Ατομική Διπλωματική Εργασία

Development of a crowdfunding platform based on Ethereum blockchain

Άγγελος Σιαμμάς

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2021

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Development of a crowdfunding platform based on Ethereum blockchain

Άγγελος Σιαμμάς

Επιβλέπων Καθηγητής

Πάλλης Γιώργος

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Acknowledgments

I would like to thank my supervising professor, Dr. George Pallis, as well as Ioannis Savvidis who allowed me to prepare this dissertation. With the help and the right guidance they gave me, as well as the patience they have shown, I was able to complete a large part of the work.

I would also like to thank my family for the psychological and financial support they provided me throughout my studies.

Summary

The current dissertation is focused on creating a crowdfunding platform based on the Ethereum blockchain. They are three key categories of which the project was designed, the first being the back-end following the design of using a factory smart contract to create a new instance of a campaign smart contract. Moreover, a user who wanted to create a new campaign on the platform will communicate with the already deployed factory smart contract and through that, it will create a new smart contract campaign. The second category of the project is testing, as mention many times in my dissertation a smart contract is immutable, meaning when they are deployed in the blockchain network it can not change. So in advance of that, a lot of testing needs to be taken into consideration before actually deploying the smart contract to the blockchain, them being variable checking, deployments checking, etc. The final category is the front-end where I was advised to make a website based on React which I will be able to visually represent the outputs of the crowdfunding platform and interact with the deployed campaign.

Contents

Chapter 1

Introduction

1.1 Motivation

Blockchain technology is something people find difficult to understand, which is not is just a specific type of database that works in its way. The main characteristic that differs from a normal database is the way it stores its information, well typically in a database we store our information in a centralized computer system called servers. Although when we use blockchain we store our information in blocks, that are then chained together creating a chain of blocks. Storing new data on the blockchain we need to add the data into a new block and when the block is filled up with the data provided then is chained with the previous block, so we can know its chronological order. The data we want to store in blocks can be any kind of data type, we name them transactions which are pieces of information described as a ledger. Usually, people confuse blockchain with Bitcoin, in simple words blockchain is the technology used behind it. In Bitcoins case, blockchain is used in a decentralized way so that not a single person or group has control over it, they all just retain control. Blockchain is a secure technology, the reason is, it uses this decentralized approach of the network which makes it immutable.

In addition to using blockchain is that it comes along with the ability to create and deploy smart contracts. Smart contracts are not other than just a piece of code a developer writes to pass something to the blockchain, as we mention before once we deploy them in our network they can never be changed. Describing smart contracts and their work is like going into a bank and want to get a loan for a house, you need to get past all these controls for u to get the verification you need. So, in other words, smart contracts make your life easier by doing this automatically, if the requirements are met and verified, then the blockchain is updated with the new transaction completed.

After smart contracts were introduced with blockchain a platform was needed to be created. Ethereum was launched in 2015, it is described as a decentralized software platform that uses its cryptocurrency which is called Ether. The programs develop on Ethereum are called dapps for a shortcut for decentralized applications. Ethereum is an open-source platform that enables developers to create smart contracts and dapps securely without the participation of third parties. The advantage of using Ethereum over anything else at the moment is because of the platform that it provides, where you can easily create a decentralized application with backend and frontend using smart contracts.

Blockchain is a new type of technology for people and strange also, however knowing what kind of blockchain and which configuration to use is where most people fail to understand. Although, there are many frameworks to address those problems, it still does not answer the famous question, why should I use blockchain? To answer the question we need to dive deeper into the decisions we need to make for the potential use of the technology. Some of the questions a person needs to make in order to decide whether to use blockchain or not are something more like, are you in need of a shared common database, multiple parties involved, involved parties have conflict interests/trust issues, etc [9]. This kind of questions will help someone to determine if they can benefit from using blockchain technology.

Blockchain might be very exciting and has the opportunity to transform how businesses are running, although this doesn't mean that is the right solution to each problem scenario. However, we still have good examples for the use of blockchain technology, for example when people want to manage and secure digital relationships or keep a decentralized, shared system of record. Another good example is when the cost of a middle-man or third party is expensive or time-consuming. Another case is when you want to record secure transactions, especially between multiple partners. Blockchain is a type of technology that has many different uses, the goal though is to benefit from it.

The key factor of using blockchain is the decentralized network approach against centralized networks, which are more commonly used. In a decentralized network, anyone can transact on the ledger versus the centralized network which only known and identified parties can transact on their ledger. On the other hand, a centralized network is made up of parties that are known to each other and for decentralized networks like blockchain, anyone can participate and transact on the ledger. As a result, in a decentralized network, different mechanisms need to be implemented to ensure the transactions on the ledger are valid. Moreover, a blockchain can either be centralized or decentralized, even though it cannot be confused with the distributed. Simply whether blockchain uses centralized or decentralized network approach, just refers to the rights of peers on the ledger.

1.2 Problem Statement

The current dissertation is focused on developing a crowdfunding platform based on Ethereum blockchain, so far we do not have anything online where is a crowdfunding platform and on backend uses blockchain technology on the market. That is what fired me up to create this project, at first I needed to establish if that is even possible because the idea was easy to think but the process of actually developing it was not so easy. One of the issues was the security of the actual transactions created when someone wanted to fund a specific project, but this was easily fixed by the fact we are using blockchain technology when is pretty hard for someone to bypass the securities of it. The best thing about this idea is that anyone can create their own project so they can get funded by anyone and is no needed for a third party to be between the two parties when the transactions are happening. This for sure makes it easier for the funders to create payments and be more flexible to projects they want to fund.

So what I will be presenting on the current dissertation is my development of a dapp on the back-end using smart contracts programmed on the programming language Solidity where they have been multiple times tested and deployed in the blockchain. Then creating a website on the frontend using React and Javascript so we can see examples of some of the interactions we can do with our crowdfunding platform, some of them are: create a new project, fund a project, withdraw funds, etc.

1.3 Methodology

A key feature of the dissertation is focused on using blockchain technology and developing a crowdfunding platform based on Ethereum blockchain. In addition, a lot of work was focused on understanding the technologies of blockchain technology, its mechanism, the consensus process, and how they all work together. In the case of development, a lot of work has been done to create the back-end of the platform which is based on the Solidity language, moreover a front-end was made for a user to interact.

The method I took first into consideration for the back-end was to design the smart contracts for the application for someone to create and interact with the crowdfunding platform. For the creation of the smart contracts the programming language Solidity was used and Javascript, in order to do the proper testing to see if our smart contracts were deployed and running correctly. In addition, a front-end was created to interact better with the smart contracts, I used React along with some HTML and CSS to make the front-end functionally working. Moreover, some of the technologies used to help contribute to the creation of this project was Ganache which is an online wallet with fake accounts credited with ETH balance to do the deployments of our smart contracts to the network. A lot of help was also the Google extension Metamask which was needed in order to do our transactions with the front-end when the platform was created.

Chapter 2

Related Work

2.1 What is Blockchain?

A blockchain is a decentralized ledger of transactions that is duplicated and distributed through the blockchain's entire network of computer systems. Each block in the chain contains several transactions, and if a new transaction occurs on the blockchain, a record of that transaction is added to the ledger of each party. The database which is used by a lot of people is called Distributed Ledger Technology.

Blockchain architecture is based on a sequence of blocks, which holds several transaction records on the ledger. A previous block hash is contained as a header in each block header and a block has only one parent block. It is worth mentioning that children of the block's ancestor's hashes are also stored inside the blockchain. The first block on the blockchain is called the genesis block and which it has no parent connected with it. The block structure contains the block header as mention and the block body. More specifically, the block header consists of the block version which is responsible to indicate which set of block validation rules to follow. The Merkle tree root hash contains the hash value of all the transactions so far made on the block, a timestamp, nbits which indicates the target threshold of a valid block hash. In addition, the block header has a header nonce which is a 4-byte field, that usually starts with 0 and increases for each hash calculation, and at last the parent block hash, a 256-bit hash value that gives as the previous block. The block body includes a transaction counter and the transactions made, the maximum amount of transaction that can be made on a block in based on the block size and the size of each transaction. Blockchain uses an asymmetric cryptographic mechanism in order to validate the authenticity of all the transactions made on a block.

Moreover, blockchain is a Distributed Ledger where the transactions made with it are passed as an immutable cryptographic signature which is known as a hash. What that means is when a block in one chain is somewhat changing in whatever form, it will immediately be known to the blockchain. For example, if someone wanted to penetrate a blockchain system they would need to change almost every block of the chain, across all of the distributed versions of the chain in order to manipulate the system.

The key characteristics of blockchain are decentralization, persistency, anonymity, auditability. In a centralized transaction system, all the transactions need to validate their authenticity by a central trusted agency, as a result of performance bottlenecks to the central entity. However on a decentralized system like blockchain doesn't need a third party to interact between the two entities, although to maintain data consistency consensus algorithms are used. The transactions that are made in the system can be validated quickly and the invalid transactions will be discarded by trusted peers. They can not be deleted or call off when they are included on the blockchain, and this because of the immutability of the technology. Blocks that include invalid transactions are immediately discovered by the network and discard by the rest of the peers. In blockchain anonymity is key, each user can interact within the blockchain anonymously with a generated address, and it does not give away the real id of the user. To the blockchain, any transactions are made need to refer to some previous unspent transactions, and once the transactions are validated and added to the blockchain the state of the unspent transactions is changed to spent. In conclusion, all transactions added to the blockchain can be easily tracked and verified by the peers in the network.

They are three classifications of blockchain: public blockchain, private blockchain, and consortium blockchain. In a public blockchain, there is no centralized peer or party which holds more power than the rest of the peers, the blockchain is publicly open and every peer has the right to validate a transaction. In the case of a private blockchain, a centralized approach is structured, where a single entity has the power to make a decision and validate the transactions on the network. On the other hand, in a consortium blockchain, not everyone has equivalent rights, for decisions making and validating transactions on the network. A smaller number of peers have the right to over-validate a transaction, although the rest of the peers can validate but they need to reach a consensus between them. Regarding the consensus process in a public blockchain, everyone can take part in the process, the reason being all records are visible publicly to the peers. Incongruously, in a private blockchain, only the peers of a specific group or organization are allowed to participate in the consensus process. In consortium blockchain, only the group that is selected can take part in the process of consensus.

The process of reaching consensus in other words means reaching an agreement. They are algorithms that aid decentralized networks to extract a decision when is need. The features of consensus algorithms involve decentralized governance, quorum structure, authentication, integrity, non-repudiation, byzantine fault-tolerant, and performance [10]. In order to reach consensus in blockchain among the untrusted peers of the network is an approach of the Byzantine General problem [11]. A summary on the Byzantine General problem, a group of generals who command a group of army surround a city, although not all generals want to participate in the fight and some want to retreat back. Although, if only one of the group attack then they will fail drastically, in addition to that they need to reach an agreement between the two groups to either attack or retreat.

Reaching a consensus in a blockchain network is a challenge as described, although there are several approaches to reach consensus in the blockchain. Proof of Work (PoW), is one of the approaches where a bitcoin network uses, as mention in a distributed or decentralized network someone needs to be selected in order to record the transactions. One easy solution given is random, if a peer wants to deploy a block of transactions to the blockchain a lot of work has to be done in order for the validation of the new block. Moreover, on PoW every node on the blockchain network calculates a hash value of the block header. The block architecture as mention contains two-part the header on the body, focusing on the block header a nonce is contained where miners in the network would change its hash values often to get different nonce values. According to consensus algorithms the calculated hash value needs to be equal to or smaller than the certain value given. When a peer reaches the target value it will broadcast the block to other peers in the network and they need all in once to confirm the validation of the hash value. If is correct, then is validated and the new block will be added to the blockchain. However, in a decentralized network validation of the blocks can happen at the same time when we have multiple peers trying to find the correct nonce. As a result of that problem, branches need to generate, on the other hand, is hard to generate next simultaneously. In conclusion of the PoW algorithm, a block that becomes a ledger is considered as the authentic ones.

2.2. Related Technologies

2.2.1 Ethereum

Ethereum is a digital platform that is used in a variety of applications these days, it adopts some of the technologies used in the blockchain technology established by bitcoin. Many confused Ethereum with ether which is the cryptocurrency used in the current network. Moreover of the history of the Ethereum platform, was created in 2015 by Vitalik Buterin who had a view of a vehicle for decentralized, collaborative applications. As mention Ether (ETH) is the current token which is used in the network to make all kind of transactions, ether works as a self-contained peer-to-peer financial system, free from governments inspections.

The way Ether works are more like other cryptocurrencies, which share a digital ledger where all the transactions made are recorded on it, this ledger is accessible by everyone in the public but very hard to change. On the other hand, the way the technology works, Ethereum blockchain is a bit similar to that on Bitcoin, but it has a programming language called Solidity which enables developers to write dapps all through the blockchain and produce specific outcomes in their favor and they called smart contracts.

The purpose of Ethereum is to put together and improve the idea of scripting, alt-coins, and on-chain meta-protocols, in order to allow the programmers to create consensus-based applications. Ethereum is doing that by constructing what is the ultimate foundation layer, which consists of a blockchain with a built-in Turing complete programming language that allows everybody to create smart contracts, according to Vitalik Buterik the creator of Ethereum [13].

2.2.2 Smart Contracts

A smart contract is an “agreement” between two entities in the form of computer code or else programming. This piece of code is deployed and run in the Ethereum blockchain where it accessible to the public peers in the network although it is immutable, meaning it can not be changed. When a transaction is made in the smart contract responsible for its process is the blockchain, which means a third party involved in the process is not needed, the transaction only are been executed only when the agreements are met automatically.

The way smart contracts works and is possible to work is because of blockchain technology, where we were able to decentralize this piece of “agreements” in order to make them trustworthy to each other peer. In addition to being decentralized is impossible for someone to manipulate them because of the blockchain technology which is not run only by one entity, so for someone to gain access to a smart contract need to "hack" more than half of the nodes running in the blockchain. So smart contracts can run safely and automatically triggered when the “agreements” are matched without interruption of anybody.

2.3 Crowdfunding Platforms

The crowd in crowdfunding refers to people or organizations that provide the money to a company or a person that wants to raise money through crowdfunding. The way they can collect money is to pitch their idea by posting it on a crowdfunding platform, this means they can avoid loans from banks.

They are many different types of crowdfunding one been the so-called investment-based crowdfunding which refers to invest in an idea with the return of shares of the company. Another type of crowdfunding is the loan-based where this works as follows u lend money to a person or a company with the return of set interest rate, in the other hand we have donation-based crowdfunding which u can donate to a person or company money without waiting for the return of that amount in any sort of way.

The way crowdfunding works is when u visit a crowdfunding-based website, you can see all the current projects that are been pitched. In most, crowdfunding platforms u don't need to be registered to view the details of a campaign being pitched but some do, although if u keen on contributing some money to a specific one u might need to register most of the time. The investment although can only go through to the person or the company only when the goal the amount of money is raised otherwise it returns to the contributors.

Chapter 3

Methodology

3.1 Design

They were different versions of the design along the way, but mostly two were the most critical of the whole design of the platform, one being the back-end and the other the front-end. Starting with the idea of the design, what was first designed was a smart contract with the ability to create and hold pieces of information about a campaign, although the idea was good at first I missed track of something which was the deployment of the smart contract, see image 3.1. For example, if someone wanted to create a campaign, that would be good only for the first campaign. The problem with this specific version is that when I needed to create another campaign I would need the user to deploy on his own the new campaign smart contract which in the end that's not how it should work. But still, the campaign smart contract works pretty well with just one u could give many attributes such as description, title, image URL, and even had functions for someone to contribute to it.

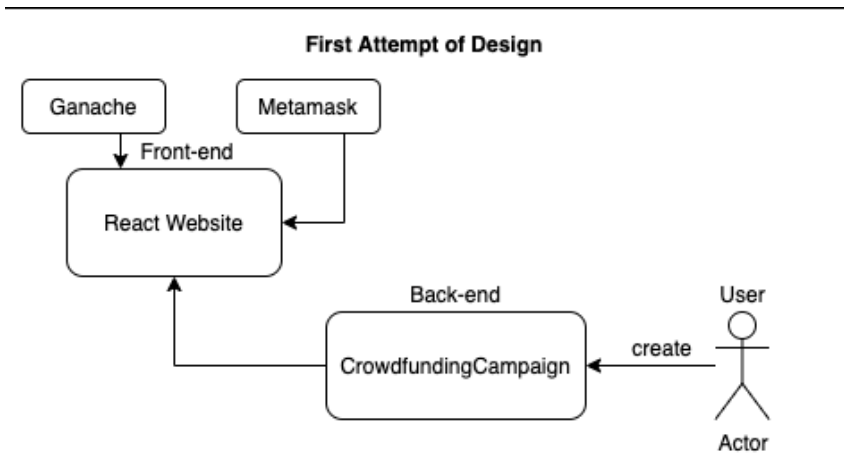


Image 3.1: This was the first attempt at the crowdfunding platform design as mentioned, only one user (Actor) is creating/deploying its own crowdfunding campaign.

The current version of the design was supposed to work great and it did actually at the start but as mention, this was not a crowdfunding solo campaign project but in fact, a crowdfunding platform that would hold several campaigns at once. So another idea was build upon the previous design and it was to create a crowdfunding factory smart contract, what this smart contract would do is when a user/actor would want to create a new campaign it would “communicate” with the factory smart contract and create a new campaign. The factory campaign in its part would create and deploy the new smart contract to the Ethereum blockchain for the user to collaborate with it. This was a big step up from the previous design because from that we could hold the addresses of the new campaigns and when we needed to access them for somewhat reason we could just pass in the address of the specific campaign we are looking for. The biggest advantage was that the factory campaign would do all the work for us instead of us doing it.

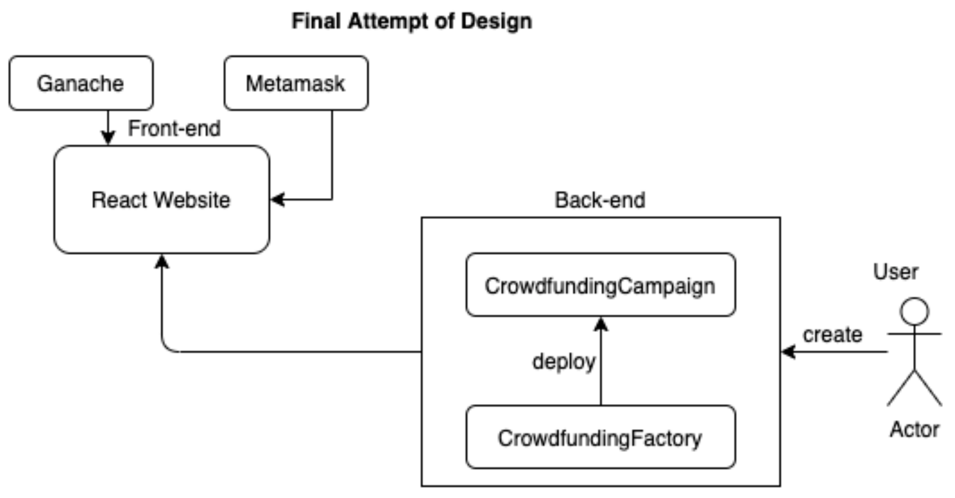


Image 3.2: This was the final design of the crowdfunding platform, where a user as mention communicate with the factory about the creation of a new campaign and the factory would deal with the deployment of the newly created smart contract.

The advantage of using this current design in opposite to my first attempt is that the deployment of the smart contracts wouldn't be the user's problem at the end, the user will just communicate with the factory smart contract for the creation of the new campaign. This helps a lot for the fact that now we can have in our new design all the addresses of the campaigns and when we need to view or interact somehow with any of the campaigns created now we can.

In advance, before creating the smart contract a class diagram was created, see image 3.3. The class diagram was built upon the two smart contracts which were decided on my final design mention above, which contain the factory and the campaign smart contract. The class diagram contains all the variables, arrays, and functions that were used in both of the smart contracts, and it was the best way to illustrate the program needed to develop in a detailed way before coding it.

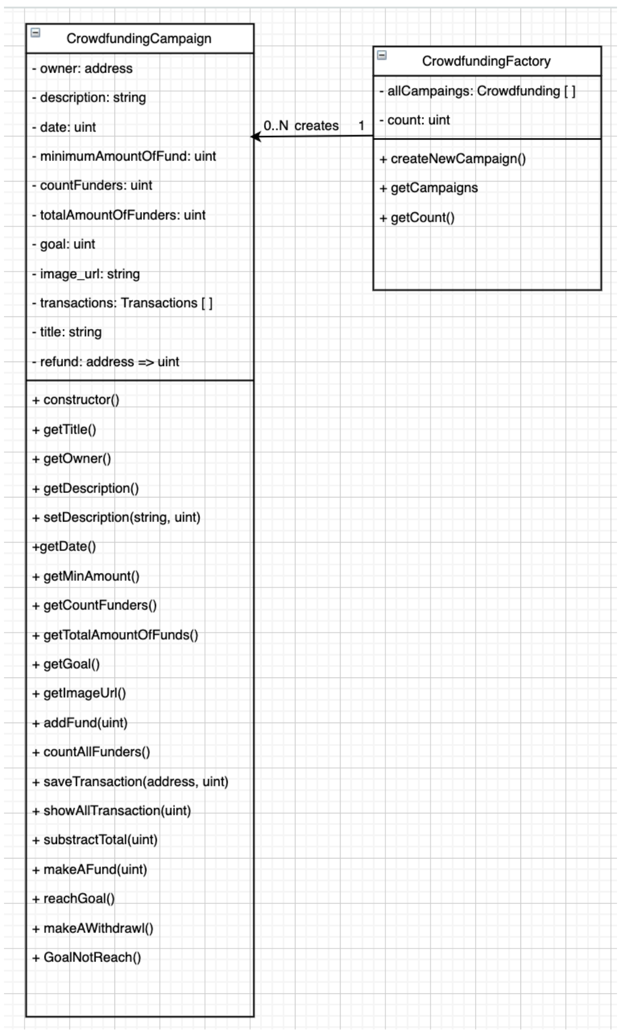


Image 3.3: This image represents the class diagram of the smart contracts.

3.2 Implementation

3.2.1 Back-end

For the creation of the crowdfunding platform a reliable programming language was needed, the Solidity programming language. Solidity as is author describes it is an object-oriented, high-level programming language for implementing smart contracts. Smart contracts are “agreements” between peers in a blockchain network in the form of computer code. More on the history of Solidity it was influenced mostly by C++, Python, Javascript and was design to target the EVM, Ethereum Virtual Machine. Moreover, some of the technical features it has are statically typed support inheritance, libraries, and complex user-defined types, etc. With Solidity of course you can create smart contracts that are meant to be for voting, crowdfunding, any type of auctions, and even multi-signature wallets.

Smart contracts were needed to be created for the back-end of the platform, so the actual implementation as mention in Chapter 3 section 3.1, was to create a smart contract that would work as a factory smart contract for the creation of the campaigns. The factory's purpose was to create a new “child” smart contract every time we triggered the create function to it and in return, we would save the current address of the newly created smart contract to an array of addresses. In addition to that, we would count the numbers of campaigns that were created at the current time, in order to have some knowledge of our platform at first because we didn’t have a visual representation of the platform so far. As mention in the factory smart contract, we would store the address of each campaign that was created and the reason being that we would want to access the data of each campaign address. A function was created afterward to give us all the addresses of the campaign which were store on the factory smart contract at that current time. In other words, the factory has three main functions which are the creation of a new campaign smart contract, a function for getting the current time campaigns addresses and a function of the total amount of campaigns were created.

As the factory was created we need to create smart contract campaigns which will store all the information for a campaign and also the ability for someone to contribute to it. For the implementation of the current smart contract, we need to have a constructor for the factory smart contract to call so it can create a new campaign smart contract from factory. The idea here is basic as in most of the object-oriented programming languages we have a constructor where we create an object based on the passed values of the constructor. The smart contract constructor takes as attributes the title, minimum amount to contribute, description, date, goal, and an image URL for someone to also upload an image for their pitch. More on the design of this smart contract we need to implement a lot of getter functions to fetch back information of the current campaign in order to preview that we are getting the correct results. Examples of some of the getter functions were to get the date, description, goal, the number of contributors for the campaign so far, the image URL, and so on.

The campaign smart contract was not only created to just hold the information on a campaign but also for the user to interact with it. So I needed to create a function in order for someone to be able to contribute back to the campaign. First, a function called makeAfund was created for a user to pick the amount of money wanted to contribute to the current campaign and if it met the requirements of the minimum goal which was set at the creation of the campaign, then it would add on. In addition to that, the number of contributors of the current campaign would have been added to plus one and after that, and then save the transactions on a struct array which kept the address and amount contributed by a funder. The history struct is called transactions in our code, see Annex A CrowdfundingCampaign contract, is an array of struct as mention which holds the address of the funder and also the amount of money contributed at the time. Moreover, on the tasks of the campaign smart contract and as mention in Chapter 2 section 2.3 on the Crowdfunding Platforms, the amount which is been gather all through the life of the campaign cannot be withdrawn unless the goal in money is met. So another function was created to settle down this purpose and it was called makeAWithdrawl, see Annex A CrowdfundingCampaign contract, and what it does it checks if the goal is met if not then u can't withdraw the funds, otherwise the funds are contributed to the creator of the campaign.

3.2.2 Testing

Those were the implementations for the current back-end without being tested because as mention in Chapter 2 section 2.2 Smart Contracts, when they are deployed there is no turning back at all because of the technology that makes them immutable. So a lot of testing was needed to implement to the smart contracts functions for more security during our usage of them. Some of the implementations been added to the smart contracts were mostly inputs checks to pass the right type of variable and if not a specific message would pop up for the error created. In order to test the smart contract, I used the developer console truffle who has the ability to run Javascript tests to see the status of the smart contracts deployed. A lot of tests were created for the safer deployment of the smart contracts to the blockchain, of which the total number of tests was 19. The tests were divided into two parts the one that tested the factory smart contract and the one that tested the campaign smart contract. The tests which I tested on the factory smart contract were more focused on the correct deployment of the smart contract and the correct value of the total amount of campaigns they were created at the time. On the side of the campaign tests, more focus on testing was needed and this is because of how many more tasks that smart contract contained. An example of some of the tests which they were created for the better usage of them was the creation correctly of a campaign, a check if the owner which created the current campaign is indeed the correct one, checks if the getter function for all the values passed on the smart contract on the creation is correct and at last, check if our goal was met then we can withdraw the money gathered. At the end of the testing, I could be more confident of deploying my smart contract to the blockchain.

3.2.3 Front-end

On the front-end, the idea was to create a crowdfunding platform which is a website-based decentralized application to communicate more visually with our Ethereum blockchain. The methodology I used here is to use React.js which is an open-source Javascript library that is used for building user interfaces (UI) and even UI components, its main use is to maintain the view layer as we call it for web and apps. React became famous over the last few years because of its simplicity and flexibility and many people refer to it as the future perhaps of web development.

Moreover, on the connections of my Ethereum blockchain with React, a library was needed to be implemented and this is where web3.js comes in. Web3 js is the collection of libraries as described which allows the user to interact with a local or remote Ethereum node using HTTP or IPC connection, in other words, web3 is a library that interacts with the Ethereum blockchain. Some of the functions it has as a library it can retrieve user accounts, send a transaction, interact with our smart contracts as mentioned, and many more.

Another mechanism used for the build-up of the front-end was Metamask. Metamask is a google extension where we can connect our cryptocurrency wallet to interact with my Ethereum blockchain and on my part was Ganache. The way it works is that you can implement an account from the Ganache application to the google extension using the private key provide for the specific account u want to use, in order to for our front-end to make transactions with the smart contracts.

All these were the technologies used to create our front-end, for the design of the front-end. At first, we were to create three pages website one been Home the other Campaign and the last Create New Campaign. Although this idea was not able to come to life for the reason of some trouble I had some trouble with connecting the three pages all along, and then a final design was created which is two pages website that contains a Home and Add New Campaign page. On the first page, someone can get just a picture and a welcoming message to my platform and some description of the work is doing, but also navigate from the navbar on top to the Add New Campaign page. On the second page what a user can see are two buttons to interact with my smart contracts, one been to create a new campaign and the other being to view all the campaigns which are created and also can contribute to whomever he wants.

On the implementation part of the two pages, the use of Router React was imported to have a two pages dapp, on the page of Home the code is just a usual React class that renders an HTML code of an image and the headers for the descriptions. On the second page where all work is made the Campaign page, we needed to implement first the web3.js on a lifecycle function in our component to connect our dapp with our smart contracts. When we are implementing the current lifecycle function to our code what we do is when something new happens on our blockchain, this function is triggered and is fetching us the new information of our blockchain. The lifecycle function is called componentDidMount, which inside the code first thing is doing is fetching the accounts with the use of web3.js for us to connect our factory smart contract with the front-end. Next in the current lifecycle function is to load our blockchain data from the smart contract to interact or manipulate somehow with them in our code, in the case of the factory, we want to fetch back each of the campaigns addresses that were created so far to view them visually in our website.

The page Campaign as mentioned is where all the work of our front-end design, the page is connected with another component called Main which is responsible for the job of implementing a form to create a new campaign. This form passes the values inputted back to the NewCampaign component and when the submit button is clicked it props the value into a function that is responsible for the creation of a new campaign to the blockchain. The function which creates the new campaign uses web3, and it can communicate with the factory smart contract in order to use the method in the smart contract which creates the new campaign.

The other task the page Campaign can do is to bring back the information from the blockchain of all campaigns to have a visual representation of the campaigns. The way this is done is to call on componentDidMount again to load the blockchain data but this time for each campaign smart contract. With the help of web3 and the getter function were created to the specific smart contract we can fetch back each information about the campaign we want and save it to a state in our React component. In other words, the second function of our page is showing on a card container information about a campaign.

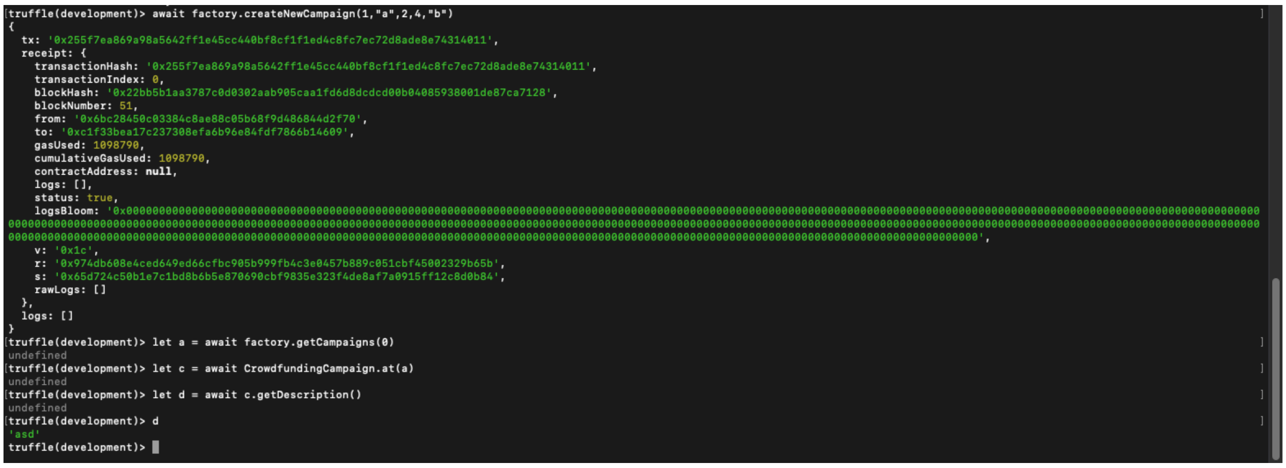
The way someone can interact with my front-end is easy, first, u need to start the client for the page to start. The first thing someone sees as mention is the Home page, then for someone to navigate to the next page is by clicking the Campaign button on the top right corner, which displays two options one to create and the other to view the current campaigns. To the option of creating a new campaign, a form would display at the side with input boxes to fill in which are just specific attributes the campaign will hold. The tricky part here is uploading an image with someone to do so is to have a string URL of the image he wants to add to upload it, using third parties applications. After submitting and refreshing the page the campaign will be created, to view the current campaign which is created is to click the second option which is the view all campaigns. The second option will display each campaign made so far wrapped around a card container for a better visual representation of the campaigns and in more detail, someone can see the image, title, description, goal and a contribute button for each campaign.

Chapter 4

Results

4.1 Results

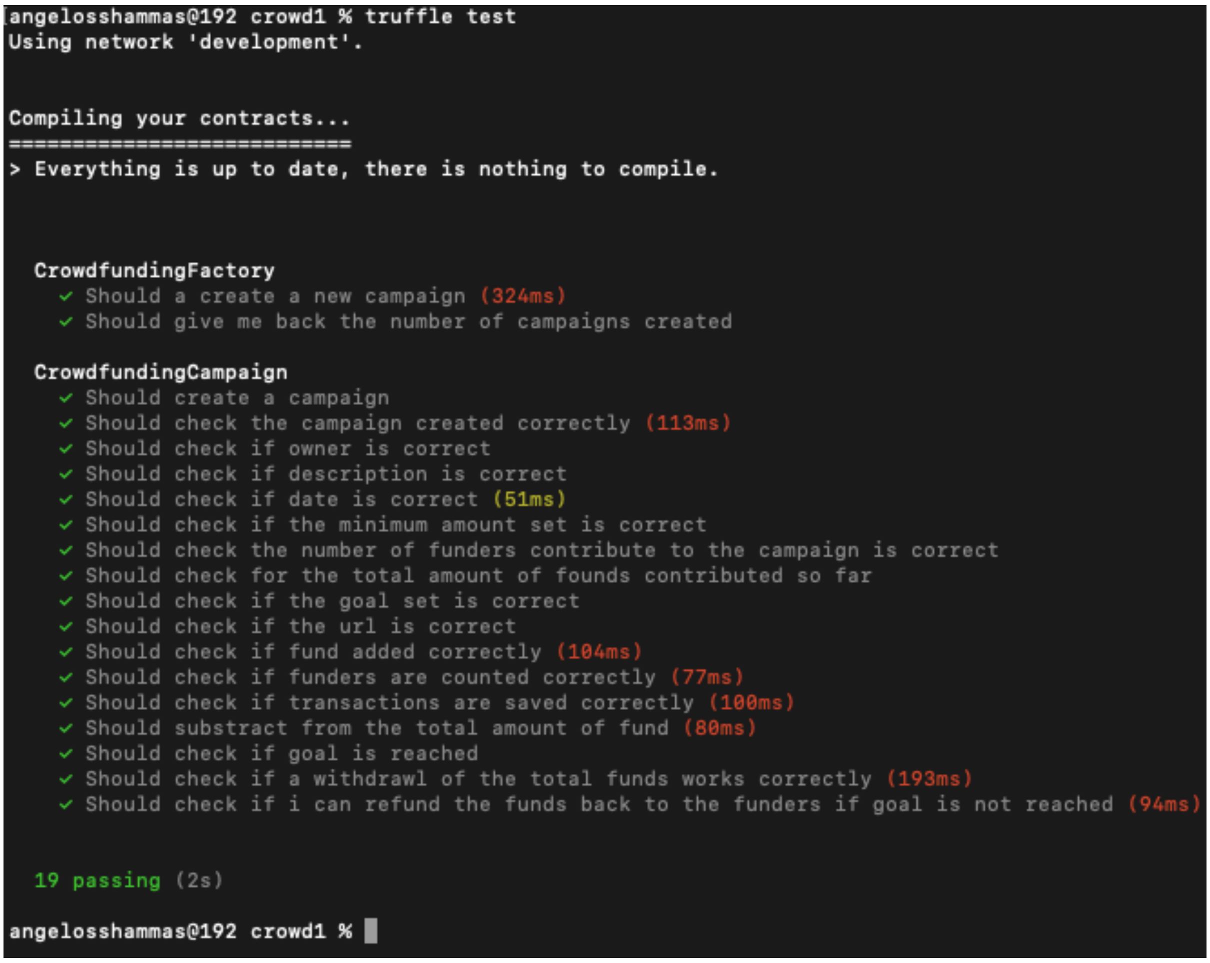
From the results that have been taken, we can split them into three different categories the back-end, testing, and the front-end results. The results are a representation of some visual values of the campaigns create and even the testing for both of the smart contracts before deploying them back to the Ethereum blockchain.



```
truffle(development)> await factory.createNewCampaign(1,"a",2,4,"b")
{
  tx: '0x255f7ea869a98a5642ff1e45cc440bf8cf1f1ed4c8fc7ec72d8ade8e74314011',
  receipt: {
    transactionHash: '0x255f7ea869a98a5642ff1e45cc440bf8cf1f1ed4c8fc7ec72d8ade8e74314011',
    transactionIndex: 0,
    blockHash: '0x22bb5b1aa3787c0d0302aab905caa1fd6d8dcdcd00b04085938001de87ca7128',
    blockNumber: 51,
    from: '0x6bc28450c03384c8ae88c05b68f9d486844d2f70',
    to: '0xc1f33bea17c237308efa6b96e84fdf7866b14609',
    gasUsed: 1098790,
    cumulativeGasUsed: 1098790,
    contractAddress: null,
    logs: [],
    status: true,
    logsBloom: '0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000',
    v: '0x1c',
    r: '0x974db608e4ced649ed66cfbc905b999fb4c3e0457b889c051cbf45002329b65b',
    s: '0x65d724c50b1e7c1bd8b6b5e870690cbf9835e323f4de8af7a0915ff12c8d0b84',
    rawLogs: []
  },
  logs: []
}
truffle(development)> let a = await factory.getCampaigns(0)
undefined
truffle(development)> let c = await CrowdfundingCampaign.at(a)
undefined
truffle(development)> let d = await c.getDescription()
undefined
truffle(development)> d
'asd'
truffle(development)> 
```

Image 4.1: This is a visual representation of our backend where we truffle console in order to interact with our smart contracts. At first, what we do is to deploy the factory smart contract and then create a new campaign, afterwards, we take the address of the newly created smart contract and we deployed it at the specific address we got to retrieve some information, and as we can see am fetching back the correct description which was first created.

What we can see from the image, see image 4.1, presented above is how at first at the creation of the back-end how I interacted with the smart contracts. I was using the truffle console which is a basic interactive console connecting to any Ethereum client. First, we deploy the smart contract using the .deployed() and store the instance of it into a let variable, and like that we deployed our first smart contract. In continues, we can then get the address of the campaign from the factory method getCampaigns() in order to fetch the newly deployed smart contract using its address. The next step is to fetch back whatever information the new campaign has and manipulated as we like.



```
angelosshammas@192 crowd1 % truffle test
Using network 'development'.


Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.



  CrowdfundingFactory
    ✓ Should a create a new campaign (324ms)
    ✓ Should give me back the number of campaigns created

  CrowdfundingCampaign
    ✓ Should create a campaign
    ✓ Should check the campaign created correctly (113ms)
    ✓ Should check if owner is correct
    ✓ Should check if description is correct
    ✓ Should check if date is correct (51ms)
    ✓ Should check if the minimum amount set is correct
    ✓ Should check the number of funders contribute to the campaign is correct
    ✓ Should check for the total amount of founds contributed so far
    ✓ Should check if the goal set is correct
    ✓ Should check if the url is correct
    ✓ Should check if fund added correctly (104ms)
    ✓ Should check if funders are counted correctly (77ms)
    ✓ Should check if transactions are saved correctly (100ms)
    ✓ Should substract from the total amount of fund (80ms)
    ✓ Should check if goal is reached
    ✓ Should check if a withdrawl of the total funds works correctly (193ms)
    ✓ Should check if i can refund the funds back to the funders if goal is not reached (94ms)


  19 passing (2s)

angelosshammas@192 crowd1 %
```

Image 4.2: This is what the compile giving back when the tests are been executed, again we are using the truffle library and the command we are giving is truffle test, which test is Javascript file in my directory.

On the above image, see image 4.2, we can see all the tests that are been executed for both smart contracts, as we mention we testing if the deployment is correct and if they are fetching back the correct information for the campaigns which are tested. We can see there are 19 tests which are been executed in which all 19 passes correctly, we are using the truffle test here which is a command that gets ur test.js file and executes the codes of the test. We can see also a checkmark on the left side of the description of the test indicating the passing of the test and also on the end the real execution time needed for each test.

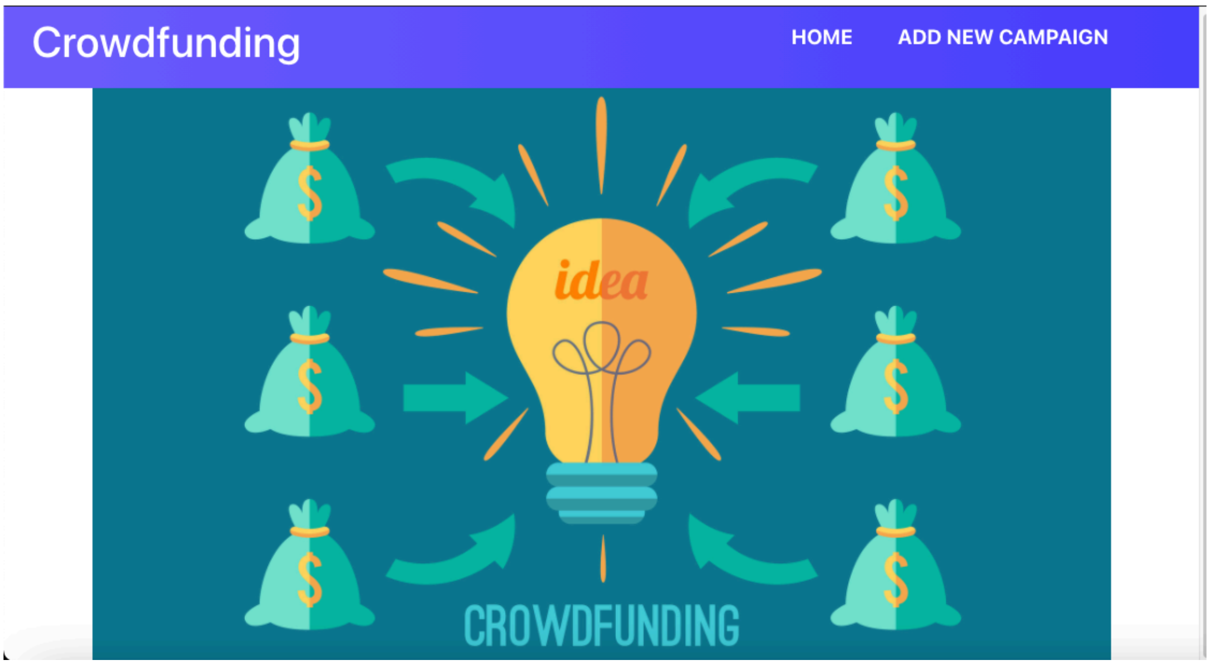
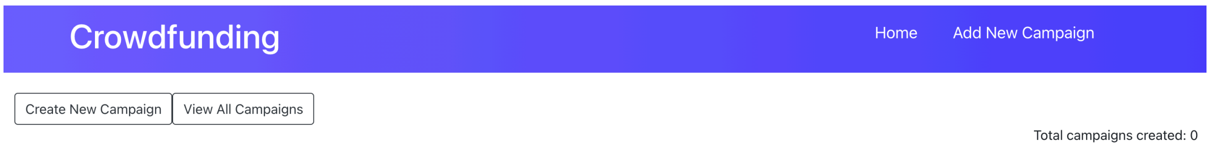
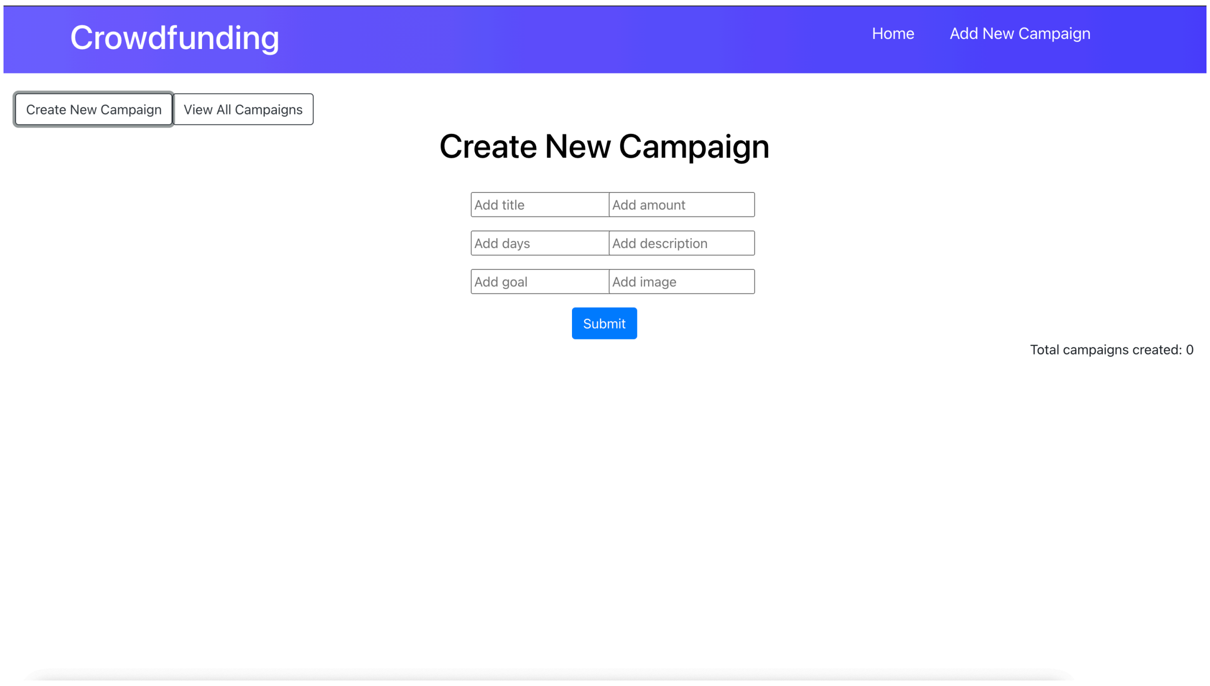


Image 4.3: The image above is a visual representation of the front-end Home page.



Total campaigns created: 0

Image 4.4: The image above is a visual representation of the front-end Add New Campaign page with the two options to create or view campaigns.

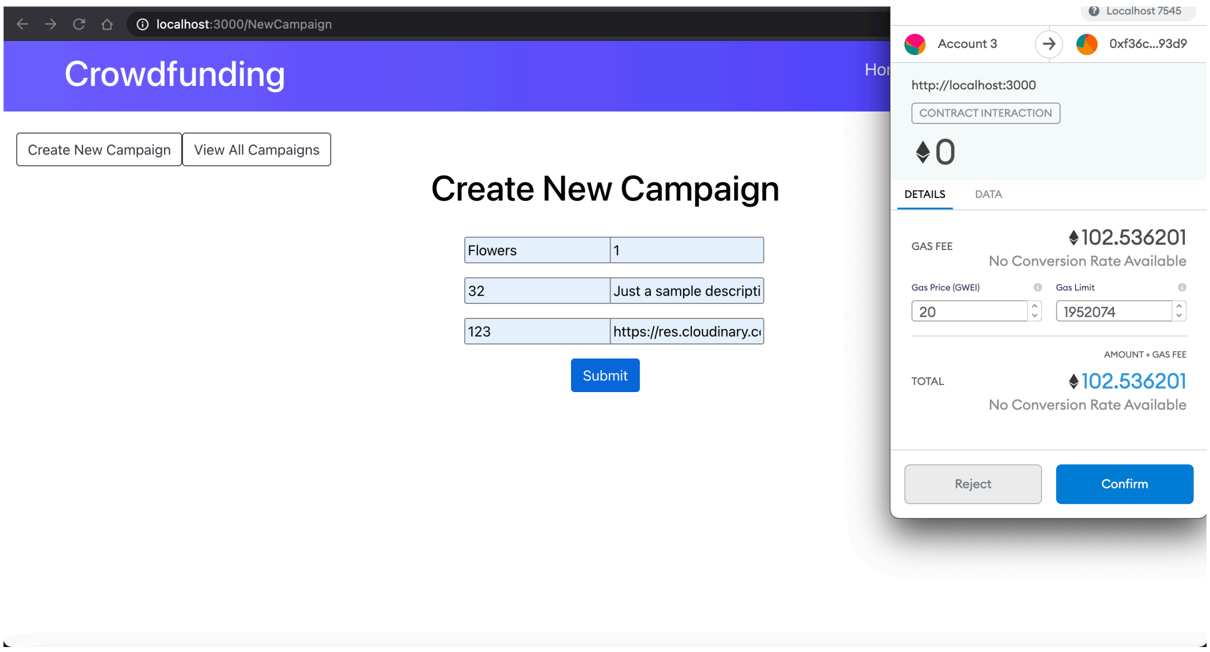


Create New Campaign

Add title	Add amount
Add days	Add description
Add goal	Add image

Total campaigns created: 0

Image 4.5: The image above is a visual representation of the front-end Add New Campaign page with the option to create a new campaign.



Create New Campaign

Flowers	1
32	Just a sample descripti
123	https://res.cloudinary.c

http://localhost:3000

CONTRACT INTERACTION

0

GAS FEE 102.536201
No Conversion Rate Available

Gas Price (GWEI) 20
Gas Limit 1952074

AMOUNT + GAS FEE
TOTAL 102.536201
No Conversion Rate Available

Image 4.6: The image above is a representation of when a new campaign is created, and on the top right we see the pop-up of metamask asking for confirmation of the transaction when clicked submit.

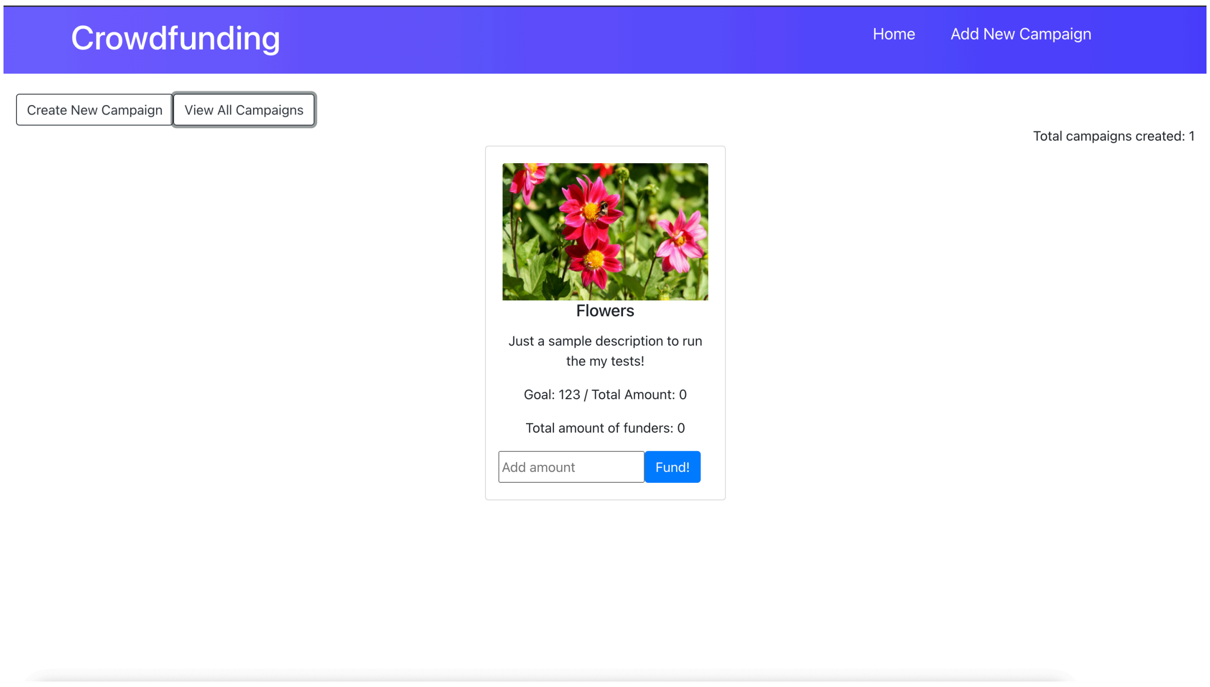

Crowdfunding
Home
Add New Campaign
Create New Campaign
View All Campaigns
Total campaigns created: 1
Flowers
Just a sample description to run the my tests!
Goal: 123 / Total Amount: 0
Total amount of funders: 0
Add amount
Fund!

Image 4.7: The above image represents in a card container the campaign on the blockchain with the ability to contribute to the campaign.

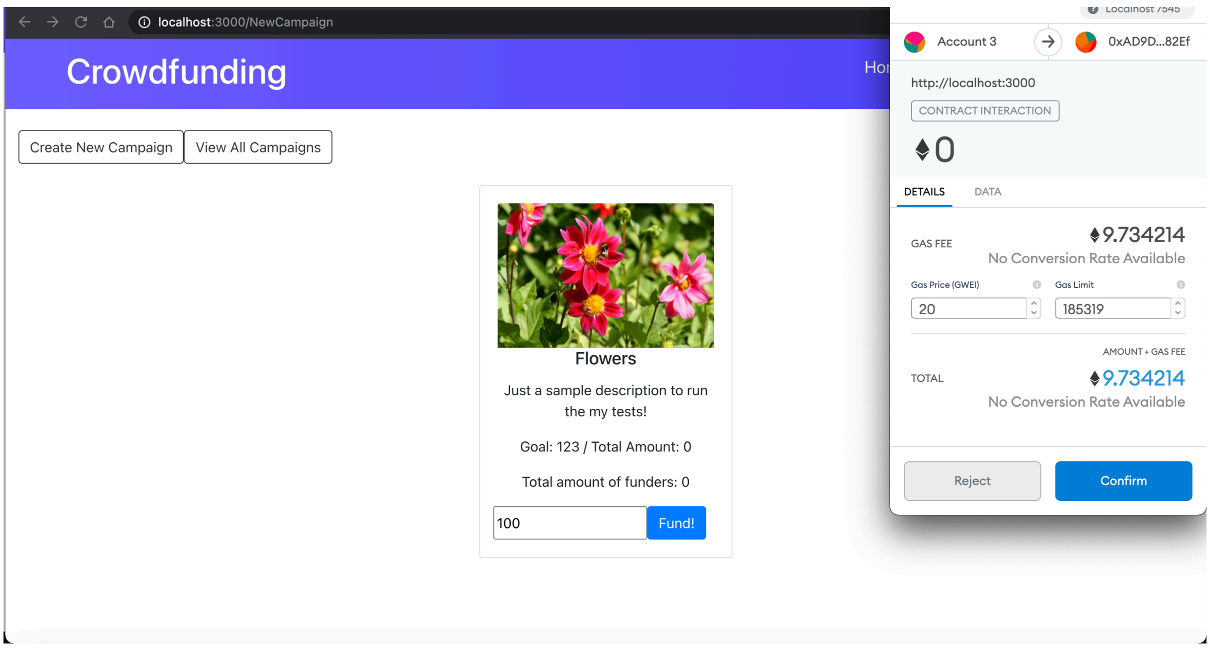
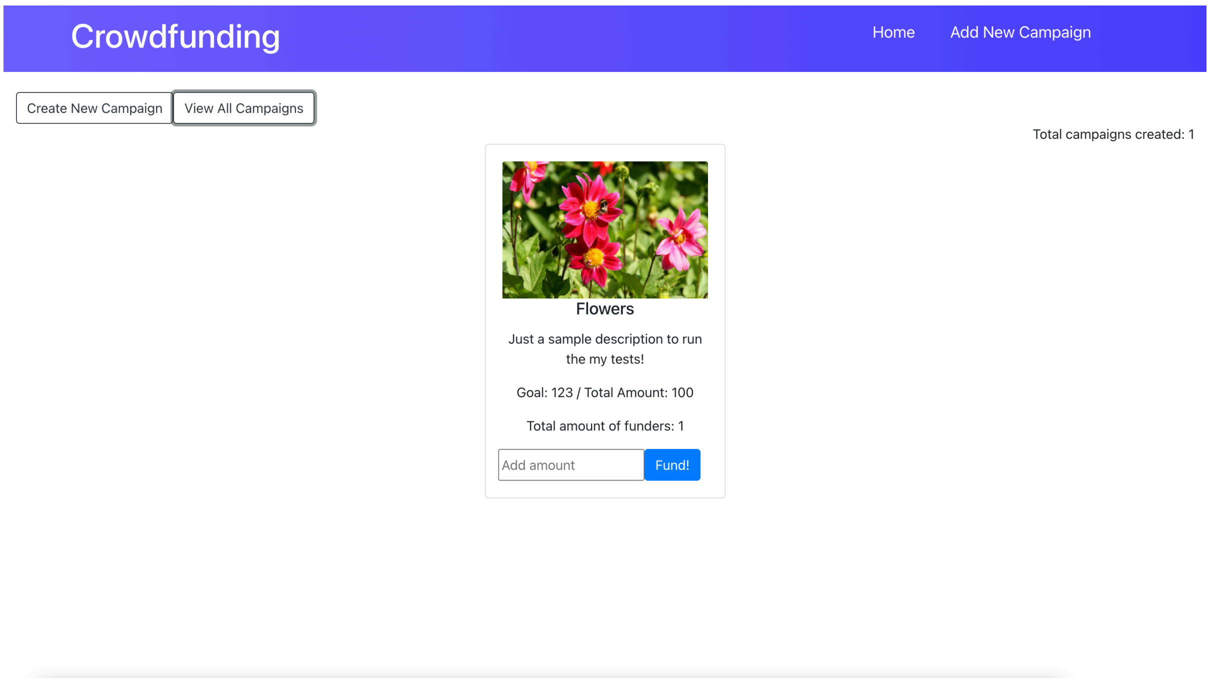


Image 4.8: The image above shows the pop-up again of metamask when a user wants to contribute to a campaign, and metamask asks for confirmation when the button Fund! In the card is clicked.



Total campaigns created: 1

Flowers

Just a sample description to run the my tests!

Goal: 123 / Total Amount: 100

Total amount of funders: 1

Image 4.9: This is image represents the result of the contribution made.

Images 4.3-4.9 representing a simple example taken into consideration, in order for someone to get an idea of how the front-end represents the campaigns. In the images shown above, we display the actions taken to display and interact with the campaign. First, we open the website and see the Home page, then we navigate to the Add New Campaign and display what it contains. Afterward, we creating a new campaign to the blockchain with the metamask popping up on the screen for confirmation of the transaction. We refresh our page and we then navigate to the View All Campaigns button in order to see the new campaign, and in the end, we try to contribute to it and displayed the updated result of the total amount and funders on the campaign.

Chapter 5

Conclusions

5.1 Future Work

I can see a lot of potential for this project to evolve and be used in real-time network environments. However, new features need to be added in order to have a more professional ready to run website. First, real-time testing on running networks needs to be made in order to see how our smart contracts interact with different scale networks. In addition, to involve in a real-time network we can add a new feature on the front-end where we can end the campaign when the number of days is reached, with the proportional act. More on the front-end, another good feature is to connect a database that holds usernames and passwords, in order to have a login page. This will allow us to differentiate the user entity into two new ones, the creator and the funder. On the back-end, a good feature to add in the future will be the implementation of an ERC20 Token, which will act as currency on our website.

5.2 Conclusions

The project was to develop a crowdfunding platform based on Ethereum blockchain as the title mention, and this is what the current project is all about. There are three key categories take into consideration on the project, front-end, testing, back-end. Each one of these categories with an equal amount of hard work. The back-end with the design problem, testing with the repeating coding, and front-end with the lack of knowledge to the errors which were displayed to me in the course of the development. Many step-backs and retries in my section of the development of the project and are all because blockchain is a new technology and one day something is working properly the next is not. A good example would be bigNumber.js dependence in React on the front-end, where you cannot send as attribute an int256 to a function, because of the potential passing object as a value. This specific problem was not taken into consideration when I started to develop the platform, but along the way, this depended needed to add to the project.

In conclusion, the current dissertation was a good example for me to get hands-on the blockchain technology and experience in advance the creation and deployment of smart contracts. In addition, the development of the front-end with React.js gave me the opportunity to create a website connected with my smart contracts, with the ability to learn a new set of skills of web development.

Bibliography

1. https://www.trufflesuite.com/
2. https://nodejs.org/en/
3. https://ethereum.org/en/developers/docs/smart-contracts/
4. https://www.blockchain.com/
5. https://reactjs.org/docs/getting-started.html
6. https://docs.soliditylang.org/en/v0.8.4/
7. https://www.trufflesuite.com/ganache
8. https://www.trufflesuite.com/docs/truffle/testing/writing-tests-in-javascript
9. Meunier, S. “When Do You Need Blockchain? Decision Models”, Medium, August 4, 2016, available at https://medium. com/@sbmeunier/when-do-you-need-blockchain-decision-modelsa5c40e7c9ba1.
10. Sigrid Seibold, George Samman, Consensus Immutable agreement for the Internet of value, 2016.
11. L. Lamport, R. Shostak, and M. Pease, “The byzantine generals problem,” ACM Transactions on Programming Languages and Systems (TOPLAS), vol. 4, no. 3, pp. 382–401, 1982
12. https://www.finra.org/sites/default/files/2017_BC_Byte.pdf
13. Vitalik Buterin. 2014. A next-generation smart contract and decentralized application platform. white paper(2014).
14. Satoshi Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System, 2008
15. Preetchi Kasireddy, How does Ethereum work, anyway?, 2017

Annex A

Back-end

```
pragma solidity >=0.4.21 <0.7.0;
pragma experimental ABIEncoderV2;



contract CrowdfundingFactory {

  CrowdfundingCampaign[] allCampaigns;
   uint public count;


      function createNewCampaign(string memory _title, uint _minAmount, string memory
_description, uint _date, uint _goal,string memory _imageURL) public {
            allCampaigns.push(new CrowdfundingCampaign(_title, _minAmount, _description,
_date, _goal, _imageURL));
      count++;
   }

   function getCampaigns(uint index) public view returns(CrowdfundingCampaign) {
      require(index >=0, "Wrong input for index!");
      CrowdfundingCampaign temp = allCampaigns[index];
      return temp;
   }

   function getCount() public view returns(uint) {
      return count;
   }

}

contract CrowdfundingCampaign {
   struct Transaction {
      address funder;
      uint amount;
   }

   address owner;
   string description;
   uint date;
   uint minimumAmountOfFund;
   uint countFunders;
   uint totalAmountOfFunds;
   uint goal;
   string image_url;
   Transaction[] public transactions;
   string title;
   mapping(address => uint) public refund;

    constructor(string memory _title, uint _minimumAmountOfFund, string memory _desc,
uint _date, uint _goal, string memory _image_url) public payable {
      require(_minimumAmountOfFund >= 0,"Wrong input minimum amount!");
      require(_date >= 0,"Wrong input for date!");
      require(_goal >= 0,"Wrong input for goal!");
      title = _title;
      minimumAmountOfFund = _minimumAmountOfFund;
      owner = msg.sender;
      description = _desc;
      date = _date;
      countFunders = 0;
      totalAmountOfFunds = 0;
      goal = _goal;
      image_url = _image_url;
   }
```

```
function getTitle() public view returns(string memory) {
   return title;
}

function getOwner() public view returns(address) {
   return owner;
}

function getDescription() public view returns(string memory) {
   return description;
}

function setDescription (string memory desc,uint amount) public {
   description = desc;
   amount = amount;
}

function getDate() public view returns(uint) {
   return date;
}

function getMinAmount() public view returns(uint) {
   return minimumAmountOfFund;
}

function getCountFunders() public view returns(uint) {
   return countFunders;
}

function getTotalAmountOfFunds() public view returns(uint) {
   return totalAmountOfFunds;
}

function getGoal() public view returns(uint) {
    return goal;
}

function getImageUrl() public view returns(string memory){
    return image_url;
}

function addFund(uint _amount) public {
    require(_amount >= 0,"Wrong input!");
    totalAmountOfFunds = totalAmountOfFunds + _amount;
}

function CountAllFunders() public {
    countFunders++;
}

function saveTransaction(address _address, uint _amount) public {
    require(_amount >= 0,"Wrong input for amount!");
    require(_address != address(0),"Wrong input for address!");
    transactions.push(Transaction(_address, _amount));
}

 function showAllTransactions(uint _index) public view returns(Transaction memory) {
    require(_index >= 0,"Wrong input!");
    Transaction memory t = transactions[_index];
    return t;
}

function substractTotal(uint _amount) public {
    require(_amount >= 0,"Wrong input for amount!");
    totalAmountOfFunds = totalAmountOfFunds - _amount;
}

function makeAFund(uint _amount) public {
        require(_amount >= 0,"Wrong input for amount!");
        require(_amount >= getMinAmount());
        addFund(_amount);
        CountAllFunders();
        saveTransaction(msg.sender, _amount);
    }

    function reachGoal() public view returns(bool) {
        return (getTotalAmountOfFunds() >= getGoal());
    }

    function reachIsGoal() public {
        totalAmountOfFunds = 0;
        description = 'You reach your goal!';
        countFunders = 0;
        goal = 0;

    }

    function makeAWithdrawl() public  {
        require(reachGoal());
        reachIsGoal();
    }

    function GoalNotReach() public {
        for(uint i=0;i<transactions.length;i++){
            refund[transactions[i].funder] = transactions[i].amount;
            substractTotal(transactions[i].amount);
            transactions[i].amount = 0;
        }
    }
}
```
Testing

```
const CrowdfundingFactory = artifacts.require('CrowdfundingFactory');
const CrowdfundingCampaign = artifacts.require('CrowdfundingCampaign');
//const truffleAssert = require('truffle-assertions');

describe('CrowdfundingFactory', async function(){

	let campaign = null;
	beforeEach(async function() {
		campaign = await CrowdfundingFactory.deployed()
	})

	it('Should a create a new campaign', async function(){
		await campaign.createNewCampaign("a",1,"a",2,4,"b")
		let a = await campaign.getCampaigns(0)
		let c = await CrowdfundingCampaign.at(a)
		let d = await c.getDescription()
		let min = await c.getMinAmount()
		let date = await c.getDate()
		let goal = await c.getGoal()
		let url = await c.getImageUrl()

		assert.equal(d, "a")
		assert.equal(min.toNumber(), 1)
		assert.equal(date.toNumber(), 2)
		assert.equal(goal.toNumber(), 4)
		assert.equal(url, "b")
	})

	it('Should give me back the number of campaigns created', async function() {
		let count = await campaign.getCount();

		assert.equal(count.toNumber(),1)
	})
})
```

```
describe('CrowdfundingCampaign', async function() {

        let campaign = null;
        beforeEach(async function() {
                campaign = await CrowdfundingCampaign.deployed()
        })

        it('Should create a campaign', async function() {
                campaign = await CrowdfundingCampaign.deployed()
        })

        it('Should check the campaign created correctly', async function() {
                let d = await campaign.getDescription()
                let min = await campaign.getMinAmount()
                let date = await campaign.getDate()
                let goal = await campaign.getGoal()
                let url = await campaign.getImageUrl()

                assert.equal(d, "A")
                assert.equal(min.toNumber(), 1)
                assert.equal(date.toNumber(), 2)
                assert.equal(goal.toNumber(), 3)
                assert.equal(url, "b")

        })

        it('Should check if owner is correct', async function() {
                let campaignAddress = await campaign.getOwner();
                let add = "0x6Bc28450C03384C8aE88c05b68f9D486844D2F70"

                assert.equal(add, campaignAddress)
        })

        it('Should check if description is correct', async function() {
```

```
        let description = await campaign.getDescription()

        assert.equal(description,"A")
    })

    it('Should check if date is correct', async function() {
        let date = await campaign.getDate()

        assert.equal(date.toNumber(), 2)
    })

    it('Should check if the minimum amount set is correct', async function() {
        let min = await campaign.getMinAmount()

        assert.equal(min.toNumber(), 1)
    })

    it('Should check the number of funders contribute to the campaign is correct', async
function() {
        let countF = await campaign.getCountFunders()

        assert.equal(countF.toNumber(), 0)
    })

    it('Should check for the total amount of founds contributed so far', async function() {
        let total = await campaign.getTotalAmountOfFunds()

        assert.equal(total.toNumber(), 0)
    })

    it('Should check if the goal set is correct', async function() {
        let goal = await campaign.getGoal()

        assert.equal(goal.toNumber(), 3)
        })

        it('Should check if the url is correct', async function() {
                let url = await campaign.getImageUrl()

                assert.equal(url, "b")
        })

        it("Should check if fund added correctly", async function() {
                await campaign.addFund(1)
                let total = await campaign.getTotalAmountOfFunds()

                assert.equal(total.toNumber(), 1)
        })

        it('Should check if funders are counted correctly', async function() {
                let count = await campaign.CountAllFunders()
                let result = await campaign.getCountFunders()

                assert.equal(result.toNumber(), 1)
        })

        it('Should check if transactions are saved correctly', async function() {
                let          transaction          =          await
campaign.saveTransaction("0x6Bc28450C03384C8aE88c05b68f9D486844D2F70", 1)
                let result = await campaign.showAllTransactions(0)


expect([result[0]]).to.eql(["0x6Bc28450C03384C8aE88c05b68f9D486844D2F70"])
                expect([result[1]]).to.eql(["1"])
        })

        it('Should substract from the total amount of fund', async function() {
                let sub = await campaign.substractTotal(1)
```

```
		let result = await campaign.getTotalAmountOfFunds()


		assert.equal(result.toNumber(), 0)
	})

	it('Should check if goal is reached', async function() {
		let goal = await campaign.reachGoal()


		assert.equal(goal, false)
	})

	it('Should check if a withdrawl of the total funds works correctly', async function() {
		let goalprints = await campaign.reachIsGoal()
		let total = await campaign.getTotalAmountOfFunds()
		let description = await campaign.getDescription()
		let count = await campaign.getCountFunders()
		let goal = await campaign.getGoal()



		assert.equal(total.toNumber(), 0)
		assert.equal(description, 'You reach your goal!')
		assert.equal(count.toNumber(), 0)
		assert.equal(goal.toNumber(), 0)
	})

	it('Should check if i can refund the funds back to the funders if goal is not reached',
async function() {
		let refund = await campaign.GoalNotReach()
		let result = await campaign.showAllTransactions(0)



expect([result[0]]).to.eql(["0x6Bc28450C03384C8aE88c05b68f9D486844D2F70"])
		expect([result[1]]).to.eql(["0"])
	})
```

```
})
```

Front-end

App.js

```
import React, {Component} from 'react'
import "./App.css"
import Home from './Home.js'
import Newcampaign from './NewCampaign.js'
import {Route, Link} from 'react-router-dom'
import Navbar from "./components/Navbar/Navbar";

class App extends Component{

constructor(props){
        super(props)
        this.state={
                campaigns:"0xc28d3006142Bb479b6C374B8615F5d82D9EA112c"
        }
this.getAddress = this.getAddress.bind(this)
}
getAddress = (campainAdd) => {
   this.setState({campaigns: campainAdd})
}

render(){
  return (

   <div className="App">
        <Navbar />
        <Route exact path='/' render = { () => < Home name={this.state.campaigns}/> } />
        <Route exact path='/Newcampaign' component={Newcampaign}
      />
   </div>
   )}
}
export default App
```

Home.js

```
import React,{Component} from 'react'
import Web3 from 'web3'
import CrowdfundingCampaign from './contracts/CrowdfundingCampaign.json'
import home from './home.css'



class Home extends Component{

        render(){
                return(
                <img style={{width:950}}src={require('./chi.jpeg')}/>
                )
        }
}


 export default Home
```

NewCampaign.js

```
import React, {Component} from 'react'
import Web3 from 'web3'
import CrowdfundingFactory from './contracts/CrowdfundingFactory.json'
import CrowdfundingCampaign from './contracts/CrowdfundingCampaign.json'
import Main from './Main.js'
import Home from './Home.js'
import App from './App.js'
import view from './view.js'
import axios from 'axios'

import MyCard from './MyCard.js'
import NumericInput from 'react-numeric-input';
var bigInt = require("big-integer");

class NewCampaign extends Component{
        componentDidMount = async () => {
                await this.loadWeb3()

                const web3 = window.web3

                const accounts = await web3.eth.getAccounts()
                this.setState({account: accounts[0]})
                const networkId = await web3.eth.net.getId()
                const networkData = CrowdfundingFactory.networks[networkId]
                if(networkData){
                        const factory = new web3.eth.Contract(CrowdfundingFactory.abi,
networkData.address)
                        this.setState({ factory })
                } else {
                        window.alert('Contract has not deployed to detected network')
                }

                await this.loadBlockchainData()
                await this.loadCampaignData()



        }

                //This function connects the app with the blockchain.
        async loadWeb3 (){
                if (window.ethereum){
                        window.web3 = new Web3(window.ethereum)
                        await window.ethereum.enable()
                }
                 else if (window.web3){
                        window.web3 = new Web3(window.web3.currentProvider)
                }
                else {
                        window.alert('Error on detection')
                }
        }

        async loadBlockchainData (){
                let countCampaigns= await this.state.factory.methods.getCount().call()
 this.setState({countCampaigns},function(){console.log('count',this.state.countCampaigns)})

                if(countCampaigns > 0){
                        for(var i=0;i<countCampaigns;i++){
                        const            campaignAdd            =            await
this.state.factory.methods.getCampaigns(i).call()
                                this.setState({
                                        campaignAdd: [...this.state.campaignAdd,
campaignAdd]
                                })

                        }
                }
```

```
            this.setState({loading: false})
        }

        async loadCampaignData(){
            await this.loadWeb3()
            const web3 = window.web3
            const accounts = await web3.eth.getAccounts()
            this.setState({account: accounts[0]})

            const networkId = await web3.eth.net.getId()
            const networkData = CrowdfundingCampaign.networks[networkId]

            for(var i=0;i<this.state.campaignAdd.length;i++){
            if(networkData){
                const campaign = new web3.eth.Contract(CrowdfundingCampaign.abi,
this.state.campaignAdd[i])
                this.setState({ campaign })

            } else {
                window.alert('Contract has not deployed to detected network')
            }

            let description = await
this.state.campaign.methods.getDescription().call({from:this.state.campaignAdd[i]})
            this.setState({
                        description: [...this.state.description, description]
                    })

            let date = await
this.state.campaign.methods.getDate().call({from:this.state.campaignAdd[i]})

            this.setState({
                        date: [...this.state.date, date]
                    })
        let goal = await
this.state.campaign.methods.getGoal().call({from:this.state.campaignAdd[i]})
        this.setState({
                goal: [...this.state.goal, goal]
            })
            let urls = await
this.state.campaign.methods.getImageUrl().call({from:this.state.campaignAdd[i]})
        this.setState({
                urls: [...this.state.urls, urls]
            })
        let titles = await
this.state.campaign.methods.getTitle().call({from:this.state.campaignAdd[i]})
        this.setState({
                titles: [...this.state.titles, titles]
            })
        let totals = await
this.state.campaign.methods.getTotalAmountOfFunds().call({from:this.state.campaignAdd[i]
})
        this.setState({
                totals: [...this.state.totals, totals]
            })
        let funders = await
this.state.campaign.methods.getCountFunders().call({from:this.state.campaignAdd[i]})
        this.setState({
                funders: [...this.state.funders, funders]
            })
        }
        }
    uploadNewCampaign= (title,amount,description,date,goal,image) => {

        this.setState({loading: true})
        this.state.factory.methods.createNewCampaign(title,
amount,description,date,goal,image).send({from:   this.state.account}).on('transactionHash',
(hash) =>{
        this.setState({loading: false})

        })
        console.log("created!")
    }

    hideComponent(name) {
        console.log(name)
        switch(name) {
            case "showHideMain":
            this.setState({showHideMain: !this.state.showHideMain});

            break;
            case "showHideView":
            this.setState({showHideView: !this.state.showHideView});

            default:
            break;
        }

    }

    uploadAFund(amount){

        this.setState({loading: true})
        this.state.campaign.methods.makeAFund(amount).send({from:
this.state.account}).on('transactionHash',(hash) =>{
        this.setState({loading: false})

        })
        console.log("Funded!")
```

```
}

constructor(props){
        super(props)

        this.state = {
                account:'',


                campaignAdd:[],
                factory: null,
                campaign:null,
                countCampaigns:'',
                description:[],
                loading: true,
                secure_url:'',
                urls:[],
                showHideMain:false,
                showHideView:false,
                date:[],
                goal:[],
                titles:[],
                totals:[],
                funders:[]

        }
        this.uploadNewCampaign = this.uploadNewCampaign.bind(this)
        this.hideComponent = this.hideComponent.bind(this)
        this.uploadAFund = this.uploadAFund.bind(this)

}

render(){
        const mystyle={margin:"auto",width:"60%",padding:"10px"}
        const items =[]
```

```
                for(var i=0;i<this.state.campaignAdd.length;i++){

                        items.push(<div key={i}><div className="card" style={{width:
'18rem'}}>

                <div className="card-body">
                <img className="card-img-top" src={this.state.urls[i]} alt="Card image
cap"/>
        <h5 className="card-title">{this.state.titles[i]}</h5>
        <p className="card-text">{this.state.description[i]}</p>
        <p className="card-text">Goal: {this.state.goal[i]} / Total Amount:
{this.state.totals[i]}</p>
        <p className="card-text">Total amount of funders: {this.state.funders[i]}</p>
        <form onSubmit={(event) =>{
                                        event.preventDefault()
                                        const contributeAmount = this.contributeAmount.value
                                        this.uploadAFund(contributeAmount)
                }}>
         <div className="form-row">
        <input id="contributeAmount" type="text" ref={(input) => {this.contributeAmount =
input}}  placeholder="Add amount" required/>
                <button type="sumbit" className="btn btn-primary">Fund!</button>
                </div>
        </form>
                </div>
                </div>
                </div>)
                }
                const showHideMain = this.state.showHideMain
                const showHideView = this.state.showHideView
        return(
                <div>

                        <div className="col-sm-10 text-left">
```

```
                    <br></br>
                              <button className="btn btn-outline-dark" onClick={() =>
this.hideComponent("showHideMain")}>
          Create New Campaign
        </button>
                              <button className="btn btn-outline-dark" onClick={() =>
this.hideComponent("showHideView")}>
          View All Campaigns
        </button>
        </div>

                {showHideMain && <Main
uploadNewCampaign={this.uploadNewCampaign} /> }

<div className="col-sm-12 text-right"><p className="card-text">Total campaigns created:
{this.state.countCampaigns}</p></div>

                <div>{showHideView && items }</div>
                </div>


          )}
 }

 export default NewCampaign
```