A

Project Report

On

TRANSPARENT FUNDING PLATFORM USING BLOCKCHAIN

Submitted for partial fulfilment of the requirements for the award of the degree of

BACHELOR OF ENGINEERING

In

COMPUTER SCIENCE AND ENGINEERING

By

Kannemolla Santhosh (2451-20-733-135)

Jatavath Jayanth (2451-20-733-137)

Under the guidance of

P. Subhashini

Assistant Professor

Department of CSE

Maturi Venkata Subba Rao (MVSR) ENGINEERING COLLEGE
Department of Computer Science and Engineering
(Affiliated to Osmania University & Recognized by AICTE)
Nadergul, Saroor Nagar Mandal, R.R Dist. Hyderabad – 501510
Academic Year: 2023 - 24

Maturi Venkata Subba Rao Engineering College

(Affiliated to Osmania University, Hyderabad)

Nadergul(V), Hyderabad-501510

CERTIFICATE

This is to certify that the project work entitled **Transparent Funding Platform using Blockchain** *is a bonafide work carried out by* ***Mr. Kannemolla Santhosh (2451-20-733-135), Mr. Jatavath Jayanth (2451-20-733-137)*** *in partial fulfilment of the requirements for the award of degree of* ***Bachelor of Engineering*** *in* ***Computer Science and Engineering*** *from* ***Maturi Venkata Subba Rao (MVSR) Engineering College,*** *affiliated to OSMANIA UNIVERSITY, Hyderabad, during the Academic Year 2023-24 under our guidance and supervision.*

The results embodied in this report have not been submitted to any other university or institute for the award of any degree or diploma to the best of our knowledge and belief.

Internal Guide	**Head of the Department**
P. Subhashini	J. Prasanna Kumar
Assistant Professor	Professor
Department of CSE	Department of CSE
MVSREC.	MVSREC.

External Examiner

DECLARATION

This is to certify that the work reported in the present project entitled "**Transparent Funding Platform using Blockchain"** is a record of bonafide work done by us in the Department of Computer Science and Engineering, Maturi Venkata Subba Rao (MVSR) Engineering College, Osmania University during the Academic Year 2023-24. The reports are based on the project work done entirely by us and not copied from any other source. The results embodied in this project report have not been submitted to any other University or Institute for the award of any degree or diploma.

Kannemolla Santhosh — 2451-20-733-135

Jatavath Jayanth — 2451-20-733-137

ACKNOWLEDGEMENTS

We would like to express our sincere gratitude and indebtedness to our major-project guide **P. Subhashini** for her valuable suggestions and interest throughout the course of this project.

We are also thankful to our principal **Dr. Vijaya Gunturu** and **J. Prasanna Kumar,** Professor and Head, Department of Computer Science and Engineering, Maturi Venkata Subba Rao Engineering College, Hyderabad for providing excellent infrastructure for completing this project successfully as a part of our B.E. Degree (CSE). We would like to thank our project coordinator **Mr. K. Murali Krishna** for his constant monitoring, guidance and support.

We convey our heartfelt thanks to the lab staff for allowing us to use the required equipment whenever needed. We sincerely acknowledge and thank all those who gave directly or indirectly their support in completion of this work.

VISION

- To impart technical education of the highest standards, producing competent and confident engineers with an ability to use computer science knowledge to solve societal problems.

MISSION

- To make learning process exciting, stimulating and interesting.
- To impart adequate fundamental knowledge and soft skills to students.
- To expose students to advanced computer technologies in order to excel in engineering practices by bringing out the creativity in students.
- To develop economically feasible and socially acceptable software.

PEOs:

PEO-1: Achieve recognition through demonstration of technical competence for successful execution of software projects to meet customer business objectives..

PEO-2: Practice life-long learning by pursuing professional certifications, higher education or research in the emerging areas of information processing and intelligent systems at a global level.

PEO-3: Contribute to society by understanding the impact of computing using a multidisciplinary and ethical approach.

PROGRAM OUTCOMES (POs)

At the end of the program the students (Engineering Graduates) will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified

needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and the need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions**.**
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Lifelong learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

13. (PSO-1) Demonstrate competence to build effective solutions for computational real-world problems using software and hardware across multi-disciplinary domains.
14. (PSO-2) Adapt to current computing trends for meeting the industrial and societal needs through a holistic professional development leading to pioneering careers or entrepreneurship.

COURSE OBJECTIVES AND OUTCOMES

Course Code: PW861 CS

Course Objectives

- To enhance practical and professional skills.
- To familiarize tools and techniques of systematic literature survey and documentation
- To expose the students to industry practices and teamwork.
- To encourage students to work with innovative and entrepreneurial ideas

Course Outcomes

CO1: Summarize the survey of the recent advancements to infer the problem statements with applications towards society

CO2: Design a software-based solution within the scope of the project.

CO3: Implement test and deploy using contemporary technologies and tools

CO4: Demonstrate qualities necessary for working in a team.

CO5: Generate a suitable technical document for the project.

ABSTRACT

The project focuses on the development of a decentralized transparent funding platform leveraging blockchain technology. A blockchain-based transparent funding platform holds immense potential to revolutionize the funding industry by introducing unprecedented levels of efficiency, transparency, and security. By leveraging blockchain technology, this platform can provide a decentralized and tamper-proof environment for funding activities. This decentralization ensures that no single entity has control over the platform, mitigating the risks of fraud or misuse of funds. Additionally, the immutable nature of blockchain ensures that transaction records cannot be altered or manipulated, enhancing transparency and accountability.

By democratizing access to funds, the platform aims to make funding more inclusive and accessible to a wider audience. The implementation of smart contracts on the blockchain transparent funding platform further enhances its efficiency and security. Smart contracts enable automated execution of funding agreements, ensuring that funds are released only when predefined conditions are met. This automation reduces the need for intermediaries and minimizes the potential for errors or disputes. Investors, donors, project creators, and fundraisers can all benefit from the transparent and secure environment provided by blockchain technology. This increased trust and confidence among participants can lead to greater participation in funding campaigns, ultimately fostering innovation and driving positive social and economic impact.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

The project is a decentralized fundraising platform leveraging blockchain technology to enable transparent, secure, and efficient fundraising campaigns. Built on the Ethereum network, the platform utilizes smart contracts to automate the process of creating and managing fundraising campaigns, ensuring that funds are collected and distributed in a transparent manner. Users can create campaigns for various causes such as education, health, and animal welfare, detailing their goals and required amounts. The decentralized nature of the platform eliminates the need for intermediaries, reducing costs and increasing trust among donors and campaigners.

The platform also integrates with Pinata for IPFS (Inter Planetary File System) storage, allowing users to upload and store campaign-related documents and images securely and immutably. By utilizing IPFS, the platform ensures that all campaign materials are accessible and tamper-proof, enhancing the overall credibility and reliability of the campaigns. Additionally, the platform incorporates a seamless wallet connection feature, enabling users to connect their cryptocurrency wallets, manage their balances, and make donations directly through the platform. This integration not only simplifies the donation process but also ensures that all transactions are recorded on the blockchain, providing complete transparency and traceability.

1.1 PROBLEM STATEMENT

In today's rapidly evolving technological landscape Crowdfunding has become one of the most popular methods for raising funds for projects, causes, or individuals in need. The onset of COVID-19 has amplified this trend, with a surge in crowdfunding activities globally. These range from small campaigns to provide oxygen and medical assistance to large-scale funds like PM Cares.

However, current crowdfunding platforms face significant issues that need addressing. One major problem is security. As fund sizes grow, so does the need for robust security measures. Although symmetric encryption is used to secure e-payments, vulnerabilities to hacking still exist. Blockchain technology, which has never been compromised, offers a higher level of security, ensuring that funds remain safe from cyber threats. Another critical issue is transparency and fraud prevention. Crowdfunding scams are a persistent problem, with no reliable way to track how funds

are utilized. Our goal is to make the entire flow of funds transparent at every stage using blockchain, eliminating any possibility of misuse. This transparency builds trust among contributors and ensures that funds are used as intended.

Global contribution is often restricted by country-specific platforms, limiting international participation. Blockchain enables anyone worldwide to contribute to campaigns, making transactions quick and convenient. This global reach democratizes crowdfunding, allowing for broader and more diverse support for campaigns, regardless of geographic boundaries.

1.2 OBJECTIVE

The primary objective of the project is to develop Transparent Funding Platform, a decentralized, transparent funding platform leveraging Ethereum blockchain technology to overcome the limitations of current centralized funding systems. Transparent Funding Platform aims to create a transparent environment where all transactions are visible and immutable, building trust and reducing fraud. By eliminating intermediaries and utilizing smart contracts, the platform significantly reduces delays and costs associated with traditional funding methods, enhancing efficiency. Transparent Funding Platform democratizes access to funding, removing restrictive participation requirements and enabling individuals and organizations, especially those in developing countries or with limited resources, to engage in crowdfunding activities.

Key features of the proposed decentralized funding system leverages blockchain technology to address limitations of existing centralized models, prioritizing transparency, efficiency, and accessibility in fundraising. Blockchain ensures transactions are transparent and tamper-proof, allowing stakeholders to track fund usage accurately. Transparent Funding Platform aims to create a building trust and reducing fraud. By eliminating intermediaries, the system reduces inefficiencies, enabling faster and more cost-effective transactions. Blockchain enables anyone worldwide to contribute to campaigns, making transactions quick and convenient. Its decentralized nature removes restrictive participation requirements, broadening access for individuals and organizations, including those in developing countries or with limited resources. Overall, this system revolutionizes fundraising by providing a transparent, efficient, and inclusive platform for all participants.

1.3 MOTIVATION

The motivation behind developing the Transparent Funding Platform stems from the need to address significant challenges associated with traditional centralized funding systems. These challenges include a lack of transparency, inefficiencies, security risks, restricted accessibility, and fraud prevention issues. Centralized systems often fail to provide clear tracking of funds, leading to mistrust among stakeholders. They involve multiple intermediaries, causing delays and higher costs, and remain vulnerable to hacking and fraud despite existing security measures. Moreover, stringent participation requirements limit access for many, particularly in developing countries.

The Transparent Funding Platform leverages blockchain technology to overcome these challenges by offering enhanced fund tracking through a transparent, tamper-proof ledger, streamlined transactions with smart contracts, and robust security with immutable records and decentralized control. The platform's decentralized nature removes geographical barriers, fostering global reach and inclusivity, and democratizes access by eliminating restrictive requirements, supporting a more equitable distribution of resources. The transparency and immutability of blockchain transactions aid in detecting and preventing fraud, ensuring funds are used as intended. By addressing these critical issues, the Transparent Funding Platform aims to create a more trustworthy, efficient, and inclusive crowdfunding environment, encouraging greater participation and support for diverse global projects and causes.

In addition to addressing the foundational challenges, the Transparent Funding Platform offers several advanced features that set it apart. The platform integrates a user-friendly interface to facilitate ease of use for all participants, including those with minimal technical expertise The platform incorporates real-time analytics and reporting tools, enabling stakeholders to monitor campaign progress and financial health dynamically. The transparency and immutability of blockchain transactions aid in detecting and preventing fraud Furthermore, it offers customizable smart contracts, allowing fundraisers to tailor funding agreements to specific needs and conditions, thus enhancing flexibility and user satisfaction. With these additional features, the Transparent Funding Platform not only addresses the inherent flaws of traditional funding systems but also provides a comprehensive, adaptable solution for modern fundraising, promoting greater transparency, security, and efficiency in the global crowdfunding landscape.

1.4 SCOPE

The project's scope is to develop a decentralized funding platform leveraging blockchain technology, notably Ethereum, and smart contracts. This platform is designed to redefine crowdfunding by emphasizing transparency, efficiency, accessibility, and security while ensuring compliance with regulatory standards. By harnessing the decentralized ledger capabilities of Ethereum, the platform aims to create an immutable record of transactions, fostering trust among stakeholders. Through smart contracts, funding agreements are automated, streamlining processes and minimizing errors.

Input encompasses the utilization of Ethereum blockchain for secure and transparent transactions. Smart contracts play a pivotal role in automating funding agreements, enhancing the efficiency and reliability of the platform. Furthermore, user-friendly interfaces tailored for investors and beneficiaries are developed, prioritizing intuitive navigation and inclusivity. The input phase sets the foundation for a robust and trustworthy crowdfunding ecosystem.

Output focuses on delivering seamless and intuitive interfaces that empower users to participate in funding campaigns effortlessly. Investors benefit from a user-friendly experience, allowing them to browse campaigns, contribute funds, and manage portfolios efficiently. Meanwhile, beneficiaries are equipped with tools to create and manage campaigns effectively, fostering community engagement and trust. The platform's inclusive design promotes global participation, enabling individuals and organizations worldwide to access funding opportunities. By prioritizing accessibility and transparency, the output phase aims to create a vibrant crowdfunding ecosystem that inspires confidence and drives positive impact.

1.5 HARDWARE REQUIREMENTS

1. Requires at least 8GB of RAM and good processor.
2. i5/ryzen5 or higher CPU
3. 2GB dedicated GPU or higher
4. Internet Connection: Stable internet connection for downloading dependencies, accessing blockchain networks, and collaborating with team members.
5. Browser - Latest version of Google Chrome, Mozilla Firefox, or Safari for testing web applications and Metamask integration.

1.6 SOFTWARE REQUIREMENTS

1. Operating System - Compatible with Windows, macOS, or Linux.
2. Development Environment
    - Node.js - Latest stable version installed.
    - Hardhat - Installed for Ethereum smart contract development.
    - Solidity Compiler - Version 0.8.10 or compatible for contract compilation.
    - Dotenv - Installed for managing environment variables.
3. Frontend Framework
    - React.js - Latest version for frontend development.
    - Styled Components - Installed for styling React components.
4. Blockchain Integration
    - Metamask - Browser extension for connecting Ethereum wallets.
5. Text Editor
    - Visual Studio Code, Sublime Text, or any preferred text editor for code editing.

CHAPTER 2

LITERATURE SURVEY

2.1 SURVEY OF MAJOR AREA

The project focuses on the development of a decentralized transparent funding platform leveraging blockchain technology. A blockchain-based transparent funding platform holds immense potential to revolutionize the funding industry by introducing unprecedented levels of efficiency, transparency, and security. By leveraging blockchain technology, this platform can provide a decentralized and tamper-proof environment for funding activities. This decentralization ensures that no single entity has control over the Transparent platform, mitigating the risks of fraud or misuse of funds. Additionally, the immutable nature of blockchain ensures that transaction records cannot be altered or manipulated, enhancing transparency and accountability. Tiganoaia and Alexandru's study explores the integration of blockchain technology in decentralized crowdfunding platforms tailored for social and educational causes, particularly in the context of sustainable development. The paper emphasizes the use of smart contracts and Polygon's scalability to ensure transparency, accountability, and efficiency in donation processes. By leveraging blockchain technology, the platform addresses security challenges inherent in traditional crowdfunding models, such as fund misappropriation and lack of transparency. Furthermore, the authors highlight the transformative potential of blockchain-based crowdfunding in advancing Sustainable Development Goals (SDGs) and facilitating charitable donations. The study provides valuable insights into the application of blockchain in socially impactful ventures, shedding light on the role of decentralized finance in supporting sustainable initiatives [1].

Tiwari et al.'s research focuses on the development of a transparent, distributed, and secure crowdfunding platform powered by blockchain technology. The study underscores the importance of transparency, accountability, and security in crowdfunding processes, particularly in mitigating risks associated with fund diversion and fraudulent activities. By leveraging blockchain's decentralized architecture, the platform enhances accountability and reduces the potential for monopolistic control, thereby fostering improved scalability and inclusivity. The paper presents a robust framework for blockchain-based crowdfunding, highlighting its potential to revolutionize fundraising practices and promote greater trust and confidence among

stakeholders. The findings underscore the significance of blockchain in reshaping the crowdfunding landscape, offering a compelling alternative to traditional fundraising mechanisms [2].

Pawar et al.'s study explores the integration of blockchain technology, particularly Ethereum smart contracts, in crowdfunding initiatives. The research highlights blockchain's potential to address traditional funding challenges by offering enhanced transparency, security, and efficiency in fundraising processes. By leveraging Ethereum's smart contract capabilities, the platform automates key aspects of crowdfunding, including transaction verification, fund allocation, and project governance. The paper underscores the transformative impact of blockchain-based crowdfunding in streamlining fundraising operations and reducing administrative overheads. Additionally, the study emphasizes the role of blockchain in fostering trust and accountability among project stakeholders, thereby enhancing investor confidence and participation in crowdfunding campaigns [3].

Chethan et al.'s paper introduces a decentralized application (Dapp) for crowdfunding large projects, emphasizing secure and transparent decision-making processes. The study highlights blockchain's role in facilitating trust and accountability in crowdfunding initiatives, thereby addressing common challenges such as information asymmetry and fund misallocation. By leveraging blockchain's immutable ledger and smart contract functionality, the platform ensures transparent governance and efficient fund management, ultimately enhancing investor trust and confidence. The findings underscore the potential of blockchain-based crowdfunding platforms to revolutionize traditional fundraising practices, offering a more democratic and inclusive approach to project financing [4].

Kumbharkar et al.'s study explores the use of blockchain technology in crowdfunding initiatives, focusing on addressing key challenges and enhancing platform security. The research emphasizes blockchain's decentralized architecture and cryptographic algorithms in ensuring the integrity and confidentiality of crowdfunding transactions. By leveraging cryptographic techniques such as SHA-256 and proof-of-stake (PoS) consensus mechanisms, the platform enhances data security and prevents unauthorized access to sensitive information. The study provides valuable insights into the technical aspects of blockchain-based crowdfunding, offering practical solutions for entrepreneurs, investors, and developers looking to leverage blockchain technology in fundraising ventures [5].

The article proposes a decision-making platform for large-scale business projects, addressing funding challenges, trust issues, and decision-making transparency. Utilizing Ethereum smart contracts, the platform enhances security, reliability, and community involvement in project investments. By leveraging blockchain technology, the platform ensures transparent governance and efficient fund management, ultimately enhancing investor trust and confidence. The findings underscore the transformative potential of blockchain-based crowdfunding platforms in reshaping traditional fundraising practices and promoting greater transparency, accountability, and inclusivity [6]. The referenced papers collectively highlight the growing significance of blockchain technology in revolutionizing crowdfunding practices. By offering enhanced transparency, security, and efficiency, blockchain-based crowdfunding platforms present a compelling alternative to traditional fundraising mechanisms, offering greater trust, confidence, and inclusivity for project stakeholders. Moving forward, continued research and development in this field are essential to unlocking the full potential of blockchain technology in supporting sustainable initiatives and driving positive social impact.

Here is the summary about the papers referred in the Table 2.1

Table 2.1 Literature Survey Summary.

SNo	Title	Authors	Published by	Published on	Paper Analysis
1	Building a Blockchain-Based Decentralized Crowdfunding Platform for Social and Educational Causes in the Context of Sustainable Devel opment	Bogdan Tiganoaia and George-Madalin Alexandru	MDPI	22 NOV 2023	This paper explores blockchain integration in a decentralized web3 app for transparent educational donations. Highlighting smart contracts and Polygon's scalability, it

					addresses security challenges, emphasizing the transformative impact on SDGs and charitable donatio ns.
2	A Transparent, Distributed & Secure Crowdfunding platform based on Blockchain	Vinita Tiwari, Sam Goundar, Karri Babu Ravi, Basant Agarwal Priyanka H	Springer, Research gate	4 OCT 2023	Blockchain-powered crowdfunding ensures transparency, accountability, and security, addressing fund diversion concerns. Decentralization enhances accountability and counters monopoly, fostering improved business scalability.
3	Blockchain based Crowdfunding using Ethereum Smart Contract	Dipali Pawar, Sanket Sangle, Aditya Muley,	IJRASET	MAY 2023	The paper explores Blockchain-based crowdfunding's potential in addressing

		Siddhesh, Ruhinaaz Shaikh			traditional funding challenges, emphasizing transparency, security, and efficiency. Integration with Ethereum smart contracts revolutionizes fundraising.
4	Fund Crypt using Sha-256 & pos alg	P.B. Kumbharkar, Geetank Asati, Aditya Valekar, Rushikesh Palaskar, Satyam Yenegure	IEEE	19 JUL 2023	Study explores blockchain in crowdfunding, addressing challenges. Blockchain's decentralization, transparency, and smart contracts enhance crowdfunding. Valuable insights for entrepreneurs, investors, and developers.
5	Crowdfunding Dapp	Chethan S, Rohan P, Roopashree CS	IRJMETS	7 JUL 2023	Paper introduces blockchain crowdfunding for large projects, emphasizing

					secure, transparent decision-making. Positive survey results affirm platform's potential to enhance transparency, trust, and security.
6	Blockchain-based Distributed Secure Crowdfunding and Decision-MakingPlatform for Large-scale Business Projects in Public and Private Sectors	Malithi mithsara, T M K K Jinnasena	Research gate	SEP 2020	The article proposes a decision-making platform for large-scale business projects. It addresses funding challenges, trust issues, and decision-making transparency, utilizing Ethereum smart contracts to enhance security, reliability, and community involvement in project investments.

2.2 SOFTWARE TECHNOLOGIES

1. **Next.js:** Next.js is a React framework for building server-rendered applications. It allows for easy setup of server-side rendering, static site generation, and more, providing a robust foundation for building web applications.
2. **Styled-components:** Styled-components is a CSS-in-JS library that allows developers to write CSS directly inside JavaScript files using template literals. It provides scoped styles and encourages component-based styling, making it easier to manage styles in React applications.
3. **ethers.js:** ether.js is a JavaScript library for interacting with the Ethereum blockchain. It provides a simple and easy-to-use API for working with Ethereum smart contracts, transactions, and more. In the provided code, ethers.js is used to interact with smart contracts deployed on the Ethereum network.
4. **JSON-RPC Provider:** JSON-RPC Provider is a remote procedure call protocol encoded in JSON. In the code, a JSON-RPC provider is used to connect to an Ethereum node to interact with the Ethereum blockchain. It enables communication with the blockchain network to read data and execute transactions.
5. **Solidity:** Solidity is a high-level programming language used to write smart contracts on the Ethereum blockchain. Smart contracts are self-executing contracts with the terms of the agreement directly written into code. In the code, smart contracts are written in Solidity to implement the crowdfunding functionality.
6. **Pinata Cloud:** Pinata Cloud is a platform for hosting and managing files on the Inter Planetary File System (IPFS), a decentralized storage network. In the code, Pinata Cloud is used to host files, such as images and story content, on IPFS and retrieve them for display in the web application.
7. **React:** React is a JavaScript library for building user interfaces. It allows developers to create reusable UI components and efficiently update the UI in response to changes in data. The provided code uses React for building the frontend of the crowdfunding platform, including components for displaying project details, accepting the user donations, and showing recent donations to the campaign. These technologies collectively enable the development of a decentralized crowdfunding platform that leverages blockchain technology for transparency, security, and immutability.

CHAPTER 3

SYSTEM DESIGN

3.1 SYSTEM ARCHITECTURE

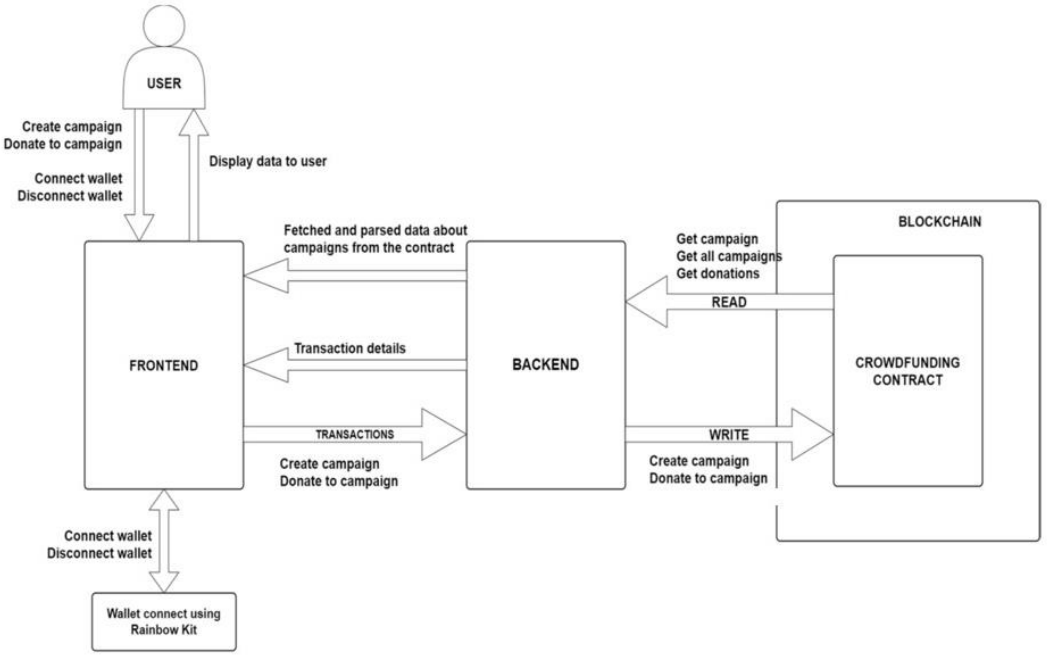

Fig 3.1 System Architecture

The system architecture of the crowdfunding platform is designed to seamlessly integrate blockchain technology, providing a secure and transparent environment for users to create and donate to campaigns. At the core of this system is the interaction between various components: the user interface (frontend), the backend server, the blockchain, and the crowdfunding smart contract.

The User component is the entry point of interaction. Users perform actions such as creating campaigns, making donations, and managing their cryptocurrency wallets. These interactions are facilitated through a user-friendly frontend interface. A toolkit called Rainbow Kit is used to manage wallet connections, allowing users to securely connect and disconnect their cryptocurrency wallets. This ensures a seamless experience for users as they engage with the platform.

The Frontend is the user interface that presents the platform's functionality to users. It handles actions such as campaign creation, donations, and wallet management. The frontend communicates with the backend to fetch and display campaign data and sends transaction details when users create campaigns or make donations. This layer is crucial for providing a responsive and interactive user experience.

The Backend serves as the intermediary between the frontend and the blockchain. It processes requests from the frontend, such as fetching campaign data or processing transactions. When a user initiates a transaction (e.g., creating a campaign or making a donation), the backend communicates with the crowdfunding contract on the blockchain to execute these operations. It also parses data from the blockchain and sends it back to the frontend to be displayed to users. The backend's role is pivotal in managing the business logic and ensuring the accurate execution of transactions.

The Blockchain hosts the crowdfunding smart contract, which is the backbone of the platform's core functionality. The smart contract includes methods for creating campaigns, processing donations, and fetching campaign data. These methods enable both read and write operations on the blockchain, ensuring the integrity and transparency of campaign and donation data. By leveraging blockchain technology, the platform guarantees that all transactions are secure and immutable.

The Crowdfunding Contract is a specialized smart contract deployed on the blockchain. It manages the creation of campaigns and the processing of donations. This contract stores all campaign data and tracks donations, making it accessible for read operations by the backend. The smart contract ensures that all transactions are executed in a trustless manner, providing transparency and security for all parties involved.

3.2 SYSTEM MODULES

1. **CampaignFactory**: This module represents a smart contract deployed on the Ethereum blockchain. It serves as a factory for creating new instances of crowdfunding campaigns. Through the createCampaign function, users can initiate new campaigns with specified parameters such as title, required amount, image URI, and category. The contract emits an event campaignCreated upon successful creation of a campaign, providing important details transparency.

2. **CreateCampaign Function**: This function is a crucial component of the CampaignFactory module. It facilitates the creation of new crowdfunding campaigns by instantiating a new Campaign contract with the provided parameters. By invoking this function, users trigger the deployment of a new campaign contract, enabling fundraising efforts with defined objectives and characteristics.

3. **Form**: As a module, the form component plays a pivotal role in facilitating user interaction and data submission within the application. It encapsulates various

input fields, buttons, and validation logic necessary for capturing user input and transmitting it to the backend for processing. The form module may incorporate functionalities such as input validation, error handling, and submission handling.

4. **FormLeftWrapper and FormRightWrapper**: These modules likely serve as styling components for structuring the layout of forms within the user interface. They encapsulate the visual presentation and organization of form elements, contributing to a cohesive and intuitive user experience. The separation into left and right wrappers may indicate a design choice for arranging form fields or sections.

5. **Components**: This overarching module encompasses a collection of React components utilized throughout the application. Each component represents a reusable building block responsible for rendering specific UI elements or encapsulating distinct functionalities. Components within this module, such as the Header, Wallet, and potentially others, contribute to the overall structure and behavior of the user interface.

6. **Wallet**: This module represents a component dedicated to displaying wallet-related information, such as the user's address and balance. It serves as a crucial element for users engaging with blockchain-related functionalities within the application, providing transparency and accessibility to their digital assets.

7. **Header**: The header module encompasses a prominent UI component situated at the top of the application interface. It typically includes branding elements, navigation links, and potentially user authentication controls. The header contributes to the overall navigation and user experience by providing consistent access to essential functionalities across different pages or views.

8. **Layout**: This module defines the overarching layout and structure of the application's user interface. It encompasses components responsible for establishing the grid system, defining spacing and alignment, and potentially incorporating global styles or themes. The layout module ensures consistency and coherence in the visual presentation of the application's content across various screens and devices.

9. **Themes**: The themes module governs the visual appearance and styling characteristics of the application. It defines sets of predefined styles, colors,

typography, and other design elements that contribute to the overall aesthetic and branding of the application. The themes module enables easy customization and theming of the user interface, catering to diverse preferences and branding.

10. **Dashboard and Inde**x: These modules represent distinct pages or views within the application, each serving specific purposes and functionalities. The dashboard module may provide a centralized hub for users to monitor campaign progress, manage contributions, and access relevant information. The index module, often serving as the application's homepage or landing page, may offer an overview of available campaigns, featured content, or navigation options to guide users through the application experience.

Each module fulfills a unique role and contributes to the overall functionality, usability, and visual presentation of the application. By understanding the purpose and characteristics of each module, developers can effectively design, implement, and maintain a robust and user-friendly crowdfunding platform. Facilitates better communication and collaboration. By providing a clear overview of all connected clients, this module enhances transparency and coordination among teammembers, making it easier to manage collaborative efforts. The Transparent Funding Platform encompasses several key system modules to create a decentralized, transparent, and user-friendly crowdfunding experience. At its core, the CampaignFactory module leverages the Ethereum blockchain to deploy and manage campaign contracts, ensuring immutable and transparent transactions. The CreateCampaign Function facilitates the creation of new campaigns, streamlining the process for users. On the frontend, modules like FormLeftWrapper, FormRightWrapper, and Form handle the user interface, enabling intuitive data entry and submission.Additionally, the platform includes a Wallet module for displaying user wallet information, enhancing accessibility to digital assets. The Components module comprises reusable elements like the Header, which provides consistent navigation across the application. The Layout module ensures a cohesive and visually appealing user experience, supported by the Themes module that allows for customizable styling. This comprehensive approach ensures the platform is efficient, transparent, and accessible to a global audience. Together, these modules ensure the platform is efficient, transparent, and accessible to a global audience, revolutionizing the crowdfunding landscape by fostering trust and inclusivity.

3.3 FLOW DIAGRAM

Fig 3.2 Flow chart

The first image is an activity diagram illustrating the process for donating to a campaign using a wallet application. The workflow starts with the user initiating the process with the "Start" action. The first key step is to connect their wallet, which is essential for accessing and managing funds. After the wallet is connected, the user proceeds to retrieve a list of available campaigns by selecting the "Get List of Campaigns" activity. This step provides the user with various campaigns to choose from. The user then selects a specific campaign and performs the "Get Campaign Information" action to gather detailed information about the selected campaign.

This information is crucial for making an informed donation decision. The next step is to check the contract balance by executing the "Get Contract Balance" activity. This step helps the user understand the current funding status and the financial health of the campaign. The diagram then guides the user to review their individual donation

contributions per campaign through the "Get User Donation per Campaign" activity. Following this, the user views the total list of donations for the selected campaign using the "Get List of Donations per Campaign" action, which provides a comprehensive overview of all contributions. The process continues with the "Get List of Transactions per Campaign" activity, where the user can review all financial transactions related to the campaign. Finally, the user completes the donation process by executing the "Donate to Campaign" action. The workflow concludes with the "End" step, indicating the completion of the donation activity.

The second image is an activity diagram outlining the steps to create and manage a campaign using a wallet application. The process begins with the user initiating the "Start" action, followed by connecting their wallet in the "Connect Wallet" step, which is necessary for setting up and managing campaign funds. The next critical step is the "Create Campaign" activity, where the user initiates the process of setting up a new campaign. Following this, the user fills in the campaign details through the "Fill Campaign Information" action, providing essential data such as campaign goals, description, and timeline. The process continues with the "Fill Contract Balance" step, where the user allocates initial funding to the campaign. This step ensures that the campaign has the necessary financial resources to begin.

The user then reviews their individual donation contributions per campaign through the "Get User Donation per Campaign" activity. This step helps the user monitor their financial involvement. Next, the user accesses the total list of donations for the campaign by executing the "Get List of Donations per Campaign" action, providing an overview of all contributions. The process proceeds with the "Get List of Transactions per Campaign" activity, where the user reviews all financial transactions related to the campaign, ensuring transparency and accountability. Following this, the user can utilize the funds for campaign-related activities through the "Use Funds" step. This action allows the user to allocate resources towards achieving the campaign's objectives. The workflow concludes with the "End Campaign" activity, indicating the successful completion of the campaign's lifecycle. Finally, the process ends with the "End" step, marking the conclusion of all activities related to campaign creation and management. A flow chart visually represents the sequence of steps in a process, illustrating how tasks and decisions progress from start to finish. In the context of our project, it outlines user interactions, backend processing, and blockchain transactions, ensuring clarity and understanding of the system’s workflow and functionality.

3.4 UML DIAGRAMS

The Unified Modeling Language (UML) is a standardized visual language for software design. It provides a collection of diagrams to document a system's structure and behavior. In this section, we leverage UML diagrams to visually represent the design of the system. These diagrams offer a clear and concise understanding of the system's architecture, including classes, interactions, and workflows. We will be utilizing different UML diagrams such as use case diagrams, class diagrams, activity, and sequence diagrams to understand the architecture and workflow of the system.

Use Case Diagram

Use case diagrams are a set of use cases, actors, and their relationships. They represent the use case view of a system. The use case diagram is used to identify the primary elements and processes that form the system. The primary elements are termed as "actors" and the processes are called "use cases." The use case diagram shows which actors interact with each use case.

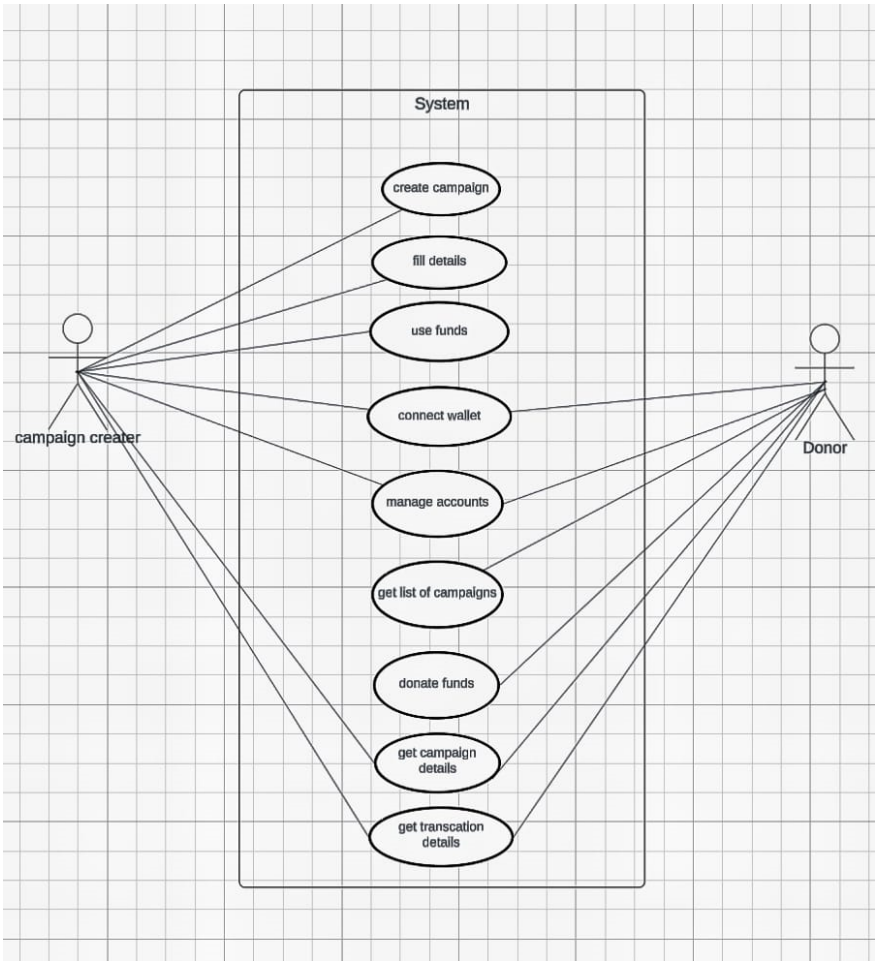

Fig 3.3 Use Case diagram

The use case diagram for the crowdfunding platform illustrates interactions between two primary actors, the Campaign Creator and the Donor, and the system. The Campaign Creator is tasked with initiating and managing crowdfunding campaigns, which involves creating a campaign, filling in necessary details like title, description, funding goal, and duration, and utilizing the funds collected. On the other hand, the Donor's role is to explore, select, and contribute funds to these campaigns. Both actors engage in several shared actions such as connecting their cryptocurrency wallets for transactions, managing their accounts by updating profiles and viewing transaction histories, and retrieving lists and detailed information about available campaigns.

The diagram highlights the sequential flow from campaign creation to fund utilization, reflecting the typical lifecycle of a crowdfunding campaign. It also emphasizes the interconnectedness of various actions, showing how foundational tasks like managing accounts and connecting wallets are essential for both campaign creation and donation processes. By detailing these interactions and functionalities, the diagram provides a comprehensive overview of the platform's operations, demonstrating the critical roles of both Campaign Creators and Donors. This inclusive approach ensures a dynamic ecosystem where both parties can efficiently engage with the platform, contributing to its overall success and user satisfaction.

Class Diagram

The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an "is-a" or "has-a" relationship. Each class in the class diagram may be capable of providing certain functionalities. These functionalities provided by the class are termed "methods" of the class.

The provided diagram illustrates the architecture of a decentralized crowdfunding platform built on Ethereum smart contracts. It includes several key components: the Campaign contract, which represents individual campaigns and manages details like the campaign's manager, minimum contribution, description, image URL, and target amount. This Class Diagram explains about how each module works and each modules functions in the project. This also describes how the each class module communicate with each other.

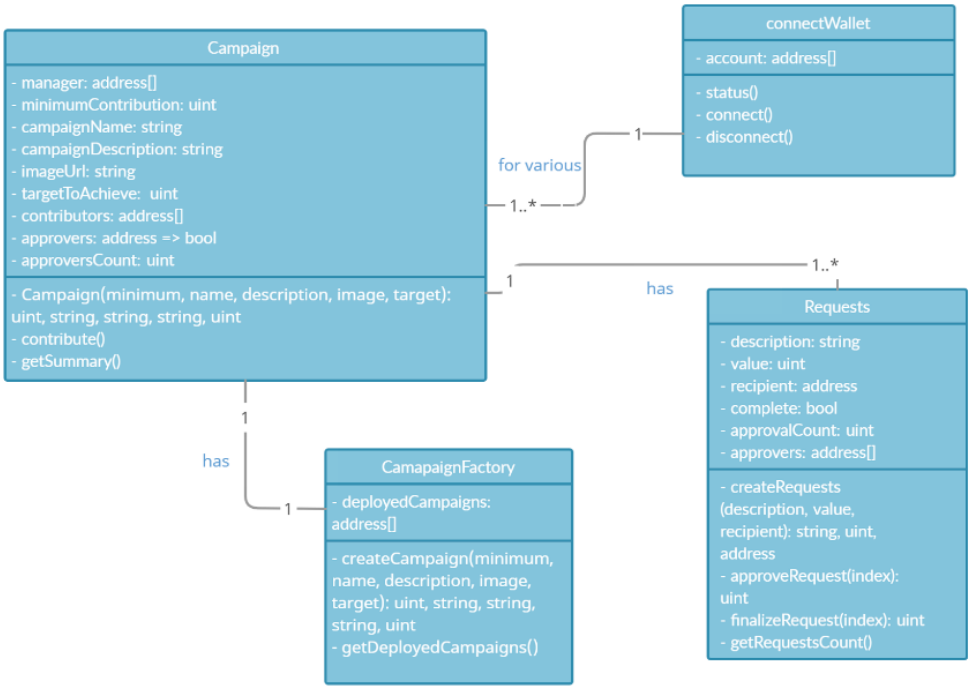

Fig 3.4 Class Diagram

The CampaignFactory contract is responsible for creating and deploying new campaign instances and maintains a list of all campaigns. The Requests module handles spending requests within campaigns, managing descriptions, values, recipients, and approval statuses, with functions to create, approve, and finalize requests. The connectWallet component integrates user Ethereum wallets, managing account connections and statuses. This architecture ensures transparency, security, and efficiency in managing and tracking crowdfunding campaigns, leveraging the immutable and decentralized nature of blockchain technology.

The Requests module enhances the platform’s functionality by providing a systematic way to manage spending requests within campaigns. The platform's Analytics module provides real-time insights and data visualization, empowering campaign creators on donor and funding progress. The Notification system ensures that users receive timely updates about their campaigns and contributions, enhancing engagement and trust. This module includes detailed tracking of request descriptions, values, recipients, and approval statuses, ensuring that funds are used appropriately and efficiently. The integration of the connectWallet component further bolsters security and user-friendliness, allowing seamless connection and status management of user Ethereum wallets. Overall, this architecture leverages blockchain's inherent advantages to create a robust, transparent, and secure crowdfunding environment.

Activity Diagram

The process flows in the system are captured in the activity diagram. Similar to a state diagram, an activity diagram also consists of actions, transitions, initial and final states, and guard conditions.

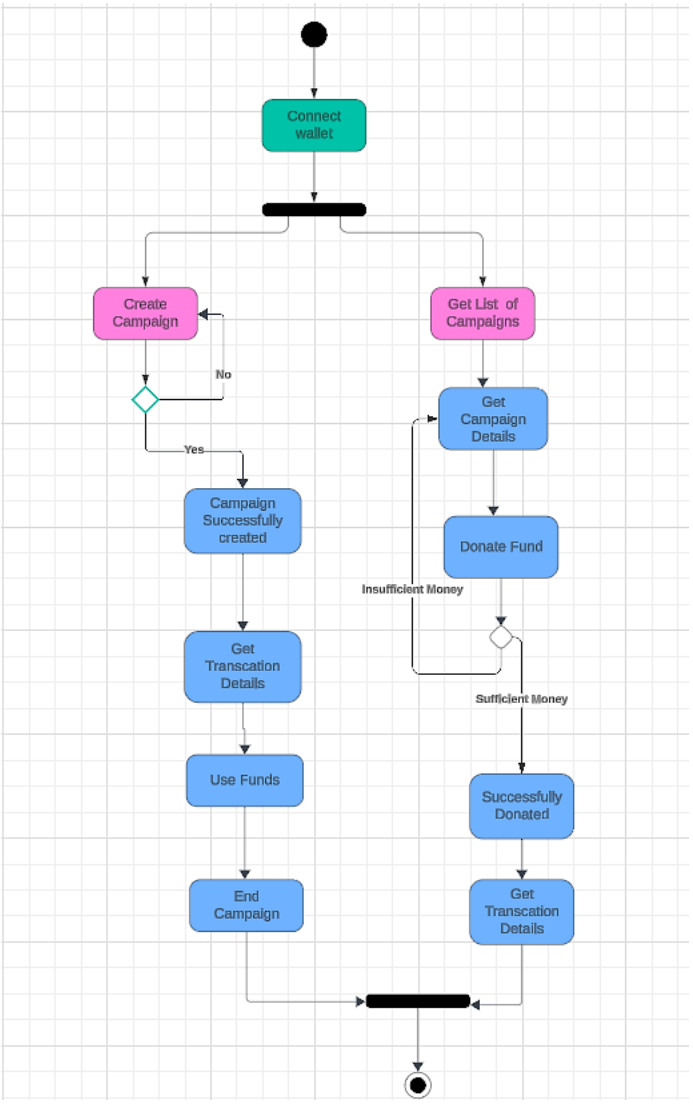

Fig 3.5 Activity Diagram

The activity diagram outlines the workflow for using a wallet application to create campaigns and make donations. The process begins with the user connecting their wallet. From this point, the user can either create a new campaign or explore existing ones.

If the user chooses to create a campaign, they proceed with the "Create Campaign" activity. A decision point determines if the campaign creation is successful. If successful, the workflow continues to fetch transaction details, use the allocated funds, and eventually end the campaign.

Alternatively, if the user opts to explore existing campaigns, they start with the "Get List of Campaigns" activity. After selecting a campaign, the user retrieves detailed information about it. The next step is attempting to donate funds. Here, another decision point checks if the user has sufficient funds. If the funds are sufficient, the donation is processed successfully, and the user is provided with the transaction details related to the donation.

The diagram concludes with the end of the activities, showing a clear and structured process for managing campaign creation, fund utilization, and donation through the wallet application. This ensures a seamless user experience for both campaign managers and donors.

3.5 PROJECT PLAN

Phase 1 - Planning

Define project scope and objectives.

Develop a detailed project plan.

Deliverables: Project charter, analysis, project plan.

Phase 2 - Design

Design smart contracts for CampaignFactory and Campaign.

Create UI/UX design prototypes.

Deliverables: Smart contract specifications, UI/UX design prototypes.

Phase 3 - Development

Implement smart contracts using Solidity.

Set up the development environment using Hardhat.

Develop frontend components using React.

Deliverables: Deployed smart contracts, frontend components.

Phase 4 - Integration

Integrate frontend with blockchain.

Implement wallet connectivity.

Deliverables: Integrated platform, wallet connectivity.

Phase 5 - Testing

Conduct unit testing for smart contracts.

Perform integration

Deliverables: Test reports, bug fixes.

Phase 6 - Deployment

Deploy the platform on the local network (Ganache).

Set up hosting for the frontend.

Deliverables: Deployed platform, user guides.

Project Schedule

Month 1-2: Planning and Design

Week 1: Project kickoff

Week 2-4: Requirements gathering and analysis

Week 5-8: Design phase, finalize smart contract and UI/UX designs

Month 3-4: Development

Week 9-12: Smart contract development

Week 13-16: Frontend development

Month 5: Integration and Testing

Week 17-18: Integration of frontend and backend

Week 19-20: Unit testing of smart contracts

Week 21-22: Integration testing

Week 23-24: User acceptance testing

Month 6: Deployment and Launch

Week 25-26: Deploy smart contracts to Ganache.

CHAPTER 4

IMPLEMENTATION

4.1 ENVIRONMENT SETUP

1. **Text Editor/IDE**: A good code editor like Visual Studio Code for developing and testingthe application.
2. **Node.js and npm**: Install Node.js, a JavaScript runtime, which includes npm (Node Package Manager) for managing project dependencies.Node.js allows you to run JavaScript code outside of a web browser, making it ideal for server-side development. npm is a package manager that simplifies the process of installing, updating, and managing project dependencies.
3. **React**: Set up a React project using Create React App or another boilerplate generator.React is a JavaScript library for building user interfaces. Create React App is a popular tool that sets up a React project with all the necessary configurations and dependencies, allowing you to focus on writing code.
4. **Solidity Compiler:** Install the Solidity compiler to compile smart contracts written in Solidity programming language.Solidity is a high-level programming language used for writing smart contracts on the Ethereum blockchain. The Solidity compiler compiles Solidity code into bytecode that can be executed on the Ethereum Virtual Machine (EVM).
5. **Ganache:** Set up Ganache, a local blockchain development environment, for testing and deploying smart contracts locally. Ganache provides a personal Ethereum blockchain that you can use for testing and development purposes. It allows you to deploy smart contracts, simulate transactions, and inspect blockchain data without interacting with the main Ethereum network.
6. **Metamask:** Install the Metamask browser extension for interacting with Ethereum-based applications. Metamask is a cryptocurrency wallet and Ethereum browser extension that allows you to manage Ethereum accounts, interact with decentralized applications (DApps), and sign transactions securely.
7. **Hardhat:** Hardhat is a development environment designed for Ethereum-based blockchain projects, facilitating tasks like compilation, deployment, testing, and debugging of smart contracts. To set up Hardhat for your project, start by installing Node.js and npm. Then, create a new project directory and run npm init to initialize it. Install Hardhat using npm install --save-dev hardhat. Set up your

Hardhat environment by running npx hardhat and following the prompts to create a basic sample project. This setup will include essential configuration files and directories, enabling efficient smart contract development and deployment within your blockchain project.

4.2 IMPLEMENTION OF EACH MODULE

1. **CampaignFactory**
   - Backend Implementation: The CampaignFactory is implemented as a Solidity smart contract using the Hardhat framework. It includes functions for creating new crowdfunding campaigns and emitting events upon creation.
   - Frontend Integration: In the frontend, the CampaignFactory contract is interacted with using web3.js or ethers.js. Users trigger the creation of new campaigns through a user interface component, which calls the createCampaign function with the necessary parameters.
2. **CreateCampaign Function**
   - Backend Implementation: The createCampaign function within the CampaignFactory contract is responsible for deploying new instances of the Campaign contract with the provided parameters.
   - Frontend Integration: Upon user input, the frontend component invokes the createCampaign function of the CampaignFactory contract, passing the required parameters such as campaign title, required amount, and category.
3. **Form**
   - Frontend Implementation: The Form module is implemented as a React component. It includes input fields, buttons, and validation logic to capture user input and handle form submissions. State management libraries like Redux or React Context may be used to manage form state.
4. **FormLeftWrapper and FormRightWrapper**
   - Frontend Implementation: These wrapper components are implemented using HTML/CSS or styled-components in a React application. They encapsulate the visual presentation and organization of form elements, contributing to a cohesive and intuitive user experience. They structure the layout of form elements, with FormLeftWrapper containing elements aligned to the left and FormRightWrapper containing elements aligned to the right.

4.3 INTEGRATION & DEPLOYMENT

1. Integration

Frontend Integration

- The frontend of the application, developed using frameworks like React.js or Vue.js, interacts with both the backend server and the blockchain network.
- User interfaces are designed to capture user input, display relevant information, and facilitate interactions with the application.
- Frontend components make API requests to the backend server to fetch data, such as campaign details, user information, and transaction history.
- User actions, such as creating a new campaign or donating to an existing campaign, trigger transactions on the blockchain network via web3.js or ethers.js libraries.
- The frontend listens for blockchain events emitted by smart contracts to update the user interface in real-time, providing feedback on transaction status and campaign updates.

Backend Integration

- The backend server, developed using frameworks like Express.js or Django, serves as the intermediary between the frontend and the blockchain network.
- It exposes RESTful APIs or GraphQL endpoints that the frontend can consume to perform various operations, such as user authentication, data retrieval, and transaction processing.
- Backend APIs handle business logic, validate user input, and interact with the blockchain network by invoking smart contract functions through web3.js or ethers.js.
- Upon receiving requests from the frontend, the backend server processes data, executes transactions, and returns responses, ensuring proper synchronization between the frontend and the blockchain.

Blockchain Integration

- The blockchain network, powered by platforms like Ethereum or Binance Smart Chain, stores immutable data and executes smart contracts governing the

crowdfunding platform's operations.

- Smart contracts are deployed on the blockchain to define the rules and behaviors of crowdfunding campaigns, including creation, donation, and withdrawal processes.
- Frontend and backend components interact with smart contracts using web3.js or ethers.js libraries, enabling the execution of transactions and retrieval of data from the blockchain.
- Transactions triggered by user actions, such as creating a campaign or donating funds, are broadcasted to the blockchain network, where they are validated, executed, and recorded on the blockchain ledger.
- Events emitted by smart contracts are monitored by the frontend and backend components, allowing them to react to changes in the state of the blockchain, such as campaign creation or donation confirmations.

2. Deployment

1. **Setup Ganache**: First, ensure that Ganache is installed and running on your local machine. Ganache provides a local Ethereum blockchain environment for development and testing purposes.
2. **Compile Smart Contracts**: Use a Solidity compiler like solc or an integrated development environment (IDE) such as Remix to compile your Solidity smart contracts (.sol files). This step generates bytecode and ABI (Application Binary Interface) for each contract.
3. **Configure Deployment Script**: Write a deployment script in JavaScript to interact with the Ganache network and deploy your compiled smart contracts. Use libraries like web3.js or ethers.js to communicate with the blockchain.
4. **Connect to Ganache**: Ensure that your deployment script is configured to connect to the Ganache network. You'll need to specify the network endpoint (usually http://localhost:7545) and provide the necessary account credentials to sign transactions.
5. **Deploy Contracts**: Use the deployment script to deploy your smart contracts to the Ganache network. This involves sending a transaction to the blockchain network, which includes the bytecode of the contract to be deployed.
6. **Verify Deployment**: Once the deployment transaction is confirmed, you'll receive a transaction receipt containing the contract address. Verify that the

contract has been successfully deployed by checking the transaction status and the contract address.

7. **Interact with Contracts**: After deployment, you can interact with the deployed smart contracts using their addresses and ABI. You can send transactions to invoke contract functions, read data from the blockchain, and listen for events emitted by the contracts.
8. **Testing**: Finally, perform thorough testing to ensure that the deployed contracts function as expected. Use tools like Truffle or Hardhat for automated testing, and simulate various scenarios to validate the behavior of your contracts.

CHAPTER 5

TESTING & RESULTS

5.1 TEST CASES

TEST CASE 1 Creating a campaign to educate the child who is working at mines.

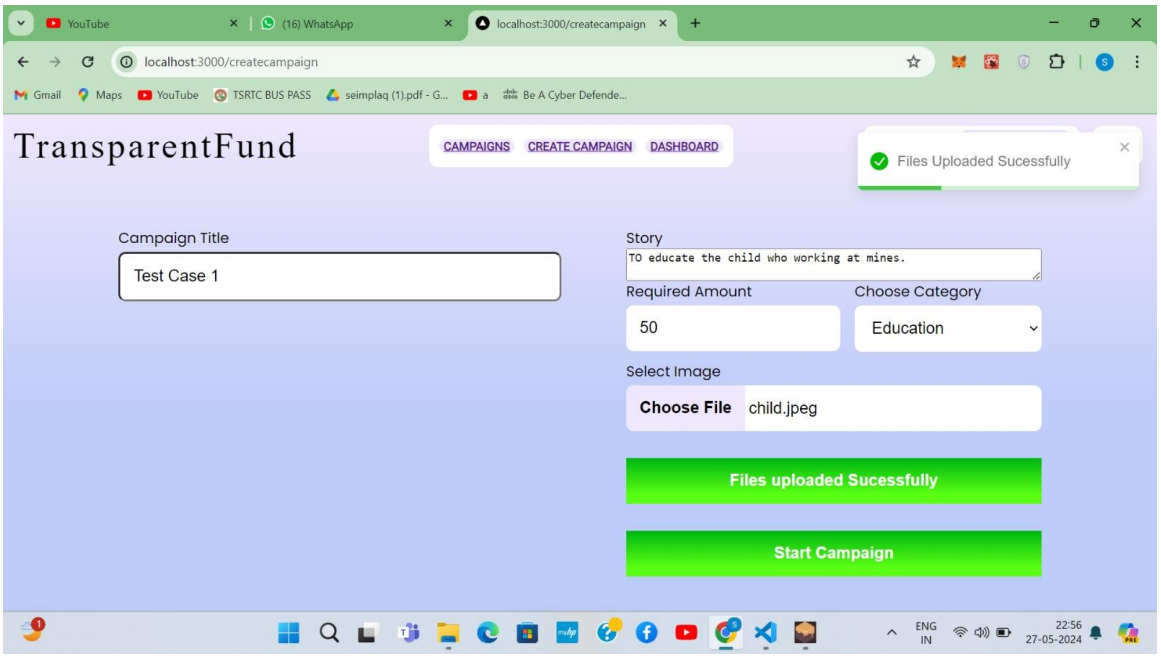

Fig 5.1 Test Case 1 Create Campaign

Creation of a crowdfunding campaign aimed at educating a child currently working in mines as shown in the Fig 5.1. This includes filling out the campaign form with specific details such as the campaign title, required amount, image, category, and story, followed by confirming the creation through MetaMask as shown in the Fig 5.2.

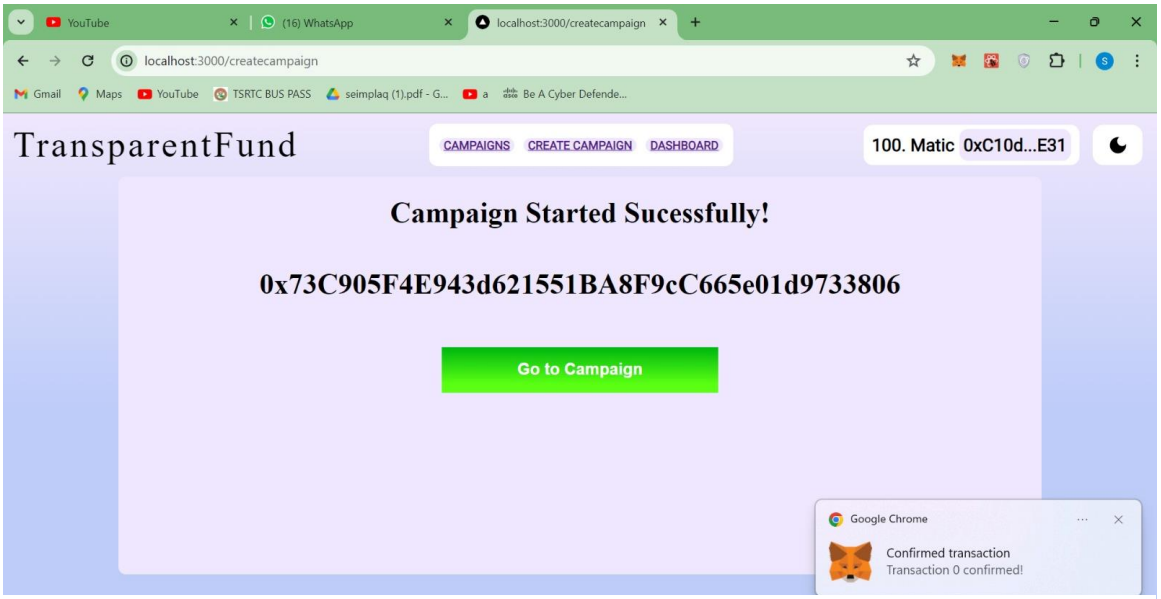

Fig 5.2 Campaign Created

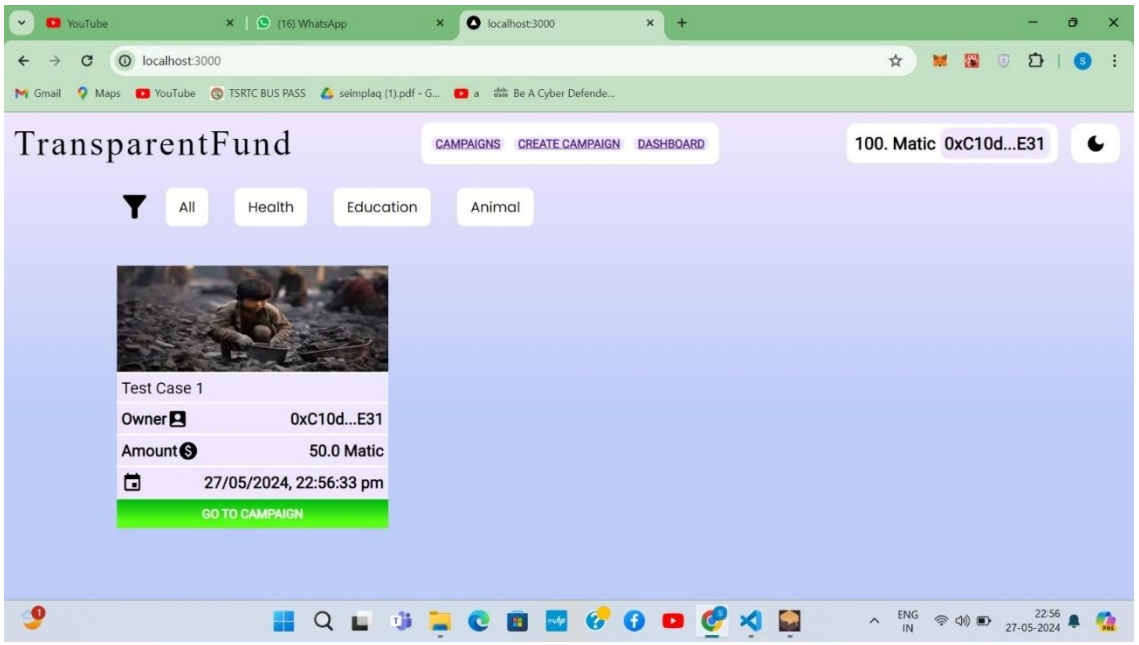

Fig 5.3 Campaign in Home Page

As shown in the above Fig 5.3, newly created campaign is displayed on the homepage and also appears in the user's dashboard. The campaign details, such as title, required amount, and image, are accurately shown in both locations.

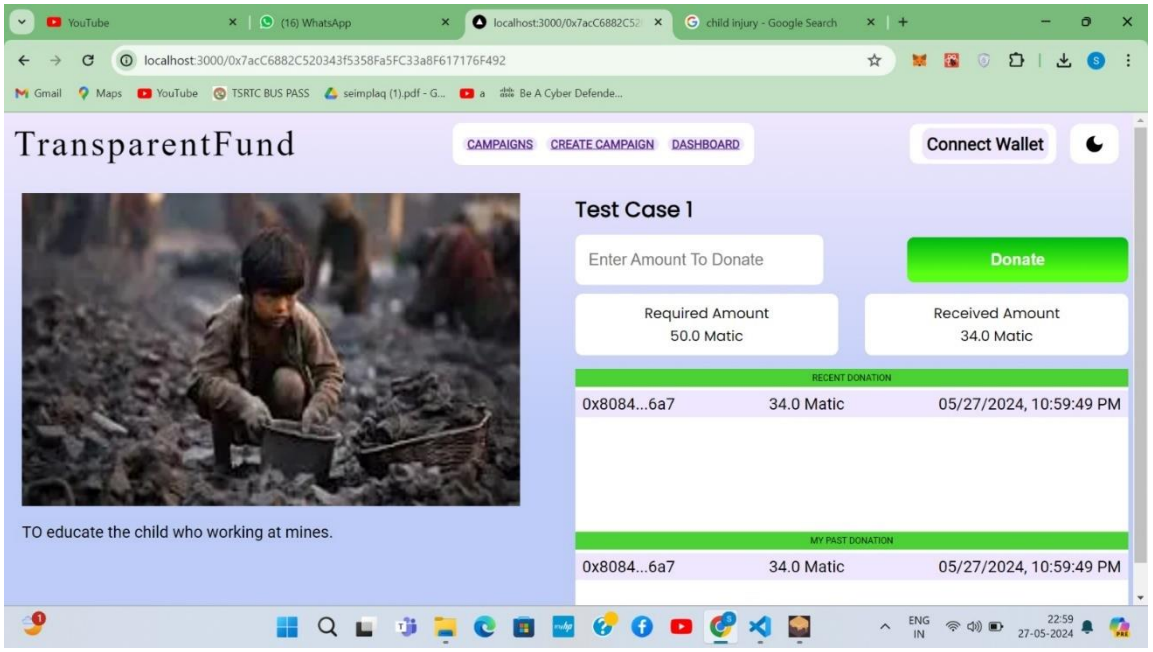

Fig 5.4 Donating to Campaign

Users can donate to the campaign aimed at educating a child working in mines as shown in the above Fig 5.4. This includes ensuring that the donation process is smooth, the funds are transferred to the campaign owner, and transactions are recorded and displayed correctly.

TEST CASE 2 – Creating another campaign to save the dog from abused owner.

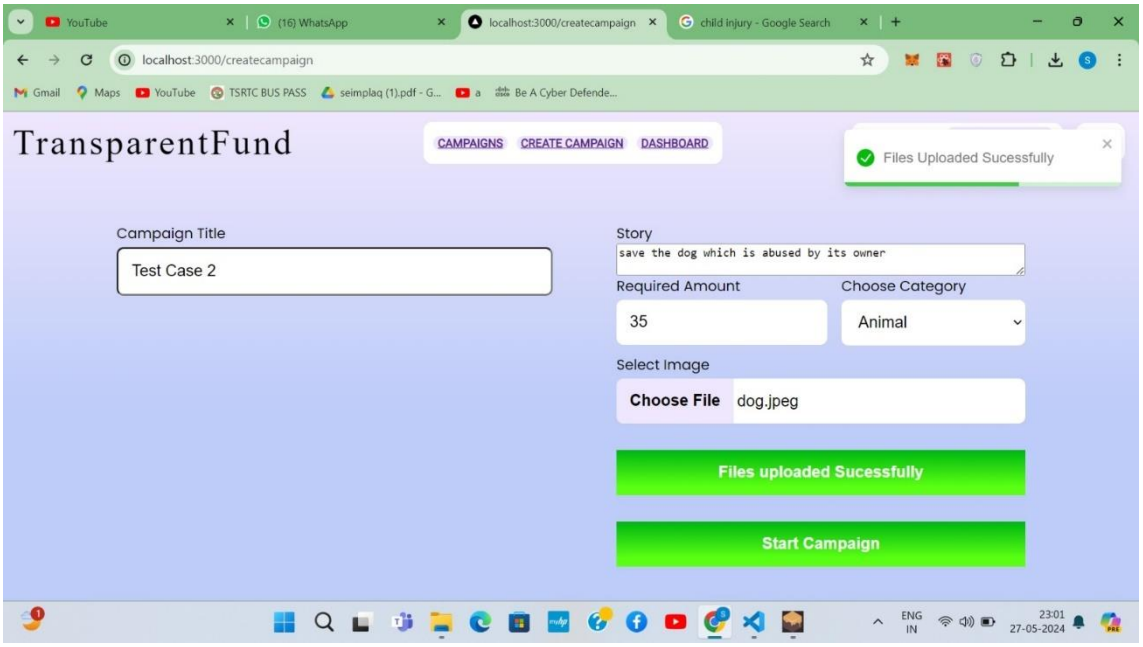

Fig 5.5 Test Case 2

As shown in the above Fig 5.5, Test Case 2 involves creating a campaign to rescue a dog from an abusive owner. The campaign aims to raise awareness, gather support, and ultimately facilitate the dog's safe relocation to a caring environment.

TEST CASE 3- Creating another campaign to help the hospitalized child.

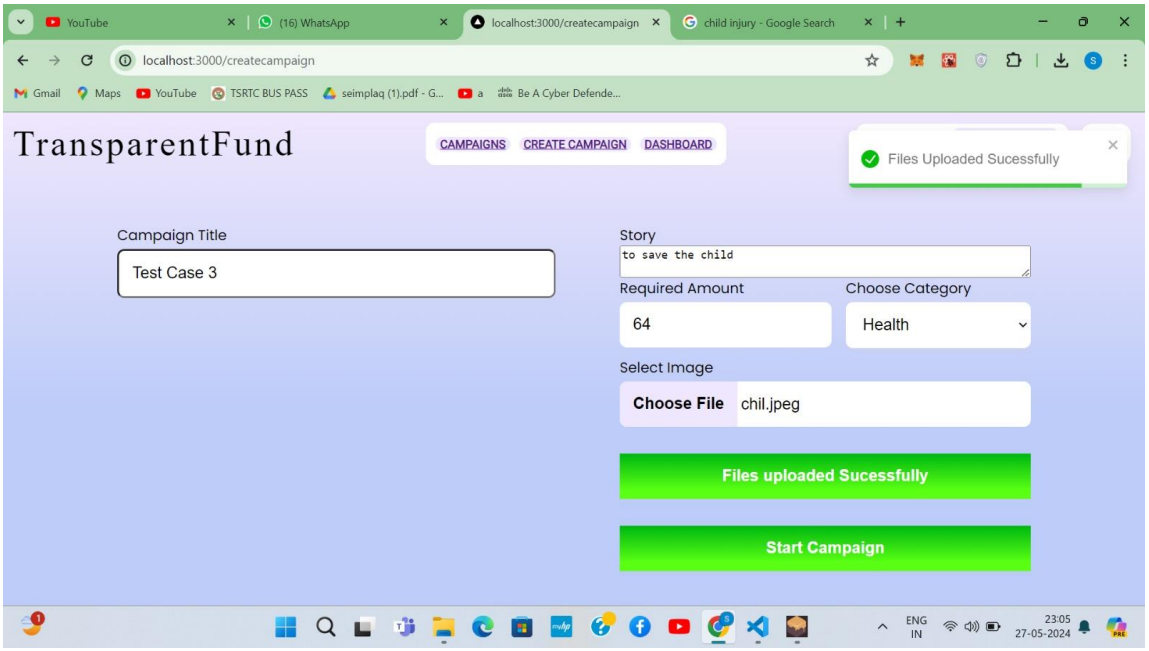

Fig 5.6 Test Case 3

As Shown in the above Fig 5.6, this test case focuses on launching a campaign to aid a hospitalized child. The campaign seeks to mobilize resources, donations, and to alleviate the child's medical expenses and provide emotional comfort.

5.2 RESULTS

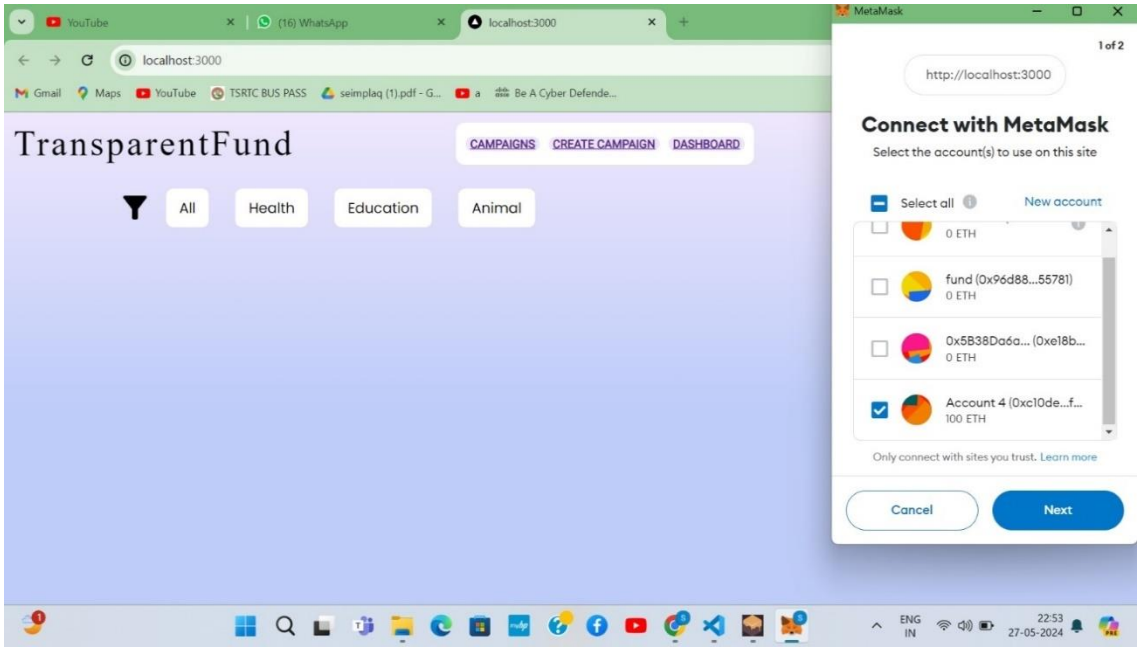

Fig 5.7 Connecting to Metamask Wallet

The above Fig 5.7 shows Connection of Metamask wallet. The above screen shows the process of connecting to the MetaMask wallet for secure blockchain interactions.

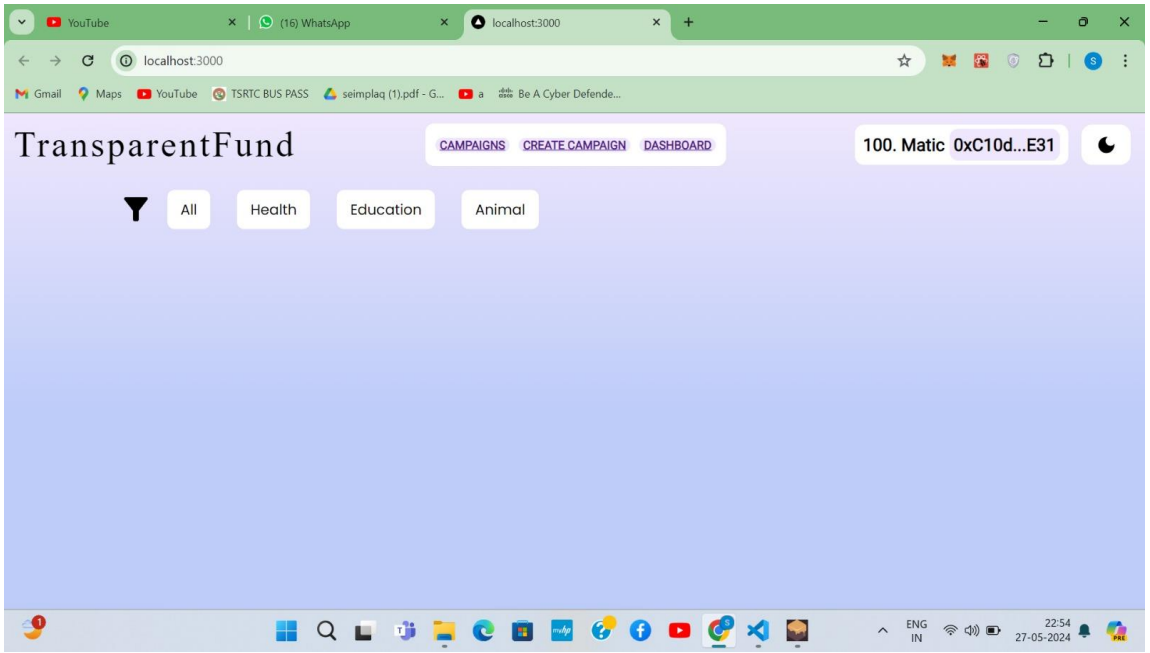

Fig 5.8 Home Page after connecting wallet

The above Fig 5.8 shows the Home Page after wallet connection. The home page displays user interface elements and options available after successfully connecting the MetaMask wallet.

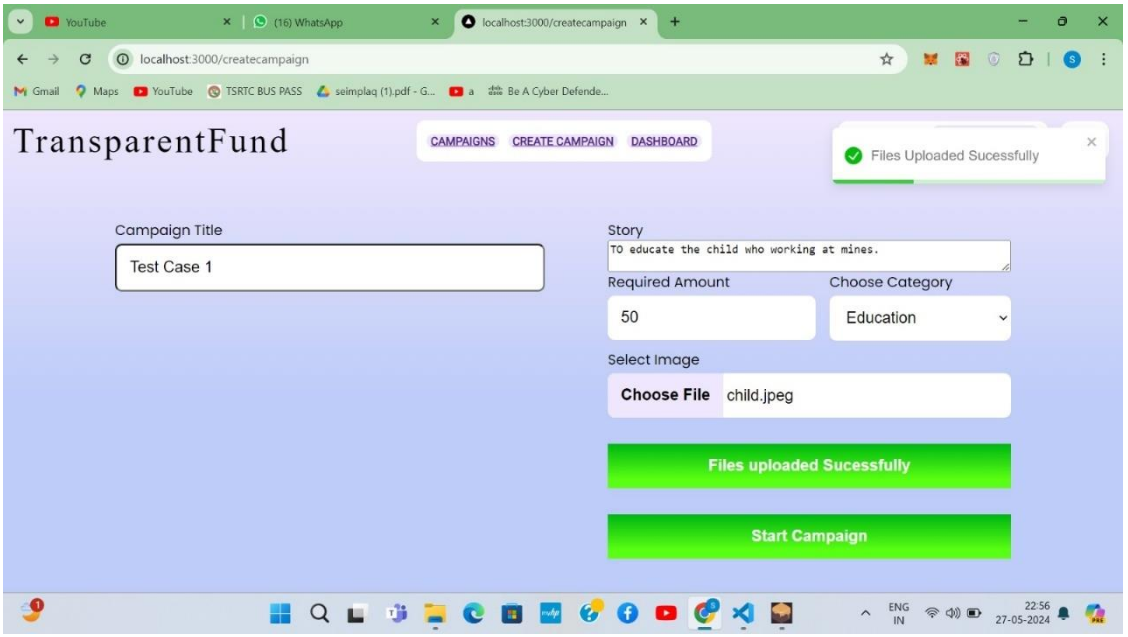

Fig 5.9 Creating a Campaign

Users can create a new campaign by filling out the necessary details in the provided form such as Title, Story, Amount Required, Image as shown in the above Fig 5.9

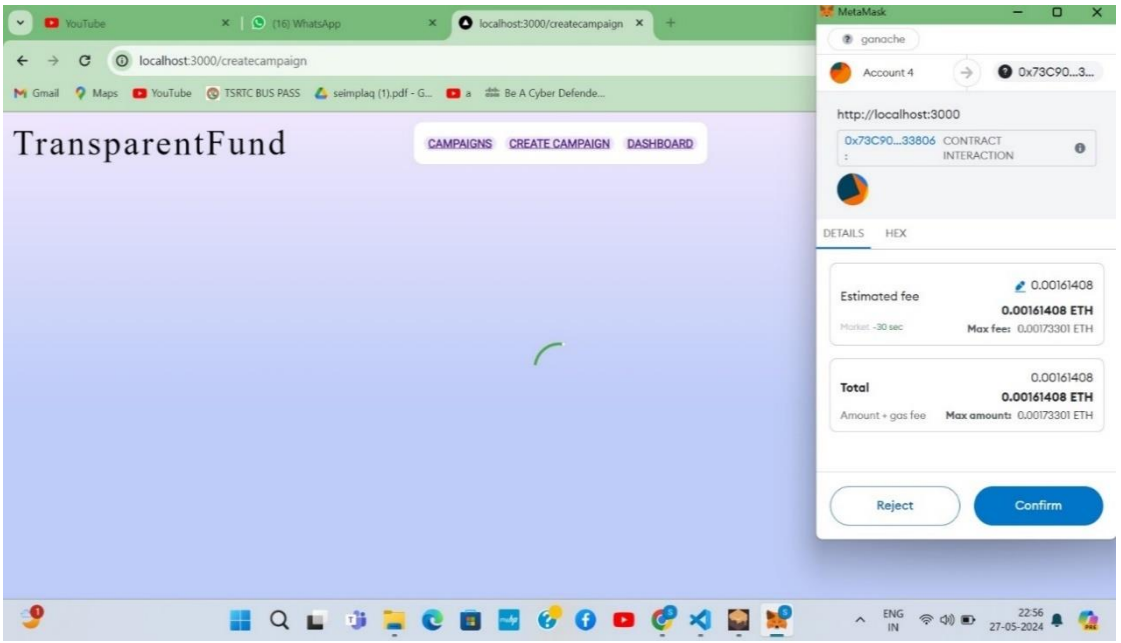

Fig 5.10 Confirmation from MetaMask

The above Fig 5.10 depicts Confirmation of MetaMask Wallet for creating a Campaign. MetaMask requests confirmation for the transaction to ensure user consent and security.

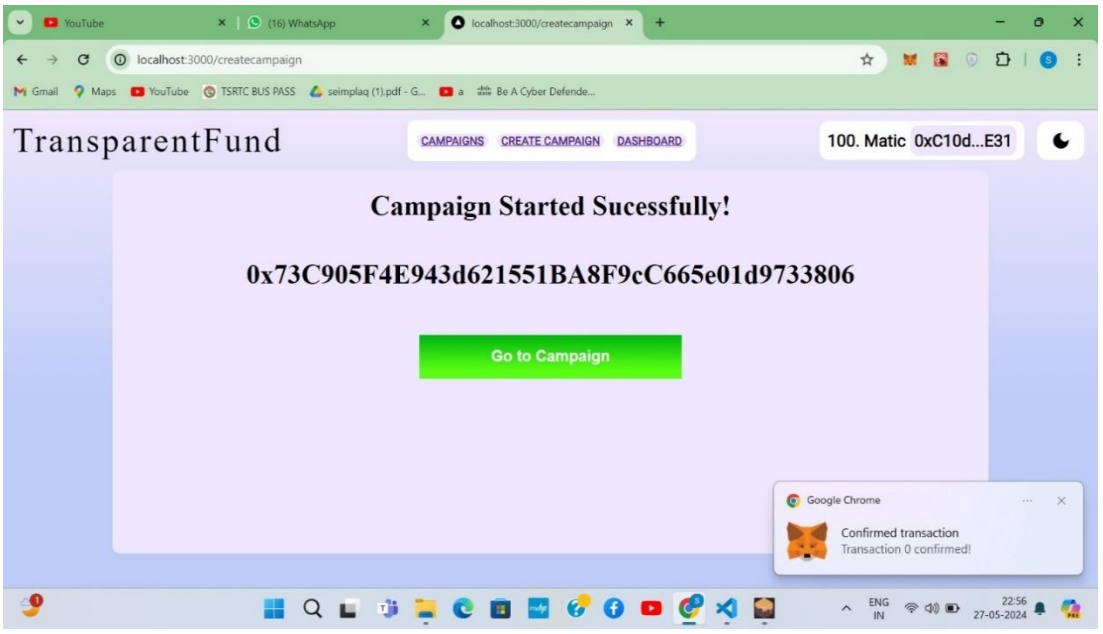

Fig 5.11 Campaign Creation Successful

The above Fig 5.11 shows the confirmation message which indicates that the campaign has been successfully created and recorded on the blockchain.

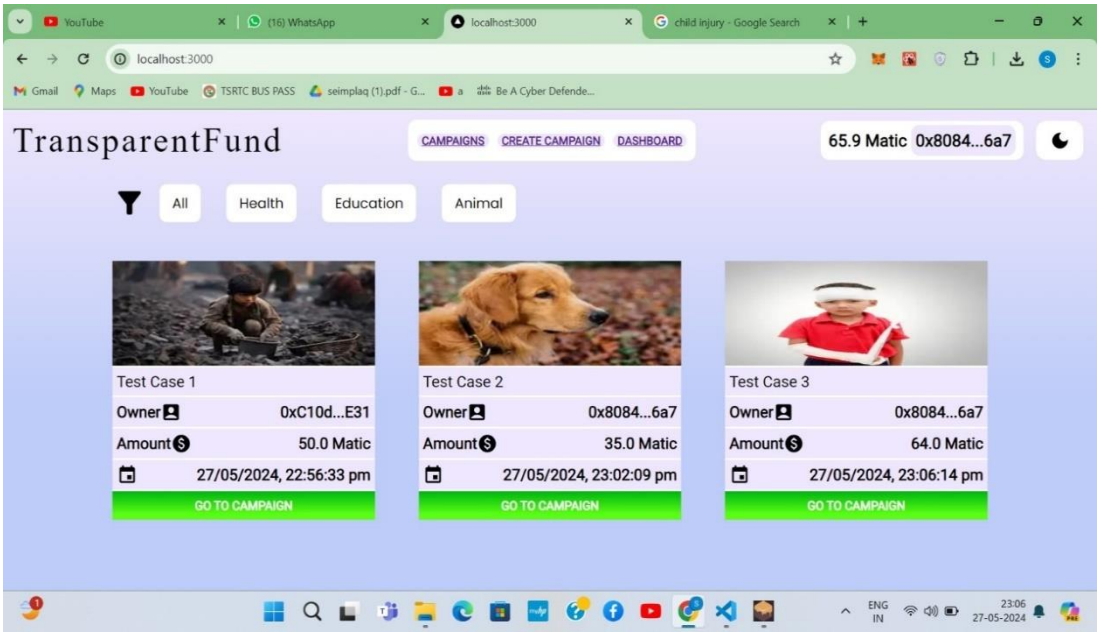

Fig 5.12 All Campaigns in Home Page

The above Fig 5.12 shows that all campaigns are displayed in homepage. The homepage lists all active campaigns available for users to view and interact with.

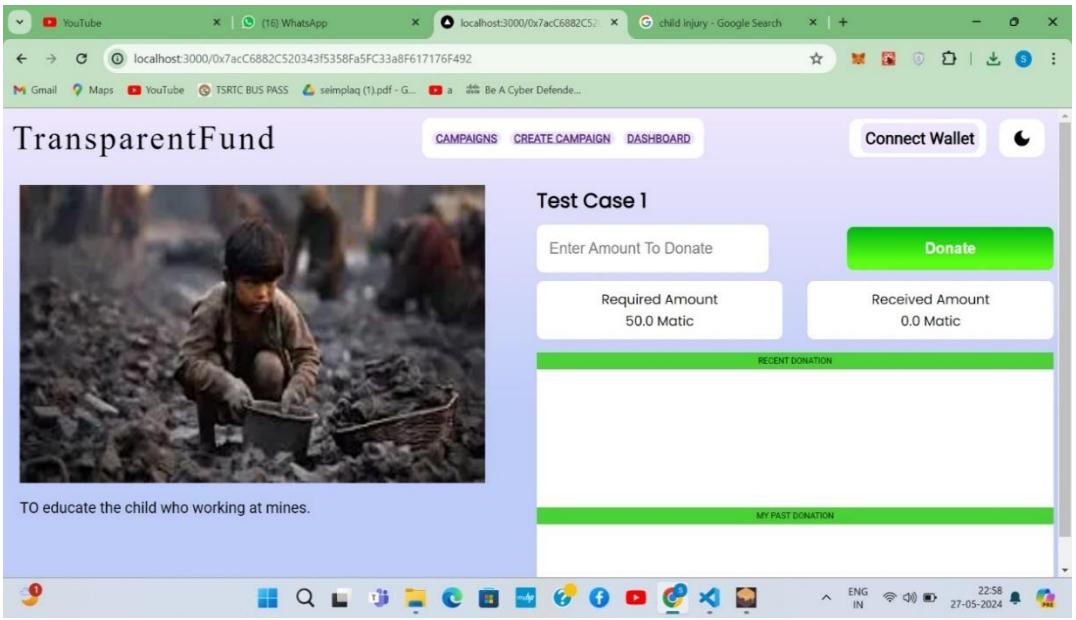

Fig 5.13 Into the Campaign

The above Fig 5.13 describes detailed view of an individual campaign, including its description, goals, status and also indicates the amount of money required.

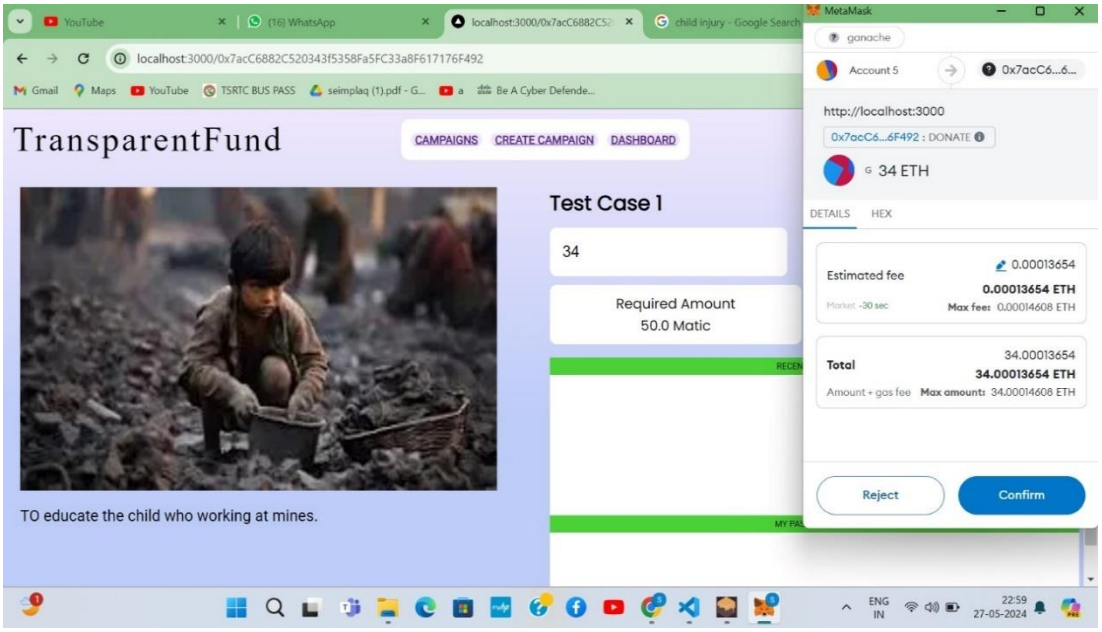

Fig 5.14 Donating to Campaign

The above Fig 5.14 shows how to donate to campaign. Users can donate to a campaign by entering the donation amount and confirming the transaction.

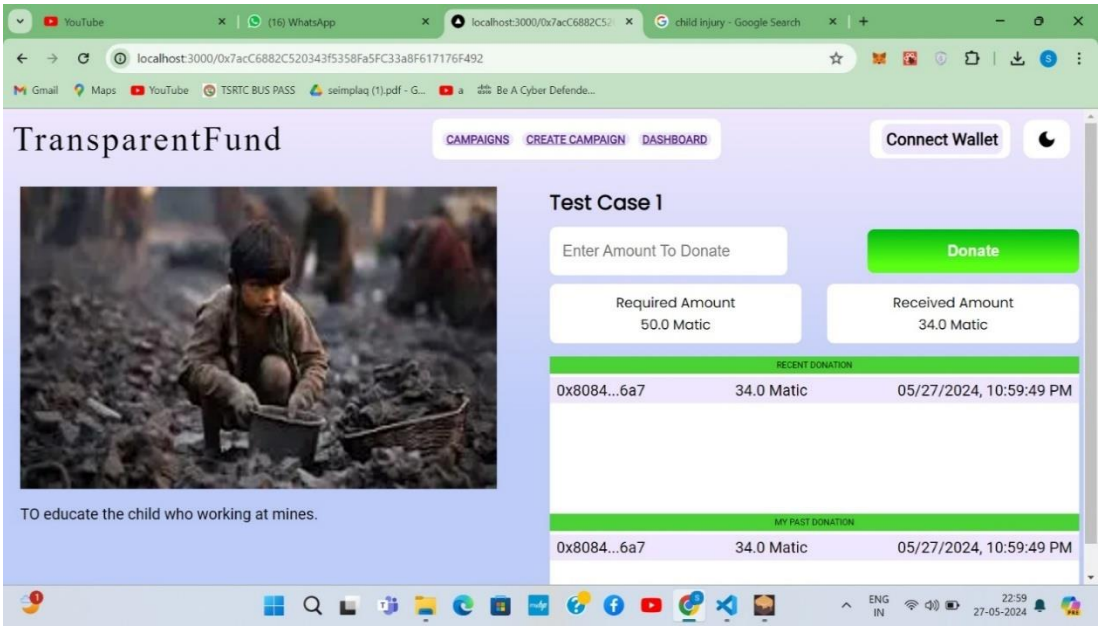

Fig 5.15 Transaction History

The above Fig 5.15 shows the transaction history. A detailed log of all transactions related to the user's account, providing transparency and traceability.

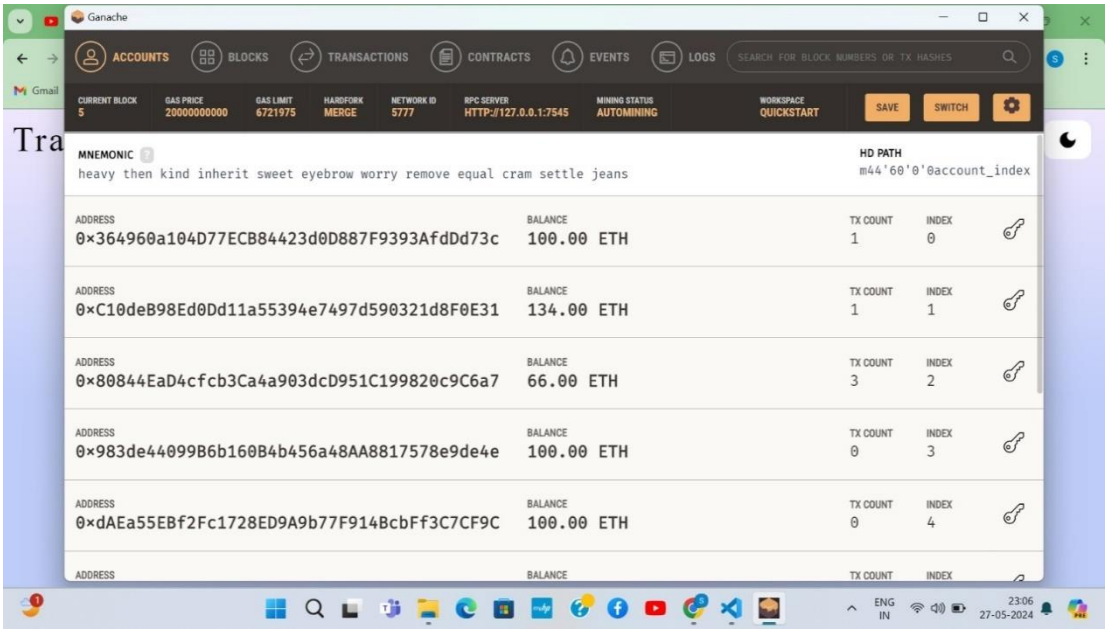

Fig 5.16 Ganache Connection

The above Fig 5.16 is the Ganache Connection. Connection to the Ganache blockchain, used for local testing and development of smart contracts.

CHAPTER 6

CONCLUSION & FUTURE ENHANCEMENTS

In conclusion, our project, "Transparent Funding Platform powered by Blockchain," is a significant advancement in the crowdfunding space, addressing long-standing issues of transparency and fraud. Traditional crowdfunding methods have often been marred by these challenges, deterring potential contributors and undermining trust. By leveraging blockchain technology, the Transparent Funding Platform offers a robust, decentralized solution that ensures a transparent and secure crowdfunding experience. The integration of Ethereum smart contracts and user-friendly interfaces revolutionizes the crowdfunding landscape, providing unparalleled transparency and security for contributors. This enables people to confidently support causes without the fear of fraud, significantly enhancing the overall trust in the system.

The platform's innovative use of CampaignFactory and Campaign contracts allows for efficient creation and management of crowdfunding campaigns. This, combined with Hardhat for smart contract development and Dotenv for environment variable management, ensures reliability and security. The Transparent Funding Platform democratizes access to funds, fostering trust among investors and donors, and encouraging global participation in crowdfunding initiatives. By addressing the limitations of traditional methods, the platform promotes a more inclusive and trustworthy fundraising ecosystem, paving the way for broader and more impactful participation in funding worthwhile causes.

FUTURE ENHANCEMENTS

The Transparent Funding Platform stands poised for a dynamic evolution, with future enhancements poised to propel it to even greater heights of innovation and effectiveness. One pivotal avenue for advancement lies in the integration of sophisticated analytics and reporting tools. By delving into deeper insights regarding campaign performance, donor behaviors, and overarching platform metrics, users can make more informed decisions and refine their fundraising strategies with precision.

Moreover, expanding the platform's compatibility to encompass multiple blockchain networks beyond Ethereum represents a significant leap forward. This broader scope also opens avenues for exploring emerging technologies and decentralized solutions, further enriching the platform's ecosystem and user experience.

Additionally, introducing decentralized identity solutions promises to revolutionize user verification and authentication protocols. By implementing robust DID mechanisms, the platform can instill heightened levels of security and regulatory compliance, fostering trust among users and safeguarding against illicit activities. Furthermore, mobile application support promises to enhance accessibility, enabling users to seamlessly manage campaigns and contributions while on the move. Lastly, nurturing a thriving community and marketplace within the platform fosters collaboration and resource sharing, enriching the user experience and solidifying the platform's position as a pioneering force in the crowdfunding landscape.

REFERENCES

[1] Building a Blockchain-Based Decentralized Crowdfunding Platform for Social and Educational Causes in the Context of Sustainable Development 2023, 15(23), 16205; https://doi.org/10.3390/su152316205.MDPI

[2] Bhaskar, P., & Mehrotra, S. (2020). Blockchain Technology and Crowdfunding: A Systematic Review. In 2020 2nd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA) (pp. 1-5). IEEE.

[3] Zhang, W., Zheng, Z., & Cai, Z. (2020). Crowdfunding based on Blockchain: A Comprehensive Review. Journal of Industrial Information Integration, 20, 100155

[4] Xu, J., Wu, J., & Dai, J. (2019). Blockchain-based decentralized crowdfunding: A survey. arXiv preprint arXiv:1908.01897.

[5] Gai, K., & Qiu, M. (2019). Blockchain-based crowdfunding: Challenges and opportunities. Electronic Commerce Research and Applications, 34, 100833.

[6] Malithi mithsara, T M K, (2020) Blockchain-based Distributed Secure Crowdfunding and Decision-Making Platform for Large-scale Business Projects in Public and Private Sectors. ResearchGate

[7] Chethan S, Rohan P, Roopashree CS, (2023) Crowdfunding Dapp. IRJMETS

[8] Dipali Pawar, Sanket Sangle, Aditya Muley,Siddhesh, Ruhinaaz Shaikh (2023) Blockchain based Crowdfunding using Ethereum Smart Contract.IJRASET

[9] Vinita, Sam, Ravi (2023) A Transparent, Distributed & Secure Crowdfunding platform based on Blockchain.Springer

APPENDIX

SOURCE CODE

Campaign.sol

```
// SPDX-License-Identifier: Unlicensed
pragma solidity >0.7.0 <=0.9.0;
contract CampaignFactory {
address[] public deployedCampaigns;
event campaignCreated(
        string title,
        uint requiredAmount,
        address indexed owner,
        address campaignAddress,
        string imgURI,
        uint indexed timestamp,
        string indexed category
    );
    function createCampaign(
        string memory campaignTitle,
        uint requiredCampaignAmount,
        string memory imgURI,
        string memory category,
        string memory storyURI) public
    {
        Campaign newCampaign = new Campaign(
            campaignTitle, requiredCampaignAmount, imgURI,
storyURI, msg.sender);
deployedCampaigns.push(address(newCampaign));
emit campaignCreated(
            campaignTitle,
            requiredCampaignAmount,
            msg.sender,
            address(newCampaign),
            imgURI,
            block.timestamp,
            category
```

```
        );
    }
}
contract Campaign {
    string public title;
    uint public requiredAmount;
    string public image;
    string public story;
    address payable public owner;
    uint public receivedAmount;
    event donated(address indexed donar, uint indexed
amount, uint indexed timestamp);
    constructor(
        string memory campaignTitle,
        uint requiredCampaignAmount,
        string memory imgURI,
        string memory storyURI,
 address campaignOwner
    ) {
        title = campaignTitle;
        requiredAmount = requiredCampaignAmount;
        image = imgURI;
        story = storyURI;
        owner = payable(campaignOwner);
    }
    function donate() public payable {
        require(requiredAmount > receivedAmount, "required
amount fullfilled");
        owner.transfer(msg.value);
        receivedAmount += msg.value;
        emit donated(msg.sender, msg.value,
block.timestamp);
    }
}
```

Form.js

```
import styled from 'styled-components';
import FormLeftWrapper from './Components/FormLeftWrapper'
import FormRightWrapper from
'./Components/FormRightWrapper'
import {createContext, useState} from 'react';
import {TailSpin} from 'react-loader-spinner';
import {ethers} from 'ethers';
import {toast} from 'react-toastify';
import CampaignFactory from
'../../artifacts/contracts/Campaign.sol/CampaignFactory.jso
n'
const FormState = createContext();
const Form=()=>{
    const [form,setForm]=useState({
        campaignTitle:"",
        requiredAmount: "",
        category:"education",
    });
    const [loading,setLoading]=useState(false);
    const[address,setAddress]=useState("");
    const [uploaded,setUploaded]=useState(false);
    const [storyUrl,setStoryUrl]=useState();
    const [imageUrl, setImageUrl]= useState();
    const FormHandler =(e)=>{
setForm({
    ...form,
    [e.target.name]: e.target.value
})
    }
    const [image,setImage]=useState(null);
    const ImageHandler=(e) =>{
        setImage(e.target.files[0]);
    }
    const startCampaign = async (e) => {
```

```
        e.preventDefault();
        const provider = new
ethers.providers.Web3Provider(window.ethereum);
        const signer = provider.getSigner();
        if(form.campaignTitle === "") {
          toast.warn("Title Field Is Empty");
        }
         else if(form.requiredAmount === "") {
          toast.warn("Required Amount Field Is Empty");
        } else if(uploaded == false){
            toast.warn("files upload required");
        }
        else {
          setLoading(true);
          const contract = new ethers.Contract(
            process.env.NEXT_PUBLIC_ADDRESS,
            CampaignFactory.abi,
            signer
                );
           const CampaignAmount =
ethers.utils.parseEther(form.requiredAmount);
          const campaignData = await
contract.createCampaign(
            form.campaignTitle,
            //parseInt(form.requiredAmount),
            CampaignAmount,
            imageUrl,
            form.category,
            storyUrl
          );
          await campaignData.wait();
          setAddress(campaignData.to);
        }
}
    return (
```

```
        <FormState.Provider value={{form, setForm,image,
setImage,ImageHandler,
FormHandler,setImageUrl,setStoryUrl,startCampaign,setUpload
ed}}>
    <FormWrapper>
        <FormMain>
        {
            loading ==true?
            address ==""?
            <Spinner><TailSpin height ={60} /></Spinner>:
            <Address>
           <h1>Campaign Started Sucessfully!</h1>
           <h1>{address}</h1>
           <Button>
            Go to Campaign
           </Button>
            </Address>
            :
            <FormInputsWrapper>
                <FormLeftWrapper />
                <FormRightWrapper />
            </FormInputsWrapper>
            }
        </FormMain>
    </FormWrapper>
    </FormState.Provider>) }
const FormWrapper = styled.div`
    width: 100%;
    display:flex;
    justify-content:center;
`
const Address = styled.div`
    width:100%;
    height:80vh;
    display:flex ;
```

```
    display:flex ;
    flex-direction:column;
    align-items:center ;
    background-color:${(props) => props.theme.bgSubDiv} ;
    border-radius:8px;
`
const FormMain = styled.div`
width: 80%;
`
const FormInputsWrapper = styled.div`
    display:flex;
    justify-content:space-between ;
    margin-top:45px ;
`
const Spinner = styled.div`
    width:100%;
    height:80vh;
    display:flex ;
    justify-content:center ;
    align-items:center ;
`
const Button = styled.button`
    display: flex;
  justify-content:center;
  width:30% ;
  padding:15px ;
  color:white ;
  background-color:#00b712 ;
  background-image:
      linear-gradient(180deg, #00b712 0%, #5aff15 80%) ;
  border:none;
  margin-top:30px ;
  cursor: pointer;
  font-weight:bold ;
  font-size:large ;
```

```
`
export default Form;
export {FormState};
Wallet.js
import styled from "styled-components";
import { ethers } from "ethers";
import { useState } from "react";
const networks = {
  polygon: {
    chainId: `0x${Number(80001).toString(16)}`,
    chainName: "Polygon Testnet",
    nativeCurrency: {
      name: "MATIC",
      symbol: "MATIC",
      decimals: 18,
    },
    rpcUrls: ["https://rpc-mumbai.maticvigil.com/"],
    blockExplorerUrls: ["https://mumbai.polygonscan.com/"],
  },
};
const Wallet = () => {
  const [address, setAddress] = useState("");
  const [balance, setBalance] = useState("");
  const connectWallet = async () => {
    await window.ethereum.request({ method:
"eth_requestAccounts" });
    const provider = new
ethers.providers.Web3Provider(window.ethereum, "any");
    if (provider.network !== "matic") {
      await window.ethereum.request({
        method: "wallet_addEthereumChain",
        params: [
          {
            ...networks["polygon"],
          },
```

```
        ],
      });
    }
      const account = provider.getSigner();
      const Address = await account.getAddress();
      setAddress(Address);
      const Balance = ethers.utils.formatEther(await
account.getBalance());
      setBalance(Balance);
  };
  return (
    <ConnectWalletWrapper onClick={connectWallet}>
      {balance == '' ? <Balance></Balance> :
<Balance>{balance.slice(0,4)} Matic</Balance> }
      {address == '' ? <Address>Connect Wallet</Address> :
<Address>{address.slice(0,6)}...{address.slice(39)}</Addres
s>}
    </ConnectWalletWrapper>
  );
};
const ConnectWalletWrapper = styled.div`
  display: flex;
  align-items: center;
  justify-content: space-between;
  background-color: ${(props) => props.theme.bgDiv};
  padding: 5px 9px;
  height: 100%;
  color: ${(props) => props.theme.color};
  border-radius: 10px;
  margin-right: 15px;
  font-family: 'Roboto';
  font-weight: bold;
  font-size: small;
  cursor: pointer;
`;
```

```
const Address = styled.h2`
    background-color: ${(props) => props.theme.bgSubDiv};
    height: 100%;
    display: flex;
    align-items: center;
    justify-content: center;
    padding: 0 5px 0 5px;
    border-radius: 10px;
`
const Balance = styled.h2`
    display: flex;
    height: 100%;
    align-items: center;
    justify-content: center;
    margin-right: 5px;
`
export default Wallet;
```

address.js

```
import styled from "styled-components";
import Image from "next/image";
import { ethers } from 'ethers';
import CampaignFactory from
'../artifacts/contracts/Campaign.sol/CampaignFactory.json'
import Campaign from
'../artifacts/contracts/Campaign.sol/Campaign.json'
import { useEffect, useState } from "react";
import { format } from 'date-fns'; // Import format
function from date-fns
export default function Detail({ Data, DonationsData }) {
  const [mydonations, setMydonations] = useState([]);
  const [story, setStory] = useState('');
  const [amount, setAmount] = useState();
  const [change, setChange] = useState(false);
  useEffect(() => {
    const Request = async () => {
```

```
      let storyData;
      await window.ethereum.request({ method:
'eth_requestAccounts' });
      const Web3provider = new
ethers.providers.Web3Provider(window.ethereum);
      const signer = Web3provider.getSigner();
      const Address = await signer.getAddress();
      const provider = new
ethers.providers.JsonRpcProvider(
        process.env.NEXT_PUBLIC_RPC_URL
      );
      const contract = new ethers.Contract(
        Data.address,
        Campaign.abi,
        provider
      );
      fetch("https://gateway.pinata.cloud/ipfs/" +
Data.storyUrl)
        .then(res => res.text()).then(data => storyData =
data);
      const MyDonations =
contract.filters.donated(Address);
      const MyAllDonations = await
contract.queryFilter(MyDonations);
      setMydonations(MyAllDonations.map((e) => {
        return {
          donar: e.args.donar,
          amount: ethers.utils.formatEther(e.args.amount),
          timestamp: format(new Date(e.args.timestamp *
1000), "MM/dd/yyyy, hh:mm:ss a") // Format timestamp using
date-fns
        }
      }));
      setStory(storyData);
    }
```

```
    Request();
  }, [change])
  const DonateFunds = async () => {
    try {
      await window.ethereum.request({ method:
'eth_requestAccounts' });
      const provider = new
ethers.providers.Web3Provider(window.ethereum);
      const signer = provider.getSigner();
      const contract = new ethers.Contract(Data.address,
Campaign.abi, signer);
      const transaction = await contract.donate({ value:
ethers.utils.parseEther(amount) });
      await transaction.wait();
      setChange(true);
      setAmount('');
    } catch (error) {
      console.log(error);
    }
  }
  return (
    <DetailWrapper>
      <LeftContainer>
        <ImageSection>
          <Image
            alt="crowdfunding dapp"
            layout="fill"
            src={
              "https://gateway.pinata.cloud/ipfs/" +
Data.image
            }
          />
        </ImageSection>
        <Text>
          {story}
```

```
        </Text>
      </LeftContainer>
      <RightContainer>
        <Title>{Data.title}</Title>
        <DonateSection>
          <Input value={amount} onChange={(e) =>
setAmount(e.target.value)} type="number" placeholder="Enter
Amount To Donate" />
          <Donate onClick={DonateFunds}>Donate</Donate>
        </DonateSection>
        <FundsData>
          <Funds>
            <FundText>Required Amount</FundText>
            <FundText>{Data.requiredAmount}
Matic</FundText>
          </Funds>
          <Funds>
            <FundText>Received Amount</FundText>
            <FundText>{Data.receivedAmount}
Matic</FundText>
          </Funds>
        </FundsData>
        <Donated>
          <LiveDonation>
            <DonationTitle>Recent Donation</DonationTitle>
            {DonationsData.map((e) => {
              return (
                <Donation key={e.timestamp}>
                  <DonationData>{e.donar.slice(0,
6)}...{e.donar.slice(39)}</DonationData>
                  <DonationData>{e.amount}
Matic</DonationData>

<DonationData>{e.timestamp}</DonationData>
                </Donation>
```

```
              )
            })
            }
          </LiveDonation>
          <MyDonation>
            <DonationTitle>My Past Donation</DonationTitle>
            {mydonations.map((e) => {
              return (
                <Donation key={e.timestamp}>
                <DonationData>{e.donar.slice(0,
6)}...{e.donar.slice(39)}</DonationData>
                  <DonationData>{e.amount}
Matic</DonationData>

<DonationData>{e.timestamp}</DonationData>
                </Donation>
              )
            })
            }
          </MyDonation>
        </Donated>
      </RightContainer>
    </DetailWrapper>
  );
}
export async function getStaticPaths() {
  const provider = new ethers.providers.JsonRpcProvider(
    process.env.NEXT_PUBLIC_RPC_URL
  );
  const contract = new ethers.Contract(
    process.env.NEXT_PUBLIC_ADDRESS,
    CampaignFactory.abi,
    provider
  );
  const getAllCampaigns =
```

```
contract.filters.campaignCreated();
  const AllCampaigns = await
contract.queryFilter(getAllCampaigns);
  return {
    paths: AllCampaigns.map((e) => ({
      params: {
        address: e.args.campaignAddress.toString(),
      }
    })),
    fallback: "blocking"
  }
}
export async function getStaticProps(context) {
  const provider = new ethers.providers.JsonRpcProvider(
    process.env.NEXT_PUBLIC_RPC_URL
  );
  const contract = new ethers.Contract(
    context.params.address,
    Campaign.abi,
    provider
  );
  const title = await contract.title();
  const requiredAmount = await contract.requiredAmount();
  const image = await contract.image();
  const storyUrl = await contract.story();
  const owner = await contract.owner();
  const receivedAmount = await contract.receivedAmount();
  const Donations = contract.filters.donated();
  const AllDonations = await
contract.queryFilter(Donations);
  //const requiredAmountWei =
ethers.utils.parseEther(requiredAmount.toString()).mul(ethe
rs.BigNumber.from(10).pow(0));
  const Data = {
    address: context.params.address,
```

```
    title,
    requiredAmount:
ethers.utils.formatEther(requiredAmount),
    image,
    receivedAmount:
ethers.utils.formatEther(receivedAmount),
    storyUrl,
    owner,
  }
  const DonationsData = AllDonations.map((e) => {
    return {
      donar: e.args.donar,
      amount: ethers.utils.formatEther(e.args.amount),
      timestamp: format(new Date(e.args.timestamp * 1000),
"MM/dd/yyyy, hh:mm:ss a") // Format timestamp using date-
fns
    }
  });
  return {
    props: {
      Data,
      DonationsData
    },
    revalidate: 10
  }}
const DetailWrapper = styled.div`
  display: flex;
  justify-content: space-between;
  padding: 20px;
  width: 98%;
`;
const LeftContainer = styled.div`
  width: 45%;
`;
const RightContainer = styled.div`
```

```
  width: 50%;`;
const ImageSection = styled.div`
  width: 100%;
  position: relative;
  height: 350px;
`;
const Text = styled.p`
  font-family: "Roboto";
  font-size: large;
  color: ${(props) => props.theme.color};
  text-align: justify;
`;
const Title = styled.h1`
  padding: 0;
  margin: 0;
  font-family: "Poppins";
  font-size: x-large;
  color: ${(props) => props.theme.color};
`;
const DonateSection = styled.div`
  display: flex;
  width: 100%;
  justify-content: space-between;
  align-items: center;
  margin-top: 10px;
`;
const Input = styled.input`
  padding: 8px 15px;
  background-color: ${(props) => props.theme.bgDiv};
  color: ${(props) => props.theme.color};
  border: none;
  border-radius: 8px;
  outline: none;
  font-size: large;
  width: 40%;
```

```
  height: 40px;
`;
const Donate = styled.button`
  display: flex;
  justify-content: center;
  width: 40%;
  padding: 15px;
  color: white;
  background-color: #00b712;
  background-image: linear-gradient(180deg, #00b712 0%,
#5aff15 80%);
  border: none;
  cursor: pointer;
  font-weight: bold;
  border-radius: 8px;
  font-size: large;
`;
const FundsData = styled.div`
  width: 100%;
  display: flex;
  justify-content: space-between;
  margin-top: 10px;
`;
const Funds = styled.div`
  width: 45%;
  background-color: ${(props) => props.theme.bgDiv};
  padding: 8px;
  border-radius: 8px;
  text-align: center;
`;
const FundText = styled.p`
  margin: 2px;
  padding: 0;
  font-family: "Poppins";
  font-size: normal;
```

```
`;
const Donated = styled.div`
  height: 280px;
  margin-top: 15px;
  background-color: ${(props) => props.theme.bgDiv};
`;
const LiveDonation = styled.div`
  height: 65%;
  overflow-y: auto;
`;
const MyDonation = styled.div`
  height: 35%;
  overflow-y: auto;
`;
const DonationTitle = styled.div`
  font-family: "Roboto";
  font-size: x-small;
  text-transform: uppercase;
  padding: 4px;
  text-align: center;
  background-color: #4cd137;
`;
const Donation = styled.div`
  display: flex;
  justify-content: space-between;
  margin-top: 4px;
  background-color: ${(props) => props.theme.bgSubDiv};
  padding: 4px 8px;`;
const DonationData = styled.p`
  color: ${(props) => props.theme.color};
  font-family: "Roboto";
  font-size: large;
  margin: 0;
  padding: 0;
`;
```