Minor Project Report

on

Crowdfunding DAPP

Submitted to Guru Gobind Singh Indraprastha University, Delhi (India)

In partial fulfilment of the requirement for the award of the degree

of

Bachelor of Technology

in

Information Technology

Under the guidance of

Dr. Tripti Rathee
Assistant Professor

Information Technology

Submitted By:

Ajay Singh Dhillon	**Himanshu Mittal**	**Gaurav Raghav**
20196303119	**20796303119**	**20896303119**

MAHARAJA SURAJMAL INSTITUTE OF TECHNOLOGY

C4, JANAKPURI, NEW DELHI – 110058

(GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY,
DWARKA, NEW DELHI)

(January, 2023)

CANDIDATE'S DECLARATION

It is hereby certified that the work which is being presented in the B.Tech. Minor Project report entitled **Crowdfunding DAPP** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** and submitted in the **Department of Information Technology, New Delhi (Affiliated to Guru Gobind Singh Indraprastha University, New Delhi)** is an authentic record of our work carried out during a period from **September 2022 to January 2023** under the guidance of Dr. Tripti Rathee (Assistant Professor)**.**

The matter presented in the B.Tech. Minor Project Report has not been submitted by us for the award of any other degree of this or any other institute.

Ajay Singh Dhillon	**Himanshu Mittal**	**Gaurav Raghav**
20196303119	**20796303119**	**20896303119**

This is to certify that the above statement made by the candidates are correct to the best of my knowledge. They are permitted to appear in the External Minor Project Examination.

Dr. Tripti Rathee	**Dr. Tripti Sharma**
(Asst. Professor, IT Department)	**(HOD, IT Department)**

The B.tech Minor Project Viva-Voce Examination of **Ajay Dhillon (20196303119), Himanshu Mittal (20796303119), Gaurav Raghav (20896303119) has been held on 31.01.2023.**

Project Coordinator **(Signature of External Examiner)**

ACKNOWLEDGEMENT

We express our deep gratitude to **Dr. Tripti Rathee** , Assistant Professor, Department of Information Technology for her valuable guidance and suggestions throughout our project work. We are thankful to **Dr. Minakshi Tomer**, Project Coordinator for her valuable guidance.

We would like to extend our sincere thanks to the **Head of Department, Dr. Tripti Sharma** for her time-to-time suggestions to complete our project work. We are also thankful to **Prof. (Col) Ranjit Singh, Director of MSIT** for providing us with the facilities to carry out our project work.

Ajay Singh Dhillon	**Himanshu Mittal**	**Gaurav Raghav**
20196303119	**20796303119**	**20896303119**

CONTENTS

ABSTRACT

Crowdfunding is a method for funding various kinds of ventures, wherein individual founders of the ventures can request for funds. The ventures may be working for profit motive, non-profit motive, cultural or social. The funds are usually given in return for future products or equity. Due to the current era of technology, the use of internet social media platforms to connect investors with entrepreneurs in order to raise capital for various kinds of ventures in return for compensation or just with a motive to help the noble cause.

The issue with today's crowdfunding method is that fundraisers and contributors have no control over the funds they donate and some more like Scam-start-ups, Intellectual Property risk, exorbitant fees. These issues in addition to solving an issue or helping a noble cause, raises few other problems and for few, it won't allow to contribute.

This project creates a platform that presents a block-chain-based crowdfunding network that can provide a private, secure, and decentralized crowdfunding path by using smart contracts. The aim is to solve these problems by applying smart contracts to the crowdfunding site so that the contracts will be fully automatically executed so that frauds can be prevented, and a healthy relationship is built between the fundraisers, platform and the contributors.

This crowdfunding application is not just like any other application which just allows people to invest their money, but this platform also gives an assurance to the backers that returns will be guaranteed. The application will also provide transparency between the backers and the start-ups so that the backers can stay updated on the progress of the project work of the respective start-ups that they invested their money in.

LIST OF FIGURES

LIST OF TABLES

CHAPTER – 1

INTRODUCTION

1. INTRODUCTION

Looking back into the history, conventional practices like raising money through small loans to poor families were practiced followed by providing microcredits to small entrepreneurs which were unable to qualify for bank loans.

New technology and innovation always make a huge impact on humans and the society. With the emergence of new technology, it will impact the existing ones. This transformation can be seen with crowdfunding. The number of projects being launched through crowdfunding platforms has increased significantly. Stakeholders would usually like to track the way the organization is working, and be a part of the same, provided financial information is 100% transparent. And therefore, it is important for the entrepreneurs as well as stakeholders to maintain transparency, credibility and also understand the legal and regulatory framework behind the project.

Crowdfunding is a new and innovative method for funding various kinds of ventures, wherein individual founders of the ventures can request for funds. The ventures may be working for profit motive, cultural or social. The funds are usually given in return for future products or equity. It includes the use of internet social media platforms to connect investors with entrepreneurs in order to raise capital for various kinds of ventures in return for compensation. Internet and social media became new platforms that emerged. Social media and internet play vital role in raising funds for entrepreneurs and other non-profit organizations.

Due to recent times of Covid-19 pandemic across various countries in the world – including developed countries like USA, Russia etc., Crowdfunding activities have been increased all around the world, ranging from tiny campaigns to help individuals acquire oxygen and medical assistance to huge funds like PM Cares. Contributors, crowdfunding platforms, and project administrators were the primary players in the crowdfunding event.

1.1. WHAT IS CROWDFUNDING?

Crowdfunding is basically the practice of funding a project or a start-up through raising money from various individuals. It is usually done using the internet, as it is easily accessible to gather contacts and determine the stakeholders of the project. Crowdfunding can be a suitable method for personal use, for real estate, loans, start-ups, and other businesses.

Crowdfunding is emerging as one of the most affordable and viable options for young entrepreneurs in the country.

“In the recent times, India has witnessed a massive growth in the start-up industry. It is because people want to break out of the barrier of working 9-5 and our government supports it. The government has a significant role to play in this, as they're offering some advantages for start- ups such as relief from paying tax for the first three years and much more. Start-ups often look out for investors and try to approach VCs to get funding. Sometimes it gets tedious. Start-ups can try using crowdfunding as it increases the brand reach and it's easier than approaching a VC.”

Some of the best examples of crowdfunding platforms are Kickstarter, Indiegogo, Startengine, Ketto, etc. These platforms encourage entrepreneurship and are most-commonly used for creative projects, everything from music, art, films as well as technology.

1.2. TYPES OF CROWDFUNDING

The type of crowdfunding to be used usually depends on the kind of business or activity that is being set up. The goals and objectives of the business influence the type as well. There are three basic types of crowdfunding. They are:

- **Debt form of Crowdfunding**: In this medium, the investors or rather contributors are paid the money back along with interest. It is also known as “peer to peer’ lending and usually does not involve the traditional banking methods.
- **Equity based Crowdfunding**: In this method, the investors become partially the owners of the business project. They are also accredited

with dividends. This is in a way kind of a gamble because, if the project is successful, the share value will go up, otherwise down.

- **Donations**: In this medium, the investors don't look for any kind of financial returns. They invest in their belief on the project which is usually cause-based. However, perks can be given in order to show gratitude towards the donators. Examples of these kind of projects are, charities, disaster relief, and other not-for-profit organizations.
  However, it does not come with its own challenges.

1.3. LIMITATIONS OF THE EXISTING CROWDFUNDING PLATFORMS

1. **Exorbitant fees**: Usually, crowdfunding platforms take a certain amount of fee for every project that is listed. Sometimes, it's a specific amount, and other times, it is taken as a percentage of the contribution made by the contributors. This is a drawback for the availability of the funds since start-ups are literally looking for every rupee to help themselves.
2. **Scam start-ups**: in some cases, start-ups turn out to be scams and leave the investors with no option but to suffer the loss of their investment.
3. **IP risk**: sometimes, few start-ups don't protect their intellectual property and in order expose them to other experienced investors who can steal their idea and enter the market with the resources they could arrange for.
4. **DIY marketing**: sometimes, its aggressive advertising and marketing that the start-ups need, rather than spreading the word. And this kind of marketing requires a huge expenditure .
5. **Fine print rules and regulations**: They have their own criterion, and if the start-up isn't meeting the criteria mentioned, then it poses as an obstacle for a start-up's innovation and business.

1.4. SYNERGY BETWEEN BLOCKCHAIN AND CROWDFUNDING

In present times, we are a generation wherein we want to see results of our contributions immediately. The upcoming generation of investors would not want to be just passive givers.

Stake holders would like to keep a track of the way the organization is functioning, whether it is achieving its goals and guide accordingly. This requires 100% transparency, and authentic financial information of the related project activity.

Blockchain solves this issue. Blockchain is basically an independent and transparent model which reduces uncertainty between parties who are exchanging the values. Though it is a new technology, it can be used in real time cases like crowdfunding.

Though, it is not a very familiar technology, but it can definitely be of help especially in the ventures that have crowdfunding as their source of finance. Crowdfunding is based on the trust between the investors and stakeholders. In order to eliminate any kind of uncertainty with respect to utilization of the funds, blockchain technology is the way.

Speaking in general, most of the crowdfunding campaigns that are considered to be successful require the trust of the investors as well as the prestige of the campaign creator. “Transparency and immutable information are key advantages of Blockchain.”

1.5. BENEFITS OF BLOCKCHAIN IN CROWDFUNDING

Excess availability:

Any company that uses blockchain technology for crowdfunding would definitely get funded. Additionally, any individual who has an internet connection will be able to contribute to such projects. To individuals who invest in crowdfunding using blockchain technology, it is beneficial in a way that "fraud" will be absent in such cases because the investors will receive their ownership or a fraction of the enterprise immediately.

Decentralization

The advantage in this model is that, start-ups are not going to utilize the services of any platform or platforms in order to raise funds. They no longer have to abide by any rules or regulations as provided by various platforms. Hence, any project that has the capability of gaining exposure can get funds and therefore can also omit the requirement of paying the fees to the platforms.

Thus, this makes crowd funding cheap and affordable to the investors and contributors.

Smart Contracts:

There are various ways in which smart contracts enabled by blockchain technology can establish more accountability when it comes to crowd funding. First and foremost, the smart contracts would create obstacles that would hinder the funds from being released without derivation with regards to a project or any other legitimate campaign. Therefore, this would in turn prevent huge sums of money from being embezzled by those who have a malafide intention or persons who are not eligible to run the campaign in the first place. It is self- executing and by default transfers the amount of capital for further development if all the criterion are met.

1.6. PROPOSED SYSTEM

This system is aimed to overcome the above major shortcomings with current crowd funding platforms. Crowd fundraising involves a large number of transactions, it is necessary to manage and document them legally. As a result, a smart contract is utilized, which is a transaction protocol that automatically executes, controls, and documents transactions on behalf of project creators and investors in accordance with the agreement.

Any web-based application is a centralized application which means that everything done on the platform is controlled by a single company server. Decentralized application is offered based on the Ethereum Blockchain, in which all campaign information, contributions, withdrawal requests, and funds are stored on a blockchain network that is open to all. The concept is called "*Distributed ledger technology*."

The distributed ledger and its contents are available to all network participants. Here, the transaction uses *PoS* in which it is more fast and secure than existing *PoW*. It is energy efficient in which the nodes are not competing against each other to attach a new block to the blockchain, energy is saved. Also, no problem has to be solved (as in case of Proof-of-Work system) thus saving energy. Proof of stake cuts out the need for complex computations. So, it beats proof of work when it comes to energy efficiency.

A transactional record that cannot be changed. Transactions are recorded only once with this shared ledger, reducing the duplication of effort. After a transaction is recorded to the shared ledger, no participant can edit or tamper with it. If a mistake is found in a transaction record, a new transaction must be entered to correct the problem, and both transactions are then visible. This means that all nodes on the blockchain may see and store funds and transactions preventing data from being held on a single or centralized server.

1.7. SYSTEM ARCHITECTURE

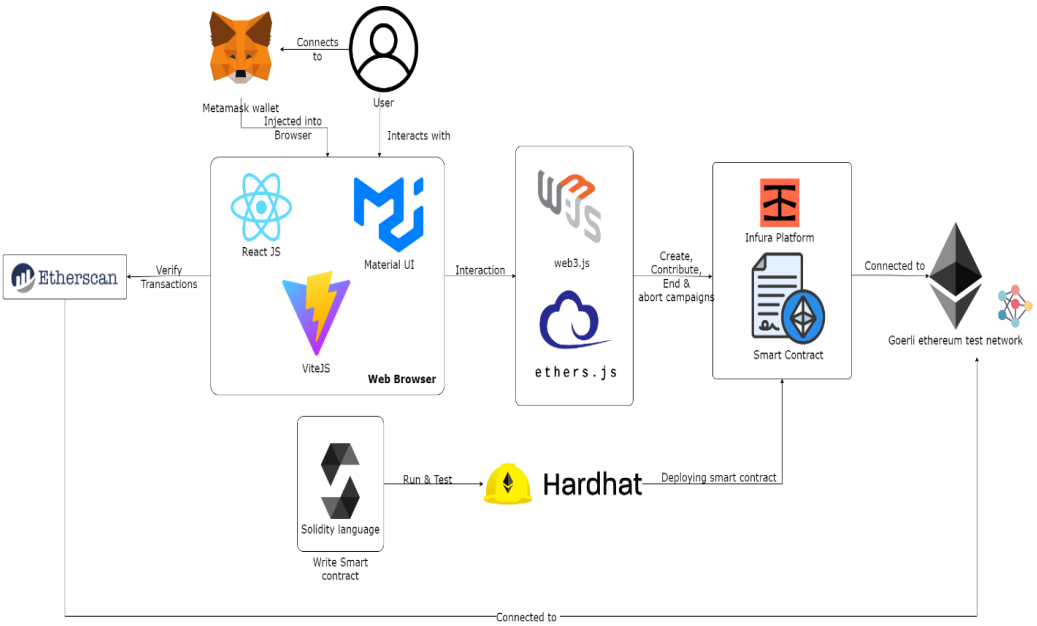

Figure-1.1: System Architecture

CHAPTER – 2

SOFTWARE REQUIREMENT SPECIFICATION

2. Software Requirement Specification

2.1. WHAT IS SRS?

Software Requirement Specification (SRS) is the starting point of the software developing activity. As system grew more complex it became evident that the goal of the entire system cannot be easily comprehended. Hence the need for the requirement phase arose. The software project is initiated by the client needs. The SRS is the means of translating the ideas of the minds of clients (the input) into a formal document (the output of the requirement phase.)

The SRS phase consists of two basic activities:

Problem/Requirement Analysis:

The process is order and more nebulous of the two, deals with understand the problem, the goal and constraints.

Requirement Specification:

Here, the focus is on specifying what has been found giving analysis such as representation, specification languages and tools, and checking the specifications are addressed during this activity. The Requirement phase terminates with the production of the validate SRS document. Producing the SRS document is the basic goal of this phase.

Role of SRS

The purpose of the Software Requirement Specification is to reduce the communication gap between the clients and the developers. Software Requirement Specification is the medium though which the client and user needs are accurately specified. It forms the basis of software development.
A good SRS should satisfy all the parties involved in the system.

2.2. REQUIREMENTS SPECIFICATION DOCUMENT

A Software Requirements Specification (SRS) is a document that describes the nature of a project, software or application. In simple words, SRS document is a manual of a project provided it is prepared before you kick-start a project/application. This document is also known by the names SRS report, software document. A software document is primarily prepared for a project, software or any kind of application.

The purpose of SRS (Software Requirement Specification) document is to describe the external behavior of the application developed or software. It defines the operations, performance and interfaces and quality assurance requirement of the application or software. The complete software requirements for the system are captured by the SRS. This section introduces the requirement specification document for "*Blockchain based crowdfunding platform*" which enlists functional as well as non-functional requirements.

2.3. FUNCTIONAL REQUIREMENTS

For documenting the functional requirements, the set of functionalities supported by the system are to be specified. A function can be specified by identifying the state at which data is to be input to the system, its input data domain, the output domain, and the type of processing to be carried on the input data to obtain the output data. Functional requirements define specific behavior or function of the application. Following are the functional requirements:

The system should

- Enable the fundraisers and backers to register with appropriate proofs.
- Allow the fundraisers to launch a campaign by filling the necessary fields.
- Allow backers to view the active campaigns and fund digital currency.
- In case of failure to meet the criteria set (either not exceeded the timeline or not reached the required funds) – should able to transfer back the funded amount or disburse the raised funds.
- Allow fundraisers to update the status of cause regularly.

- Disburse fraction of funds automatically upon meeting the goals saved in smart- contracts. **7.** Disburse the raised funds *(after successful campaign)* after verification. The system should enable the fundraisers to

2.4. NON-FUNCTIONAL REQUIREMENTS

A non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviours. Especially these are the constraints the system must work within. Following are the non-functional requirements.

The system should

- Provide transparency and security with blockchain.
- Updating the status of block mining as fast as possible.
- Not allow anyone to tamper the smart-contract and achieve a mischievous act.
- Provide cross-platform login and viewing access.

2.5. SOFTWARE REQUIREMENTS

- **Operating System** Any OS capable of running a browser (Mac, Windows, Linux)
- **Front-end** HTML, CSS, JS(React JS)
- **Libraries** MUI, Web3.js, ether.js
- **Language** JS, Solidity
- **Tools Required** Metamask (Browser Extension), npm
- **IDEs** VS-Code, Remix-Ethereum IDE
- **Hosting** Github (code), Netlify (website hosting)

2.6. HARDWARE REQUIREMENTS

- **Processor** Intel core i3 and above
- **Hard Disk** 2 GB or above
- **RAM** 2 GB or above
- **Internet** 4 Mbps or above *(Wired or Wireless)*.

CHAPTER – 3

LITERATURE SURVEY

3. LITERATURE SURVEY

The process of summarization of the previous research works on the particular field of research. The aim of every research is deriving a new solution or invention.

The below table gives the overview of the central concept illustrated in the research papers collected on the project title.

S.No.	Title	Author	Publisher	Concept
1	Blockchain based Crowdfunding	Viren Patil Vasvi Gupta Rohini Sarode	IEEE - 2022	The authors gave the - Gave the overview of the project along with implementation screenshots with architecture diagram, class-diagram and state-diagram.
2	BitFund: A Blockchain-based Crowd Funding Platform for Future Smartand Connected Nation	Vikas Hassija, Vinay hamola,and Sherali Zeadally	Research Gate	The authors Proposed a new system similar to the freelancers system in which "**Backers can give project to the developers**". The developers will bid for the project, the one who bids best as per the requirements of backer, wins.
3	Role of Blockchain Technology in Crowdfunding	Atluri Divija Choudary	ICMEF	The author gave case studies of 2 successful startups and 2 failures raised via crowdfunding, along with blockchain based solutions to those.
4	Smart Contract and	Firmansyah Ashari1,Tetuko	WARSE	Shown 3 popular crowdfunding

	Blockchain for Crowdfunding Platform	Catonsukmoro, Wilyu Mahendra Bad,Sfenranto, Gunawan wang		schemes (traditional) - Way to increase trust - via offline registration, and provided legal ways (waiting few days, for verifying..).
5	Blockchain based crowdfunding systems	Md NazmusSaadat, Syed Abdul Halim, Husna Osman, Rasheed Mohammad Nassr,Megat F. Zuhairi	IAES	Presented with the implementation in MERN stack of platform along with detailed flow and necessary algorithms.
6	Application of blockchain technology in crowdfunding - a case study of the eu - researchgate	Micheal Gebert	Research Gate	A thorough case study on crowd funding in Europe
7	Crowdfunding Platform Using Blockchain Technology	Dr.R.Senthamil Selvi, SuryaPrakash R , Vishnu , Priyadharsana, S.Prasanna Venkateshwar D B	IJIRT	Highlighted few potential issues of current crowdfunding platforms and proposed solution using Blockchain along with description of few main use-cases.

Table 3.1- Literature survey of project

CHAPTER – 4

SYSTEM DESIGN

4. SYSTEM DESIGN

The Unified Modeling Language allows the software engineer to express an analysis model using the modeling notation that is governed by a set of syntactic, semantic and pragmatic rules. A UML system is represented using five different views that describe the system from distinctly different perspective. Each view is defined by a set of diagram, which is as follows:

1. **User Model View** This view represents the system from the users' perspective. The analysis representation describes a usage scenario from the end-users' perspective.
2. **Structural Model View** In this model, the data and functionality are arrived from inside the system. This model view models the static structures.
3. **Behavioral Model** View It represents the dynamic of behavioral as parts of the system, depicting he interactions of collection between various structural elements described in the user model and structural model view.
4. **Implementation Model View** In this view, the structural and behavioral as parts of the system are represented as they are to be built.
5. **Environmental Model** View In this view, the structural and behavioral aspects of the environment in which the system is to be implemented are represented.

UML Diagrams

4.1 Use Case Diagram

To model a system, the most important aspect is to capture the dynamic behavior. To clarifya bit in details, dynamic behavior means the behavior of the system when it is running/operating. So only static behavior is not sufficient to model a system rather dynamic behavior is more important than static behavior. In UML there are five diagrams available to model dynamic nature and use case diagram is one of them. Now as we have to discuss that the use case diagram is dynamic

in nature there should be some internal or external factors for making the interaction.

These internal and external agents are known as actors. So use case diagrams are consisting of actors, use cases and their relationships. The diagram is used to model the system/subsystem of an application. A single use case diagram captures a particular functionality of a system. So to model the entire system numbers of use case diagrams are used.

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. So when a system is analyzed to gather its functionalities use cases are prepared and actors are identified. In brief, the purposes of use case diagrams can be as follows:

1. Used to gather requirements of a system.
2. Used to get an outside view of a system.
3. Identify external and internal factors influencing the system.
4. Show the interacting among the requirements are actors.

Actors of the system

There are 3 main actors who interact with the system. They are:

1. Public viewer
2. Fund Raiser
3. **Backer (Contributor)**

- **Use-cases of viewer**

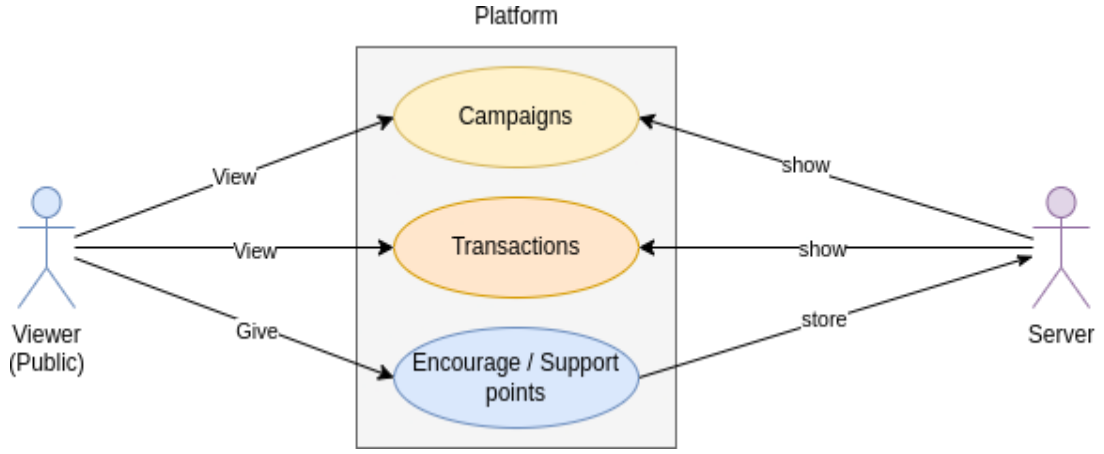

Figure 4.1: Viewing campaigns, transactions and giving encourage/support points

- **Use cases of Fundraiser**

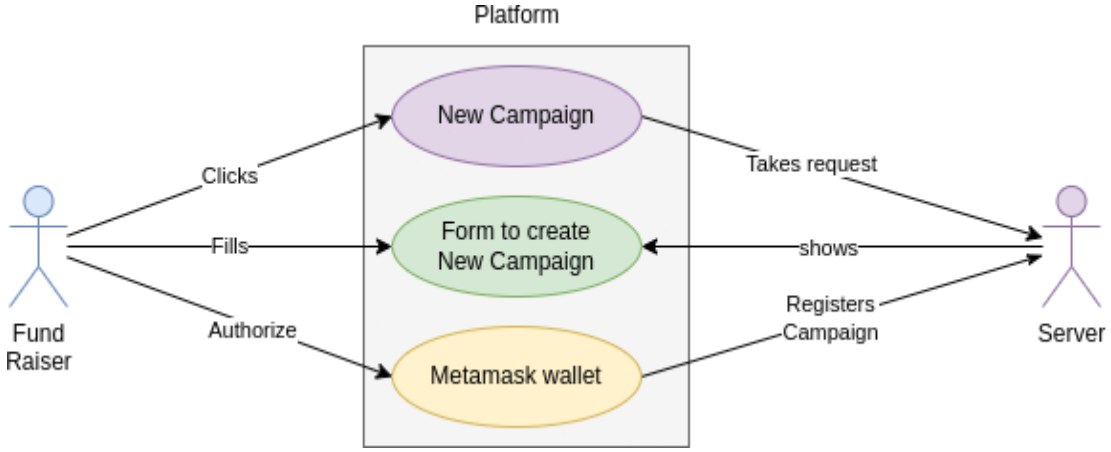

Figure 4.2: New Campaign Creation

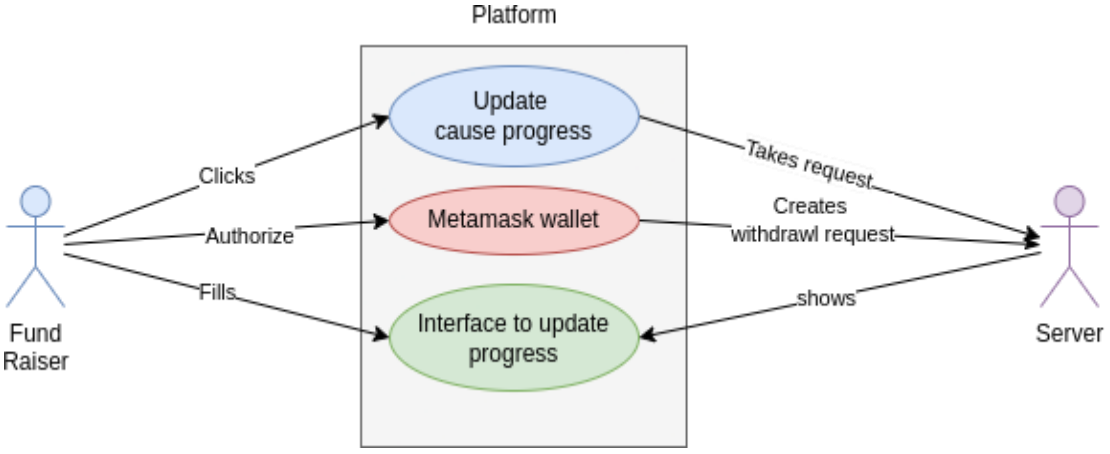

Figure 4.3: Updating the progress of cause

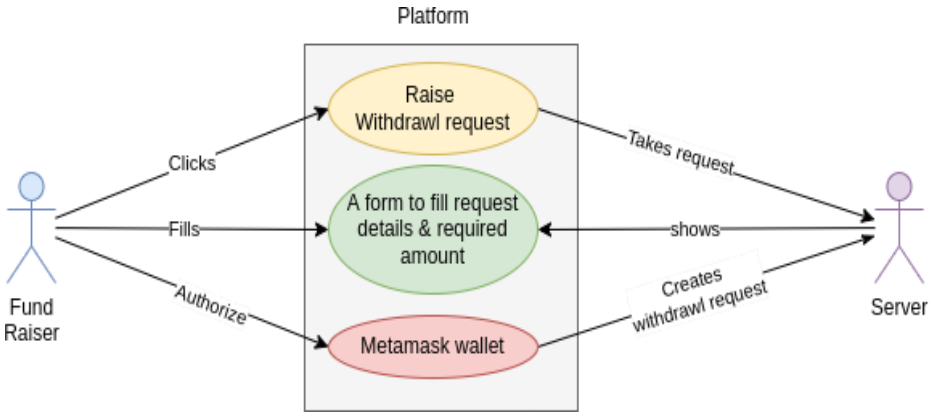

Figure 4.4: Raising withdrawal request by connecting recipient's wallet address

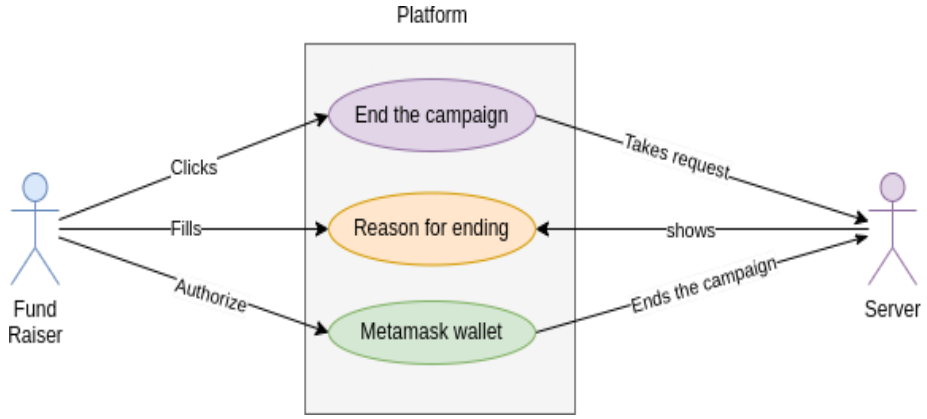

Figure 4.5: Ending campaigns

- **Use cases of the backer**

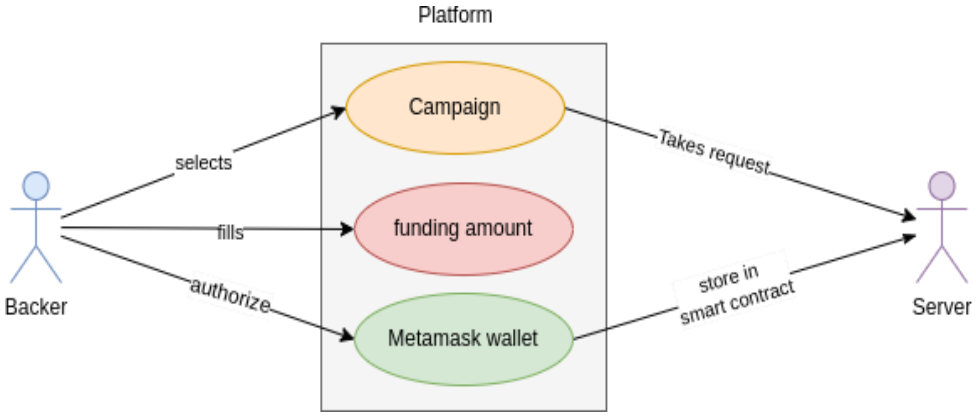

Figure 4.6: Contributing to campaigns

4.2. Class Diagram

The UML Class diagram is a graphical notation used to construct and visualize object oriented systems. A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's: classes, their attributes, operations (or methods), and the relationships among objects.

A Class is a blueprint for an object. Objects and classes go hand in hand. We can't talk about one without talking about the other. And the entire point of Object-Oriented Design is not about objects, it's about classes, because we use classes to create objects. So a class describes what an object will be, but it isn't the object itself. In fact, classes describe the type of objects, while objects are usable instances of classes.

Class diagram of the application.

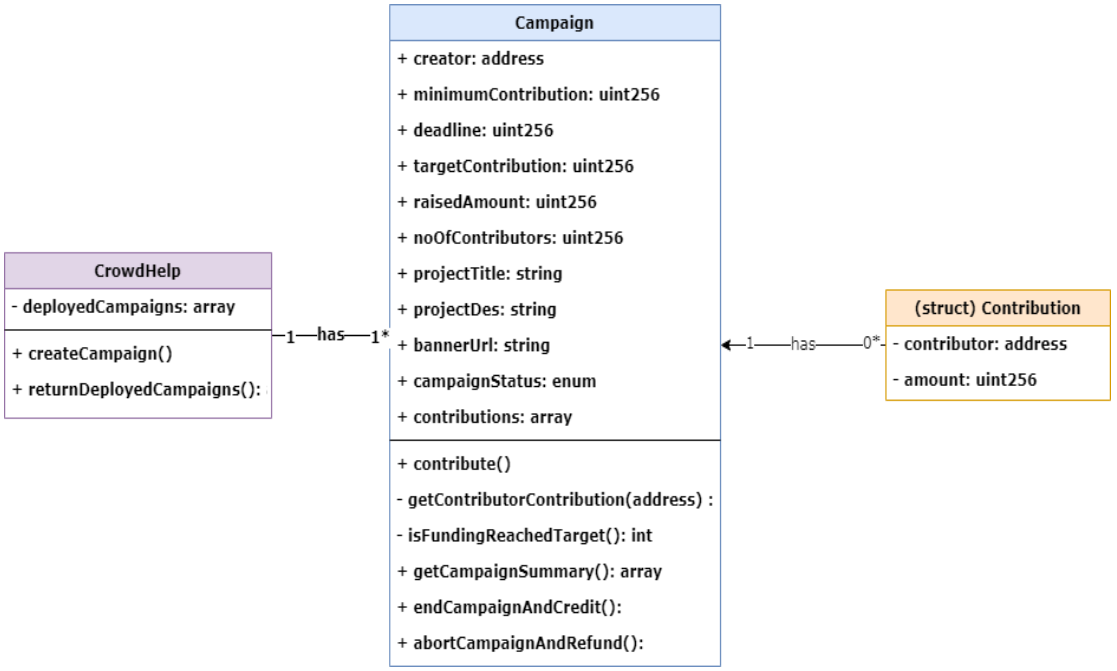

Figure 4.8: Class diagram showing various classes (contract in solidity)

Description:

CrowdHelp contract (class) is represents this whole application. It contains a method called createCampaign(), in which it instantiates the Campaign contract (class) for every new create campaign call. After successful instantiation, a new address will be returned, and this is stored in a private array deployedCampaigns of type address. From Homepage, the method returnDeployedCampaigns() will be called, which returns all the addresses of deployed campaigns, through which each individual campaign can be accessed.

Campaign contract (class) contains all the public attributes as all those will be accessed outside of the contract.

Method contribute() will be called when a backer initiates contribution. It checks whether the contribution amount is >= minimumContribution, which is set at the time of creation. If passed, it adds the backer's wallet address in contributions array along with the amount they contributed. If a single backer contributes multiple times, then their existing amount will be incremented. Finally, raisedAmount will be incremented by the contribution amount.

Method endCampaignAndSummary() will get called when by the fundraiser. This will internally check, whether

1. The deadline has passed or not &
2. Campaign has reached target amount or not.

If passed, credits all the contributed amount to the fund raiser's wallet address, which is stored in contributor value.

If fails, reports with error.

After successful end, it changes the campaignStatus value from SUCCESS to EXPIRED. This method internally takes the help of getContributorContribution() method to check whether he is the existing backer or not.

It also takes help of one more internal method named isFundingReachedTarget(). This gets called at the end, to update the status from ACTIVE to SUCCESS.

Method abortCampaignAndRefund() will be called by fundraiser when the creator wants to abort campaign before reaching deadline. In this case all the contributed amount (if any), will be transferred back to contributors. This method internally takes the help of getContributorContribution() method. It uses the contributions array to

get how many backers have contributed & with what amount (wholesome). Even if the backer contributed multiple times, all his amount will be transferred as whole in a single go.

4.3. SEQUENCE DIAGRAMS

A sequence diagram is an **interaction diagram** that emphasizes the time-ordering of messages. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

Sequence diagrams describe interactions among classes in terms of an exchange of messages over time. They're also called event diagrams. A sequence diagram is a good way to visualize and validate various runtime scenarios. These can help to predict how a system will behave and to discover responsibilities a class may need to have in the process of modelling a new system.

The aim of a sequence diagram is to define event sequences, which would have a desired outcome. The focus is more on the order in which messages occur than on the message per se. However, the majority of sequence diagrams will communicate what messages are sent and the order in which they tend to occur.

Basic Sequence Diagram Notations Class Roles or Participants

Class roles describe the way an object will behave in context. Use the UML object symbol to illustrate class roles, but don't list object attributes.

Activation or Execution Occurrence

Activation boxes represent the time an object needs to complete a task. When an object is busy executing a process or waiting for a reply message, use a thin grey rectangle placed vertically on its lifeline.

Messages

Messages are arrows that represent communication between objects. Use half arrowed lines to represent asynchronous messages. Asynchronous messages are sent from an object that will not wait for a response from the receiver before continuing its tasks.

Lifelines

Lifelines are vertical dashed lines that indicate the object's presence over time.

Destroying Object

Objects can be terminated early using an arrow labelled "<< destroy >>" that points to an X. This object is removed from memory. When that object's lifeline ends, you can place an X at the end of its lifeline to denote a destruction occurrence.

Loops

A repetition or loop within a sequence diagram is depicted as a rectangle. Place the condition for exiting the loop at the bottom left corner in square brackets [].

Guards

When modelling object interactions, there will be times when a condition must be met for a message to be sent to an object. Guards are conditions that need to be used throughout UML diagrams to control flow.

Sequence diagrams of the system:

1. Campaign creation by fundraiser.
2. Contributing to campaign by backer
3. Withdraw requests by fundraiser (currently, this is shifted to next version)

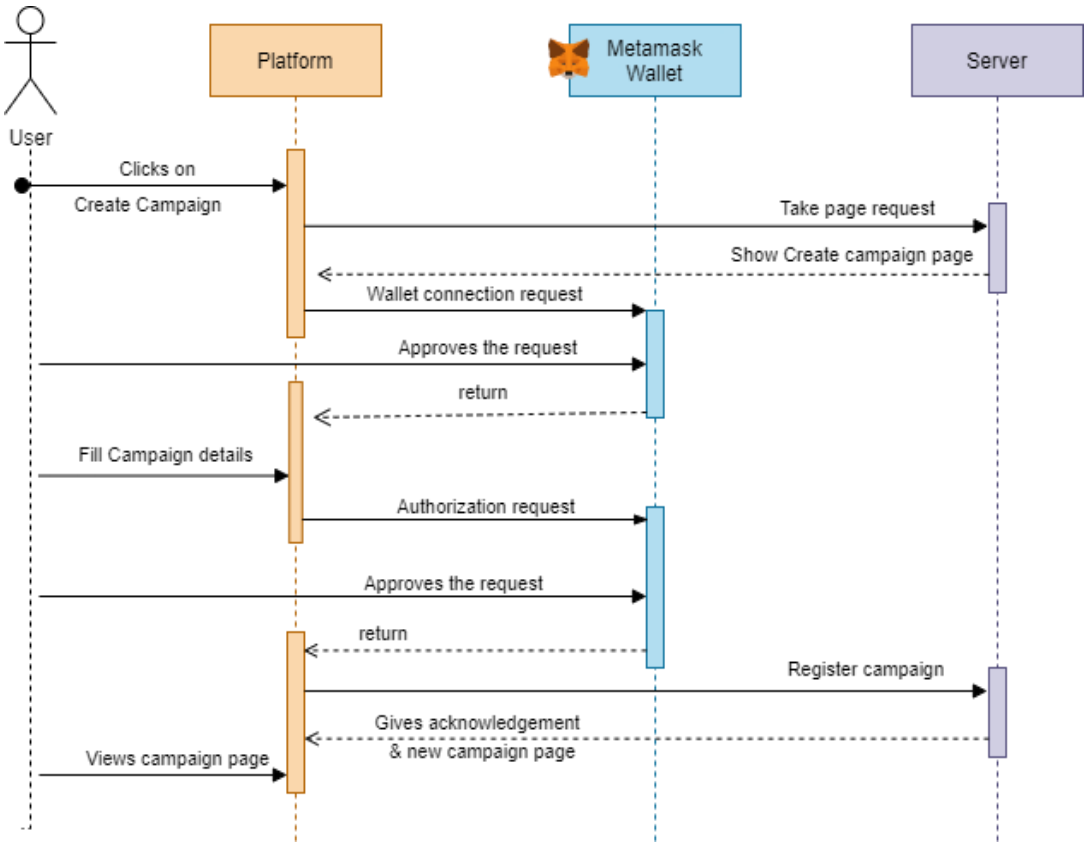

Figure 4.9: New Campaign creation by fund raiser

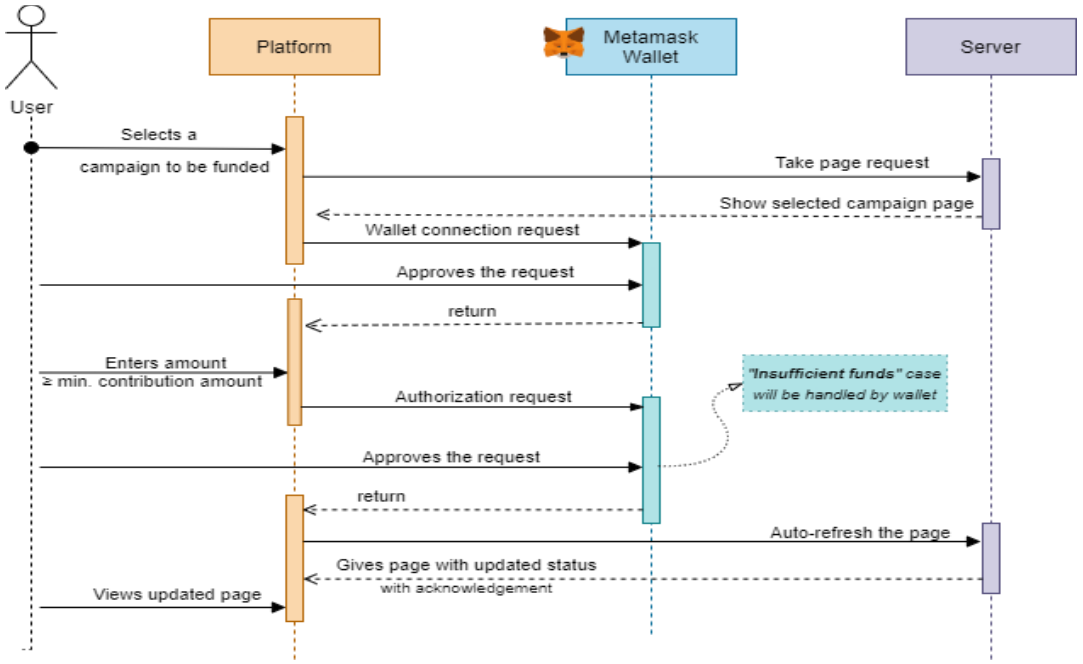

Figure 4.10: Backers contributing to the campaign

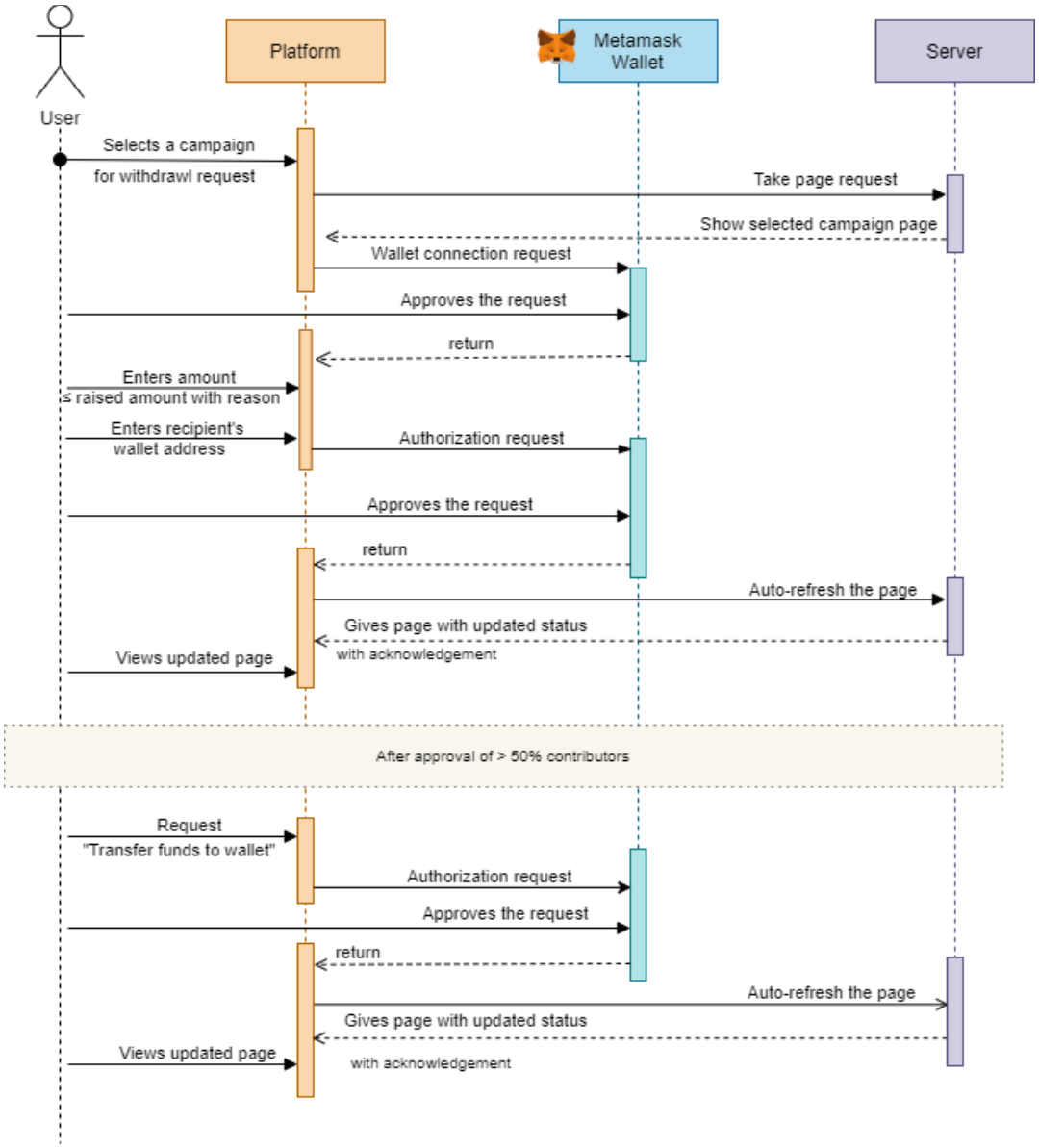

Figure 4.11: Raising withdraw request & transferring the funds to wallet (after approval).

4.4. ACTIVITY DIAGRAMS

Activity diagram is another important **behavioral diagram** in UML diagram to describe dynamic aspects of the system. Activity diagram is essentially an advanced version of flow chart that modeling the flow from one activity to another activity.

Activity Diagrams describe how activities are coordinated to provide a service which can be at different levels of abstraction. Typically, an event needs to be achieved by some operations, particularly where the operation is intended to achieve a number of different things that require coordination, or how the events in a single use case relate to one another, in particular, use cases where activities may overlap and require

coordination. It is also suitable for modeling how a collection of use cases coordinate to represent business workflows Identify candidate use cases through the examination of business workflows ,Identify pre- and post-conditions (the context) for use cases Model workflows between/within use cases

Activity diagrams of the application

1. Activity Diagram - Creating a new campaign

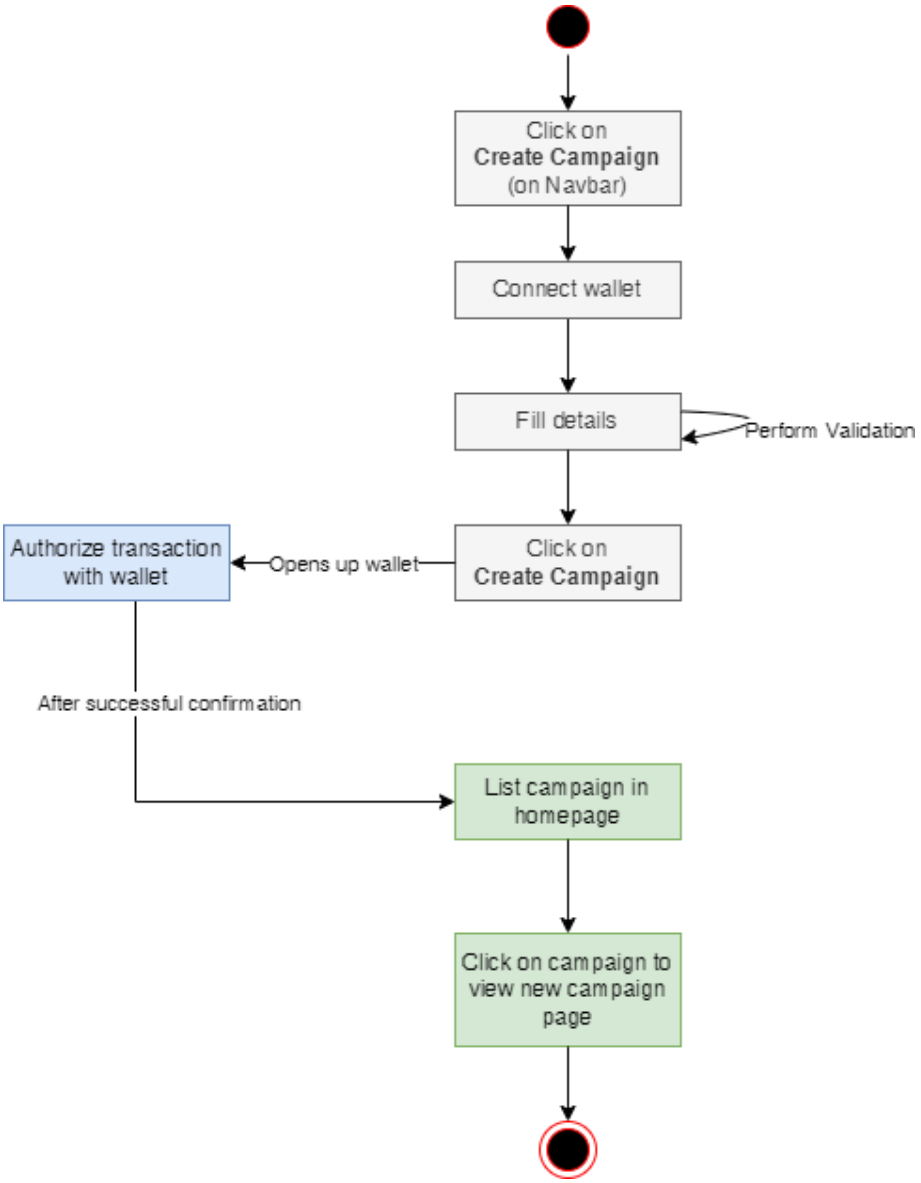

Figure: 4.12: Activity diagram showing the various activities while creating a campaign.

2. Activity Diagram – Contributing to a campaign

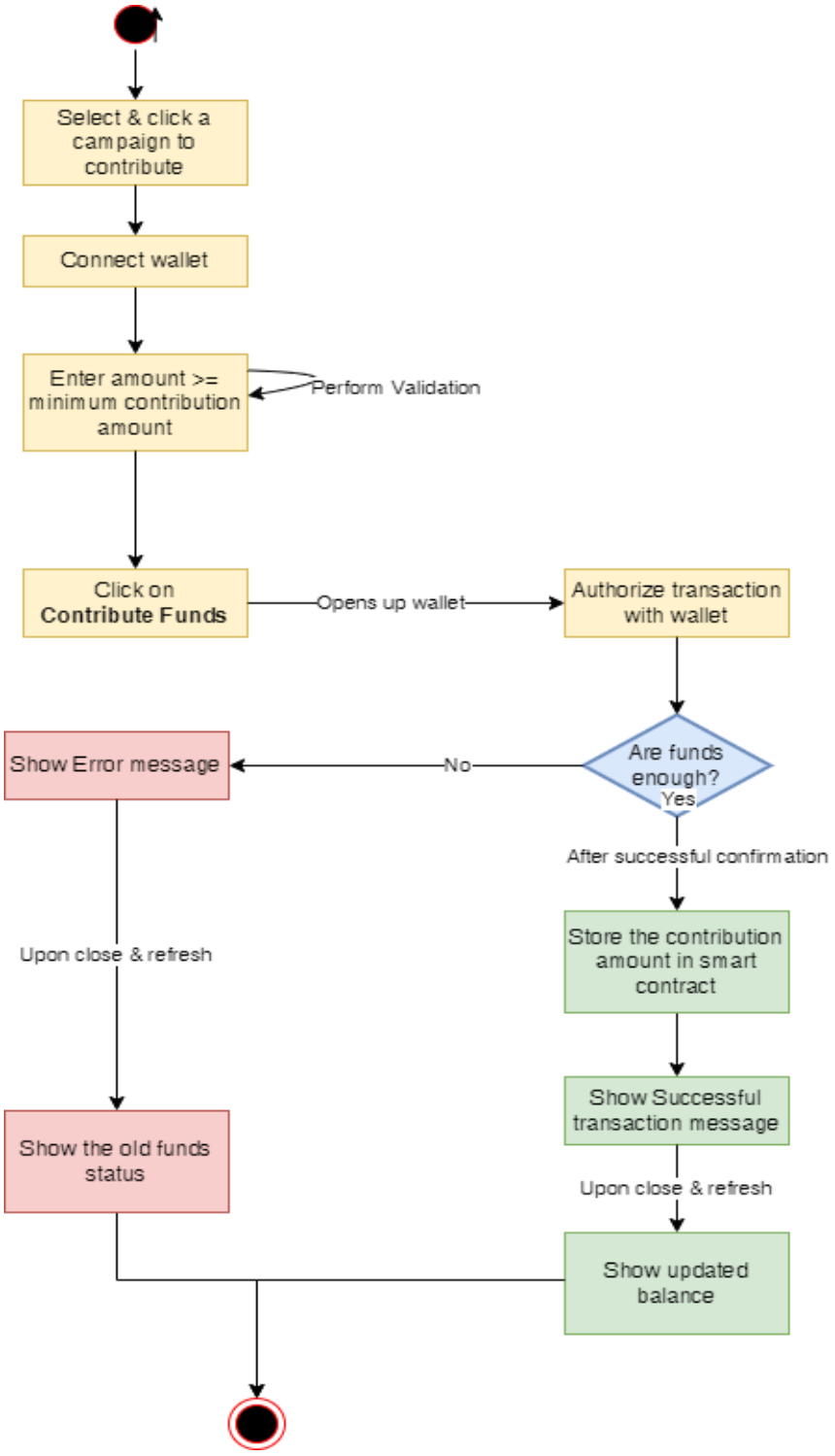

Figure: 4.13: Activity diagram showing the various activities while contributing to a campaign.

3. Activity Diagram – Ending a campaign & Withdrawing raised funds

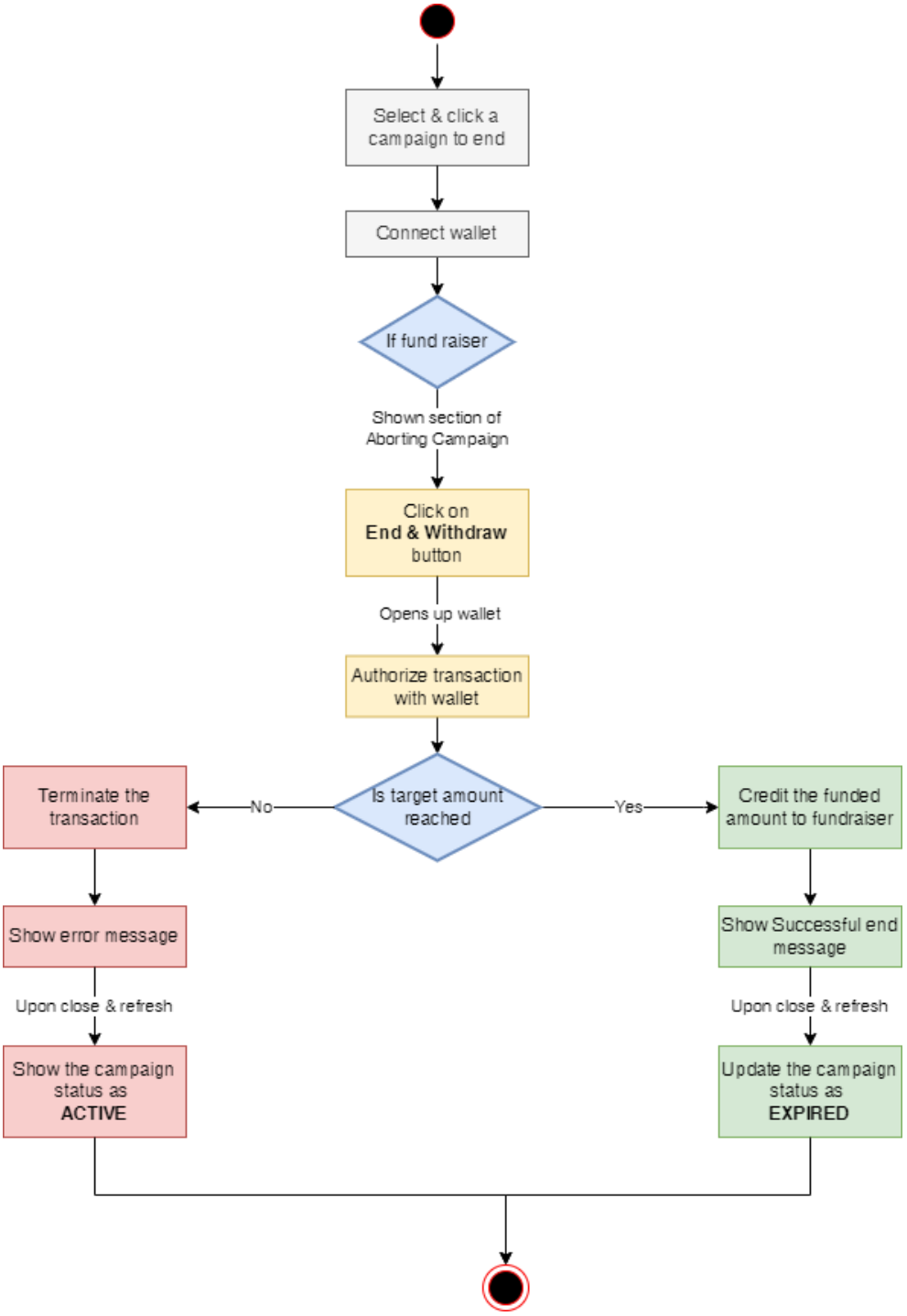

Figure: 4.14: Activity diagram showing the various activities while ending a campaign & withdrawing funds.

4. Activity Diagram – Aborting a campaign & Refunding raised funds

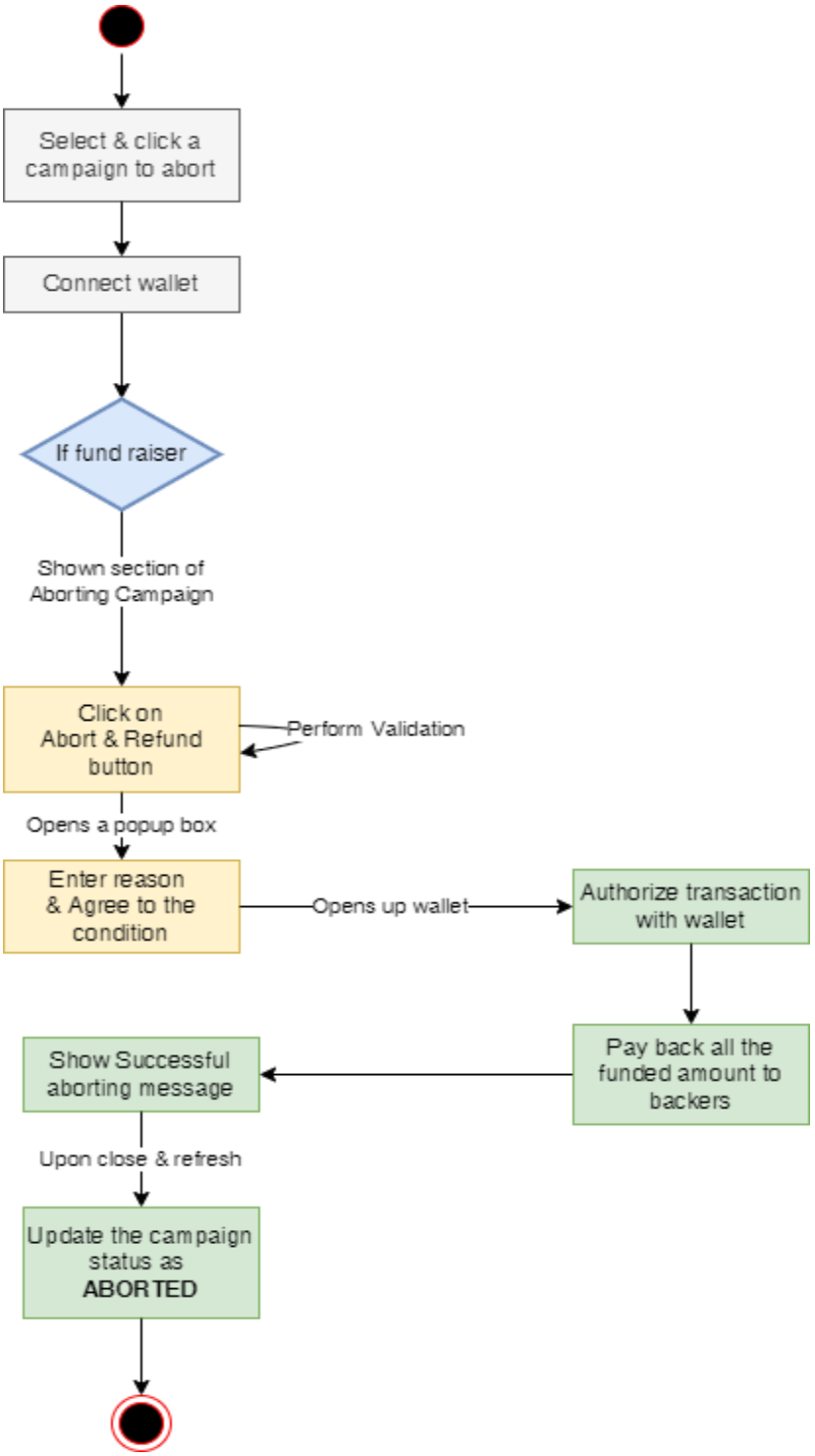

Figure: 4.15: Activity diagram showing the various activities while aborting a campaign & refunding contributed amount to backers.

CHAPTER – 5

IMPLEMENTATION

5. IMPLEMENTATION

5.1. CODE SNIPPET

Main code snippet is the Solidity Smart contract, which is the core of the application. It is

```
//SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract CrowdHelp{

// events

event CampaignStarted(

    address projectContractAddress ,

    address creator,

    uint256 minContribution,

    uint256 projectDeadline,

    uint256 goalAmount,

    uint256 currentAmount,

    uint256 noOfContributors,

    string title,

    string desc,

    uint256 currentState

);

event ContributionReceived(

   address projectAddress,

   uint256 contributedAmount,

   address indexed contributor

);

Campaign[] private deployedCampaigns;

 // @dev Anyone can start a fund rising

 // @return null
```

```
function createCampaign(
        string memory projectTitle,
        string memory projectDesc,
        uint256 minimumContribution,
        uint256 targetContribution,
        uint256 deadline,
        string memory bannerUrl     )
public {
 // deadline = deadline;

Campaigncampaign=newCampaign(msg.sender,minimumContribution,deadline,targetContribut
ion,projectTitle,projectDesc, bannerUrl);
 deployedCampaigns.push(campaign);     }
  // @dev Get deployedCampaigns list
  // @return array
function returnDeployedCampaigns() external view returns(Campaign[] memory){
return deployedCampaigns;
    }
}contract Campaign{
  // Campaign state
enum State {
 ACTIVE,     // Campaign can receive funds / contribution
 SUCCESS,    // Campaign has reached its goal before the deadling
 EXPIRED,    // Campaign has withdrawn amount of successfully ended campaign
ABORTED     // Campaign has terminted in-between, all the raised amount has refunded back to
backers.
    }  struct Contribution{
        address payable contributor;
        uint256 amount;
```

```
    // Variables
address payable public creator;
uint256 public minimumContribution;
uint256 public deadline;
uint256 public targetContribution; // required to reach at least this much amount
uint public reachedTargetAt; // stores when it has ended (by the fundraiser)
    uint256 public raisedAmount; // Total raised amount till now
    uint256 public noOfContributors;
    string public projectTitle;
    string public projectDes;
    string public bannerUrl;
    State public state = State.ACTIVE;


 // mapping (uint => Contribution) public contributors;          // use indices as address with amountContributed as its values.
    Contribution [] contributions;
    // after work... replace this with contributions.. -- as of type array
    // Modifiers
    modifier isCreator(){
  require(msg.sender == creator,'You dont have access to perform this operation !');_;
    } modifier canContribute(){
  require(state == State.ACTIVE || state  == State.SUCCESS ,'Invalid state');
 require(block.timestamp < deadline,'Sorry backer, deadline has passed! No contributions can be accepted now.')     _;
    // Events
    // Event that will be emitted whenever funding will be received
    event FundingReceived(address contributor, uint amount, uint currentTotal);
    // Event gets emitted when amount gets credited to fund raiser - when campaign has ended
    event AmountCredited(address contributor, uint amountTotal);
    // Event gets emitted when campaign gets aborted [before deadline]
```

```
  event AmountRefunded(uint noOfContributors, uint amountTotal);

  constructor(
      address _creator,
      uint256 _minimumContribution,
      uint256 _deadline,
      uint256 _targetContribution,
      string memory _projectTitle,
      string memory _projectDes,
      string memory _bannerUrl
  ) {
      creator = payable(_creator);
      minimumContribution = _minimumContribution;
      deadline = _deadline;
      targetContribution = _targetContribution;
      projectTitle = _projectTitle;
      projectDes = _projectDes;
      raisedAmount = 0;
      bannerUrl=_bannerUrl;
  }

 function getContributorContribution(address  _contributor) internal view returns(int){

// Takes the contributor's address and returns their contribution amount (if found) else 0.

       int contribIndex=-1;

       for(int idx=0; idx<int(noOfContributors); idx++){

          if(_contributor == contributions[uint(idx)].contributor){

              contribIndex =  idx;

              break;

          } }

       return contribIndex;    // when not found -- send "-1" to indicate as not found. NOTE: Only
this is THE negative value used.     }
```

```
    // @dev Anyone can contribute
    function contribute() public canContribute() payable {
        // validation
        require(msg.value >= minimumContribution,'Contribution amount is too low !');
        address payable contributor = payable(msg.sender);
        int contributionIdx = getContributorContribution(contributor);
        if(contributionIdx == -1){    // if contributing for the first time..
            noOfContributors++;
  contributions.push(Contribution(contributor, msg.value));// store the amount of funds     funded
        }
        // if contributed already..
        else{
            contributions[uint(contributionIdx)].amount += msg.value;   // update the contribution
        }
        // contributors[contributor] += msg.value;      // store the amount of funds funded -- older
version
        // update the global value
        raisedAmount += msg.value;
        emit FundingReceived(contributor,msg.value,raisedAmount);
        isFundingReachedTarget();
    }
  function isFundingReachedTarget() internal {
        if(raisedAmount >= targetContribution && block.timestamp < deadline){
            state = State.SUCCESS;
            reachedTargetAt = block.timestamp;
        }
    }
    function endCampaignAndCredit() public isCreator() payable {
        // perform validation..
```

```
        require(state == State.SUCCESS, 'Goal not reached, amount cannot be withdrawn. Please abort to refund to backers');

        require(block.timestamp < deadline, 'Campaign cannot be ended, deadline not reached.');

        // transfer the WHOLE amount raised to fund raiser

        creator.transfer(raisedAmount);

        // Update the campaign state

        state = State.EXPIRED;

        // emit the event..

        emit AmountCredited(creator, raisedAmount); }

    function abortCampaignAndRefund() public isCreator() payable{

        // perform validation..

        require(block.timestamp < deadline, 'Campaign cannot be aborted, deadline passed.');

        require(state == State.ACTIVE || state == State.SUCCESS, 'Invalid state. Cannot abort campaign.');

        // refund money to backers

        for(uint idx=0; idx<noOfContributors; idx++){ // iterate through all addresses of contrdibutors

            contributions[idx].contributor.transfer(contributions[idx].amount);

            delete contributions[idx];        }

        // update the state..

        state = State.ABORTED;

        // emit the event..

        emit AmountRefunded(noOfContributors, raisedAmount);

    }

  function getContractBalance() public view returns(uint256){

        return address(this).balance;

    }   function getCampaignSummary() public view returns(

    address payable projectStarter,

    uint256 minContribution,

    uint256  projectDeadline,
```

```
    uint256 goalAmount,
    uint completedTime,
    uint256 currentAmount,
    string memory title,
    string memory desc,
    State currentState,
    uint256 balance,
    string memory imageUrl,
    uint256 numBackers
    ){
        projectStarter=creator;
        minContribution=minimumContribution;
        projectDeadline=deadline;
        goalAmount=targetContribution;
        completedTime=reachedTargetAt;
        currentAmount=raisedAmount;
        title=projectTitle;
        desc=projectDes;
        currentState=state;
        balance=address(this).balance;
        imageUrl=bannerUrl;
        numBackers=noOfContributors;
    }
}
```

CHAPTER – 6

TESTING

6.1. INTRODUCTION TO TESTING

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. Testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

According to ANSI/IEEE 1059 standard, Testing can be defined as - A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.

Who does Testing?

It depends on the process and the associated stakeholders of the project(s). In the IT industry, large companies have a team with responsibilities to evaluate the developed software in context of the given requirements. Moreover, developers also conduct testing which is called Unit Testing. In most cases, the following professionals are involved in testing a system within their respective capacities:

- Software Tester
- Software Developer
- Project Lead/Manager
- End Use

Levels of testing include different methodologies that can be used while conducting software testing.

The main levels of software testing are:

- Functional Testing
- Non-functional Testing
- Functional Testing

This is a type of black-box testing that is based on the specifications of the software that is to be tested. The application is tested by providing input and then the results are examined that need to conform to the functionality it was intended for. Functional

testing of a software is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.

Software Testing Life Cycle

The process of testing a software in a well-planned and systematic way is known as software testing lifecycle (STLC). Different organizations have different phases in STLC however generic Software Test Life Cycle (STLC) for waterfall development model consists of the following phases.

- Requirements Analysis
- Test Planning Test Analysis
- Test Design
- Requirements Analysis

In this phase testers analyze the customer requirements and work with developers during the design phase to see which requirements are testable and how they are going to test those requirements. It is very important to start testing activities from the requirements phase itself because the cost of fixing defect is very less if it is found in requirements phase rather than in future phases .

Test Planning

In this phase all the planning about testing is done like what needs to be tested, how the testing will be done, test strategy to be followed, what will be the test environment, what test methodologies will be followed, hardware and software availability, resources, risks etc. A high level test plan document is created which includes all the planning inputs mentioned above and circulated to the stakeholders. **Test Analysis**

After test planning phase is over test analysis phase starts, in this phase we need to dig deeper into project and figure out what testing needs to be carried out in each SDLC phase. Automation activities are also decided in this phase, if automation needs to be done for software product, how will the automation be done, how much time will it take to automate and which features need to be automated. Non-functional testing areas (Stress and performance testing) are also analyzed and defined in this phase.

6.2. TEST CASES DESIGN & EXECUTION

Test Scenario ID		Campaign Creation		**Test case ID**	TEST001	
Test case description		Testing creation		**Test Priority**	HIGH	
Pre-Requisite		Wallet connection		**Post - Requisite**	NA	
S.No.	**Action**	**Inputs**	**Expected output**	**Actual Output Test Browser**	**Test Result**	**Test Comments**
1	Filling all the fields	Entering values in each field matching constraints.	Allow to create campaign	Allowing user to click Create Campaign button	Pass	NA
2	Filling all the fields	Missing few input fields	Restrict user to click	Showing validation error.	Pass	NA
3	Filling all the fields	Entered all the fields but mismatching constraints of the fields.	Restrict user to click **Create Campaign** button	Showing validation error.	Pass	NA
4	Authorizing creation	Insufficient funds in wallet to deploy the smart contract.	Report **Insufficient Funds** error	Report **Insufficient Funds** error.	Pass	NA
5	Authorizing creation	Sufficient funds in wallet to deploy the smart contract.	Authorizing with wallet	Report **Insufficient Funds** error.	Pass	NA

Table 6.1: Campaign Creation

Test Scenario ID		Contributing to campaign		**Test case ID**	TEST002	
Test case description		Testing contribution		**Test Priority**	HIGH	
Pre-Requisite		Wallet connection		**Post - Requisite**	NA	
S.No	**Action**	**Inputs**	**Expected output**	**Actual Output Test Browser**	**Test Result**	**Test Comments**
1	Contributing funds	Entering < minimum amount & Clicking Contribute Funds button.	Should not allow to pass to next step.	Showing validation error.	Pass	NA
2	Contributing funds	Entering >= minimum amount & Clicking Contribute Funds button.	Should allow to pass to next step.	Allowing user to authenticate the transaction with wallet.	Pass	NA
3	Contributing funds	Entering >= target amount & Clicking Contribute Funds button.	Should allow to pass to next step.	Allowing user to authenticate the transaction with wallet.	Pass	NA

Table 6.2: Contributing to campaign

Test Scenario ID		Campaign Ending		**Test case ID**	TEST003	
Test case description		Testing ending		**Test Priority**	HIGH	
Pre-Requisite		Connected with fundraiser's wallet account.		**Post - Requisite**	NA	
S.No.	**Action**	**Inputs**	**Expected output**	**Actual Output Test Browser**	**Test Result**	**Test Comments**
1	Ending Campaign	Clicking End campaign button (Before deadline)	Transaction fail	Reporting the transaction failed error.	PASS	NA
2	Ending Campaign	Clicking End campaign button (After deadline & campaign has reached the target)	Transaction success & amount has transferred to fund raiser's account.	Amount crediting to fund raiser's account.	PASS	NA
3	Ending Campaign	Clicking End campaign button (After deadline & campaign has **not** reached the target)	Transaction success & amount has transferred to fund raiser's account.	Amount crediting to fund raiser's account.	FAIL	NA

Table 6.3: Campaign Ending

Test Scenario ID		Campaign Aborting		**Test case ID**	TEST004	
Test case description		Testing aborting		**Test Priority**	HIGH	
Pre-Requisite		Connected with fundraiser's wallet account.		**Post - Requisite**	NA	
S.No.	**Action**	**Inputs**	**Expected output**	**Actual Output Test Browser**	**Test Result**	**Test Comments**
1	Aborting campaign	**Clicking Abort Campaign button (before deadline)**	Allow to pass to next step.	Showing wallet to authorize transaction	PASS	NA
2	Aborting campaign	**Clicking Abort Campaign button (after deadline)**	Should not allow to pass to next step.	Showing wallet to authorize transaction	PASS	NA
3	Aborting campaign	**Clicking Abort Campaign button (before deadline)**	Allow to pass to next step.	Showing wallet to authorize transaction	PASS	NA
4	Authorizing valid abort	Clicking **Confirm button**	Funds getting credited back to backers	Funds getting credited back to backers	PASS	NA

Table 6.4: Campaign Aborting

Test Scenario ID		Displaying Campaign		**Test case ID**	TEST005	
Test case description		Testing displaying		**Test Priority**	HIGH	
Pre-Requisite		Connected with fundraiser's wallet account.		**Post - Requisite**	NA	
S.No.	**Action**	**Inputs**	**Expected output**	**Actual Output Test Browser**	**Test Result**	**Test Comments**
1	Clicking on campaign card	Clicking on a campaign card in home page	Showing respective campaign page	Showing respective campaign page	PASS	NA

Table 6.5: Displaying Campaign

CHAPTER – 7

SCREENSHOTS OF THE APPLICATION

7. SCREENSHOTS OF APPLICATION

HOME PAGE

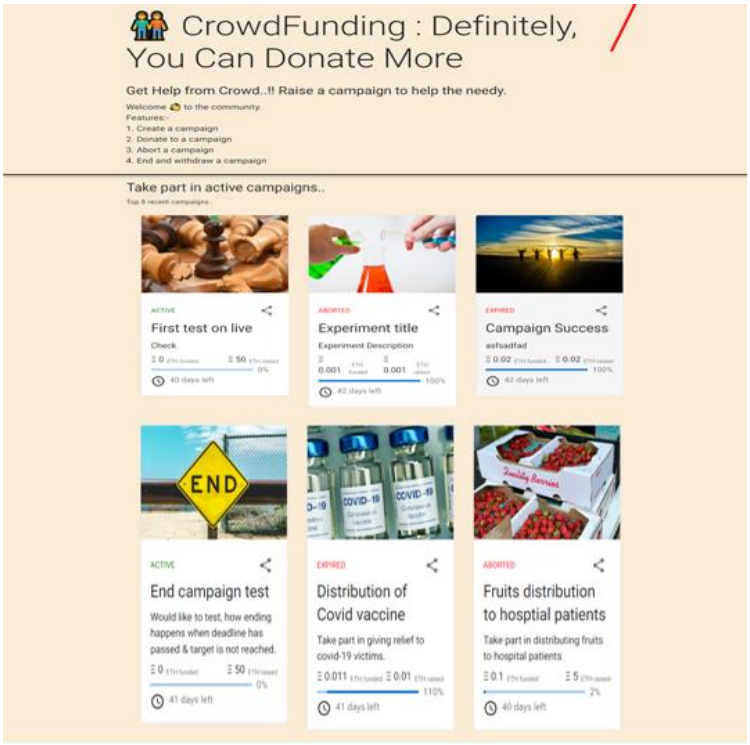

Figure 7.0: Showing list of campaigns in Home Page

7.1. WALLET CONNECTION

1. Click on **Connect Wallet** at top-right in navbar.

Figure 7.1: Showing list of accounts to be connected

2. Select the accounts you would like to connect with the site.

3. Click on **Connect**

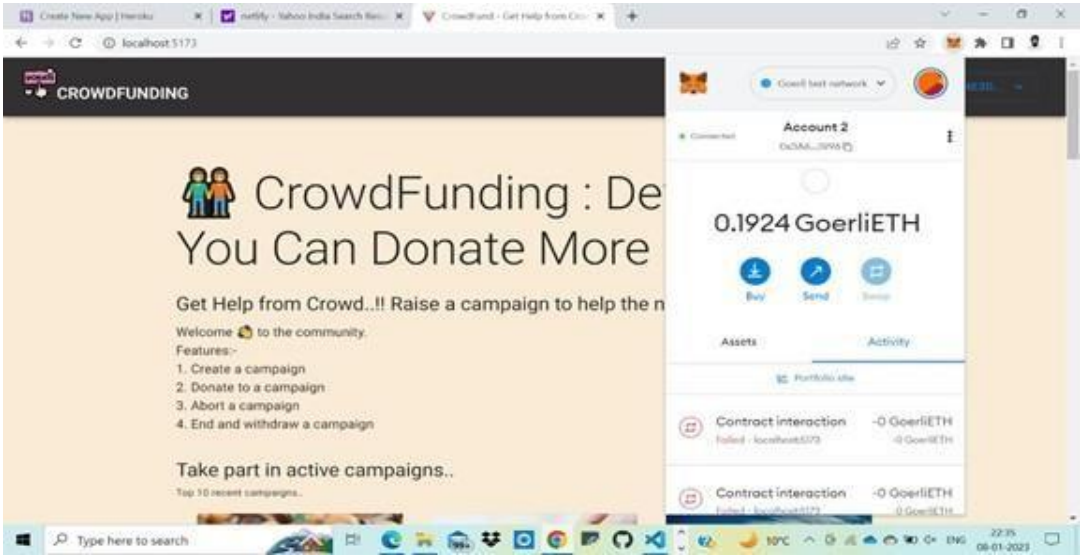

Figure 7.2: Account Information

- Notice at the top-right of navbar, that showing connnected wallet's address instead of **Connect wallet** button. The user can click on it, to get option to disconnect the wallet.

Figure 7.3: Displaying the wallet address at the top-right of navbar.

7.2. CAMPAIGN CREATION & DISPLAYING

1. Click on **Create campaign** button at top-right in navbar.

2. Fill the details.

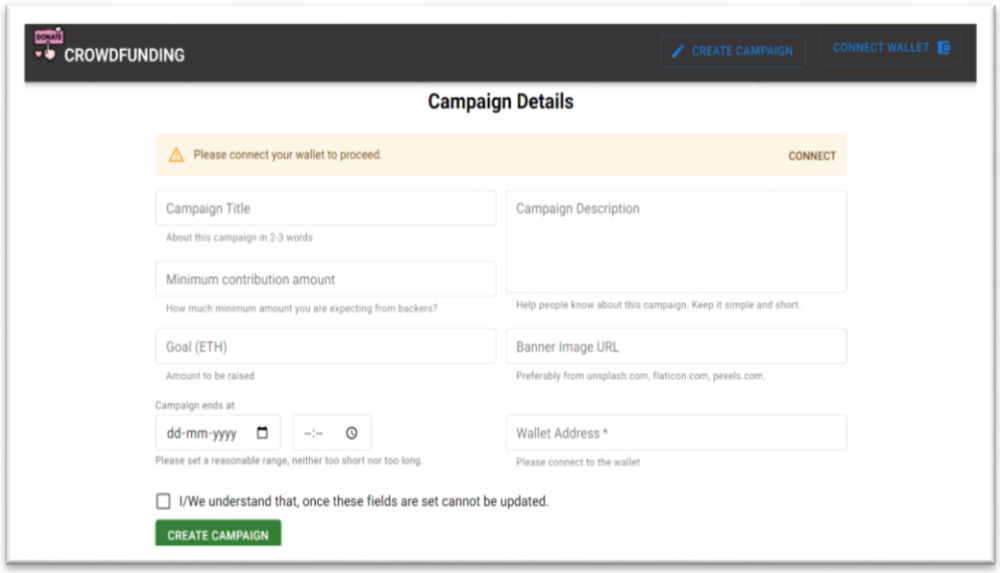

Figure 7.4: Showing the fields to be filled for campaign creation.

3. Authenticate with wallet.

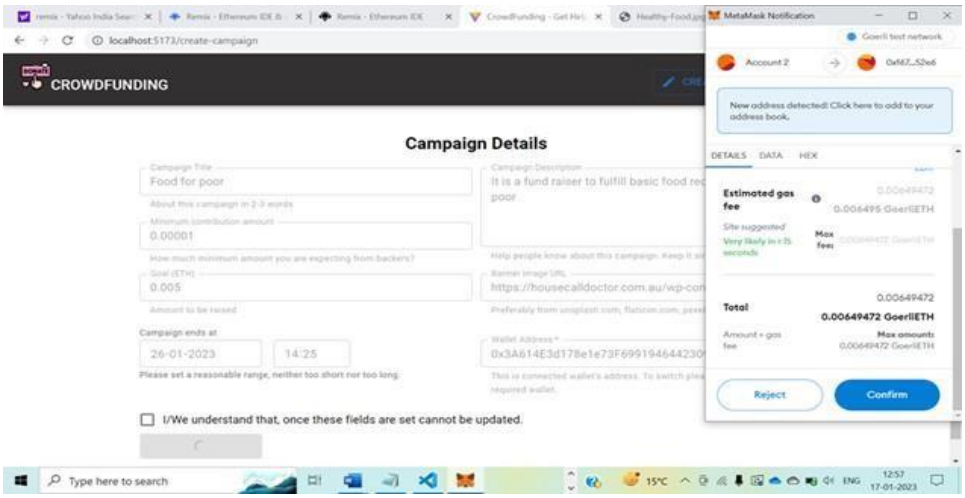

Figure 7.5: Wallet Authentication for creation

4. After successful creation, navigates to the homepage that shows the newly created campaign.

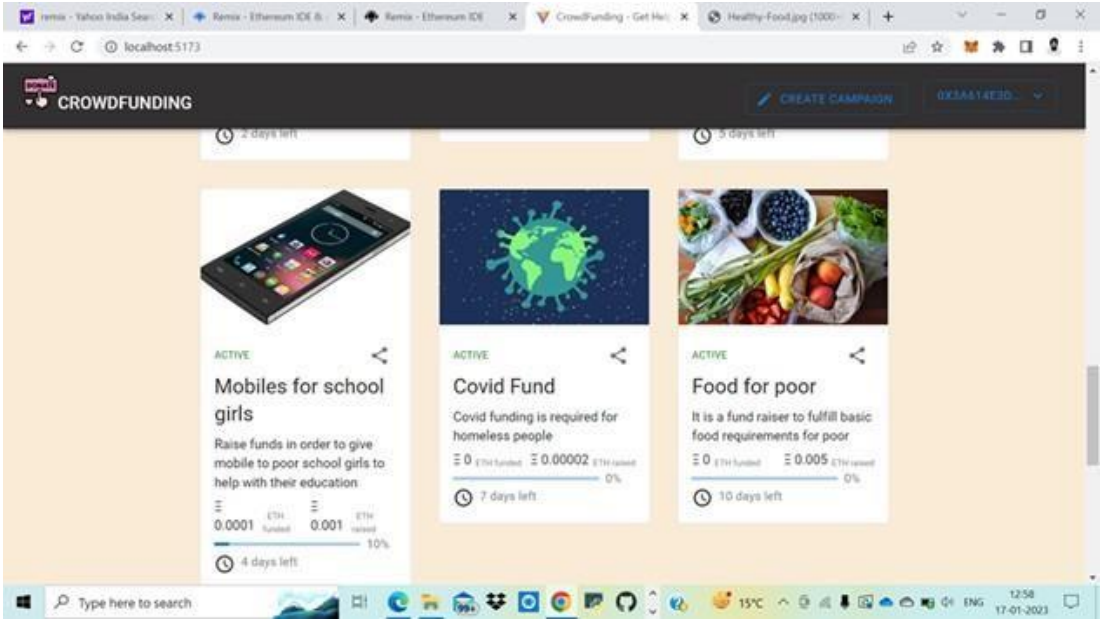

Figure 7.6: Showing the newly created campaign in home page.

5. [Page Appearance] When fundraiser clicks on the campaign which they had created. Notice that, both are of same addresses, the connected wallet address (at top-right) and in the *Wallet address of Fundraiser*.

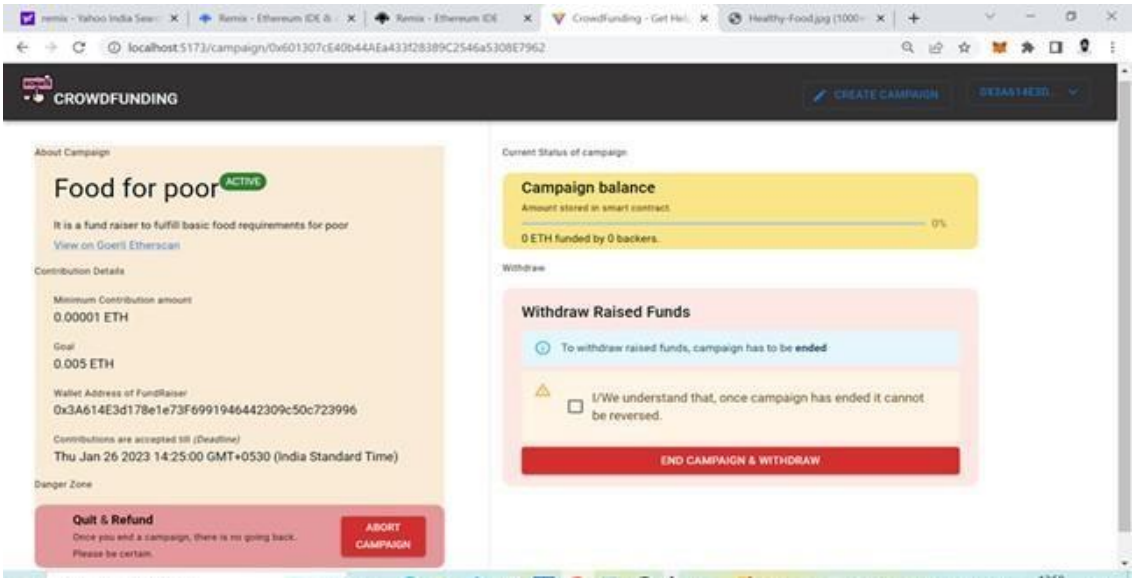

Figure 7.7: Campaign page for fund raiser

6. [Page Appearance] When public/backer clicks on the campaign which others had created. Notice that, both are of different addresses, the connected wallet address (at top-right) and in the Wallet address of Fundraiser.

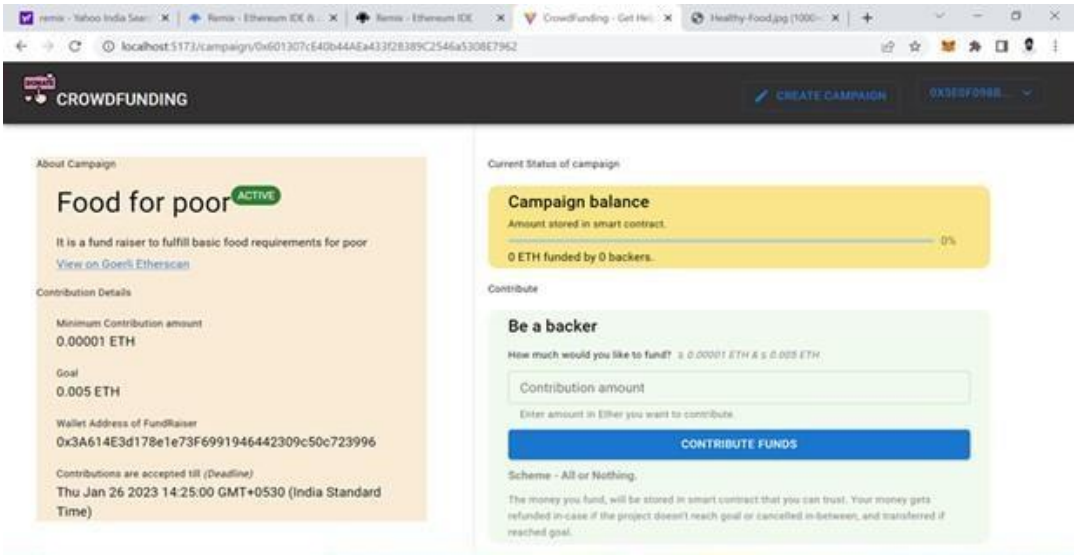

Figure 7.8: Campaign page for public.

7.3. CONTRIBUTING TO CAMPAIGNS

1. Select the campaign you would like to fund/contribute.
2. Enter the amount >= minimum amount and click on **Contribute funds**. Click on **Confirm** to finish authentication with the wallet.

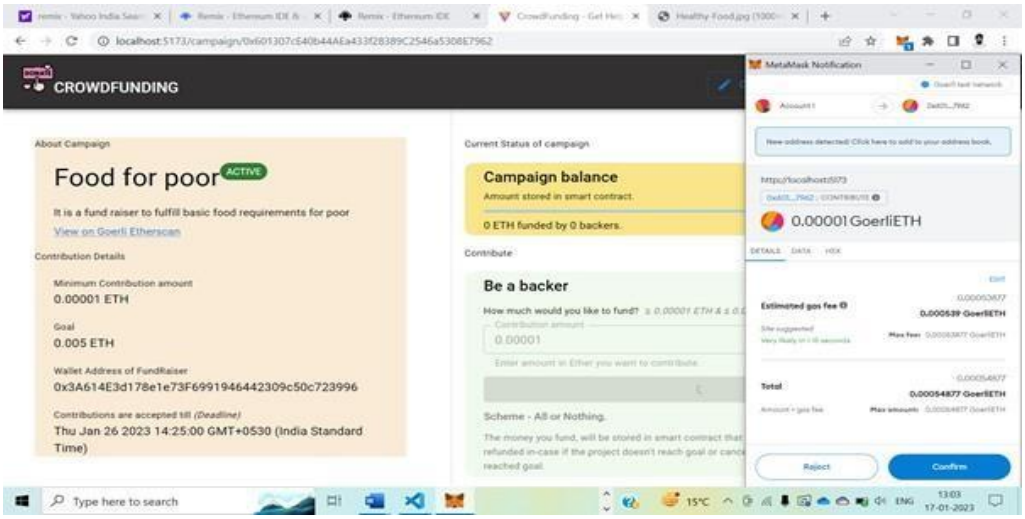

Figure 7.9: Backer contributing to a campaign

3. Status of contribution Showing Success message after funding

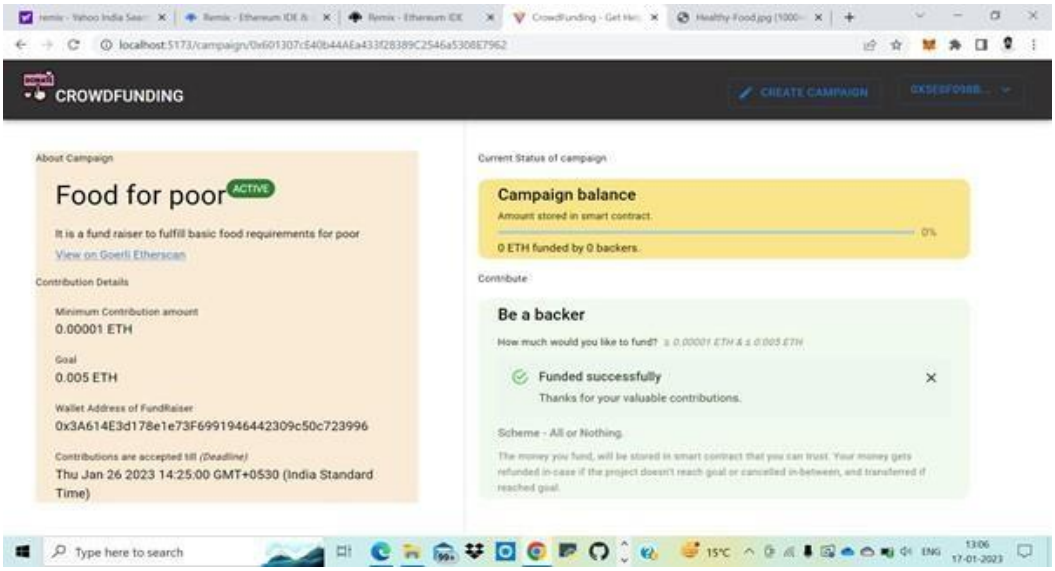

Figure 7.10: Showing contribution success message to backer

4. Showing updated status (after closing the message, page gets reloaded).

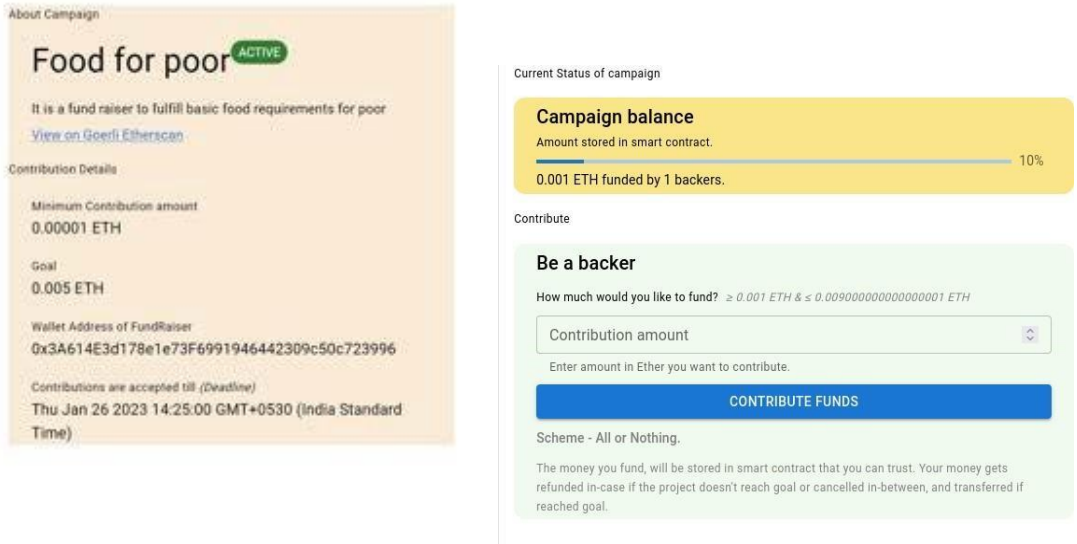

Figure 7.11: Showing updated balance

7.4. ENDING CAMPAIGN

1. FundRaiser's wallet balance before ending the campaign.

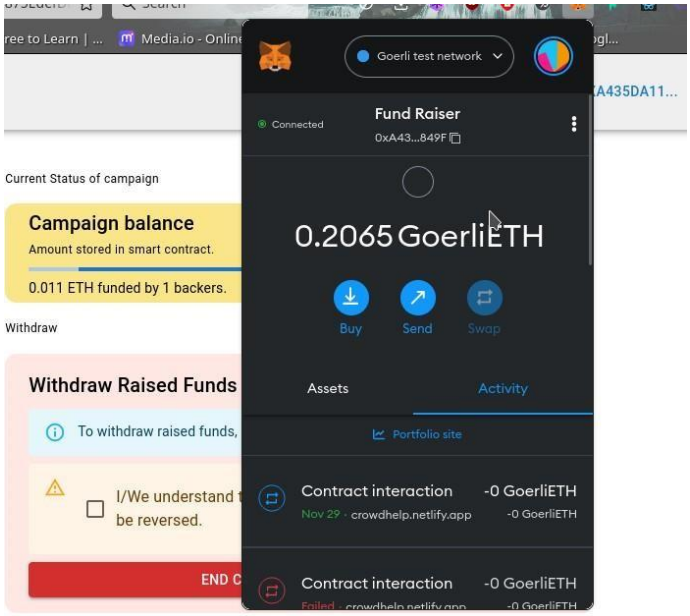

Figure 7.12: Showing Backer balance before fund raiser ending campaign.

2. After fundraiser clicks on **End Campaign & withdraw**, asking for wallet authentication.

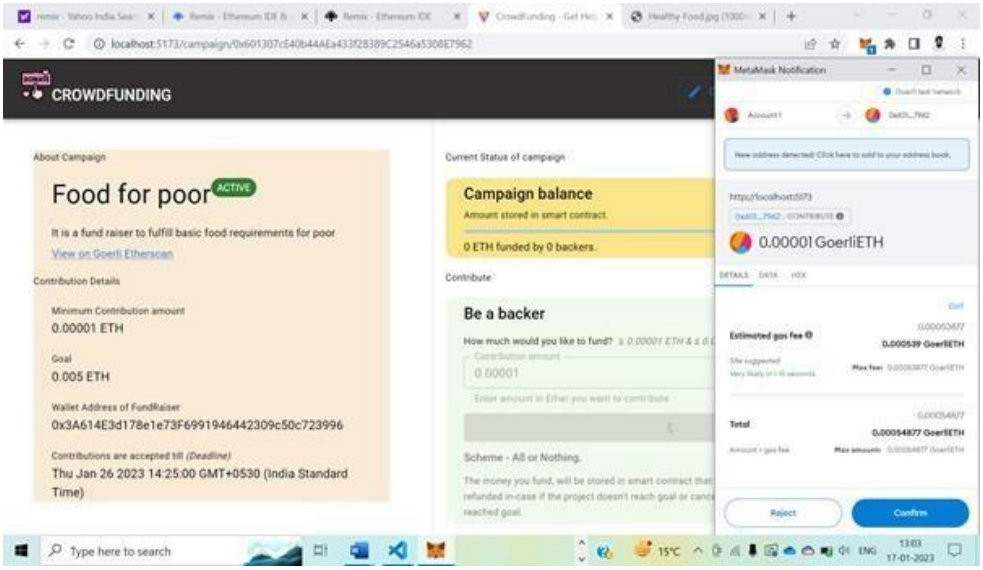

Figure 7.13: Wallet authentication to approve contribution

3. Fundraiser's wallet after successful end. Notice that, the wallet balance has increased from 0.2065 ETH to 0.2173 ETH.

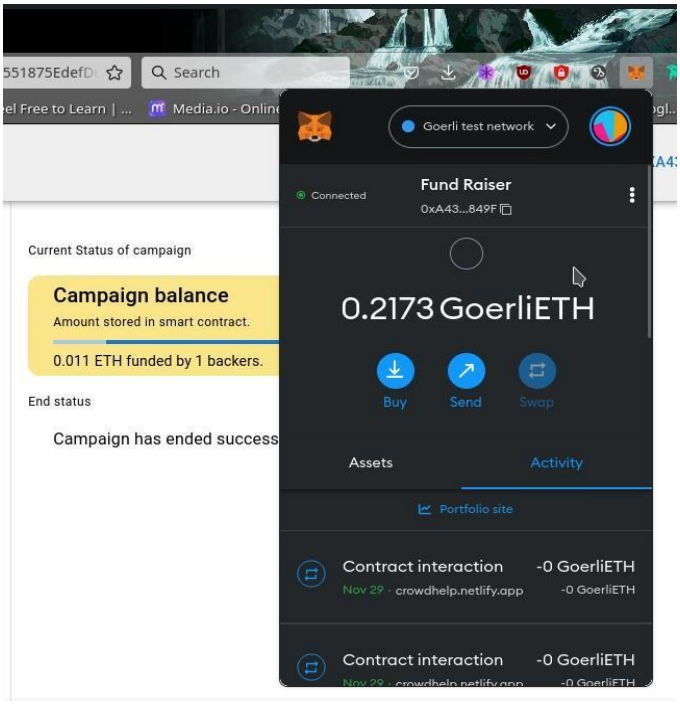

Figure 7.14: Showing fundraiser’s wallet after campaign end.

7.5. ABORTING CAMPAIGN

1. Backer's wallet balance before the fundraiser aborting the campaign.

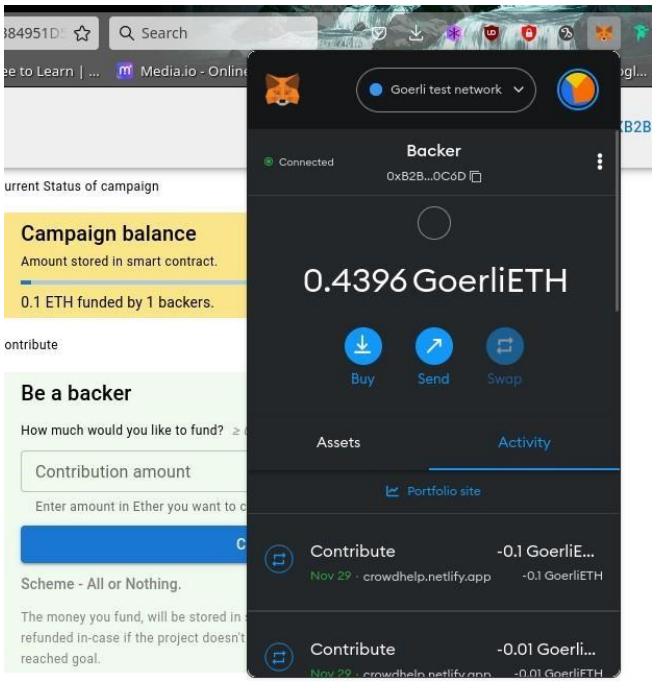

Figure 7.15. Showing backer's balance before aborting

2. Fund raiser clicking on Aborting campaign.

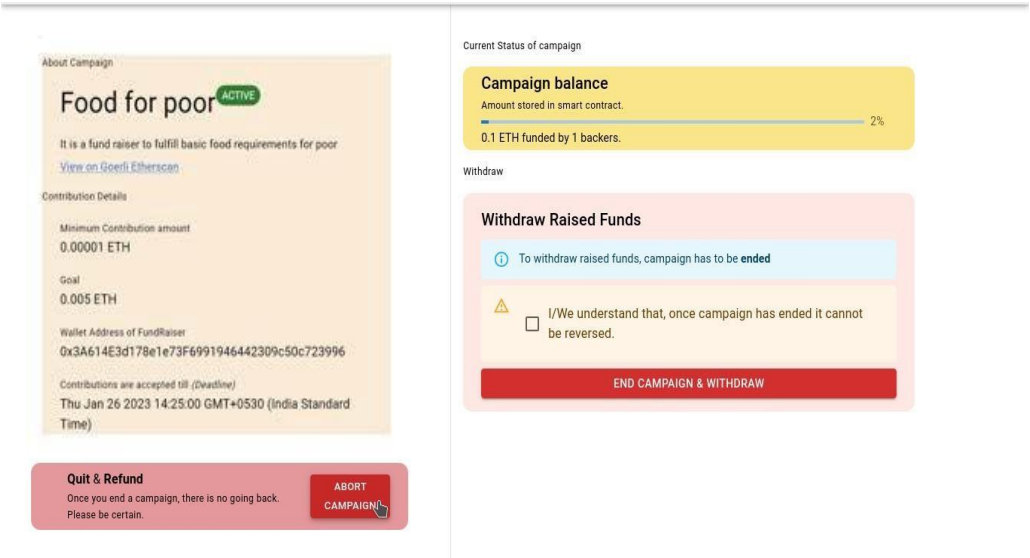

Figure 7.16: Fund raiser clicking button of Aborting campaign

3. Fund raiser filling the reason for aborting and accepting the condition of refund to backers.

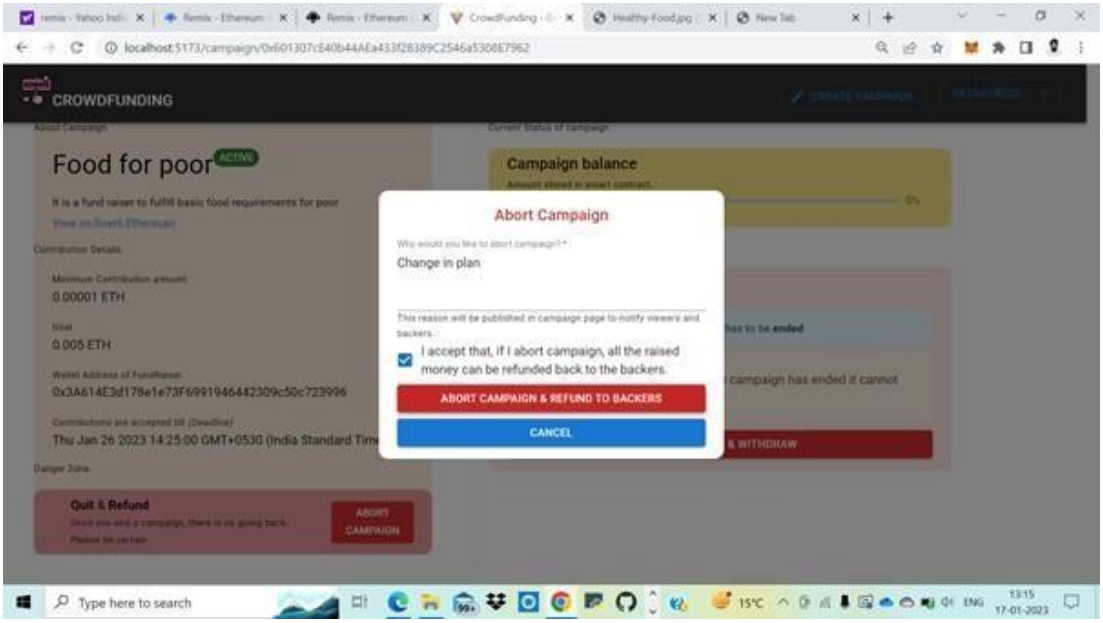

Figure 7.17: Fund raiser filled the details to abort campaign and clicking on Abort campaign button

4. Fund raiser authenticating the transaction with the wallet.

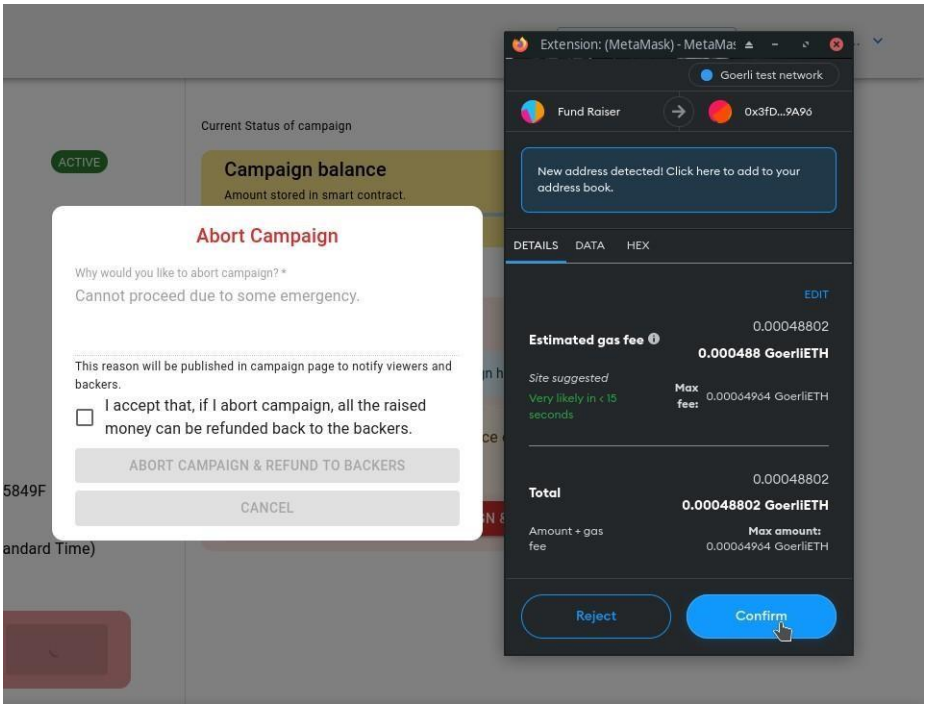

Figure 7.18: Wallet Authentication to approve aborting campaign

5. Backer's wallet balance after fund raiser aborting the campaign. Notice that, the balance of backer has increased from 0.4396 ETH to 0.5396 ETH.

After aborting, each backer will get their **whole** amount, even if they contribute partially multiple times.

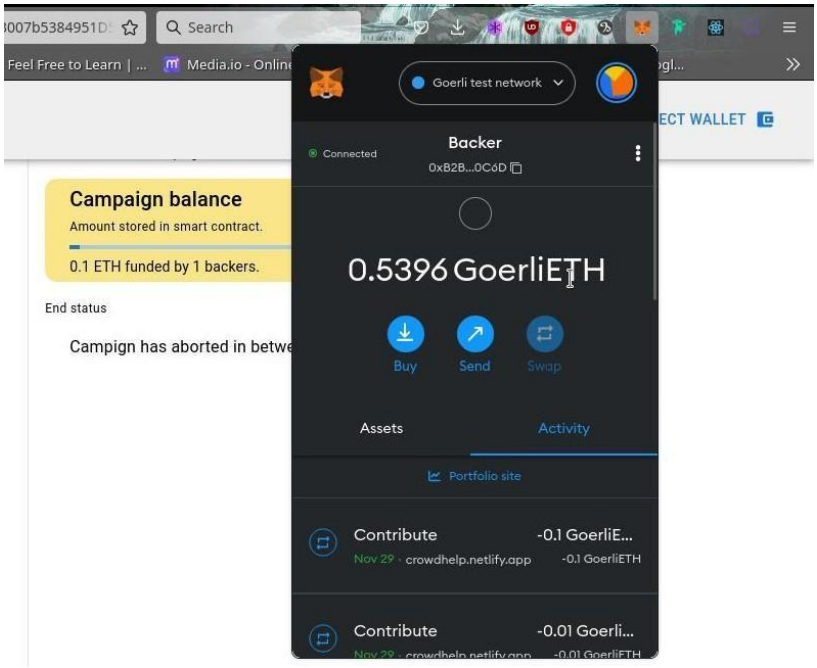

Figure 7.19: Showing increase in wallet's balance of backer after aborting.

7.6. VIEWING [ANY] CAMPAIGN'S TRANSACTIONS

1. Any user can view the campaign's transactions in etherscan.io -- By clicking on **View on Goerli therscan** link on any campaign page.

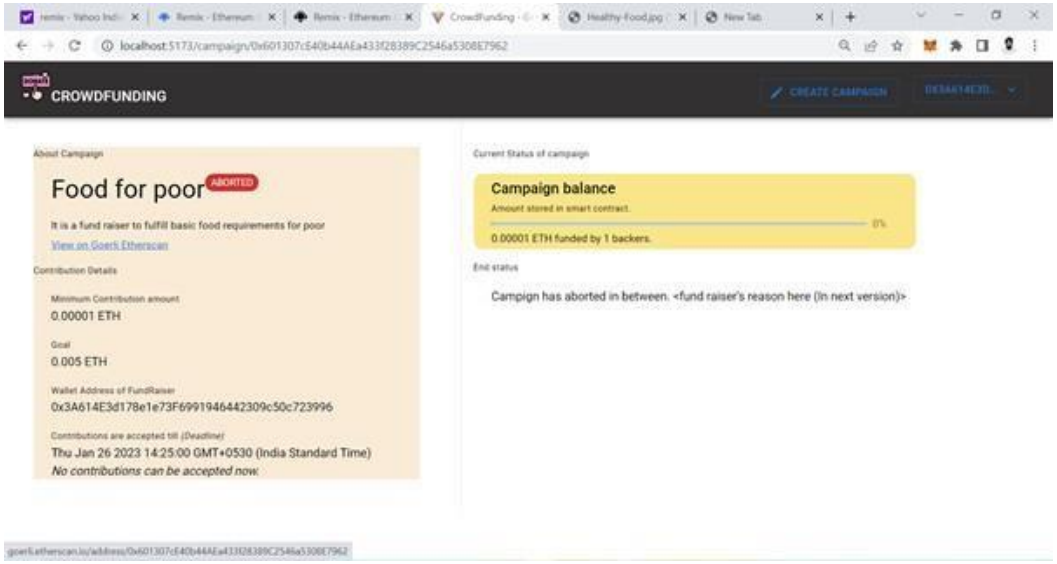

Figure 7.20: Viewer (public) clicking the link to view the transaction history in etherscan.

2. This opens up a new page of goerli.etherscan.io, showing the transactions.

- This provides transparency and trust that,
- When backer has funded, with what amount.

When the campaign has ended, aborted...

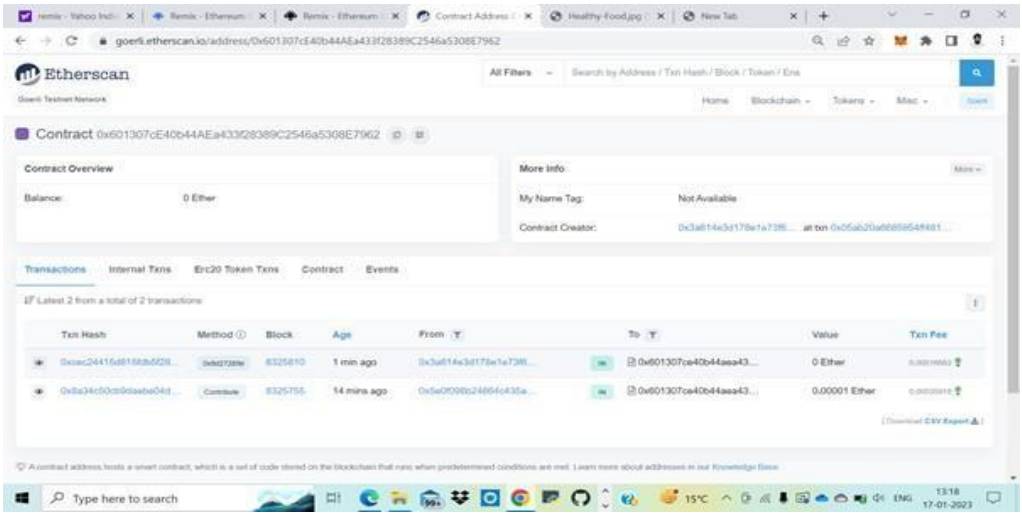

Figure 7.21: Showing the transaction history of campaign in etherscan

FUTURE SCOPE

The working prototype of the application currently works on a local blockchain that does not involve the use of real cryptocurrency. As a future update, the application can be deployed to production on an actual blockchain thereby allowing the masses to make use of it.

The application can be used for secure crowdfunding. Actual investors can invest money in innovative projects that have the potential to grow in the market. It will also benefit the nascent startups to showcase their products on a legitimate platform.

Other features thought to be having worthy, planned as a future update

Feature	**User**	**Purpose**
Setting Milestones	Fund raiser	Has to be set at time of campaign creation. Why? To protect the funds of the backer. The money will be distributed in chunks by showing the progress of funds usage.
Updating campaign page	Fund raiser	To elaborate the cause.
Raising Withdraw requests	Fund raiser	A request with the amount needed and reason. Why? The funds raised from backers. The request will be approved only if > 50% of contributors accept it.
Approving withdraw requests	Backer	Depending on the progress achieved campaign & amount needed, he can approve request. NOTE: Any backer who has contributed >= Minimum amount will be considered as approver.
Contributions page	Backer	A separate page, where backers can only view the campaigns they funded with the amount of contributions. Why? To facilitate how many campaigns they are supporting & to know SPECIFIC campaigns status to monitor how

		their contributed funds are in usage.
Funds usage	Backer / public	A page which shows the flow of funds (Inflow:backers, Outflow: by fundraiser with a reason) with transaction IDs can be checked publicly
Feature	User	Via etherscan.io Why? To provide transparency.
Searching campaigns with filters	Public	Mainly to facilitate backers & public, to search campaigns & know the status of it, usage of funds ... etc.,

CONCLUSION

It is quite evident that crowdfunding has huge potential in present times as well as in the future. Even though it comes with its own drawbacks and challenges, this form of funding is helping new start- ups and innovators, entrepreneurs and other creators. In the future, when blockchain will the backbone of major investor contributions, it will make crowdfunding easier, transparent and accessible. It is only a matter of time.

A. Security

Although blockchain is not immune to hacking, its decentralized nature gives it a greater line of defense. A hacker or criminal would need control of more than half of all machines in a distributed ledger to change it.

This application provides a good amount of security, the reason being, firstly the application is decentralized which ensures that no single person has authority over the whole data that is flowing through the blockchain. A blockchain is made up of several nodes, all these nodes collectively store the data that is flowing through the blockchain.

B. Reliability

The funds will be added to the smart contract only till the point when the requirement of the startup is met, hence, ensuring that no extra funds are transferred to the smart contract. This application also ensures that the maximum funding that the backers can provide at a time is no more than the total funding required by the project.

C. Transparency

The application maintains transparency between the backers and startups, i.e., the backers can keep checking the progress of the startup's project. Backers can also be rest assured that if at any point in time a startup abandons a project then, the remaining amount left in the smart contract is refunded to the backer in the appropriate proportion that they had funded.

D. Interactive User Interface

The interface of this application is user-friendly, it also has a dedicated “Help” page which explains in detail how to use the application, both for the backers and for the startups.

In order to conclude we would like to affirm that the potential and influence that the emerging technologies possess, for crowdfunding, is immense. The current solutions for the challenges that the usual crowdfunding platforms pose, are now able to transform the society for better. Crowdfunding platforms using the blockchain technology hold more credibility and therefore, we believe are the future for the right investment for investors.

REFERENCES

INSPIRATION

1. CryptoRelief — Help India fight the Covid emergency – Existing & **SUCCESSFULLY WORKED** project, ran in covid time, raised $475,067,752 (~4 billion USD) for India from all over the world in crypto currency – check on etherscan.

For getting the idea and clarity on Crowdfunding & crowdfunding on blockchain:

Research papers:

1. Blockchain-Based Crowdfunding Application - IEEE - From Manan's Ref. of Google drive - publicly viewing access allowed - practical demo
2. BitFund: A Blockchain-based Crowd Funding Platform for Future Smart and Connected Nation - researchgate
3. Role of Blockchain Technology in Crowdfunding - ICMEF
4. Smart Contract and Blockchain for Crowdfunding Platform – warse
5. Blockchain based crowdfunding systems – iaes
6. Crowdfunding Platform Using Blockchain Technology - IJIRT

Articles

1. Create a Crowdfund Smart Contract using Solidity - DEV Community– solidity part of crowdfunding with explanation
2. Create a Crowdfunding Smart Contract using Solidity | Hackmamba – for writing smart contract
3. Crowd Funding Smart Contract | DApp World (dapp-world.com)

Videos

1. Blockchain Based Crowdfunding Application Demonstration – Viren Patil – video implementation of 1st IEEE research paper.
2. What are Smart Contracts in Crypto? (4 Examples + Animated)
3. Build Your First Blockchain App Using Ethereum Smart Contracts and Solidity – freecodecamp

Attributions:

Images & logos used in system architecture: Taken from Wikimedia commons and flaticon.