaction allows users to create and invest ether into campaigns that interest them.

Without much efforts campaign creators and campaign investors can perform their intended activities using the crowdfunding platform. There are new emerging blockchain platforms such as EOS, Stellar, Cardano and NEO [17] that provide more language choices and platform configuration choices compared to Ethereum but these platforms haven’t proved themselves yet. EOS looks like a promising platform and in future this project can be moved to EOS if it proves to be a better choice than Ethereum.

BIBLIOGRAPHY

[1] A. Rosic, "Blockgeeks," [Online]. Available: https://blockgeeks.com/blockchain-crowdfunding/. [Accessed 18 October 2018].

[2] Blogger, "Finance Magnates," 22 May 2015. [Online]. Available: https://www.financemagnates.com/cryptocurrency/bloggers/why-the-blockchain-will-soon-become-a-household-name/. [Accessed 19 August 2018].

[3] J. S. Calvery, "Statement of Jennifer Shasky Calvery, Director Financial Crimes Enforcement Network United States Department of the Treasury Before the United States Senate Committee on Homeland Security and Government Affairs," 18 November 2013. [Online]. Available: https://www.fincen.gov/sites/default/files/2016-08/20131118.pdf. [Accessed 20 July 2018].

[4] L.S., "Who is Satoshi Nakamoto?," 2 November 2015. [Online]. Available: https://www.economist.com/the-economist-explains/2015/11/02/who-is-satoshi-nakamoto. [Accessed 19 October 2017].

[5] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 18 Oct 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf. [Accessed 24 August 2017].

[6] C. McNally, "Coin Clarity," 09 July 2018. [Online]. Available: https://coinclarity.com/predecessors-of-bitcoin/. [Accessed 16 May 2018].

[7] V. Buterin, "Ethereum Whitepaper," 2013. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper. [Accessed 21 September 2017].

[8] Solidity. [Online]. Available: https://solidity.readthedocs.io/en/develop/.

[9] ChristophM, "Intoduction to the DAO," [Online]. Available: https://daowiki.atlassian.net/wiki/spaces/DAO/pages/5996662/Introduction+to+the+DAO.

[10] "Attack," [Online]. Available: https://daowiki.atlassian.net/wiki/spaces/DAO/pages/7209155/Attack.

[11] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Solidity.

[12] "Remix, Ethereum-IDE," [Online]. Available: https://remix.readthedocs.io/en/latest/run_tab.html.

[13] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Visual_Studio_Code.

[14] G. Hayes, "The Beginners Guide to Using an Ethereum Test Network," 16 February 2018. [Online]. Available: https://medium.com/compound-finance/the-beginners-guide-to-using-an-ethereum-test-network-95bbbc85fc1d. [Accessed 26 March 2018].

[15] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/InterPlanetary_File_System.

[16] L. Mearian, "Ethereum explores a fix for blockchain's performance problem," 5 January 2018. [Online]. Available: https://www.computerworld.com/article/3245928/emerging-technology/ethereum-explores-a-fix-for-blockchains-performance-problem.html.

[17] A. Lielacher, "The Top Five Ethereum Competitors," 21 September 2018. [Online]. Available: https://btcmanager.com/the-top-five-ethereum-competitors/.