

## Crowdfunding Using Blockchain

Sushanth kumar reddy kura<sup>1</sup>, Trupthi M<sup>2</sup>

<sup>1,2</sup>(IT, CBIT, India)

### Abstract.

*There are lots of developments in the fields of science and technology but even in these days very important agreements related to properties, Crowdfunding are made on paper which is not at all safe as they can be duplicated or forgery can be done. Even if the agreements are digital they can be hacked if no security surrounds them. But if agreements, applications are made using smart contracts the security will be increased by a large scale. In normal crowd funding the entire amount is directly in the hands of manager and also the amount used by manager doesn't involve investors opinion. So, there is no security ensured in this case. The proposed methodology creates a smart contract in such a way that the entire funding amount will be inside the smart contract and when the manager wants to make use of funding amount and buy something from a vendor, he/she has to make a spending request and then investors have to vote on that request. If the contract gets majority amount of votes the amount will be automatically sent to the vendor. In this way process of crowd funding can be made better and trustworthy.*

**Keywords:**Blockchain, Smart Contracts, Crowdfunding,

## 1 INTRODUCTION

Crowdfunding is the practice of funding a project or venture by raising small amounts of money from a large number of people, typically via the internet [1]. Before Crowdfunding were started people would really struggle a lot to get funding for startups. It was difficult for people to convince investors to invest in their ideas. But now there is no need of such struggle as many Crowdfunding platforms like Kickstarter have evolved which are helping a lot of people for getting the required funding thereby making their dreams come true.

In existing system [1] of Crowdfunding platform like Kickstarter in order to list any campaign you have to first sign in the web application and select your category, verify your bank account, government issued id and debit or credit card. Fig. 1.1. shows an example of a campaign in Kickstarter.



Fig. 1.1.A campaign of Kickstarter

The campaign creator should give details like the purpose of startup, In which category do their idea fall and he/she needs to keep a funding goal and if amount reaches the goal then entire amount excluding some Kickstarter fee will be given to campaign creator. But if the funding doesn't then it will be declared as a unsuccessful and the money reverts back to the people who ever invested in it. Fig. 1.2. shows process involved in creating a campaign on Kickstarter site.



Fig. 1.2. The process involved in creating a campaign on Kickstarter

But despite of having many advantages it also has some cons like no ensured security, directly giving whole funding amount to the person who created the campaign and so on.

In normal Crowdfunding platforms like Kickstarter there are many loop holes. In previous years many frauds have taken place where millions were stolen. People just come and list their startup by giving a fake idea and raise the

money. All the money donated by the investors are being stolen. Fig.1.3. shows the results when Kickstarter frauds is typed in google search. By this we can realize how critical the problem is.

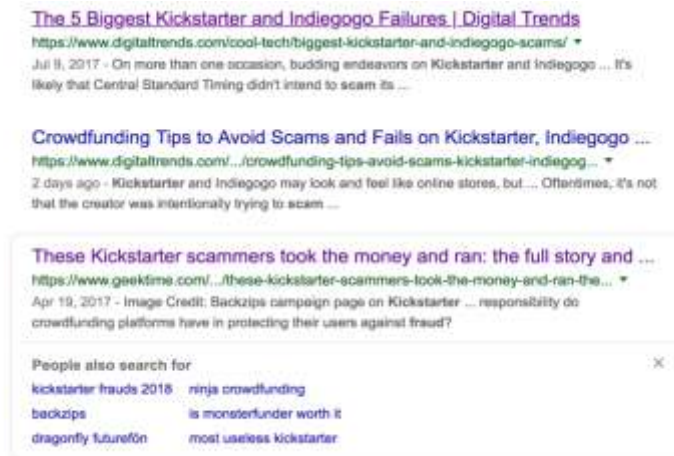


Fig. 1.3. google search result of Kickstarter frauds

So, our project focuses on increasing the security with the help of an application of blockchain technology called smart contract.

## 2 PROPOSED SYSTEM

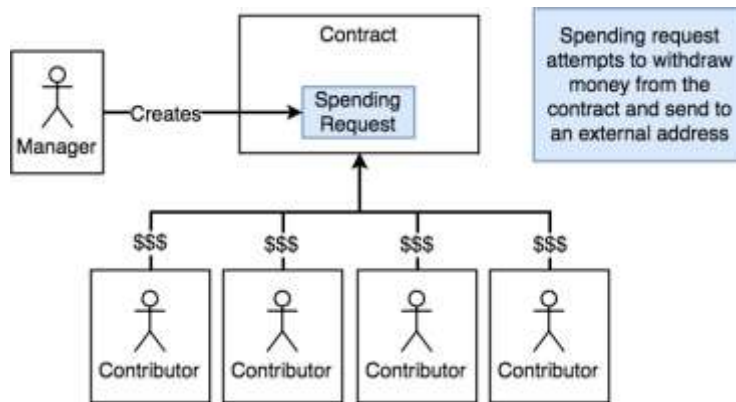
### 2.1 Methodology

The project is a web application which is basically an enhancement of the existing crowd funding systems. In normal Crowdfundings there is no security ensured for contribution amount. So, smart contract of Ethereum platform which is an application of blockchain can be used in order to solve this issue. Blockchain is a decentralized distributed ledger system that accesses, verifies, and transmits network data through distributed nodes [2].

An Ethereum based smart contract is a cryptographic box which stores information, processes inputs, writes outputs and is only accessible to the outside if certain predefined conditions are met [3] and the contracts in Ethereum are written in special language called solidity [4]. In practice, Ethereum allows for an easy implementation of such smart contracts [5] and in addition Ethereum offers developers online compilers of solidity code. Smart contract is written in such a way that the entire amount funded by the contributors will safely be kept in smart contracts so that no one can modify it or steal it. The amount will not be directly given to campaign creator rather it will be held in

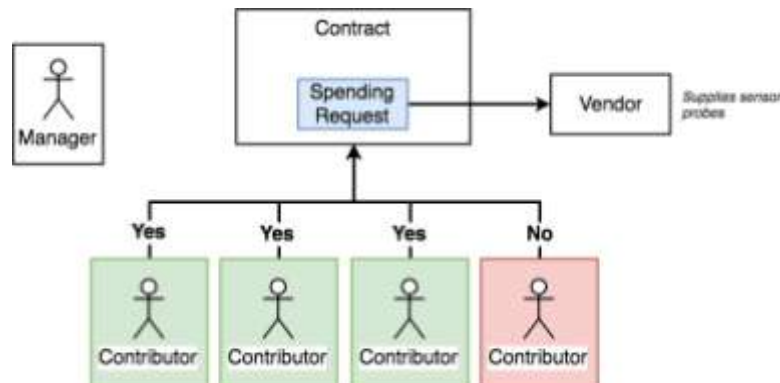
smart contract itself. If the campaign creator wants to use this amount he/she has to create a spending request as shown in Fig.2.1 by giving the following details:

- Why he/she wants to spend the money?
- How much he/she wants to spend?
- To whom(vendor address) he/she wants to send the money?



**Fig.2.1.** Campaign creator or manager creating a spending request

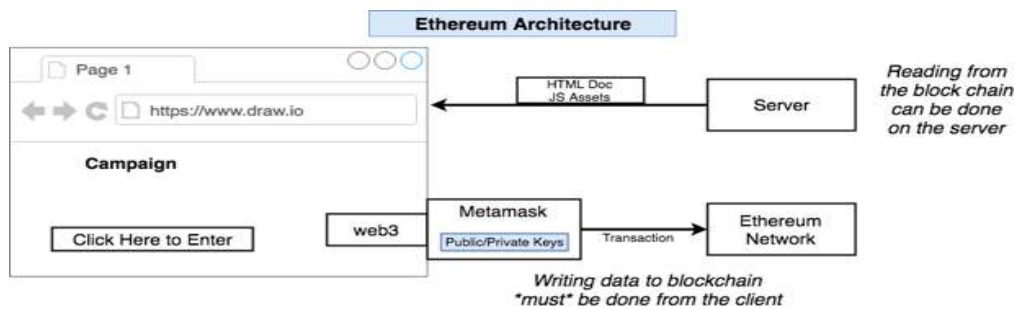
Then the approvers(people who have contributed to the campaign) should approve the request created by the campaign creator. If the request gets majority no of approvals/votes as shown in Fig.2.2 then the amount can be sent to the vendor specified by the campaign creator. The voting system used is decentralized as blockchain technology is used in implementing it. This makes the voting system more secure and also cost efficient while guaranteeing the voters privacy.



**Fig. 2.2.** Contributors voting for the spending request created by manager

Campaign creator will be able to finalize the payment once the required votes are obtained. In this the security is increased and also the peoples/contributors opinion is taken.

## 2.2 Architecture of The Proposed System

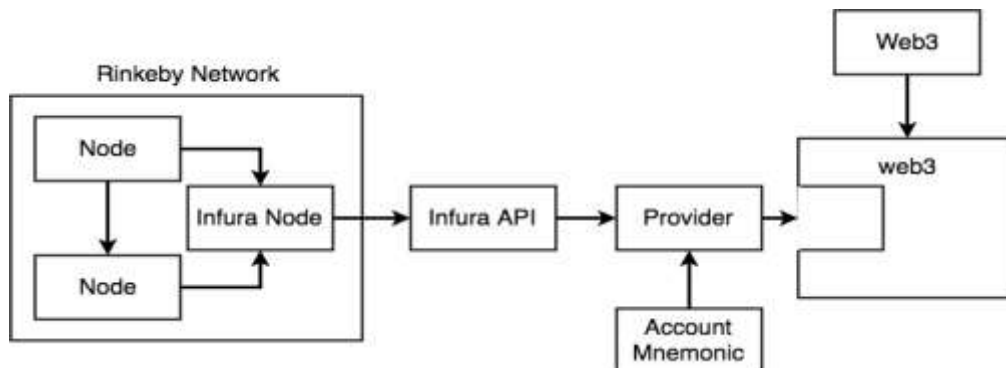


**Fig. 2.3.** Architecture of the Project

Fig.2.3 represents the architecture of the Ethereum which shows how web application with solidity as backend works. Here server does far less work than what it does in a traditional web application. Server just sends some html, javascript content to the browser and but when user does an action or clicks on a button they do not reach server instead Ethereum application running inside the web browser uses web3 and communicates with Metamask [6], then Metamask creates a transaction signs it with user's private key and sends that transaction to Ethereum network. These transactions can be tracked at Etherscan [7].

These public and private keys will be never sent to server because you can never ask for your users private keys. Hence writing to database part is done by the client. So, here the client has to be more powerful that's why react js has been as frontend in this project as it helps in building complex applications.

### 2.2.1 Connecting to the Rinkeby test network(Ethereum network)



**Fig. 2.4.** connecting web3 to Rinkeby



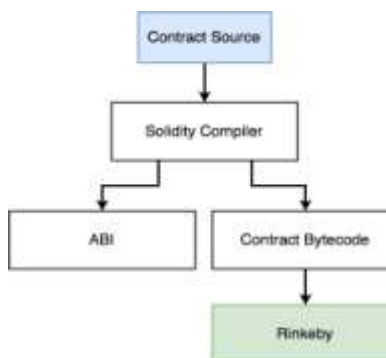
After writing solidity smart contract it follows deploying it to a network. To do that we need a local node running in our machine, provider, Metamask account with some ether in it belonging to Rinkeby test network and web3. In order to connect to any network web3 needs a provider. This particular provider should also contain an account mnemonic which is a key to an account. That particular account should also contain some ether as while deploying the smart contract we need to pay some fees. Now the provider should be connected to a node running in our local machine. But it takes a lot of effort to get that node. So, instead we can create an account in Infuraapi [8] which basically provides nodes to connect to Rinkeby network [7], [9]. Rinkeby is a test network which is suitable to perform experiments as it is the most stable network that developers can use for experimenting with their ideas using Ethereum tokens instead of real ether money. In this way we connect to an actual Ethereum network. To pay for transactions we used ether tokens requested from faucet [10]. Fig.2.4 shows how web3 is connected to Rinkeby.

### 2.2.2 The flow of the Smart Contract

First the smart contract are written with the functionalities that are required in remix solidity open source tool. Then smart contract functionalities are checked and verified before compiling it. Once the functionalities are verified smart contracts are compiled to get the abi and byte code. As the browser cannot understand the byte code, abi is used as an interface for browser and the bytecode to connect to any network like Rinkeby. Fig.2.5 represents the smart contract compilation process.

Then to add user interface to this smart contract web3 and react js are used with semantic-ui. For react to get access to the contract functions web3 is used [11]. Then tests are performed using mocha test runner to check if the functionalities are properly working. All the code is written in node using node js as java script is being run outside the browser. This also helps project to use third party libraries.

Nextjs is used for server side rendering and routing of the web pages. Finally solidity contract is deployed to the Ethereum test network using Infura and gets the address where it is deployed and gives it to the application. Then server is started and at localhost: 3000 web application can be accessed.



**Fig. 2.5.** Smart contract compilation process

### 3 SYSTEM DESIGN

#### 3.1 Flow chart



**Fig.3.1.** Flow chart of the project

Fig.3.1 represents the complete flow of creating the project. First smart contract is created according to our needs. Then the ABI(Application Binary Interface) code and interface of the contract are used to interact with it and then deploy the contract. Then frontend is added to the contract using react js. In this way all web pages are created and nextjs is used as routing mechanism to route this pages and provide them a hyperlink. Then finally server is started. At localhost:3000 we will find our web application running.

#### 3.2 Use case diagram

Fig.3.2 represents the use case diagram of the project. This represents how the application actually works by showing the tasks done by manager and contributor. First the manager creates the campaign. Then contributor will contribute to the campaign. Then after contributors contribute manager creates a spending request. Every contributor becomes an approver after they donate some money. Then contributors vote to this spending request created by the

manager. After at least 51% of the contributors approve the spending request then the amount will be automatically transfer to the vendor specified in the spending request by the manager.

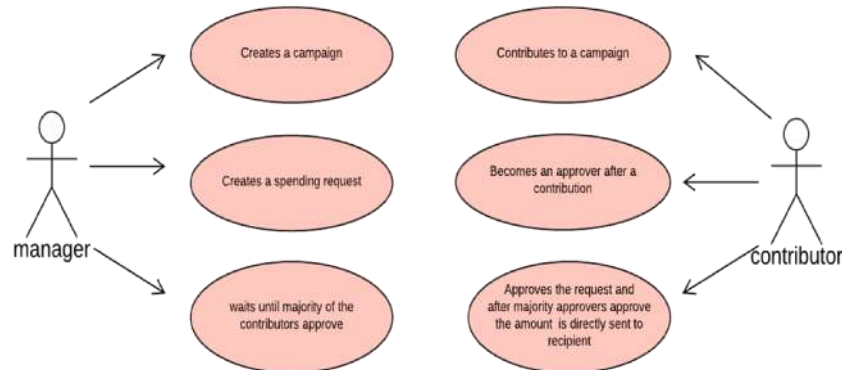


Fig. 3.2. Use case diagram of the project

## 4 IMPLEMENTATION

In order to implement and run the project there are some software requirements that need to be installed and configured as mentioned below:

- Metamask wallet

Metamask allows to run Ethereum decentralized applications in the browser itself without running a full Ethereum node and it is a self-hosted wallet to store, send, and receive Ethereum or ERC20 tokens. It allows to create n number of accounts which are just like bank accounts. Metamask wallet has to be installed from the chrome browser and network has to be set to Rinkeby test network which is available in options at top of the wallet. Then in order to test and run the project some fake Ethereum(currency) is transferred from Rinkeby faucet [10] to the account being used in the project by giving its address.

- Infura key

This key can be obtained from the infura.io website which is specific to the Rinkeby test network so that an Ethereum environment can be established.

- Atom

It is a free and open-source source code editor. First tests have to be performed on the smart contract in remix solidity. After that a folder has to be created in the directory which can have any name for ex: kickstart. Then the contract tested in remix solidity is taken and stored in folder which was created in the



beginning. Then before performing any further activities dependencies mentioned in Fig. 4.1 have to be installed by just typing this command in terminal inside the folder kickstart npm install—save dependency name .

```

"author": "",
"license": "ISC",
"dependencies": {
  "fs-extra": "^7.0.1",
  "ganache-cli": "^5.4.1",
  "nocha": "^6.0.2",
  "next": "^4.1.4",
  "next-routes": "^1.4.2",
  "react": "^16.2.0",
  "react-dom": "^16.2.0",
  "semantic-ui-css": "^2.4.1",
  "semantic-ui-react": "^0.85.0",
  "solc": "^0.4.17",
  "truffle-hdwallet-provider": "0.0.3",
  "web3": "^1.0.0-beta.36"
}

```

**Fig. 4.1.**list of dependencies required for the project.

Then the compile script and deploy script have to be written in order to deploy the contract in actual Ethereum network. Then two other files have to be created in the Ethereum folder called web3.js which is used to interact with web3 by creating a instance of web3 and factory.js that is used to create an instance of the deployed contract. The deployed contract address is placed in factory.js file as shown in Fig. 4.2.

```

import web3 from './web3';
import CampaignFactory from './build/CampaignFactory.json';
const instance= new web3.eth.Contract(
  JSON.parse(CampaignFactory.interface),
  '0x76f12e75C61A81D5A28aAB8898369406ca30FFF6' //address of deployed contract
);

export default instance;

```

**Fig. 4.2.**Placing address of deployed contract in factory.js file

Then front end to this application has to be created. We start by creating index.js file and implement our required functions in it. The components folder is created and is used it to store the components in it which are utilized in project so, that there is no need of implementing the functions again and again.

The deployed contract Abi code and interface code are stored in two files so that they can be imported and used where ever required. The next jsgetInitialProps() method should be used to render the front end of the web application. After creating all the frontend pages they need to be connected to backend i.e. to web3 using routing which is part of next js. A file routes.js is created routes are specified as shown in Fig. 4.3.

```
const routes=require('next-routes')();

routes
.add('/campaigns/new','/campaigns/new')
.add('/campaigns/:address','/campaigns/show')
.add('/campaigns/:address/requests','/campaigns/requests/index')
.add('/campaigns/:address/requests/new','/campaigns/requests/new');

module.exports=routes;
```

**Fig. 4.3.**routes of the webpages

Then a custom server is made which is required to run the web application and it is stored in server.js file as shown in Fig. 4.4. The mocha test runner may also be included inorder to perform some tests if needed. These packages should be included in package.json.

```
{
  "name": "kickstart",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "mocha",
    "dev": "node server.js"
  }
}
```

**Fig. 4.4**including custom server in package.json file

This web application can be executed at the terminal by routing to the foldermain folder i.e. kickstart which was created earlier and then the commandnpm run dev is used to run the web application which can be found at localhost:3000 where we can perform the desired operations on the deployed web application.

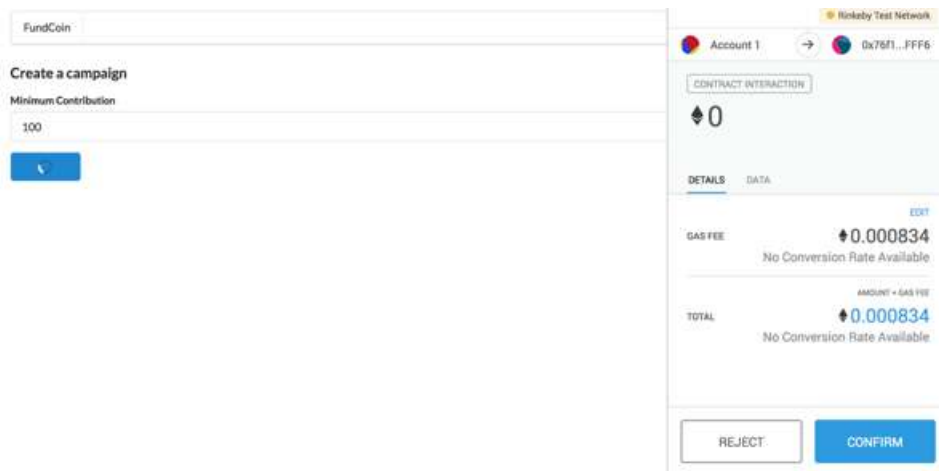
## 5 RESULTS

When the web application is started the first page that is seen is campaign page where existing campaigns are displayed and a new campaign can also be created as shown in Fig. 5.1. All these operations can be performed provided that Metamask is installed in the browser.



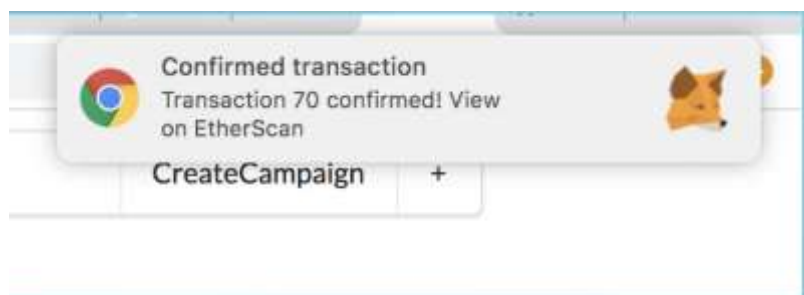
**Fig. 5.1.** view of campaigns web page

When create campaign is pressed it will redirect to the page where a transaction needs to be performed in order to create a new campaign as shown in Fig. 5.2.



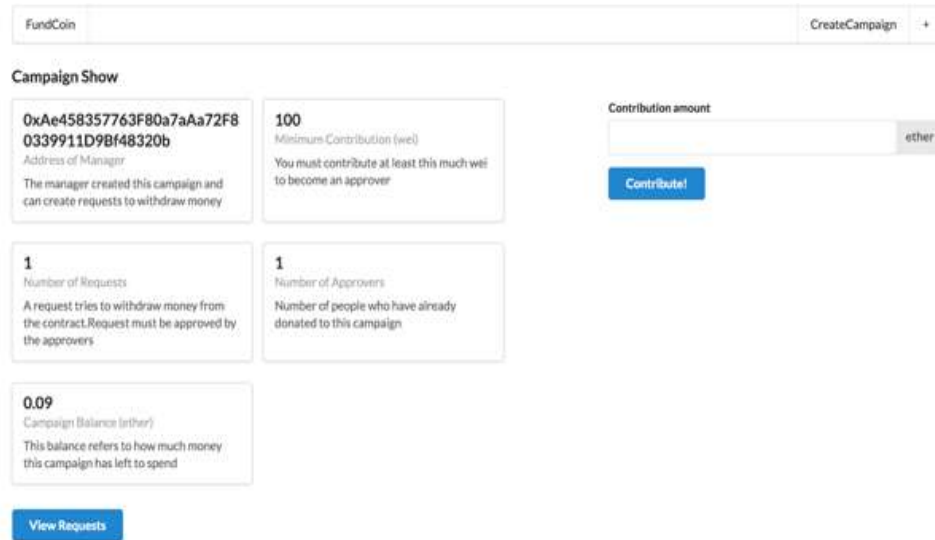
**Fig. 5.2.** Campaigns creation web page

Whatever changes/operations are to be made a transaction has to be performed and that change will reflect only after transaction is confirmed. Fig. 5.3 represents how a confirmed transaction notification looks.



**Fig. 5.3.** Metamask wallet confirmation

When view campaigns button is pressed which is present in first web page it will be redirected to this page which consists of campaigns details like address of person who created the campaign(Manager), minimum contribution, number of requests created by manager, number of approvers and campaign balance as. Investors can contribute to the campaign and become an approver. Fig. 5.4 below represents the campaigns show webpage.



FundCoin CreateCampaign +

### Campaign Show

**0xAe458357763F80a7aAa72F80339911D9Bf48320b**  
Address of Manager  
The manager created this campaign and can create requests to withdraw money

**100**  
Minimum Contribution (wei)  
You must contribute at least this much wei to become an approver

**1**  
Number of Requests  
A request tries to withdraw money from the contract. Request must be approved by the approvers

**1**  
Number of Approvers  
Number of people who have already donated to this campaign

**0.09**  
Campaign Balance (ether)  
This balance refers to how much money this campaign has left to spend

Contribution amount ether Contribute!

View Requests

**Fig. 5.4.** Campaigns show webpage

When view requests button is pressed it will be redirected to the view requests page as shown in Fig. 5.5. It consists of all the requests created by the manager. It also consists of approve and finalize button. Approve button is used by approvers to vote and Finalize button is used by manager to finalize the payment once the request gets enough number of votes.



FundCoin CreateCampaign +

### Requests

Add request

ID	Description	Amount	Recipient	Approval count	Approve	Finalize
0	Buy batteries	0.01	0xE791607697504E3EC362f312BAA57acb0A70AAf4	0/1	<span>Approve</span>	<span>Finalize</span>

Found 1 requests.

**Fig. 5.5.** View request webpage

When add request button is pressed it will be redirected to create request web page as shown in Fig. 5.6 where a request can be created by specifying description of spending request, value in ether that you want to spend and address of recipient to whom manager wants to send money (vendor address).

**Fig. 5.6.** Create a request web page

After approving the request if the request gets enough approvals/ majority votes then the request turns green indicating that it can be finalized by manager as shown in Fig. 5.7. It also consists of add request button which is used to create a spending request and this can be only done by the manager (person who created campaign).

Requests

ID	Description	Amount	Recipient	Approval count	Approve	Finalize
0	Buy batteries	0.01	0xE791607697506E3EC362F312BAA57acb0A70AA14	1/1	Approve	Finalize

Found 1 requests.

**Fig. 5.7.** After approving request

After finalizing the request the amount goes to the vendor address that was specified and the whole request turns grey as shown in Fig. 5.8 indicating that the request has been closed. The money/currency in the wallet is checked before and after finalizing the request in order to ensure that the transaction executed successfully.



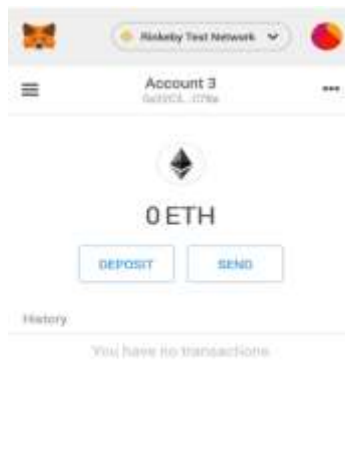
#### Requests

ID	Description	Amount	Recipient	Approval count	Approve	Finalize
0	Buy batteries	0.01	0xE7916076975063EC362D312BAAS7x3QA7DAW4	1/1		

Found 1 requests.

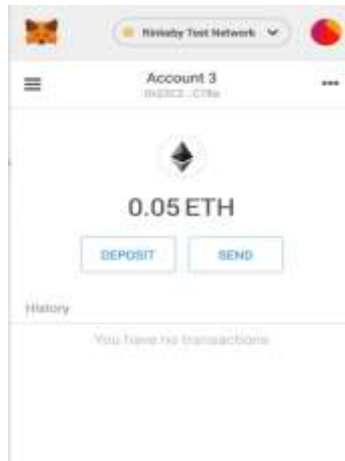
**Fig. 5.8.** After finalizing request

Before finalizing the request the vendor's wallet is empty as shown in Fig. 5.9.



**Fig. 5.9.** Wallet balance before finalizing request

After finalizing the request the vendor receives the amount and wallet balance gets updated as shown in Fig. 5.10.



**Fig. 5.10.** Wallet balance after finalizing the request

## 5.1 Observation

In Crowd Funding using blockchain to raise money for a startup first a campaign has to be created and a unique 64 character public key is assigned to that particular campaign to which investors can use as an address to donate. Investors can contribute the money in the form of ether. The contributed money will remain in the smart contract itself and will not be directly given to the campaign creator. If the campaign creator wants to spend the money he/she has to make a spending request which will consist of the amount of the ether needed and also the address or the public key of vendor to whom the ether should be sent so that campaign creator can get the required material which ensures that the contributed ether is always securely kept inside the smart contract and contributors have to vote to spending request created. If the percentage of the votes is at least 51 % then the amount will automatically be transferred to the vendor.

This method of Crowd Funding is lot more faster and secure as compared to the existing system. In current system the contributed amount is in the direct hands of the campaign creator and he/she can run away with that money, the time taken to transfer of the money raised to the campaign creator or the startup is slow and also there is fee that will be cut from the contributed amount as a third party like Kickstarter is involved or used in raising the required capital. Hence, the Crowd Funding using blockchain is more revolutionary and efficient method for raising capital for the startups .

## 6 CONCLUSION

The paper proposes a smart contract based solution to enable secure way of crowd funding by ensuring that the money donated by the investors is safe and also each and every step taken in the startup with help of donated money involves investor's opinion i.e. whenever the campaign creator wants to spend the money he/she has to make a spending request where the purpose of using the money, to whom the money is being sent(vendor) and the amount needed should be mention. The main advantage of using the smart contract which is a concept of blockchain is that it is resilient against many threats. Also it provides many features like improved reliability, faster and efficient operation. Web Application has been computed successfully and was also tested by taking "test cases". It is user friendly, and has required options, which can be utilized by user to perform the desired operation.

The goals achieved by the software are:

- Creating a campaign
- Contributing to campaign
- Creating a spending request
- Approving the spending request
- Finalising payment

Limitations regarding the project are that the vendor address is not verified by which campaign creator and the vendor can get together and scam the contributors. Also one person can contribute only one time for campaign/startup from a account as of now. The Ethereum accounts from which the contributors are investing also not verified.

Future works will aim to find a way to verify the Ethereum accounts whether it may be Metamask or MyEtherWallet(Ethereum wallet similar to metamask) accounts with help of a government id so that one person cannot have more than one account and also make sure that a person is able to contribute as many times he/she wants, so that the Crowdfunding using blockchain becomes more secure and reliable.

## REFERENCES

- [1] E. Mollick, The dynamics of crowdfunding: An exploratory study, Journal of business venturing, vol. 29, no. 1, pp. 1-16, Jan, 2014.
- [2] M. E. Peck, "Blockchains: How they work and why they'll change the world," in IEEE Spectrum, vol. 54, no. 10, pp. 26-35, 2017
- [3] VitalikButerin, A next-generation smart contract and decentralized application platform. white paper ,2014

- [4] PéterHegedus, Towards Analysing the Complexity Landscape of Solidity Based Ethereum Smart Contracts, Proc. 1<sup>st</sup> IEEE International Workshop on Emerging Trends in Software Engineering for Blockchain, Gothenburg, Sweden, 2018.
- [5] Kevin Delmolino, Mitchell Arnett, Ahmed E Kosba, Andrew Miller, and Elaine Shi, Step by Step Towards Creating a Safe Smart Contract: Lessons and Insights from a Cryptocurrency Lab, International Conference on Financial Cryptography and Data Security, Springer, pp. 79-94, 2015.
- [6] MetaMask Brings Ethereum to your browser, Accessed on September 21, 2019.
- [7] Etherscan, The Ethereum Block Explorer, <https://etherscan.io>, Accessed on September 18, 2019.
- [8] Infura, <https://infura.io>, Accessed on March 18, 2019, 35-39.
- [9] Ethereum: Comparison of the different test-nets [online] Available:  
<https://ethereum.stackexchange.com/questions/27048/comparison-of-the-different-testnets>.
- [10] Rinkeby authenticated faucet [online] Available: <https://faucet.rinkeby.io>
- [11] web3.js - Ethereum JavaScript API [Internet]. GitHub. [cited 2018 Jan 25].  
Available from : <https://github.com/etherum/web3.js>