



iJRASET

International Journal For Research in
Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 12 **Issue:** V **Month of publication:** May 2024

DOI: <https://doi.org/10.22214/ijraset.2024.60981>

www.ijraset.com

Call: ☎ 08813907089

E-mail ID: ijraset@gmail.com

The Crowdfunding Application by Blockchain and ThirdWeb Hybrid Model

Prof. Meera Sawalkar¹, Viraj Shewale², Prasad Sutar³, Vinit Kawachale⁴

JSPM Narhe Technical Campus, Narhe - Pune.

Abstract: *The advancement of Web3 and blockchain is happening rapidly in various fields, including healthcare, social services, and electronic voting. Blockchain technology is being used by crowdfunding platforms to combine its benefits of speed and low fees with traditional finance, creating a new way of raising money. By integrating Reacts user-friendly interface, Solidity smart contracts, Meta-Mask wallet integration and Hardhat development and testing capabilities, it forms a versatile and secure platform. This project aims to find out how much money is missing in the current crowdfunding market and offer them smart contracts and tools that use Ethereum to create their own application businesses. Hybrid Model Platform is a ThirdWeb tool that allows for easy and flexible scaling and visibility. this new crowdfunding app is a major leap in fundraising, fulfilling numerous customer requirements and poised to revolutionize crowdfunding with its fresh and inventive strategy.*

Keywords: *Blockchain, Web3, Smart Contract, Meta Mask, Third-Web, Hybrid Models, Crowdfunding.*

I. INTRODUCTION

Funding a project or some type of business with the aid of gathering cash from many people, specially using the internet, is referred to as crowdfunding. Crowdfunding has come to be a effective device to elevate price range and aid new initiatives, agencies and ventures. the integration of blockchain technology into public carrier applications has created a brand new technology of transparency, safety and performance. one of the primary players in this space is the “Binance Blockchain Crowdfunding App”, a platform that uses the ability of Binance smart Chain (BSC) to provide a unique and convenient crowdfunding enjoy.

Crowdfunding apps in general have confined security, so traders danger their cash to guide startups. Our effort to create this project is to create communicate among investors and startups to save you such frauds. surely put, our aim is to create a cozy platform in which cash can be raised with out worry of fraud or corruption. this is wherein blockchain is available in reachable. Blockchain is a dispensed ledger used to document all transactions throughout multiple structures in order that information can not be changed later. Hybrid models constitute a revolution in the discipline of crowdsourcing as they connect conventional domains and blockchain domains. This version combines the transparency, security and private networks, enhancing scalability and protection of blockchain. as a result we've bear in mind thirdweb platform. Binance Smart Chain is a blockchain network created by Binance that has many advantages such as high scalability, low transaction fees and fast confirmation times. These features make it an attractive option for public service projects trying to overcome some of the challenges associated with traditional blockchain networks. Additionally, the large user base and strong infrastructure of the Binance ecosystem provide a good foundation for the crowdfunding scheme to be successful. Combining the benefits of blockchain technology with traditional financial systems, the platform solves many problems while providing many benefits to creators and supporters. From start to finish, we'll cover the key features and benefits of the Binance Blockchain crowdfunding app, define its dedication to compliance and investor protection, and examine how a hybrid model can bridge the gap between blockchain and traditional finance. Whether you are an investor looking to raise capital for a project or an investor looking for an exciting opportunity, reviewing the Binance Crowdfunding app will shed light on the platform that is changing against the crowd.

II. LITERATURE SURVEY

A. Crowdfunding in Blockchain

In this project, we are investigating the integration of blockchain into crowdfunding. Researchers have examined how blockchain can increase the transparency, security and efficiency of the fundraising process.

B. Binance Smart Chain(BSC)

BSC process is important to understand, such as it provides effectiveness and low costs. Research on BSC architecture, consensus mechanisms, and network dynamics provides insight.

C. Hybrid Models

We are investigating the idea of hybrid models and how they might integrate traditional banking with blockchain technology to offer a wider range of payment choices and improve accessibility.

The blockchain platform Thirweb has a strong emphasis on security, scalability, and anonymity. It uses a special consensus method known as Proof-of-Merit to verify transactions. Thirweb seeks to protect user privacy and data integrity while facilitating decentralized applications (dApps) and smart contracts. Efficiency and inclusivity are given top priority in its architecture for blockchain operations.

D. Crowdfunding Smart Contract Security and Challenges

This paper will discuss the security variations and challenges that you may face both during and after using a blockchain-based crowdfunding platform. Smart contracts are starting to play a bigger role in the crowdfunding ecosystem.

E. Blockchain is Revolution Crowdfunding

In this paper, we will explain the limitations of crowdfunding platforms and the benefits of blockchain technology and how it is the future of crowdfunding owing to the ease and transparency of this model.

F. Cross-Platform Compatibility

Explore cross-platform compatibility and the impact of expanding cross-platform collaboration across multiple blockchain networks.

G. Blockchain and its Needs

A distributed, decentralized, and unchangeable database and ledger shared by all nodes in a computer network is called a blockchain. Blockchain functions as an electronic database that records and manages transactions in a digital manner. Blockchain keeps a decentralized, safe record of all transactions. Blockchain's unique feature is its fidelity and security throughout the record.

Data in the blockchain is stored in groups called "blocks". A block is a structure that stores a set of data. Blocks have a storage function that allows them to be linked to previous blocks, thus forming a blockchain, hence the name blockchain.

A decentralized, transparent, and safe method of recording and verifying transactions is offered by blockchain technology. Its capacity to foster confidence and do away with the necessity of middlemen in all aspects of business accounts for its allure. Blockchain guarantees data integrity, lowers fraud, and guards against manipulation. Its decentralized structure boosts efficiency and lowers the possibility of a single failure. Furthermore, digital currencies and blockchain-based smart contracts can streamline procedures, cut expenses, and boost productivity.

Overall, blockchain solves trust, security, and efficiency issues, making it useful in finance, supply chain, healthcare, and many other industries looking to update and improve performance.

H. Hashing in Blockchain

Binance Blockchain and Hybrid Model Hashing of large amounts of money is a simple security measure. It involves converting sensitive data or transaction details into a long alphanumeric string called a hash. Hashes serve several purposes: First, they protect user privacy and increase data security by hiding personal information. Second, they ensure the integrity of financial transactions and smart contracts by creating unique identifiers for each document. This will help prevent fraud and tampering. Finally, hashes facilitate the consensus in blockchain collaboration necessary to authenticate and verify transactions. In general, hash is an important factor in ensuring the stability and trust of crowd and hybrid models in the Binance blockchain ecosystem.

I. Smart Contracts

Because smart contracts write everything straight into the line of code, they automate the execution of contracts between buyers and sellers. As a result, it is beneficial to show the outcomes to each and every participant without the assistance of outside parties.

"If/when...then..." conditions that are encoded into code on a blockchain are how smart contracts function. Once these preset criteria have been confirmed and fulfilled, a network of computer systems executes the commands. The majority of these actions consist of issuing tickets, registering a car, notifying recipients, and releasing monies to the rightful parties. After the transaction is finished, the Blockchain is updated. This implies that only those parties with permission can view the outcomes, and that the transaction itself cannot be altered.

III. METHODOLOGY

A. Architecture

Fig. 1. represents the architecture of our Crowdfunding application. This shows how our web application with Solidity as our backend works. All the smart contracts that interact with the Ethereum blockchain are written in Solidity language . We have also used Hardhat ,which is an Ethereum development environment, along with the Chai assertion library to perform various tests on our smart contracts. And then later ThirdWeb is used to deploy the smart contract which the hybrid model for Blockchain.

We use React.js and TailwindCss as a frontend to serve JavaScript content to the browser and to provide responsiveness. When a user performs an action, it does not reach the server. Instead, the app runs in a web browser using web3 and ThirdWeb and communicates with the Binance Smart Chain (BSC). enter the key and send the transaction to the EVM network. These changes can be tracked in BSC to ensure transparency throughout the entire process. These public and private keys are never sent to the server because you cannot ask the user for their private number. So here the customer has more power and privacy.

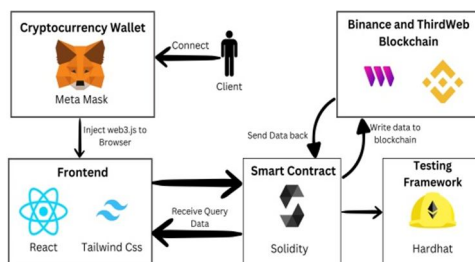


Fig -1: System Architecture of Application

B. Proposed System:

A custom script is used to construct smart contracts on Binance Smart Chain. A smart contract is an encrypted box that records outputs, processes inputs, stores information, and can only be read from the outside if specific requirements are met. We refer to it as "robustness". Actually, Binance can execute smart contracts with ease, and BSC offers online scripts for dependable codes to developers. All of the money that the provider receives is saved in the smart contract, where it cannot be altered or stolen, thanks to the way it is written. This money is stored in the smart contract itself rather than being delivered to the event creator directly. The following are the 6 modules that our application takes care of:

- 1) **Wallet Connect:** Users must have a cryptocurrency wallet (such as Metamask) to interact with our application. Initially, when a user enters our decentralized crowdfunding app, the cryptocurrency wallet is connected and then the user can make various transactions.
- 2) **Campaign Creation:** Users can create a campaign by providing relevant details such as campaign name, campaign description, donation amount, minimum donation amount, application deadline. A small gas fee is charged for each transaction. So for every transaction that needs to happen on the blockchain, we need to provide some amount of money for that transaction to be valid. When the transaction is completed after a few seconds, a new block containing the address of the contract will be added to the Ethereum blockchain. The Home Page then also displays this newly created campaign, with which the user as well as the contributor can interact.
- 3) **Fund the Campaigns:** Donors can search our app to look for events that might interest them. When donors find a campaign they like, they can support the campaign by donating Ether. After that, a Metamask pop-up will appear to confirm the change. The donor now becomes a supporter of the campaign and plays a role in deciding whether sellers will receive the revenue generated to date.
- 4) **Hybrid Model:** Thirweb is a blockchain platform emphasizing privacy, scalability, and security. It employs a unique consensus mechanism called Proof-of-Merit to validate transactions. Thirweb aims to enable decentralized applications (dApps) and smart contracts while ensuring user anonymity and data integrity. Its architecture prioritizes efficiency and inclusivity in blockchain operations.
- 5) **Withdrawal Request:** When a developer wants to withdraw some Ether from the funds he has earned so far for his campaign, he must first create a withdrawal request. The proposal must be approved by at least half of the campaign sponsors. If 50% is not voted, the seller will not be able to withdraw the money and will have to wait for people to vote again.

- 6) *Approval*: When a seller wants to spend money, the cancellation request is notified to all sponsors. Therefore, the buyer must approve the request if he wishes. A participant can vote on a proposal for a sponsor, for example. Supporters can show their approval by clicking the "Vote" button next to the seller's removal request. This will then process transactions, pay a small fuel fee and add a block to the blockchain.
- 7) *End of Campaign*: Campaign that has been created has the end date after that you cannot fund the campaign. Whatever the fund is raised is stored on your blockchain and after that the Campaigns will be terminated. All transactions taking place will be stored on the blockchain to ensure transparency in the entire process.

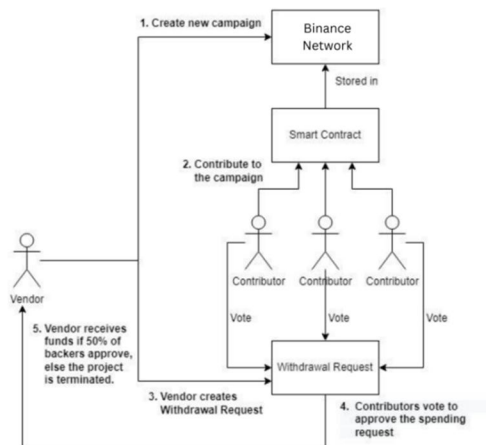


Fig-2: Flow of Crowdfunding the Application.

C. Technology Used

There are various technology and tools used while building Crowdfunding application in which some of them are :

- 1) *React and Tailwind CSS*: React is used for building the user interface. It enables the creation of dynamic, responsive, and user-friendly web pages, making it an essential technology for user interactions. And for responsive and interactiveness is handle by Tailwind Css.
- 2) *Solidity*: Solidity is a programming language specifically designed for creating smart contracts on blockchain networks. It is used to enforce the rules of the fund committee, including planning, financial management and payment rules.
- 3) *Hardhat*: Hardhat work as a development environment for Ethereum compatible blockchains such as Binance Smart Chain. It allows developers to write and test smart contracts, run tests, and upload them to the blockchain.
- 4) *Email js*: Email js is a platform that allows developers to send emails directly from client-side JavaScript. It simplifies the process of integrating email functionality into web applications by providing an API for sending emails without the need for server-side code. EmailJS supports various email services and templates, offering flexibility and ease of use.
- 5) *ThirdWeb*: Thirweb is a blockchain platform emphasizing privacy, scalability, and security. It employs a unique consensus mechanism called Proof-of-Merit to validate transactions. Thirweb aims to enable decentralized applications (dApps) and smart contracts while ensuring user anonymity and data integrity. Its architecture prioritizes efficiency and inclusivity in blockchain operations.
- 6) *Binance Smart Chain*: Based on blockchain technology, BSC provides an honest ledger for the handling of large amounts of money. It has a fast market and low cost, making it ideal for crowdfunding.
- 7) *Web3.js*: Web3.js is used to interact with the blockchain from the frontend. It enables communication with smart contracts, retrieval of blockchain data, and transaction handling.
- 8) *GitHub*: GitHub or similar version control platforms are used to manage the source code, collaborate among development teams, and track changes.

D. Deployment

First, we need to create account on thirdWeb and we load the local blockchain created by Hardhat to test and deploy our smart contract written in solidity. You can enter `npx hardhat node` to know addresses of various users.

As BSC is EVM-compatible, which means you can use the same Solidity language and many of the Ethereum development tools. However, you need to connect your development environment to the Meta Mask network. Ensure that the development environment is setup and the contract written is completely fine.

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\acer\OneDrive\Desktop\Profund\Web3> npx hardhat node
Started HTTP and WebSocket JSON-RPC server at http://127.0.0.1:8545/

Accounts
=====

WARNING: These accounts, and their private keys, are publicly known.
Any funds sent to them on Mainnet or any other live network WILL BE LOST.

Account #0: 0xf39fd651aa888f646c6a8882729c9ff1b92266 (10000 ETH)
Private Key: 0xac0974bec39a17e36ba4a6b4d238ff944baca78cb5d5e5cae784d7bf4f2ff80

Account #1: 0x7995797c31812dc3a01bc7d01b58e0d17dc79c8 (10000 ETH)
Private Key: 0x59c695e98f97a5a0044966f0945399dc5e8daec8c7a8412f46036b78690d

Account #2: 0xc344cdd86a900fa2b585dd299e03d12fA42938C (10000 ETH)
Private Key: 0x5de4111fa1ad9998f83103eb1f1766367c2e6cac87f3fb9a804cdab365a

Account #3: 0x90f79b66f6b2c4f870365e785982e1f01e93b906 (10000 ETH)
Private Key: 0x7c852118294e51e653712a81e05800f419141751be58f605c371e15141b007a6

Account #4: 0x15d34Aaf542670B7D7C367839Aaf71A00a2C6A65 (10000 ETH)
Private Key: 0x47e179ec197488593b187f80a00eb0da91f1b9d0b13f8733639f19c30a34926a

Account #5: 0x9965507D1a55bc2695C58ba16F837d81900A4dc (10000 ETH)
Private Key: 0x8b3a350cf5c34c9194ca85829a2df0ec3153be03185e2d3348e872092dfbba

Account #6: 0x976EA74026E726554dB657FA54763abd0C3a0aa9 (10000 ETH)
Private Key: 0x92db14e483b83df3d233f83dfa3a0d7096f213ca9bedd068b882b4ec1564e

```

Fig-3: Hardhat Node Command

The Crowdfunding instance was then deployed to the Hardhat test network and the address of the deployed Crowdfunding contract was console logged in the terminal using the command `npx hardhat run scripts/deploy.js --network localhost`.

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
at processTicksAndRejections (node:internal/process/task_queues:95:5)
PS C:\Users\acer\OneDrive\Desktop\Profund\Web3> npx hardhat run scripts/deploy.js --network localhost
Compiled 1 Solidity file successfully

```

Fig-4: Deployed Smart Contract

The Crowdfunding application was then started by navigating into the client directory (which contains our frontend code) and running the command `npm run dev`.

After this you require Thirdweb login you should connect your wallet to thirdweb and sign-in. After that you have open cmd and type `npm thirdweb install`. After thirdweb is setup then you need to deploy your smart contract to thirdweb sever by command `npx thirdweb deploy`. After deploying contract to thirdweb then you need to authorize and you need to have secret and API key.

Once the contract is deployed successfully on thirdweb then you just copy the contract address and can use in your application.

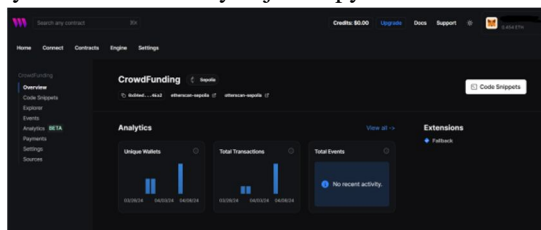


Fig-5: ThirdWeb Contract deployment

IV. RESULTS

We are able create Dapp application for Crowdfunding with help of React, Solidity, ThirdWeb and responsive UI due to Tailwind Css. As because of ThirdWeb, Meta Mask and Binance it provides more transparency, scalability, and flexibility. The smart contract used in our application is shown below:

```

1 // SPDX-License-Identifier: UNLICENSED
2 pragma solidity ^0.8.0;
3
4 contract Crowdfunding {
5     struct Campaign {
6         string title;
7         string description;
8         uint256 target;
9         uint256 deadline;
10        uint256 amountCollected;
11        string image;
12        address[] donors;
13        uint256[] donations;
14    }
15
16    mapping(uint256 => Campaign) public campaigns;
17
18    uint256 public numberOfCampaigns = 0;
19
20    function createCampaign(
21        address _owner,
22        string memory _title,
23        string memory _description,
24        uint256 _target,
25        uint256 _deadline,
26        string memory _image
27    ) public {
28        require(_deadline > block.timestamp, "The deadline should be a date in the future.");
29
30        Campaign memory campaign = Campaign({
31            title: _title,
32            description: _description,
33            target: _target,
34            deadline: _deadline,
35            amountCollected: 0,
36            image: _image,
37            donors: new address[](0),
38            donations: new uint256[](0)
39        });
40        campaigns[numberOfCampaigns] = campaign;
41        numberOfCampaigns++;
42    }
43
44    function donate(
45        uint256 campaignId,
46        uint256 amount
47    ) public {
48        Campaign memory campaign = campaigns[campaignId];
49        require(campaign.deadline > block.timestamp, "The deadline should be a date in the future.");
50        require(campaign.donors.length == 0, "Donors array is not empty");
51        require(campaign.donations.length == 0, "Donations array is not empty");
52        campaign.donors.push(msg.sender);
53        campaign.donations.push(amount);
54        campaign.amountCollected += amount;
55    }
56
57    function getDonors(
58        uint256 campaignId
59    ) public view returns (address[]) {
60        return campaigns[campaignId].donors;
61    }
62
63    function getDonations(
64        uint256 campaignId
65    ) public view returns (uint256[]) {
66        return campaigns[campaignId].donations;
67    }
68
69    function getAmountCollected(
70        uint256 campaignId
71    ) public view returns (uint256) {
72        return campaigns[campaignId].amountCollected;
73    }
74
75    function getTarget(
76        uint256 campaignId
77    ) public view returns (uint256) {
78        return campaigns[campaignId].target;
79    }
80
81    function getDeadline(
82        uint256 campaignId
83    ) public view returns (uint256) {
84        return campaigns[campaignId].deadline;
85    }
86
87    function getImage(
88        uint256 campaignId
89    ) public view returns (string) {
90        return campaigns[campaignId].image;
91    }
92
93    function getNumberOfCampaigns() public view returns (uint256) {
94        return numberOfCampaigns;
95    }
96
97    function getOwner(
98        uint256 campaignId
99    ) public view returns (address) {
100        return campaigns[campaignId].owner;
101    }
102
103    function getTitle(
104        uint256 campaignId
105    ) public view returns (string) {
106        return campaigns[campaignId].title;
107    }
108
109    function getDescription(
110        uint256 campaignId
111    ) public view returns (string) {
112        return campaigns[campaignId].description;
113    }
114
115    function getTargetValue(
116        uint256 campaignId
117    ) public view returns (uint256) {
118        return campaigns[campaignId].target;
119    }
120
121    function getDeadlineValue(
122        uint256 campaignId
123    ) public view returns (uint256) {
124        return campaigns[campaignId].deadline;
125    }
126
127    function getImageValue(
128        uint256 campaignId
129    ) public view returns (string) {
130        return campaigns[campaignId].image;
131    }
132
133    function getAmountCollectedValue(
134        uint256 campaignId
135    ) public view returns (uint256) {
136        return campaigns[campaignId].amountCollected;
137    }
138
139    function getDonorsValue(
140        uint256 campaignId
141    ) public view returns (address[]) {
142        return campaigns[campaignId].donors;
143    }
144
145    function getDonationsValue(
146        uint256 campaignId
147    ) public view returns (uint256[]) {
148        return campaigns[campaignId].donations;
149    }
150
151    function getNumberOfCampaignsValue() public view returns (uint256) {
152        return numberOfCampaigns;
153    }
154
155    function getOwnerValue(
156        uint256 campaignId
157    ) public view returns (address) {
158        return campaigns[campaignId].owner;
159    }
160
161    function getTitleValue(
162        uint256 campaignId
163    ) public view returns (string) {
164        return campaigns[campaignId].title;
165    }
166
167    function getDescriptionValue(
168        uint256 campaignId
169    ) public view returns (string) {
170        return campaigns[campaignId].description;
171    }
172
173    function getTargetValue2(
174        uint256 campaignId
175    ) public view returns (uint256) {
176        return campaigns[campaignId].target;
177    }
178
179    function getDeadlineValue2(
180        uint256 campaignId
181    ) public view returns (uint256) {
182        return campaigns[campaignId].deadline;
183    }
184
185    function getImageValue2(
186        uint256 campaignId
187    ) public view returns (string) {
188        return campaigns[campaignId].image;
189    }
190
191    function getAmountCollectedValue2(
192        uint256 campaignId
193    ) public view returns (uint256) {
194        return campaigns[campaignId].amountCollected;
195    }
196
197    function getDonorsValue2(
198        uint256 campaignId
199    ) public view returns (address[]) {
200        return campaigns[campaignId].donors;
201    }
202
203    function getDonationsValue2(
204        uint256 campaignId
205    ) public view returns (uint256[]) {
206        return campaigns[campaignId].donations;
207    }
208
209    function getNumberOfCampaignsValue2() public view returns (uint256) {
210        return numberOfCampaigns;
211    }
212
213    function getOwnerValue2(
214        uint256 campaignId
215    ) public view returns (address) {
216        return campaigns[campaignId].owner;
217    }
218
219    function getTitleValue2(
220        uint256 campaignId
221    ) public view returns (string) {
222        return campaigns[campaignId].title;
223    }
224
225    function getDescriptionValue2(
226        uint256 campaignId
227    ) public view returns (string) {
228        return campaigns[campaignId].description;
229    }
230
231    function getTargetValue3(
232        uint256 campaignId
233    ) public view returns (uint256) {
234        return campaigns[campaignId].target;
235    }
236
237    function getDeadlineValue3(
238        uint256 campaignId
239    ) public view returns (uint256) {
240        return campaigns[campaignId].deadline;
241    }
242
243    function getImageValue3(
244        uint256 campaignId
245    ) public view returns (string) {
246        return campaigns[campaignId].image;
247    }
248
249    function getAmountCollectedValue3(
250        uint256 campaignId
251    ) public view returns (uint256) {
252        return campaigns[campaignId].amountCollected;
253    }
254
255    function getDonorsValue3(
256        uint256 campaignId
257    ) public view returns (address[]) {
258        return campaigns[campaignId].donors;
259    }
260
261    function getDonationsValue3(
262        uint256 campaignId
263    ) public view returns (uint256[]) {
264        return campaigns[campaignId].donations;
265    }
266
267    function getNumberOfCampaignsValue3() public view returns (uint256) {
268        return numberOfCampaigns;
269    }
270
271    function getOwnerValue3(
272        uint256 campaignId
273    ) public view returns (address) {
274        return campaigns[campaignId].owner;
275    }
276
277    function getTitleValue3(
278        uint256 campaignId
279    ) public view returns (string) {
280        return campaigns[campaignId].title;
281    }
282
283    function getDescriptionValue3(
284        uint256 campaignId
285    ) public view returns (string) {
286        return campaigns[campaignId].description;
287    }
288
289    function getTargetValue4(
290        uint256 campaignId
291    ) public view returns (uint256) {
292        return campaigns[campaignId].target;
293    }
294
295    function getDeadlineValue4(
296        uint256 campaignId
297    ) public view returns (uint256) {
298        return campaigns[campaignId].deadline;
299    }
300
301    function getImageValue4(
302        uint256 campaignId
303    ) public view returns (string) {
304        return campaigns[campaignId].image;
305    }
306
307    function getAmountCollectedValue4(
308        uint256 campaignId
309    ) public view returns (uint256) {
310        return campaigns[campaignId].amountCollected;
311    }
312
313    function getDonorsValue4(
314        uint256 campaignId
315    ) public view returns (address[]) {
316        return campaigns[campaignId].donors;
317    }
318
319    function getDonationsValue4(
320        uint256 campaignId
321    ) public view returns (uint256[]) {
322        return campaigns[campaignId].donations;
323    }
324
325    function getNumberOfCampaignsValue4() public view returns (uint256) {
326        return numberOfCampaigns;
327    }
328
329    function getOwnerValue4(
330        uint256 campaignId
331    ) public view returns (address) {
332        return campaigns[campaignId].owner;
333    }
334
335    function getTitleValue4(
336        uint256 campaignId
337    ) public view returns (string) {
338        return campaigns[campaignId].title;
339    }
340
341    function getDescriptionValue4(
342        uint256 campaignId
343    ) public view returns (string) {
344        return campaigns[campaignId].description;
345    }
346
347    function getTargetValue5(
348        uint256 campaignId
349    ) public view returns (uint256) {
350        return campaigns[campaignId].target;
351    }
352
353    function getDeadlineValue5(
354        uint256 campaignId
355    ) public view returns (uint256) {
356        return campaigns[campaignId].deadline;
357    }
358
359    function getImageValue5(
360        uint256 campaignId
361    ) public view returns (string) {
362        return campaigns[campaignId].image;
363    }
364
365    function getAmountCollectedValue5(
366        uint256 campaignId
367    ) public view returns (uint256) {
368        return campaigns[campaignId].amountCollected;
369    }
370
371    function getDonorsValue5(
372        uint256 campaignId
373    ) public view returns (address[]) {
374        return campaigns[campaignId].donors;
375    }
376
377    function getDonationsValue5(
378        uint256 campaignId
379    ) public view returns (uint256[]) {
380        return campaigns[campaignId].donations;
381    }
382
383    function getNumberOfCampaignsValue5() public view returns (uint256) {
384        return numberOfCampaigns;
385    }
386
387    function getOwnerValue5(
388        uint256 campaignId
389    ) public view returns (address) {
390        return campaigns[campaignId].owner;
391    }
392
393    function getTitleValue5(
394        uint256 campaignId
395    ) public view returns (string) {
396        return campaigns[campaignId].title;
397    }
398
399    function getDescriptionValue5(
400        uint256 campaignId
401    ) public view returns (string) {
402        return campaigns[campaignId].description;
403    }
404
405    function getTargetValue6(
406        uint256 campaignId
407    ) public view returns (uint256) {
408        return campaigns[campaignId].target;
409    }
410
411    function getDeadlineValue6(
412        uint256 campaignId
413    ) public view returns (uint256) {
414        return campaigns[campaignId].deadline;
415    }
416
417    function getImageValue6(
418        uint256 campaignId
419    ) public view returns (string) {
420        return campaigns[campaignId].image;
421    }
422
423    function getAmountCollectedValue6(
424        uint256 campaignId
425    ) public view returns (uint256) {
426        return campaigns[campaignId].amountCollected;
427    }
428
429    function getDonorsValue6(
430        uint256 campaignId
431    ) public view returns (address[]) {
432        return campaigns[campaignId].donors;
433    }
434
435    function getDonationsValue6(
436        uint256 campaignId
437    ) public view returns (uint256[]) {
438        return campaigns[campaignId].donations;
439    }
440
441    function getNumberOfCampaignsValue6() public view returns (uint256) {
442        return numberOfCampaigns;
443    }
444
445    function getOwnerValue6(
446        uint256 campaignId
447    ) public view returns (address) {
448        return campaigns[campaignId].owner;
449    }
450
451    function getTitleValue6(
452        uint256 campaignId
453    ) public view returns (string) {
454        return campaigns[campaignId].title;
455    }
456
457    function getDescriptionValue6(
458        uint256 campaignId
459    ) public view returns (string) {
460        return campaigns[campaignId].description;
461    }
462
463    function getTargetValue7(
464        uint256 campaignId
465    ) public view returns (uint256) {
466        return campaigns[campaignId].target;
467    }
468
469    function getDeadlineValue7(
470        uint256 campaignId
471    ) public view returns (uint256) {
472        return campaigns[campaignId].deadline;
473    }
474
475    function getImageValue7(
476        uint256 campaignId
477    ) public view returns (string) {
478        return campaigns[campaignId].image;
479    }
480
481    function getAmountCollectedValue7(
482        uint256 campaignId
483    ) public view returns (uint256) {
484        return campaigns[campaignId].amountCollected;
485    }
486
487    function getDonorsValue7(
488        uint256 campaignId
489    ) public view returns (address[]) {
490        return campaigns[campaignId].donors;
491    }
492
493    function getDonationsValue7(
494        uint256 campaignId
495    ) public view returns (uint256[]) {
496        return campaigns[campaignId].donations;
497    }
498
499    function getNumberOfCampaignsValue7() public view returns (uint256) {
500        return numberOfCampaigns;
501    }
502
503    function getOwnerValue7(
504        uint256 campaignId
505    ) public view returns (address) {
506        return campaigns[campaignId].owner;
507    }
508
509    function getTitleValue7(
510        uint256 campaignId
511    ) public view returns (string) {
512        return campaigns[campaignId].title;
513    }
514
515    function getDescriptionValue7(
516        uint256 campaignId
517    ) public view returns (string) {
518        return campaigns[campaignId].description;
519    }
520
521    function getTargetValue8(
522        uint256 campaignId
523    ) public view returns (uint256) {
524        return campaigns[campaignId].target;
525    }
526
527    function getDeadlineValue8(
528        uint256 campaignId
529    ) public view returns (uint256) {
530        return campaigns[campaignId].deadline;
531    }
532
533    function getImageValue8(
534        uint256 campaignId
535    ) public view returns (string) {
536        return campaigns[campaignId].image;
537    }
538
539    function getAmountCollectedValue8(
540        uint256 campaignId
541    ) public view returns (uint256) {
542        return campaigns[campaignId].amountCollected;
543    }
544
545    function getDonorsValue8(
546        uint256 campaignId
547    ) public view returns (address[]) {
548        return campaigns[campaignId].donors;
549    }
550
551    function getDonationsValue8(
552        uint256 campaignId
553    ) public view returns (uint256[]) {
554        return campaigns[campaignId].donations;
555    }
556
557    function getNumberOfCampaignsValue8() public view returns (uint256) {
558        return numberOfCampaigns;
559    }
560
561    function getOwnerValue8(
562        uint256 campaignId
563    ) public view returns (address) {
564        return campaigns[campaignId].owner;
565    }
566
567    function getTitleValue8(
568        uint256 campaignId
569    ) public view returns (string) {
570        return campaigns[campaignId].title;
571    }
572
573    function getDescriptionValue8(
574        uint256 campaignId
575    ) public view returns (string) {
576        return campaigns[campaignId].description;
577    }
578
579    function getTargetValue9(
580        uint256 campaignId
581    ) public view returns (uint256) {
582        return campaigns[campaignId].target;
583    }
584
585    function getDeadlineValue9(
586        uint256 campaignId
587    ) public view returns (uint256) {
588        return campaigns[campaignId].deadline;
589    }
590
591    function getImageValue9(
592        uint256 campaignId
593    ) public view returns (string) {
594        return campaigns[campaignId].image;
595    }
596
597    function getAmountCollectedValue9(
598        uint256 campaignId
599    ) public view returns (uint256) {
600        return campaigns[campaignId].amountCollected;
601    }
602
603    function getDonorsValue9(
604        uint256 campaignId
605    ) public view returns (address[]) {
606        return campaigns[campaignId].donors;
607    }
608
609    function getDonationsValue9(
610        uint256 campaignId
611    ) public view returns (uint256[]) {
612        return campaigns[campaignId].donations;
613    }
614
615    function getNumberOfCampaignsValue9() public view returns (uint256) {
616        return numberOfCampaigns;
617    }
618
619    function getOwnerValue9(
620        uint256 campaignId
621    ) public view returns (address) {
622        return campaigns[campaignId].owner;
623    }
624
625    function getTitleValue9(
626        uint256 campaignId
627    ) public view returns (string) {
628        return campaigns[campaignId].title;
629    }
630
631    function getDescriptionValue9(
632        uint256 campaignId
633    ) public view returns (string) {
634        return campaigns[campaignId].description;
635    }
636
637    function getTargetValue10(
638        uint256 campaignId
639    ) public view returns (uint256) {
640        return campaigns[campaignId].target;
641    }
642
643    function getDeadlineValue10(
644        uint256 campaignId
645    ) public view returns (uint256) {
646        return campaigns[campaignId].deadline;
647    }
648
649    function getImageValue10(
650        uint256 campaignId
651    ) public view returns (string) {
652        return campaigns[campaignId].image;
653    }
654
655    function getAmountCollectedValue10(
656        uint256 campaignId
657    ) public view returns (uint256) {
658        return campaigns[campaignId].amountCollected;
659    }
660
661    function getDonorsValue10(
662        uint256 campaignId
663    ) public view returns (address[]) {
664        return campaigns[campaignId].donors;
665    }
666
667    function getDonationsValue10(
668        uint256 campaignId
669    ) public view returns (uint256[]) {
670        return campaigns[campaignId].donations;
671    }
672
673    function getNumberOfCampaignsValue10() public view returns (uint256) {
674        return numberOfCampaigns;
675    }
676
677    function getOwnerValue10(
678        uint256 campaignId
679    ) public view returns (address) {
680        return campaigns[campaignId].owner;
681    }
682
683    function getTitleValue10(
684        uint256 campaignId
685    ) public view returns (string) {
686        return campaigns[campaignId].title;
687    }
688
689    function getDescriptionValue10(
690        uint256 campaignId
691    ) public view returns (string) {
692        return campaigns[campaignId].description;
693    }
694
695    function getTargetValue11(
696        uint256 campaignId
697    ) public view returns (uint256) {
698        return campaigns[campaignId].target;
699    }
700
701    function getDeadlineValue11(
702        uint256 campaignId
703    ) public view returns (uint256) {
704        return campaigns[campaignId].deadline;
705    }
706
707    function getImageValue11(
708        uint256 campaignId
709    ) public view returns (string) {
710        return campaigns[campaignId].image;
711    }
712
713    function getAmountCollectedValue11(
714        uint256 campaignId
715    ) public view returns (uint256) {
716        return campaigns[campaignId].amountCollected;
717    }
718
719    function getDonorsValue11(
720        uint256 campaignId
721    ) public view returns (address[]) {
722        return campaigns[campaignId].donors;
723    }
724
725    function getDonationsValue11(
726        uint256 campaignId
727    ) public view returns (uint256[]) {
728        return campaigns[campaignId].donations;
729    }
730
731    function getNumberOfCampaignsValue11() public view returns (uint256) {
732        return numberOfCampaigns;
733    }
734
735    function getOwnerValue11(
736        uint256 campaignId
737    ) public view returns (address) {
738        return campaigns[campaignId].owner;
739    }
740
741    function getTitleValue11(
742        uint256 campaignId
743    ) public view returns (string) {
744        return campaigns[campaignId].title;
745    }
746
747    function getDescriptionValue11(
748        uint256 campaignId
749    ) public view returns (string) {
750        return campaigns[campaignId].description;
751    }
752
753    function getTargetValue12(
754        uint256 campaignId
755    ) public view returns (uint256) {
756        return campaigns[campaignId].target;
757    }
758
759    function getDeadlineValue12(
760        uint256 campaignId
761    ) public view returns (uint256) {
762        return campaigns[campaignId].deadline;
763    }
764
765    function getImageValue12(
766        uint256 campaignId
767    ) public view returns (string) {
768        return campaigns[campaignId].image;
769    }
770
771    function getAmountCollectedValue12(
772        uint256 campaignId
773    ) public view returns (uint256) {
774        return campaigns[campaignId].amountCollected;
775    }
776
777    function getDonorsValue12(
778        uint256 campaignId
779    ) public view returns (address[]) {
780        return campaigns[campaignId].donors;
781    }
782
783    function getDonationsValue12(
784        uint256 campaignId
785    ) public view returns (uint256[]) {
786        return campaigns[campaignId].donations;
787    }
788
789    function getNumberOfCampaignsValue12() public view returns (uint256) {
790        return numberOfCampaigns;
791    }
792
793    function getOwnerValue12(
794        uint256 campaignId
795    ) public view returns (address) {
796        return campaigns[campaignId].owner;
797    }
798
799    function getTitleValue12(
800        uint256 campaignId
801    ) public view returns (string) {
802        return campaigns[campaignId].title;
803    }
804
805    function getDescriptionValue12(
806        uint256 campaignId
807    ) public view returns (string) {
808        return campaigns[campaignId].description;
809    }
810
811    function getTargetValue13(
812        uint256 campaignId
813    ) public view returns (uint256) {
814        return campaigns[campaignId].target;
815    }
816
817    function getDeadlineValue13(
818        uint256 campaignId
819    ) public view returns (uint256) {
820        return campaigns[campaignId].deadline;
821    }
822
823    function getImageValue13(
824        uint256 campaignId
825    ) public view returns (string) {
826        return campaigns[campaignId].image;
827    }
828
829    function getAmountCollectedValue13(
830        uint256 campaignId
831    ) public view returns (uint256) {
832        return campaigns[campaignId].amountCollected;
833    }
834
835    function getDonorsValue13(
836        uint256 campaignId
837    ) public view returns (address[]) {
838        return campaigns[campaignId].donors;
839    }
840
841    function getDonationsValue13(
842        uint256 campaignId
843    ) public view returns (uint256[]) {
844        return campaigns[campaignId].donations;
845    }
846
847    function getNumberOfCampaignsValue13() public view returns (uint256) {
848        return numberOfCampaigns;
849    }
850
851    function getOwnerValue13(
852        uint256 campaignId
853    ) public view returns (address) {
854        return campaigns[campaignId].owner;
855    }
856
857    function getTitleValue13(
858        uint256 campaignId
859    ) public view returns (string) {
860        return campaigns[campaignId].title;
861    }
862
863    function getDescriptionValue13(
864        uint256 campaignId
865    ) public view returns (string) {
866        return campaigns[campaignId].description;
867    }
868
869    function getTargetValue14(
870        uint256 campaignId
871    ) public view returns (uint256) {
872        return campaigns[campaignId].target;
873    }
874
875    function getDeadlineValue14(
876        uint256 campaignId
877    ) public view returns (uint256) {
878        return campaigns[campaignId].deadline;
879    }
880
881    function getImageValue14(
882        uint256 campaignId
883    ) public view returns (string) {
884        return campaigns[campaignId].image;
885    }
886
887    function getAmountCollectedValue14(
888        uint256 campaignId
889    ) public view returns (uint256) {
890        return campaigns[campaignId].amountCollected;
891    }
892
893    function getDonorsValue14(
894        uint256 campaignId
895    ) public view returns (address[]) {
896        return campaigns[campaignId].donors;
897    }
898
899    function getDonationsValue14(
900        uint256 campaignId
901    ) public view returns (uint256[]) {
902        return campaigns[campaignId].donations;
903    }
904
905    function getNumberOfCampaignsValue14() public view returns (uint256) {
906        return numberOfCampaigns;
907    }
908
909    function getOwnerValue14(
910        uint256 campaignId
911    ) public view returns (address) {
912        return campaigns[campaignId].owner;
913    }
914
915    function getTitleValue14(
916        uint256 campaignId
917    ) public view returns (string) {
918        return campaigns[campaignId].title;
919    }
920
921    function getDescriptionValue14(
922        uint256 campaignId
923    ) public view returns (string) {
924        return campaigns[campaignId].description;
925    }
926
927    function getTargetValue15(
928        uint256 campaignId
929    ) public view returns (uint256) {
930        return campaigns[campaignId].target;
931    }
932
933    function getDeadlineValue15(
934        uint256 campaignId
935    ) public view returns (uint256) {
936        return campaigns[campaignId].deadline;
937    }
938
939    function getImageValue15(
940        uint256 campaignId
941    ) public view returns (string) {
942        return campaigns[campaignId].image;
943    }
944
945    function getAmountCollectedValue15(
946        uint256 campaignId
947    ) public view returns (uint256) {
948        return campaigns[campaignId].amountCollected;
949    }
950
951    function getDonorsValue15(
952        uint256 campaignId
953    ) public view returns (address[]) {
954        return campaigns[campaignId].donors;
955    }
956
957    function getDonationsValue15(
958        uint256 campaignId
959    ) public view returns (uint256[]) {
960        return campaigns[campaignId].donations;
961    }
962
963    function getNumberOfCampaignsValue15() public view returns (uint256) {
964        return numberOfCampaigns;
965    }
966
967    function getOwnerValue15(
968        uint256 campaignId
969    ) public view returns (address) {
970        return campaigns[campaignId].owner;
971    }
972
973    function getTitleValue15(
974        uint256 campaignId
975    ) public view returns (string) {
976        return campaigns[campaignId].title;
977    }
978
979    function getDescriptionValue15(
980        uint256 campaignId
981    ) public view returns (string) {
982        return campaigns[campaignId].description;
983    }
984
985    function getTargetValue16(
986        uint256 campaignId
987    ) public view returns (uint256) {
988        return campaigns[campaignId].target;
989    }
990
991    function getDeadlineValue16(
992        uint256 campaignId
993    ) public view returns (uint256) {
994        return campaigns[campaignId].deadline;
995    }
996
997    function getImageValue16(
998        uint256 campaignId
999    ) public view returns (string) {
1000        return campaigns[campaignId].image;
1001    }
1002
1003    function getAmountCollectedValue16(
1004        uint256 campaignId
1005    ) public view returns (uint256) {
1006        return campaigns[campaignId].amountCollected;
1007    }
1008
1009    function getDonorsValue16(
1010        uint256 campaignId
1011    ) public view returns (address[]) {
1012        return campaigns[campaignId].donors;
1013    }
1014
1015    function getDonationsValue16(
1016        uint256 campaignId
1017    ) public view returns (uint256[]) {
1018        return campaigns[campaignId].donations;
1019    }
1020
1021    function getNumberOfCampaignsValue16() public view returns (uint
```

Moving on to the front-end part as it is been developed using react and tailwind css. It is simple page having Navbar with app name, About us, Contact us, Team member and Crowd funding button.

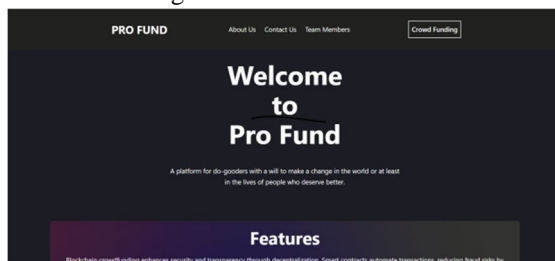


Fig-6: Front-end website page

In the Contact Us page comes with email js where you can send us the email and we will responded to your query.

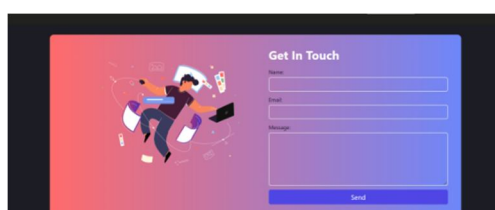


Fig-7: Contact Us page

Now when we click on Crowd Funding button then we are redirected to Crowd Funding website.

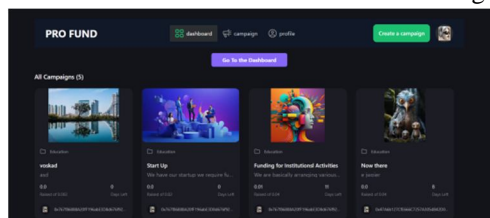


Fig-8: Crowd Funding Page

Here Firstly we have connect to the wallet and then you are further able create and fund the Camapigns. You have three sections:

- 1) Dashboard: Were you are able see all the Campaigns created by all users and organizations.
- 2) Campaign: In this section you are able to create the campaign. It is basically a form having fields such as Name, Title, Description, Fund Raise, End Date and Image of your Campaign.
- 3) Profile: This section contains the campaigns that you have created of your organization.

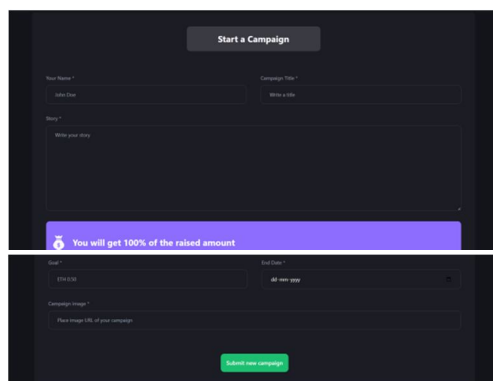


Fig-9: Create Campaign Page

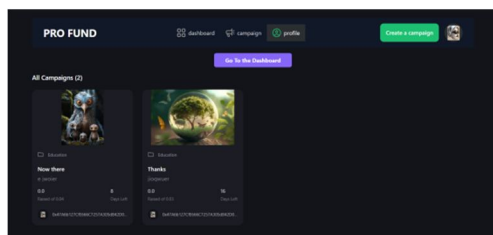


Fig-10: Profile Page

After the Campaigns are created the other users are ready to donate the Campaigns to fund the Campaigns you need to click on the Campaign and the fund card will pop-up as shown in below fig-11.

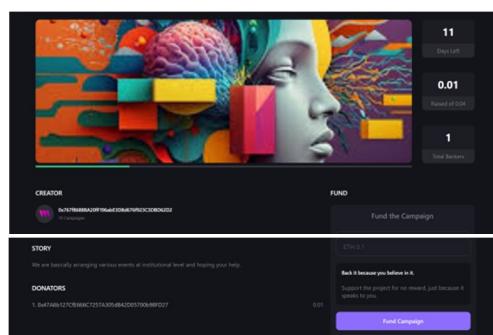


Fig-11: Funding Page

Funding Page contains the fields such as Owner of Campaign, Days left, Raised Fund, No of contributors, Story, and Fund Card where you put the Eth and donate to the Campaign.

V. CONCLUSION AND FUTURE WORK

In conclusion, a big step forward in the crowdfunding industry has been made with the Binance Blockchain and hybrid model crowdfunding application developed with React, Solidity, Harhat, ThirdWeb and Binance Smart Chain (BSC). The platform effectively blends the capabilities of blockchain technology with the creation of hybrid models, resulting in an easily navigable, safe, and transparent crowdfunding experience. The software is more dependable and draws in a diverse user base because to its low cost, security features, user training, and compliance management.

In this paper, we have discovered that the many flaws in conventional crowdfunding processes have been eliminated with the help of blockchain technology integration. This is achieved by removing the concept of central authority from the platform, which makes it decentralized and eliminates the need for middlemen while maintaining transaction transparency.

In order to comply with new standards, more work is required to assure coordination with diverse blockchain networks, scale worldwide to meet regulatory needs, and continue research. In order to gather user feedback and promote ongoing innovation, community interaction is essential for keeping the platform competitive and relevant in a rapidly evolving market.

REFERENCES

- [1] Lee, Jei Young. "A decentralized token economy: How blockchain and cryptocurrency can revolutionize business." *Business Horizons* 62.6 (2019): 773-784.
- [2] Geetika Jhanji, Vidushi Tyagi, Aditya Gaur, Yogesh Sharma. "Development of a Crowdfunding application Powered by Ethereum Blockchain." 2023. 1-7.
- [3] Alkhana Abuhashim, Chiu C. Tan. "Smart Contract Design on Blockchain Applications." 2020.
- [4] Faten Adel Alabdulwahhab. "Web3.0 : The Decentralized Web." 2018.
- [5] D Sathya, S Nithyaroopa, D Jagadeesan, I Jeena Jacob. "Blockchain Technology for Food Supply Chains." 2021.
- [6] Shivendra, Dr.Kasa Chiranjeevi, Mukesh Kumar Tripathi, Dr. Dhananjay D. Maktedar. "Block chain Technology in Agriculture Product Supply Chain." 2021.
- [7] Mrs.M.C.Jayaprasanna, Ms.V.A.Soundharya, Ms.M.Suhana, Dr.S.Sujatha. "A Blockchain based Management System for Detecting Counterfit Product in Supply Chain." 2021.
- [8] Lee, Wei-Meng. "Using the Metamask chrome extension." *Beginning Ethereum Smart Contract Programming*. Apress, Berkeley, CA, 2019.
- [9] Faheem Ahmad Reegu, Salwani Mohd Daud, Shadab Alam, Mohammed Shuaib. "Blockchain-based Electronic Health Record System for efficient Covid-19 Pandemic Management." 2020.
- [10] Wathan, A. "Tailwind CSS: A utility-first CSS framework for rapid UI development." 2021.



- [11] Banks, Alex, and Eve Porcello. "Learning React: functional web development with React and Redux." 2017.
- [12] Palechor, Luisa, and Cor-Paul Benzemer. "How are Solidity Smart contracts tested in open source projects? An exploratory study." 2022.
- [13] Olivier, Starkenmann, Karl Schmedders, and José Parra Moyano. "Implementation of a Crowdfunding Decentralized Application on Ethereum Master Thesis."
- [14] Sarmah, Simanta Shekhar. "Understanding blockchain technology." Computer Science and Engineering 8.2 (2018).



10.22214/IJRASET



45.98



IMPACT FACTOR:
7.129



IMPACT FACTOR:
7.429



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089  (24*7 Support on Whatsapp)