



# 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.

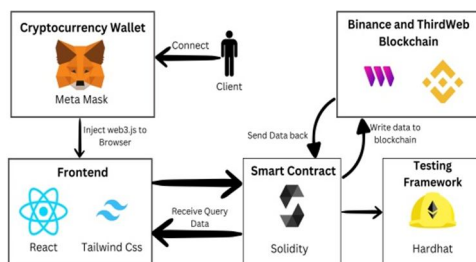


Fig -1: System Architecture of Application

#### 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.

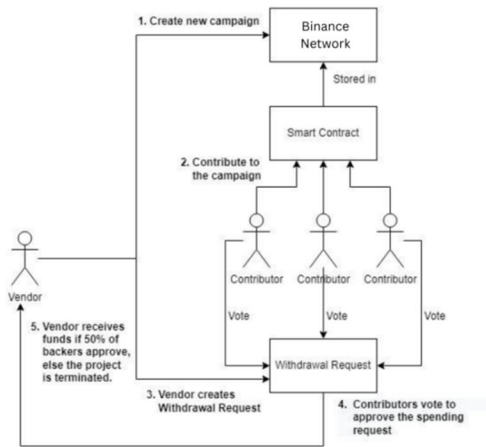


Fig-2: Flow of Crowdfunding the Application.

### 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 interactivity is handled 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 works 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*: Thirdweb is a blockchain platform emphasizing privacy, scalability, and security. It employs a unique consensus mechanism called Proof-of-Merit to validate transactions. Thirdweb 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: 0xf39fd651aa8889f64c6a882729cffff92266 (10000 ETH)
Private Key: 0xac9974bec39a17e3b4a4b44d238f944ba94c63b02047d81f47e4603eb3201000

Account #1: 0x78909797c51812dc3a018c7d01b90e817dc79c8 (10000 ETH)
Private Key: 0xc39cc995e98f97a5a0044266f09453894c9e86dae88c73841274603b678690d

Account #2: 0x3c44cdd86a90ffa2b585dd299e03d12fa4293bc (10000 ETH)
Private Key: 0xc5de4111af1a4ab94988f83109eb1f176637c2ed6ca870cf3f9a804cdab365a

Account #3: 0x90f79bf6EB2c4f80365E785982E1f01E93b906 (10000 ETH)
Private Key: 0x7c852118294e51e65372a81e05800f419141751be58f605c371e15141b007a6

Account #4: 0x15d34AAf54267D07C36783AAf71A00a2C6A65 (10000 ETH)
Private Key: 0x47e179ec197488593b187f80a00eb0da91f1b9d0b13f8733639f19c30a34926a

Account #5: 0x9965507D1A95bC2695C8Ba16fB37d8190A04dc (10000 ETH)
Private Key: 0x8b3a350c5fc34c9194c88529a2df6ccf153be03185e2d348e872092edffba

Account #6: 0x976EA7A026E72654d8657FA54763ab0C380a9 (10000 ETH)
Private Key: 0x92db14e403b83dfc3d233f83d4fa3a0d7096f21ca9bed6db88b2b4ec1564e
    
```

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 `npm 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.

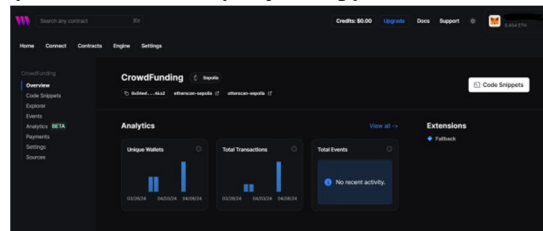


Fig-5: ThirdWeb Contract deployment

#### 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: MIT
2 pragma solidity ^0.8.0;
3
4 contract Crowdfunder {
5     struct Campaign {
6         address owner;
7         string title;
8         string description;
9         address target;
10        string deadline;
11        string amountCollected;
12        string image;
13        address[] donators;
14        uint256[] amounts;
15    }
16
17    mapping(uint256 => Campaign) public campaigns;
18
19    uint256 public numberOfCampaigns = 0;
20
21    function createCampaign(address _owner, string memory _title, string memory _description, uint256 _target, uint256 _deadline, string memory _image) public {
22        createCampaign(Campaign({owner:_owner, title:_title, description:_description, target:_target, deadline:_deadline, amountCollected:"", image:_image, donators: new address[](0), amounts: new uint256[](0)}));
23    }
24
25    function createCampaign(Campaign memory campaign) public {
26        require(campaign.deadline > block.timestamp, "The deadline should be a date in the future.");
27
28        campaign.owner = _owner;
29        campaign.title = _title;
30        campaign.description = _description;
31        campaign.target = _target;
32        campaign.deadline = _deadline;
33        campaign.amountCollected = "";
34        campaign.image = _image;
35        numberOfCampaigns++;
36    }
37
38    function getCampaign() public view returns (Campaign memory) {
39        Campaign memory campaign;
40        return campaign;
41    }
42
43    function getAllCampaigns() public view returns (Campaign[] memory) {
44        Campaign[] memory allCampaigns;
45        for (uint256 i = 0; i < numberOfCampaigns; i++) {
46            allCampaigns.push(campaigns[i]);
47        }
48        return allCampaigns;
49    }
50
51    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
52        address[] memory donators;
53        uint256[] memory amounts;
54        Campaign memory campaign;
55        return (donators, amounts);
56    }
57
58    function addDonatorAndAmount() public {
59        address[] memory donators;
60        uint256[] memory amounts;
61        Campaign memory campaign;
62        (donators, amounts) = getDonatorsAndAmounts();
63        donators.push(msg.sender);
64        amounts.push(msg.value);
65    }
66
67    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
68        address[] memory donators;
69        uint256[] memory amounts;
70        Campaign memory campaign;
71        (donators, amounts) = getDonatorsAndAmounts();
72        return (donators, amounts);
73    }
74
75    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
76        address[] memory donators;
77        uint256[] memory amounts;
78        Campaign memory campaign;
79        (donators, amounts) = getDonatorsAndAmounts();
80        return (donators, amounts);
81    }
82
83    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
84        address[] memory donators;
85        uint256[] memory amounts;
86        Campaign memory campaign;
87        (donators, amounts) = getDonatorsAndAmounts();
88        return (donators, amounts);
89    }
90
91    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
92        address[] memory donators;
93        uint256[] memory amounts;
94        Campaign memory campaign;
95        (donators, amounts) = getDonatorsAndAmounts();
96        return (donators, amounts);
97    }
98
99    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
100        address[] memory donators;
101        uint256[] memory amounts;
102        Campaign memory campaign;
103        (donators, amounts) = getDonatorsAndAmounts();
104        return (donators, amounts);
105    }
106
107    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
108        address[] memory donators;
109        uint256[] memory amounts;
110        Campaign memory campaign;
111        (donators, amounts) = getDonatorsAndAmounts();
112        return (donators, amounts);
113    }
114
115    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
116        address[] memory donators;
117        uint256[] memory amounts;
118        Campaign memory campaign;
119        (donators, amounts) = getDonatorsAndAmounts();
120        return (donators, amounts);
121    }
122
123    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
124        address[] memory donators;
125        uint256[] memory amounts;
126        Campaign memory campaign;
127        (donators, amounts) = getDonatorsAndAmounts();
128        return (donators, amounts);
129    }
130
131    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
132        address[] memory donators;
133        uint256[] memory amounts;
134        Campaign memory campaign;
135        (donators, amounts) = getDonatorsAndAmounts();
136        return (donators, amounts);
137    }
138
139    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
140        address[] memory donators;
141        uint256[] memory amounts;
142        Campaign memory campaign;
143        (donators, amounts) = getDonatorsAndAmounts();
144        return (donators, amounts);
145    }
146
147    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
148        address[] memory donators;
149        uint256[] memory amounts;
150        Campaign memory campaign;
151        (donators, amounts) = getDonatorsAndAmounts();
152        return (donators, amounts);
153    }
154
155    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
156        address[] memory donators;
157        uint256[] memory amounts;
158        Campaign memory campaign;
159        (donators, amounts) = getDonatorsAndAmounts();
160        return (donators, amounts);
161    }
162
163    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
164        address[] memory donators;
165        uint256[] memory amounts;
166        Campaign memory campaign;
167        (donators, amounts) = getDonatorsAndAmounts();
168        return (donators, amounts);
169    }
170
171    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
172        address[] memory donators;
173        uint256[] memory amounts;
174        Campaign memory campaign;
175        (donators, amounts) = getDonatorsAndAmounts();
176        return (donators, amounts);
177    }
178
179    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
180        address[] memory donators;
181        uint256[] memory amounts;
182        Campaign memory campaign;
183        (donators, amounts) = getDonatorsAndAmounts();
184        return (donators, amounts);
185    }
186
187    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
188        address[] memory donators;
189        uint256[] memory amounts;
190        Campaign memory campaign;
191        (donators, amounts) = getDonatorsAndAmounts();
192        return (donators, amounts);
193    }
194
195    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
196        address[] memory donators;
197        uint256[] memory amounts;
198        Campaign memory campaign;
199        (donators, amounts) = getDonatorsAndAmounts();
200        return (donators, amounts);
201    }
202
203    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
204        address[] memory donators;
205        uint256[] memory amounts;
206        Campaign memory campaign;
207        (donators, amounts) = getDonatorsAndAmounts();
208        return (donators, amounts);
209    }
210
211    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
212        address[] memory donators;
213        uint256[] memory amounts;
214        Campaign memory campaign;
215        (donators, amounts) = getDonatorsAndAmounts();
216        return (donators, amounts);
217    }
218
219    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
220        address[] memory donators;
221        uint256[] memory amounts;
222        Campaign memory campaign;
223        (donators, amounts) = getDonatorsAndAmounts();
224        return (donators, amounts);
225    }
226
227    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
228        address[] memory donators;
229        uint256[] memory amounts;
230        Campaign memory campaign;
231        (donators, amounts) = getDonatorsAndAmounts();
232        return (donators, amounts);
233    }
234
235    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
236        address[] memory donators;
237        uint256[] memory amounts;
238        Campaign memory campaign;
239        (donators, amounts) = getDonatorsAndAmounts();
240        return (donators, amounts);
241    }
242
243    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
244        address[] memory donators;
245        uint256[] memory amounts;
246        Campaign memory campaign;
247        (donators, amounts) = getDonatorsAndAmounts();
248        return (donators, amounts);
249    }
250
251    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
252        address[] memory donators;
253        uint256[] memory amounts;
254        Campaign memory campaign;
255        (donators, amounts) = getDonatorsAndAmounts();
256        return (donators, amounts);
257    }
258
259    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
260        address[] memory donators;
261        uint256[] memory amounts;
262        Campaign memory campaign;
263        (donators, amounts) = getDonatorsAndAmounts();
264        return (donators, amounts);
265    }
266
267    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
268        address[] memory donators;
269        uint256[] memory amounts;
270        Campaign memory campaign;
271        (donators, amounts) = getDonatorsAndAmounts();
272        return (donators, amounts);
273    }
274
275    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
276        address[] memory donators;
277        uint256[] memory amounts;
278        Campaign memory campaign;
279        (donators, amounts) = getDonatorsAndAmounts();
280        return (donators, amounts);
281    }
282
283    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
284        address[] memory donators;
285        uint256[] memory amounts;
286        Campaign memory campaign;
287        (donators, amounts) = getDonatorsAndAmounts();
288        return (donators, amounts);
289    }
290
291    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
292        address[] memory donators;
293        uint256[] memory amounts;
294        Campaign memory campaign;
295        (donators, amounts) = getDonatorsAndAmounts();
296        return (donators, amounts);
297    }
298
299    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
300        address[] memory donators;
301        uint256[] memory amounts;
302        Campaign memory campaign;
303        (donators, amounts) = getDonatorsAndAmounts();
304        return (donators, amounts);
305    }
306
307    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
308        address[] memory donators;
309        uint256[] memory amounts;
310        Campaign memory campaign;
311        (donators, amounts) = getDonatorsAndAmounts();
312        return (donators, amounts);
313    }
314
315    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
316        address[] memory donators;
317        uint256[] memory amounts;
318        Campaign memory campaign;
319        (donators, amounts) = getDonatorsAndAmounts();
320        return (donators, amounts);
321    }
322
323    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
324        address[] memory donators;
325        uint256[] memory amounts;
326        Campaign memory campaign;
327        (donators, amounts) = getDonatorsAndAmounts();
328        return (donators, amounts);
329    }
330
331    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
332        address[] memory donators;
333        uint256[] memory amounts;
334        Campaign memory campaign;
335        (donators, amounts) = getDonatorsAndAmounts();
336        return (donators, amounts);
337    }
338
339    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
340        address[] memory donators;
341        uint256[] memory amounts;
342        Campaign memory campaign;
343        (donators, amounts) = getDonatorsAndAmounts();
344        return (donators, amounts);
345    }
346
347    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
348        address[] memory donators;
349        uint256[] memory amounts;
350        Campaign memory campaign;
351        (donators, amounts) = getDonatorsAndAmounts();
352        return (donators, amounts);
353    }
354
355    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
356        address[] memory donators;
357        uint256[] memory amounts;
358        Campaign memory campaign;
359        (donators, amounts) = getDonatorsAndAmounts();
360        return (donators, amounts);
361    }
362
363    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
364        address[] memory donators;
365        uint256[] memory amounts;
366        Campaign memory campaign;
367        (donators, amounts) = getDonatorsAndAmounts();
368        return (donators, amounts);
369    }
370
371    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
372        address[] memory donators;
373        uint256[] memory amounts;
374        Campaign memory campaign;
375        (donators, amounts) = getDonatorsAndAmounts();
376        return (donators, amounts);
377    }
378
379    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
380        address[] memory donators;
381        uint256[] memory amounts;
382        Campaign memory campaign;
383        (donators, amounts) = getDonatorsAndAmounts();
384        return (donators, amounts);
385    }
386
387    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
388        address[] memory donators;
389        uint256[] memory amounts;
390        Campaign memory campaign;
391        (donators, amounts) = getDonatorsAndAmounts();
392        return (donators, amounts);
393    }
394
395    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
396        address[] memory donators;
397        uint256[] memory amounts;
398        Campaign memory campaign;
399        (donators, amounts) = getDonatorsAndAmounts();
400        return (donators, amounts);
401    }
402
403    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
404        address[] memory donators;
405        uint256[] memory amounts;
406        Campaign memory campaign;
407        (donators, amounts) = getDonatorsAndAmounts();
408        return (donators, amounts);
409    }
410
411    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
412        address[] memory donators;
413        uint256[] memory amounts;
414        Campaign memory campaign;
415        (donators, amounts) = getDonatorsAndAmounts();
416        return (donators, amounts);
417    }
418
419    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
420        address[] memory donators;
421        uint256[] memory amounts;
422        Campaign memory campaign;
423        (donators, amounts) = getDonatorsAndAmounts();
424        return (donators, amounts);
425    }
426
427    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
428        address[] memory donators;
429        uint256[] memory amounts;
430        Campaign memory campaign;
431        (donators, amounts) = getDonatorsAndAmounts();
432        return (donators, amounts);
433    }
434
435    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
436        address[] memory donators;
437        uint256[] memory amounts;
438        Campaign memory campaign;
439        (donators, amounts) = getDonatorsAndAmounts();
440        return (donators, amounts);
441    }
442
443    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
444        address[] memory donators;
445        uint256[] memory amounts;
446        Campaign memory campaign;
447        (donators, amounts) = getDonatorsAndAmounts();
448        return (donators, amounts);
449    }
450
451    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
452        address[] memory donators;
453        uint256[] memory amounts;
454        Campaign memory campaign;
455        (donators, amounts) = getDonatorsAndAmounts();
456        return (donators, amounts);
457    }
458
459    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
460        address[] memory donators;
461        uint256[] memory amounts;
462        Campaign memory campaign;
463        (donators, amounts) = getDonatorsAndAmounts();
464        return (donators, amounts);
465    }
466
467    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
468        address[] memory donators;
469        uint256[] memory amounts;
470        Campaign memory campaign;
471        (donators, amounts) = getDonatorsAndAmounts();
472        return (donators, amounts);
473    }
474
475    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
476        address[] memory donators;
477        uint256[] memory amounts;
478        Campaign memory campaign;
479        (donators, amounts) = getDonatorsAndAmounts();
480        return (donators, amounts);
481    }
482
483    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
484        address[] memory donators;
485        uint256[] memory amounts;
486        Campaign memory campaign;
487        (donators, amounts) = getDonatorsAndAmounts();
488        return (donators, amounts);
489    }
490
491    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
492        address[] memory donators;
493        uint256[] memory amounts;
494        Campaign memory campaign;
495        (donators, amounts) = getDonatorsAndAmounts();
496        return (donators, amounts);
497    }
498
499    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
500        address[] memory donators;
501        uint256[] memory amounts;
502        Campaign memory campaign;
503        (donators, amounts) = getDonatorsAndAmounts();
504        return (donators, amounts);
505    }
506
507    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
508        address[] memory donators;
509        uint256[] memory amounts;
510        Campaign memory campaign;
511        (donators, amounts) = getDonatorsAndAmounts();
512        return (donators, amounts);
513    }
514
515    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
516        address[] memory donators;
517        uint256[] memory amounts;
518        Campaign memory campaign;
519        (donators, amounts) = getDonatorsAndAmounts();
520        return (donators, amounts);
521    }
522
523    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
524        address[] memory donators;
525        uint256[] memory amounts;
526        Campaign memory campaign;
527        (donators, amounts) = getDonatorsAndAmounts();
528        return (donators, amounts);
529    }
530
531    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
532        address[] memory donators;
533        uint256[] memory amounts;
534        Campaign memory campaign;
535        (donators, amounts) = getDonatorsAndAmounts();
536        return (donators, amounts);
537    }
538
539    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
540        address[] memory donators;
541        uint256[] memory amounts;
542        Campaign memory campaign;
543        (donators, amounts) = getDonatorsAndAmounts();
544        return (donators, amounts);
545    }
546
547    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
548        address[] memory donators;
549        uint256[] memory amounts;
550        Campaign memory campaign;
551        (donators, amounts) = getDonatorsAndAmounts();
552        return (donators, amounts);
553    }
554
555    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
556        address[] memory donators;
557        uint256[] memory amounts;
558        Campaign memory campaign;
559        (donators, amounts) = getDonatorsAndAmounts();
560        return (donators, amounts);
561    }
562
563    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
564        address[] memory donators;
565        uint256[] memory amounts;
566        Campaign memory campaign;
567        (donators, amounts) = getDonatorsAndAmounts();
568        return (donators, amounts);
569    }
570
571    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
572        address[] memory donators;
573        uint256[] memory amounts;
574        Campaign memory campaign;
575        (donators, amounts) = getDonatorsAndAmounts();
576        return (donators, amounts);
577    }
578
579    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
580        address[] memory donators;
581        uint256[] memory amounts;
582        Campaign memory campaign;
583        (donators, amounts) = getDonatorsAndAmounts();
584        return (donators, amounts);
585    }
586
587    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
588        address[] memory donators;
589        uint256[] memory amounts;
590        Campaign memory campaign;
591        (donators, amounts) = getDonatorsAndAmounts();
592        return (donators, amounts);
593    }
594
595    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
596        address[] memory donators;
597        uint256[] memory amounts;
598        Campaign memory campaign;
599        (donators, amounts) = getDonatorsAndAmounts();
600        return (donators, amounts);
601    }
602
603    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
604        address[] memory donators;
605        uint256[] memory amounts;
606        Campaign memory campaign;
607        (donators, amounts) = getDonatorsAndAmounts();
608        return (donators, amounts);
609    }
610
611    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
612        address[] memory donators;
613        uint256[] memory amounts;
614        Campaign memory campaign;
615        (donators, amounts) = getDonatorsAndAmounts();
616        return (donators, amounts);
617    }
618
619    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
620        address[] memory donators;
621        uint256[] memory amounts;
622        Campaign memory campaign;
623        (donators, amounts) = getDonatorsAndAmounts();
624        return (donators, amounts);
625    }
626
627    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
628        address[] memory donators;
629        uint256[] memory amounts;
630        Campaign memory campaign;
631        (donators, amounts) = getDonatorsAndAmounts();
632        return (donators, amounts);
633    }
634
635    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
636        address[] memory donators;
637        uint256[] memory amounts;
638        Campaign memory campaign;
639        (donators, amounts) = getDonatorsAndAmounts();
640        return (donators, amounts);
641    }
642
643    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
644        address[] memory donators;
645        uint256[] memory amounts;
646        Campaign memory campaign;
647        (donators, amounts) = getDonatorsAndAmounts();
648        return (donators, amounts);
649    }
650
651    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
652        address[] memory donators;
653        uint256[] memory amounts;
654        Campaign memory campaign;
655        (donators, amounts) = getDonatorsAndAmounts();
656        return (donators, amounts);
657    }
658
659    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
660        address[] memory donators;
661        uint256[] memory amounts;
662        Campaign memory campaign;
663        (donators, amounts) = getDonatorsAndAmounts();
664        return (donators, amounts);
665    }
666
667    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
668        address[] memory donators;
669        uint256[] memory amounts;
670        Campaign memory campaign;
671        (donators, amounts) = getDonatorsAndAmounts();
672        return (donators, amounts);
673    }
674
675    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
676        address[] memory donators;
677        uint256[] memory amounts;
678        Campaign memory campaign;
679        (donators, amounts) = getDonatorsAndAmounts();
680        return (donators, amounts);
681    }
682
683    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
684        address[] memory donators;
685        uint256[] memory amounts;
686        Campaign memory campaign;
687        (donators, amounts) = getDonatorsAndAmounts();
688        return (donators, amounts);
689    }
690
691    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
692        address[] memory donators;
693        uint256[] memory amounts;
694        Campaign memory campaign;
695        (donators, amounts) = getDonatorsAndAmounts();
696        return (donators, amounts);
697    }
698
699    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
700        address[] memory donators;
701        uint256[] memory amounts;
702        Campaign memory campaign;
703        (donators, amounts) = getDonatorsAndAmounts();
704        return (donators, amounts);
705    }
706
707    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
708        address[] memory donators;
709        uint256[] memory amounts;
710        Campaign memory campaign;
711        (donators, amounts) = getDonatorsAndAmounts();
712        return (donators, amounts);
713    }
714
715    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
716        address[] memory donators;
717        uint256[] memory amounts;
718        Campaign memory campaign;
719        (donators, amounts) = getDonatorsAndAmounts();
720        return (donators, amounts);
721    }
722
723    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
724        address[] memory donators;
725        uint256[] memory amounts;
726        Campaign memory campaign;
727        (donators, amounts) = getDonatorsAndAmounts();
728        return (donators, amounts);
729    }
730
731    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
732        address[] memory donators;
733        uint256[] memory amounts;
734        Campaign memory campaign;
735        (donators, amounts) = getDonatorsAndAmounts();
736        return (donators, amounts);
737    }
738
739    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
740        address[] memory donators;
741        uint256[] memory amounts;
742        Campaign memory campaign;
743        (donators, amounts) = getDonatorsAndAmounts();
744        return (donators, amounts);
745    }
746
747    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
748        address[] memory donators;
749        uint256[] memory amounts;
750        Campaign memory campaign;
751        (donators, amounts) = getDonatorsAndAmounts();
752        return (donators, amounts);
753    }
754
755    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
756        address[] memory donators;
757        uint256[] memory amounts;
758        Campaign memory campaign;
759        (donators, amounts) = getDonatorsAndAmounts();
760        return (donators, amounts);
761    }
762
763    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
764        address[] memory donators;
765        uint256[] memory amounts;
766        Campaign memory campaign;
767        (donators, amounts) = getDonatorsAndAmounts();
768        return (donators, amounts);
769    }
770
771    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
772        address[] memory donators;
773        uint256[] memory amounts;
774        Campaign memory campaign;
775        (donators, amounts) = getDonatorsAndAmounts();
776        return (donators, amounts);
777    }
778
779    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
780        address[] memory donators;
781        uint256[] memory amounts;
782        Campaign memory campaign;
783        (donators, amounts) = getDonatorsAndAmounts();
784        return (donators, amounts);
785    }
786
787    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
788        address[] memory donators;
789        uint256[] memory amounts;
790        Campaign memory campaign;
791        (donators, amounts) = getDonatorsAndAmounts();
792        return (donators, amounts);
793    }
794
795    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
796        address[] memory donators;
797        uint256[] memory amounts;
798        Campaign memory campaign;
799        (donators, amounts) = getDonatorsAndAmounts();
800        return (donators, amounts);
801    }
802
803    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
804        address[] memory donators;
805        uint256[] memory amounts;
806        Campaign memory campaign;
807        (donators, amounts) = getDonatorsAndAmounts();
808        return (donators, amounts);
809    }
810
811    function getDonatorsAndAmounts() public view returns (address[] memory, uint256[] memory) {
812        address[] memory donators;
813        uint256[] memory amounts;
814        Campaign memory campaign;
815        (donators, amounts) = getDonatorsAndAmounts();
816        return (donators, amounts
```

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

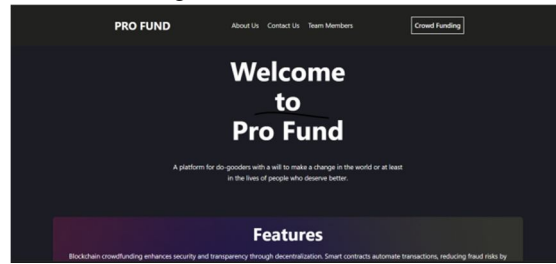


Fig-6: Front-end website page

In the Contact Us page, there is an email JS where you can send us the email and we will respond to your query.

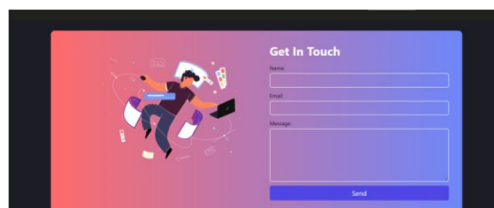


Fig-7: Contact Us page

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

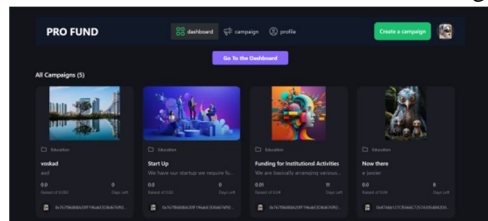


Fig-8: Crowd Funding Page

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

- 1) **Dashboard:** Where you are able to 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 for your organization.

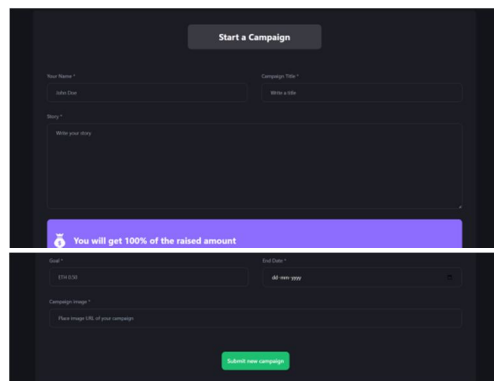


Fig-9: Create Campaign Page

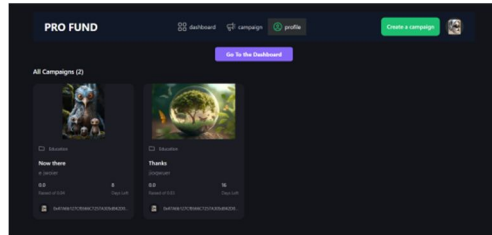


Fig-10: Profile Page

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.

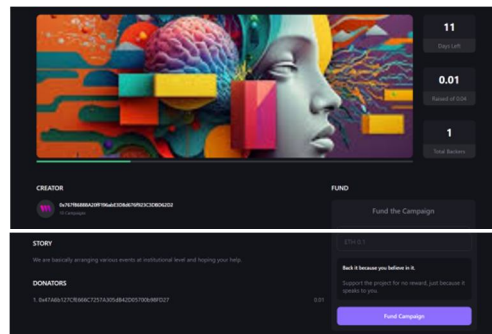


Fig-11: Funding Page

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. "Learing 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)