



# **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](http://www.ijraset.com)**

**Call:** ☎ 08813907089

**E-mail ID:** [ijraset@gmail.com](mailto:ijraset@gmail.com)

# The Crowdfunding Application by Blockchain and ThirdWeb Hybrid Model

Prof. Meera Sawalkar<sup>1</sup>, Viraj Shewale<sup>2</sup>, Prasad Sutar<sup>3</sup>, Vinit Kawachale<sup>4</sup>

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.

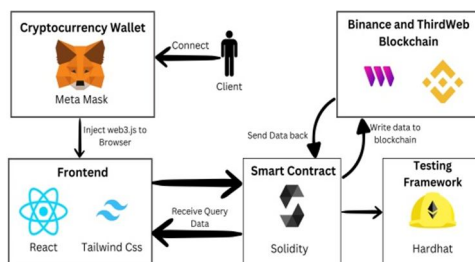


Fig -1: System Architecture of Application

#### 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.

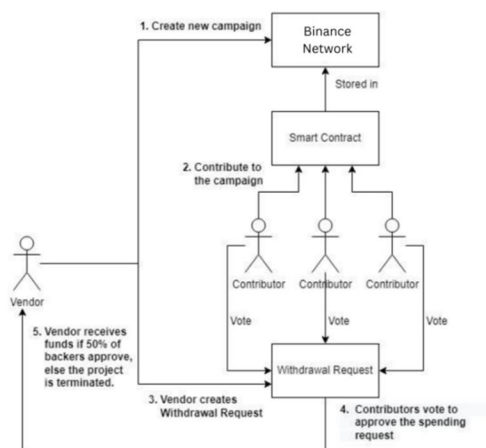


Fig-2: Flow of Crowdfunding the Application.

### 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: 0xac0974bec39a17e36ba4a6b4d238ff944baca78cb5d5e5cae784d7bf4f2ff08

Account #1: 0x7995797c31812dc3a01bc7d01b58e0d17dc79c8 (10000 ETH)
Private Key: 0xc59c995e988f97a5a0044966f0945389dc9e8daec8c7a8412f46036b78690d

Account #2: 0xc344cdd86a900fa2b585dd299e03d12fA42938C (10000 ETH)
Private Key: 0xc5de4111af1a1ad94988f83103eb1f1766367c2e6ca870fc3fb9a804cdab365a

Account #3: 0x90f79b66f6b2c4f870365e785982e1f01E93b906 (10000 ETH)
Private Key: 0x7c852118294e51e653712a81e05800f419141751be58f605c371e15141b007a6

Account #4: 0x15d34Aaf542670B7D7C367839AAf71A00a2C6A65 (10000 ETH)
Private Key: 0x47e179ec197488593b187f80a00eb0da91f1b9d0b13f8733639f19c30a34926a

Account #5: 0x9965507D1a55bc2695C58ba16F837d81900A4dc (10000 ETH)
Private Key: 0x8b3a350cf5c34c9194ca85829a2df0ec3153be03185e2d3348e872092edfba

Account #6: 0x976EA74026E726554dB657FA54763abd0C3a0aa9 (10000 ETH)
Private Key: 0x92db14e483b83df3d233f83dfa3a0d7096f213ca9bedd06888b244c1564e

```

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.

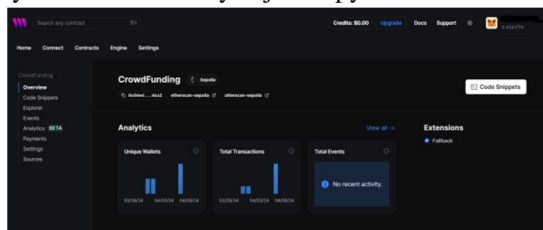


Fig-5: ThirdWeb Contract deployment

## 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 Profund {
5     struct Campaign {
6         string title;
7         string description;
8         string target;
9         string deadline;
10        string image;
11        address[] donors;
12        uint256[] amounts;
13    }
14
15    mapping(uint256 => Campaign) public campaigns;
16
17    uint256 public numberOfCampaigns = 0;
18
19    function createCampaign(address _owner, string memory _title, string memory _description, uint256 _target, uint256 _deadline, string memory _image) public {
20        require(_owner == msg.sender, "Only the owner can create a campaign");
21        require(_target > 0, "Target must be greater than 0");
22        require(_deadline > block.timestamp, "The deadline should be a date in the future.");
23
24        Campaign memory campaign = Campaign({
25            title: _title,
26            description: _description,
27            target: _target,
28            deadline: _deadline,
29            image: _image,
30            donors: new address[](0),
31            amounts: new uint256[](0)
32        });
33
34        campaigns[numberOfCampaigns] = campaign;
35        numberOfCampaigns++;
36    }
37
38    function numberOfCampaigns() public view returns (uint256) {
39        return numberOfCampaigns;
40    }
41
42    function get Campaign(uint256 _id) public view returns (Campaign memory) {
43        return campaigns[_id];
44    }
45
46    function get Campaign(uint256 _id) public view returns (string memory) {
47        return campaigns[_id].title;
48    }
49
50    function get Campaign(uint256 _id) public view returns (string memory) {
51        return campaigns[_id].description;
52    }
53
54    function get Campaign(uint256 _id) public view returns (string memory) {
55        return campaigns[_id].target;
56    }
57
58    function get Campaign(uint256 _id) public view returns (string memory) {
59        return campaigns[_id].deadline;
60    }
61
62    function get Campaign(uint256 _id) public view returns (string memory) {
63        return campaigns[_id].image;
64    }
65
66    function get Campaign(uint256 _id) public view returns (address[] memory) {
67        return campaigns[_id].donors;
68    }
69
70    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
71        return campaigns[_id].amounts;
72    }
73
74    function get Campaign(uint256 _id) public view returns (uint256) {
75        return campaigns[_id].target;
76    }
77
78    function get Campaign(uint256 _id) public view returns (uint256) {
79        return campaigns[_id].deadline;
80    }
81
82    function get Campaign(uint256 _id) public view returns (string memory) {
83        return campaigns[_id].image;
84    }
85
86    function get Campaign(uint256 _id) public view returns (address[] memory) {
87        return campaigns[_id].donors;
88    }
89
90    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
91        return campaigns[_id].amounts;
92    }
93
94    function get Campaign(uint256 _id) public view returns (uint256) {
95        return campaigns[_id].target;
96    }
97
98    function get Campaign(uint256 _id) public view returns (uint256) {
99        return campaigns[_id].deadline;
100    }
101
102    function get Campaign(uint256 _id) public view returns (string memory) {
103        return campaigns[_id].image;
104    }
105
106    function get Campaign(uint256 _id) public view returns (address[] memory) {
107        return campaigns[_id].donors;
108    }
109
110    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
111        return campaigns[_id].amounts;
112    }
113
114    function get Campaign(uint256 _id) public view returns (uint256) {
115        return campaigns[_id].target;
116    }
117
118    function get Campaign(uint256 _id) public view returns (uint256) {
119        return campaigns[_id].deadline;
120    }
121
122    function get Campaign(uint256 _id) public view returns (string memory) {
123        return campaigns[_id].image;
124    }
125
126    function get Campaign(uint256 _id) public view returns (address[] memory) {
127        return campaigns[_id].donors;
128    }
129
130    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
131        return campaigns[_id].amounts;
132    }
133
134    function get Campaign(uint256 _id) public view returns (uint256) {
135        return campaigns[_id].target;
136    }
137
138    function get Campaign(uint256 _id) public view returns (uint256) {
139        return campaigns[_id].deadline;
140    }
141
142    function get Campaign(uint256 _id) public view returns (string memory) {
143        return campaigns[_id].image;
144    }
145
146    function get Campaign(uint256 _id) public view returns (address[] memory) {
147        return campaigns[_id].donors;
148    }
149
150    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
151        return campaigns[_id].amounts;
152    }
153
154    function get Campaign(uint256 _id) public view returns (uint256) {
155        return campaigns[_id].target;
156    }
157
158    function get Campaign(uint256 _id) public view returns (uint256) {
159        return campaigns[_id].deadline;
160    }
161
162    function get Campaign(uint256 _id) public view returns (string memory) {
163        return campaigns[_id].image;
164    }
165
166    function get Campaign(uint256 _id) public view returns (address[] memory) {
167        return campaigns[_id].donors;
168    }
169
170    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
171        return campaigns[_id].amounts;
172    }
173
174    function get Campaign(uint256 _id) public view returns (uint256) {
175        return campaigns[_id].target;
176    }
177
178    function get Campaign(uint256 _id) public view returns (uint256) {
179        return campaigns[_id].deadline;
180    }
181
182    function get Campaign(uint256 _id) public view returns (string memory) {
183        return campaigns[_id].image;
184    }
185
186    function get Campaign(uint256 _id) public view returns (address[] memory) {
187        return campaigns[_id].donors;
188    }
189
190    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
191        return campaigns[_id].amounts;
192    }
193
194    function get Campaign(uint256 _id) public view returns (uint256) {
195        return campaigns[_id].target;
196    }
197
198    function get Campaign(uint256 _id) public view returns (uint256) {
199        return campaigns[_id].deadline;
200    }
201
202    function get Campaign(uint256 _id) public view returns (string memory) {
203        return campaigns[_id].image;
204    }
205
206    function get Campaign(uint256 _id) public view returns (address[] memory) {
207        return campaigns[_id].donors;
208    }
209
210    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
211        return campaigns[_id].amounts;
212    }
213
214    function get Campaign(uint256 _id) public view returns (uint256) {
215        return campaigns[_id].target;
216    }
217
218    function get Campaign(uint256 _id) public view returns (uint256) {
219        return campaigns[_id].deadline;
220    }
221
222    function get Campaign(uint256 _id) public view returns (string memory) {
223        return campaigns[_id].image;
224    }
225
226    function get Campaign(uint256 _id) public view returns (address[] memory) {
227        return campaigns[_id].donors;
228    }
229
230    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
231        return campaigns[_id].amounts;
232    }
233
234    function get Campaign(uint256 _id) public view returns (uint256) {
235        return campaigns[_id].target;
236    }
237
238    function get Campaign(uint256 _id) public view returns (uint256) {
239        return campaigns[_id].deadline;
240    }
241
242    function get Campaign(uint256 _id) public view returns (string memory) {
243        return campaigns[_id].image;
244    }
245
246    function get Campaign(uint256 _id) public view returns (address[] memory) {
247        return campaigns[_id].donors;
248    }
249
250    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
251        return campaigns[_id].amounts;
252    }
253
254    function get Campaign(uint256 _id) public view returns (uint256) {
255        return campaigns[_id].target;
256    }
257
258    function get Campaign(uint256 _id) public view returns (uint256) {
259        return campaigns[_id].deadline;
260    }
261
262    function get Campaign(uint256 _id) public view returns (string memory) {
263        return campaigns[_id].image;
264    }
265
266    function get Campaign(uint256 _id) public view returns (address[] memory) {
267        return campaigns[_id].donors;
268    }
269
270    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
271        return campaigns[_id].amounts;
272    }
273
274    function get Campaign(uint256 _id) public view returns (uint256) {
275        return campaigns[_id].target;
276    }
277
278    function get Campaign(uint256 _id) public view returns (uint256) {
279        return campaigns[_id].deadline;
280    }
281
282    function get Campaign(uint256 _id) public view returns (string memory) {
283        return campaigns[_id].image;
284    }
285
286    function get Campaign(uint256 _id) public view returns (address[] memory) {
287        return campaigns[_id].donors;
288    }
289
290    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
291        return campaigns[_id].amounts;
292    }
293
294    function get Campaign(uint256 _id) public view returns (uint256) {
295        return campaigns[_id].target;
296    }
297
298    function get Campaign(uint256 _id) public view returns (uint256) {
299        return campaigns[_id].deadline;
300    }
301
302    function get Campaign(uint256 _id) public view returns (string memory) {
303        return campaigns[_id].image;
304    }
305
306    function get Campaign(uint256 _id) public view returns (address[] memory) {
307        return campaigns[_id].donors;
308    }
309
310    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
311        return campaigns[_id].amounts;
312    }
313
314    function get Campaign(uint256 _id) public view returns (uint256) {
315        return campaigns[_id].target;
316    }
317
318    function get Campaign(uint256 _id) public view returns (uint256) {
319        return campaigns[_id].deadline;
320    }
321
322    function get Campaign(uint256 _id) public view returns (string memory) {
323        return campaigns[_id].image;
324    }
325
326    function get Campaign(uint256 _id) public view returns (address[] memory) {
327        return campaigns[_id].donors;
328    }
329
330    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
331        return campaigns[_id].amounts;
332    }
333
334    function get Campaign(uint256 _id) public view returns (uint256) {
335        return campaigns[_id].target;
336    }
337
338    function get Campaign(uint256 _id) public view returns (uint256) {
339        return campaigns[_id].deadline;
340    }
341
342    function get Campaign(uint256 _id) public view returns (string memory) {
343        return campaigns[_id].image;
344    }
345
346    function get Campaign(uint256 _id) public view returns (address[] memory) {
347        return campaigns[_id].donors;
348    }
349
350    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
351        return campaigns[_id].amounts;
352    }
353
354    function get Campaign(uint256 _id) public view returns (uint256) {
355        return campaigns[_id].target;
356    }
357
358    function get Campaign(uint256 _id) public view returns (uint256) {
359        return campaigns[_id].deadline;
360    }
361
362    function get Campaign(uint256 _id) public view returns (string memory) {
363        return campaigns[_id].image;
364    }
365
366    function get Campaign(uint256 _id) public view returns (address[] memory) {
367        return campaigns[_id].donors;
368    }
369
370    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
371        return campaigns[_id].amounts;
372    }
373
374    function get Campaign(uint256 _id) public view returns (uint256) {
375        return campaigns[_id].target;
376    }
377
378    function get Campaign(uint256 _id) public view returns (uint256) {
379        return campaigns[_id].deadline;
380    }
381
382    function get Campaign(uint256 _id) public view returns (string memory) {
383        return campaigns[_id].image;
384    }
385
386    function get Campaign(uint256 _id) public view returns (address[] memory) {
387        return campaigns[_id].donors;
388    }
389
390    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
391        return campaigns[_id].amounts;
392    }
393
394    function get Campaign(uint256 _id) public view returns (uint256) {
395        return campaigns[_id].target;
396    }
397
398    function get Campaign(uint256 _id) public view returns (uint256) {
399        return campaigns[_id].deadline;
400    }
401
402    function get Campaign(uint256 _id) public view returns (string memory) {
403        return campaigns[_id].image;
404    }
405
406    function get Campaign(uint256 _id) public view returns (address[] memory) {
407        return campaigns[_id].donors;
408    }
409
410    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
411        return campaigns[_id].amounts;
412    }
413
414    function get Campaign(uint256 _id) public view returns (uint256) {
415        return campaigns[_id].target;
416    }
417
418    function get Campaign(uint256 _id) public view returns (uint256) {
419        return campaigns[_id].deadline;
420    }
421
422    function get Campaign(uint256 _id) public view returns (string memory) {
423        return campaigns[_id].image;
424    }
425
426    function get Campaign(uint256 _id) public view returns (address[] memory) {
427        return campaigns[_id].donors;
428    }
429
430    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
431        return campaigns[_id].amounts;
432    }
433
434    function get Campaign(uint256 _id) public view returns (uint256) {
435        return campaigns[_id].target;
436    }
437
438    function get Campaign(uint256 _id) public view returns (uint256) {
439        return campaigns[_id].deadline;
440    }
441
442    function get Campaign(uint256 _id) public view returns (string memory) {
443        return campaigns[_id].image;
444    }
445
446    function get Campaign(uint256 _id) public view returns (address[] memory) {
447        return campaigns[_id].donors;
448    }
449
450    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
451        return campaigns[_id].amounts;
452    }
453
454    function get Campaign(uint256 _id) public view returns (uint256) {
455        return campaigns[_id].target;
456    }
457
458    function get Campaign(uint256 _id) public view returns (uint256) {
459        return campaigns[_id].deadline;
460    }
461
462    function get Campaign(uint256 _id) public view returns (string memory) {
463        return campaigns[_id].image;
464    }
465
466    function get Campaign(uint256 _id) public view returns (address[] memory) {
467        return campaigns[_id].donors;
468    }
469
470    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
471        return campaigns[_id].amounts;
472    }
473
474    function get Campaign(uint256 _id) public view returns (uint256) {
475        return campaigns[_id].target;
476    }
477
478    function get Campaign(uint256 _id) public view returns (uint256) {
479        return campaigns[_id].deadline;
480    }
481
482    function get Campaign(uint256 _id) public view returns (string memory) {
483        return campaigns[_id].image;
484    }
485
486    function get Campaign(uint256 _id) public view returns (address[] memory) {
487        return campaigns[_id].donors;
488    }
489
490    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
491        return campaigns[_id].amounts;
492    }
493
494    function get Campaign(uint256 _id) public view returns (uint256) {
495        return campaigns[_id].target;
496    }
497
498    function get Campaign(uint256 _id) public view returns (uint256) {
499        return campaigns[_id].deadline;
500    }
501
502    function get Campaign(uint256 _id) public view returns (string memory) {
503        return campaigns[_id].image;
504    }
505
506    function get Campaign(uint256 _id) public view returns (address[] memory) {
507        return campaigns[_id].donors;
508    }
509
510    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
511        return campaigns[_id].amounts;
512    }
513
514    function get Campaign(uint256 _id) public view returns (uint256) {
515        return campaigns[_id].target;
516    }
517
518    function get Campaign(uint256 _id) public view returns (uint256) {
519        return campaigns[_id].deadline;
520    }
521
522    function get Campaign(uint256 _id) public view returns (string memory) {
523        return campaigns[_id].image;
524    }
525
526    function get Campaign(uint256 _id) public view returns (address[] memory) {
527        return campaigns[_id].donors;
528    }
529
530    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
531        return campaigns[_id].amounts;
532    }
533
534    function get Campaign(uint256 _id) public view returns (uint256) {
535        return campaigns[_id].target;
536    }
537
538    function get Campaign(uint256 _id) public view returns (uint256) {
539        return campaigns[_id].deadline;
540    }
541
542    function get Campaign(uint256 _id) public view returns (string memory) {
543        return campaigns[_id].image;
544    }
545
546    function get Campaign(uint256 _id) public view returns (address[] memory) {
547        return campaigns[_id].donors;
548    }
549
550    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
551        return campaigns[_id].amounts;
552    }
553
554    function get Campaign(uint256 _id) public view returns (uint256) {
555        return campaigns[_id].target;
556    }
557
558    function get Campaign(uint256 _id) public view returns (uint256) {
559        return campaigns[_id].deadline;
560    }
561
562    function get Campaign(uint256 _id) public view returns (string memory) {
563        return campaigns[_id].image;
564    }
565
566    function get Campaign(uint256 _id) public view returns (address[] memory) {
567        return campaigns[_id].donors;
568    }
569
570    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
571        return campaigns[_id].amounts;
572    }
573
574    function get Campaign(uint256 _id) public view returns (uint256) {
575        return campaigns[_id].target;
576    }
577
578    function get Campaign(uint256 _id) public view returns (uint256) {
579        return campaigns[_id].deadline;
580    }
581
582    function get Campaign(uint256 _id) public view returns (string memory) {
583        return campaigns[_id].image;
584    }
585
586    function get Campaign(uint256 _id) public view returns (address[] memory) {
587        return campaigns[_id].donors;
588    }
589
590    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
591        return campaigns[_id].amounts;
592    }
593
594    function get Campaign(uint256 _id) public view returns (uint256) {
595        return campaigns[_id].target;
596    }
597
598    function get Campaign(uint256 _id) public view returns (uint256) {
599        return campaigns[_id].deadline;
600    }
601
602    function get Campaign(uint256 _id) public view returns (string memory) {
603        return campaigns[_id].image;
604    }
605
606    function get Campaign(uint256 _id) public view returns (address[] memory) {
607        return campaigns[_id].donors;
608    }
609
610    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
611        return campaigns[_id].amounts;
612    }
613
614    function get Campaign(uint256 _id) public view returns (uint256) {
615        return campaigns[_id].target;
616    }
617
618    function get Campaign(uint256 _id) public view returns (uint256) {
619        return campaigns[_id].deadline;
620    }
621
622    function get Campaign(uint256 _id) public view returns (string memory) {
623        return campaigns[_id].image;
624    }
625
626    function get Campaign(uint256 _id) public view returns (address[] memory) {
627        return campaigns[_id].donors;
628    }
629
630    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
631        return campaigns[_id].amounts;
632    }
633
634    function get Campaign(uint256 _id) public view returns (uint256) {
635        return campaigns[_id].target;
636    }
637
638    function get Campaign(uint256 _id) public view returns (uint256) {
639        return campaigns[_id].deadline;
640    }
641
642    function get Campaign(uint256 _id) public view returns (string memory) {
643        return campaigns[_id].image;
644    }
645
646    function get Campaign(uint256 _id) public view returns (address[] memory) {
647        return campaigns[_id].donors;
648    }
649
650    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
651        return campaigns[_id].amounts;
652    }
653
654    function get Campaign(uint256 _id) public view returns (uint256) {
655        return campaigns[_id].target;
656    }
657
658    function get Campaign(uint256 _id) public view returns (uint256) {
659        return campaigns[_id].deadline;
660    }
661
662    function get Campaign(uint256 _id) public view returns (string memory) {
663        return campaigns[_id].image;
664    }
665
666    function get Campaign(uint256 _id) public view returns (address[] memory) {
667        return campaigns[_id].donors;
668    }
669
670    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
671        return campaigns[_id].amounts;
672    }
673
674    function get Campaign(uint256 _id) public view returns (uint256) {
675        return campaigns[_id].target;
676    }
677
678    function get Campaign(uint256 _id) public view returns (uint256) {
679        return campaigns[_id].deadline;
680    }
681
682    function get Campaign(uint256 _id) public view returns (string memory) {
683        return campaigns[_id].image;
684    }
685
686    function get Campaign(uint256 _id) public view returns (address[] memory) {
687        return campaigns[_id].donors;
688    }
689
690    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
691        return campaigns[_id].amounts;
692    }
693
694    function get Campaign(uint256 _id) public view returns (uint256) {
695        return campaigns[_id].target;
696    }
697
698    function get Campaign(uint256 _id) public view returns (uint256) {
699        return campaigns[_id].deadline;
700    }
701
702    function get Campaign(uint256 _id) public view returns (string memory) {
703        return campaigns[_id].image;
704    }
705
706    function get Campaign(uint256 _id) public view returns (address[] memory) {
707        return campaigns[_id].donors;
708    }
709
710    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
711        return campaigns[_id].amounts;
712    }
713
714    function get Campaign(uint256 _id) public view returns (uint256) {
715        return campaigns[_id].target;
716    }
717
718    function get Campaign(uint256 _id) public view returns (uint256) {
719        return campaigns[_id].deadline;
720    }
721
722    function get Campaign(uint256 _id) public view returns (string memory) {
723        return campaigns[_id].image;
724    }
725
726    function get Campaign(uint256 _id) public view returns (address[] memory) {
727        return campaigns[_id].donors;
728    }
729
730    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
731        return campaigns[_id].amounts;
732    }
733
734    function get Campaign(uint256 _id) public view returns (uint256) {
735        return campaigns[_id].target;
736    }
737
738    function get Campaign(uint256 _id) public view returns (uint256) {
739        return campaigns[_id].deadline;
740    }
741
742    function get Campaign(uint256 _id) public view returns (string memory) {
743        return campaigns[_id].image;
744    }
745
746    function get Campaign(uint256 _id) public view returns (address[] memory) {
747        return campaigns[_id].donors;
748    }
749
750    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
751        return campaigns[_id].amounts;
752    }
753
754    function get Campaign(uint256 _id) public view returns (uint256) {
755        return campaigns[_id].target;
756    }
757
758    function get Campaign(uint256 _id) public view returns (uint256) {
759        return campaigns[_id].deadline;
760    }
761
762    function get Campaign(uint256 _id) public view returns (string memory) {
763        return campaigns[_id].image;
764    }
765
766    function get Campaign(uint256 _id) public view returns (address[] memory) {
767        return campaigns[_id].donors;
768    }
769
770    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
771        return campaigns[_id].amounts;
772    }
773
774    function get Campaign(uint256 _id) public view returns (uint256) {
775        return campaigns[_id].target;
776    }
777
778    function get Campaign(uint256 _id) public view returns (uint256) {
779        return campaigns[_id].deadline;
780    }
781
782    function get Campaign(uint256 _id) public view returns (string memory) {
783        return campaigns[_id].image;
784    }
785
786    function get Campaign(uint256 _id) public view returns (address[] memory) {
787        return campaigns[_id].donors;
788    }
789
790    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
791        return campaigns[_id].amounts;
792    }
793
794    function get Campaign(uint256 _id) public view returns (uint256) {
795        return campaigns[_id].target;
796    }
797
798    function get Campaign(uint256 _id) public view returns (uint256) {
799        return campaigns[_id].deadline;
800    }
801
802    function get Campaign(uint256 _id) public view returns (string memory) {
803        return campaigns[_id].image;
804    }
805
806    function get Campaign(uint256 _id) public view returns (address[] memory) {
807        return campaigns[_id].donors;
808    }
809
810    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
811        return campaigns[_id].amounts;
812    }
813
814    function get Campaign(uint256 _id) public view returns (uint256) {
815        return campaigns[_id].target;
816    }
817
818    function get Campaign(uint256 _id) public view returns (uint256) {
819        return campaigns[_id].deadline;
820    }
821
822    function get Campaign(uint256 _id) public view returns (string memory) {
823        return campaigns[_id].image;
824    }
825
826    function get Campaign(uint256 _id) public view returns (address[] memory) {
827        return campaigns[_id].donors;
828    }
829
830    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
831        return campaigns[_id].amounts;
832    }
833
834    function get Campaign(uint256 _id) public view returns (uint256) {
835        return campaigns[_id].target;
836    }
837
838    function get Campaign(uint256 _id) public view returns (uint256) {
839        return campaigns[_id].deadline;
840    }
841
842    function get Campaign(uint256 _id) public view returns (string memory) {
843        return campaigns[_id].image;
844    }
845
846    function get Campaign(uint256 _id) public view returns (address[] memory) {
847        return campaigns[_id].donors;
848    }
849
850    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
851        return campaigns[_id].amounts;
852    }
853
854    function get Campaign(uint256 _id) public view returns (uint256) {
855        return campaigns[_id].target;
856    }
857
858    function get Campaign(uint256 _id) public view returns (uint256) {
859        return campaigns[_id].deadline;
860    }
861
862    function get Campaign(uint256 _id) public view returns (string memory) {
863        return campaigns[_id].image;
864    }
865
866    function get Campaign(uint256 _id) public view returns (address[] memory) {
867        return campaigns[_id].donors;
868    }
869
870    function get Campaign(uint256 _id) public view returns (uint256[] memory) {
871        return campaigns[_id].amounts;
872    }
873
874    function get Campaign(uint256 _id) public view returns (uint256) {
875        return campaigns[_id].target;
876    }
877
878    function get Campaign(uint256 _id) public view returns (uint256) {
879        return campaigns[_id].deadline;
880    }
881
88
```

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.

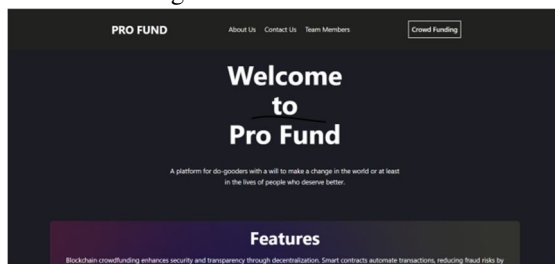


Fig-6: Front-end website page

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

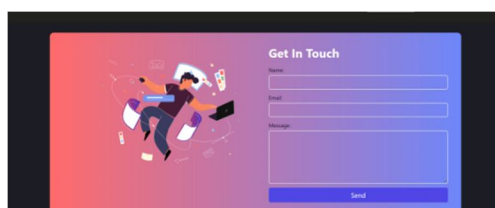


Fig-7: Contact Us page

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

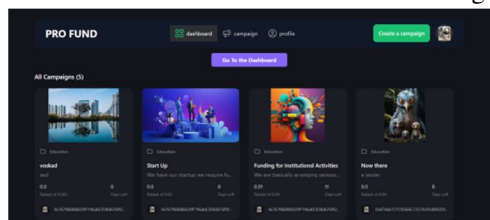


Fig-8: Crowd Funding Page

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.

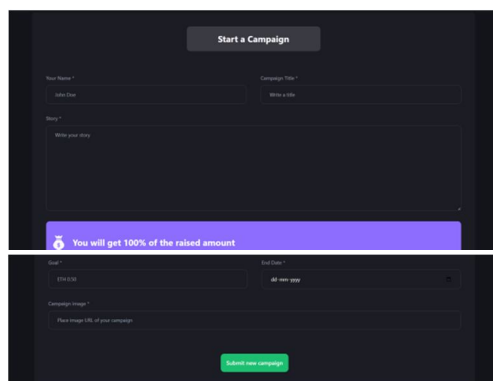


Fig-9: Create Campaign Page

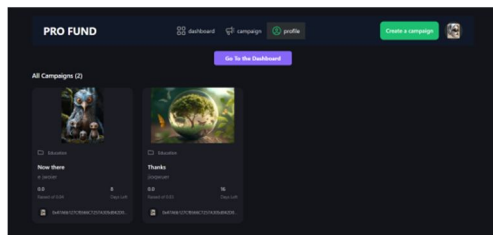


Fig-10: Profile Page

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.

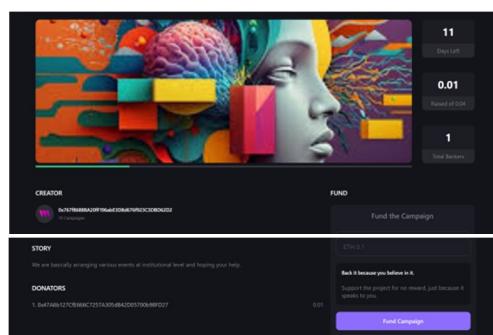


Fig-11: Funding Page

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 Nithyaroop, 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)