

PROJECT REPORT ON
BeSW: Utilizing Blockchain to Streamline Social Welfare

*Report submitted to the SASTRA Deemed to be University
as the requirement for the course*

CSE300 - MINI PROJECT

Submitted by

SANTOSH KUMAR DOODALA
(Reg. No.: 124003280, CSE)

AMAL PRAKASH
(Reg. No.: 124003025, CSE)

ANMOL RAJ
(Reg. No.: 124157004, CSE-CSBT)

May 2023



SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act, 1956)



THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

SCHOOL OF COMPUTING
THANJAVUR, TAMIL NADU, INDIA – 613 401



SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA



SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

Bonafide certificate

This is to certify that the report titled “**BeSW: Utilizing Blockchain to Streamline Social Welfare**” submitted as a requirement for the course, **CSE300 - MINI PROJECT** for B. Tech. is a bonafide record of the work done by **SANTOSH KUMAR DOODALA** (Reg. No.: **124003280**, CSE), **AMAL PRAKASH** (Reg. No.: **124003025**, CSE), **ANMOL RAJ** (Reg. No.: **124157004**, CSE-CSBT) during the academic year 2022-2023, in the School of Computing.

Signature of Project Supervisor : *B. Karthikeyan*

Name with Affiliation : Dr. Karthikeyan B, SAP, SoC

Date : 11/May/2023

Mini Project Viva voce held on held on _____

Examiner 1

Examiner 2

ACKNOWLEDGEMENT

We would like to thank our Honorable Chancellor **Prof. R. Sethuraman** for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our Honorable Vice-Chancellor **Dr. S. Vaidhyasubramaniam** and **Dr. S. Swaminathan**, Dean, Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks to **Dr. R. Chandramouli**, Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to **Dr. A. Umamakeswari**, Dean, School of Computing, **Dr. S. Gopalakrishnan**, Associate Dean, Department of Computer Application, **Dr. B. Santhi**, Associate Dean, Research, **Dr. V. S. Shankar Sriram**, Associate Dean, Department of Computer Science and Engineering, **Dr. R. Muthaiah**, Associate Dean, Department of Information Technology and Information and Communication Technology for their constant support, motivation and academic help extended for the past three years of my life in the School of Computing.

We would like to especially thank and express our sincere gratitude to our guide **Dr. Karthikeyan B**, Sr. Asst. Professor, School of Computing for his support throughout the whole project. His deep insight into the field of BCT and invaluable suggestions helped us in making progress throughout our project work. We also thank the project review panel members **Dr. Suriya Praba T**, Asst. Professor - II and **Dr. Anushiaadevi R**, Asst. Professor - III for their valuable suggestions and insights which enabled us to delve more into the concept of the BCT.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from my family, friends and the support of my teammates resulting in the successful completion of this project. We thank you all for providing me with an opportunity to showcase my skills through the project.

ABSTRACT

Donating money is seen as a moral responsibility when crises happen all over the world. People's willingness to give, however, is hampered by the charity sector's lack of openness and accountability. Donors are unwilling to provide contributions unless they know exactly how their funds will be used. This issue can be addressed utilizing blockchain technology (BCT), which provides a decentralized and secure ecosystem that eliminates the need for third-party middlemen between charity and donors. This project proposes a blockchain-based donation mechanism to assist charities, donors, and beneficiaries in times of natural disasters, pandemics like Covid-19, and emergencies. The blockchain keeps track of every transaction, giving contributors knowledge of how their money is being spent. The purpose of this paper is to increase contribution transparency and advocate for the application of BCT in charitable endeavours. This framework is implemented using the Ethereum blockchain, which offers a user-friendly platform for donations. Trust between donors, beneficiaries, and charitable organisations is increased through the usage of smart contracts. This kind of donation saves time, and money, and avoids the potential of counterfeit campaign donations. Overall, this study helps to make emergency recovery operations better.

Keywords: Blockchain technology (BCT), Charity sector, Emergencies, Ethereum, Smart contracts

TABLE OF CONTENT

ACKNOWLEDGEMENT	ii
ABSTRACT	iii
TABLE OF CONTENT	iv
LIST OF FIGURES	v
LIST OF ABBREVIATIONS	vii
TOOLS USED	vii
CHAPTER 1 - BASE PAPER DETAILS	1
CHAPTER 2 - MERITS AND DEMERITS OF THE BASE PAPER	3
CHAPTER 3 - TECHNOLOGY USED	6
1. HARDHAT	6
2. THIRDWEB	6
3. YARN	7
4. ETHER.JS	7
5. SOLIDITY	7
6. ETERNAL	7
CHAPTER 4 - IMPLEMENTED FRAMEWORK	8
Home Page	8
DoNation Platform	9
DoRequest Platform	14
CHAPTER 5 - SOURCE CODE	19
Algorithms in the Base Paper	19
Explanation of Solidity Functions	21
Code	22
CHAPTER 6 - SNAPSHOT IN DIFFERENT SCENARIOS	32
Home Page	32
DoNation Working	32
DoRequest Working	39
CHAPTER 7 - Conclusion and Future Works	44
CHAPTER 8 - References	45

LIST OF FIGURES

Figure no.	Title	Page No.
Fig 3.1	thirdweb architecture	6
Fig 4.1	Flow of Home Page	8
Fig 4.2	Flow of DoNation Platform usage by various actors	9
Fig 4.3	The Admin's MetaMask account public address	9
Fig 4.4	The use of the .env variable to allow only the admin to access the functionalities	9
Fig 4.5	Flow of the DoNation Platform	10
Fig 4.6	Use Case Diagram for DoNation Platform	11
Fig 4.7	Sequence Diagram for the DoNation Platform	11
Fig 4.8	DoNation SC address	12
Fig 4.9	`yarn hardhat node` command used to initiate a node for local testing.	12
Fig 4.10	Block #1	13
Fig 4.11	hardhat.config.js	13
Fig 4.12	Flow of DoRequest Platform usage by various actors	14
Fig 4.13	Different metamask wallet for different actors	15
Fig 4.14	hardhat.config.js	15
Fig 4.15	`npm run deploy` – to deploy the smart contract.	16
Fig 4.16	Deployed smart contract address stored in an environment variable.	16
Fig 4.17	Use of environment variable to access the deployed smart contract.	16
Fig 4.18	Flow of DoRequest	17
Fig 4.19	Use Case Diagram for DoRequest Platform	17
Fig 4.20	Sequence Diagram for DoRequest Platform	18
Fig 4.21	`yarn dev` – to run the application	18
Fig 5.1	register()	19
Fig 5.2	Working of admin()	19

Figure no.	Title	Page No.
Fig 5.3	Working of Beneficiary()	20
Fig 5.4	Working of Donor()	20
Fig 6.1	Home Page	32
Fig 6.2	Admin View	32
Fig 6.3	Block #2	33
Fig 6.4	Contributor Payment	33
Fig 6.5	Contributor View	34
Fig 6.6	Admin adding Mission	34
Fig 6.7	Mission's View (for admin)	35
Fig 6.8	Mission's View (for contributor)	35
Fig 6.9	Contributing to the Mission	36
Fig 6.10	List of contributors	36
Fig 6.11	DoNation Dashboard View	37
Fig 6.12	Ethernal Transaction's View	37
Fig 6.13	Beneficiary View	38
Fig 6.14	Normal Use Case View	38
Fig 6.15	Beneficiary View	39
Fig 6.16	Beneficiary Filing the request	39
Fig 6.17	RequestID generated	40
Fig 6.18	Checking status using RequestID - pending request	40
Fig 6.19	Admin View	41
Fig 6.20	Admin View approving Request	41
Fig 6.21	Admin View approving Request	42
Fig 6.22	Request approved relected on Beneficiary Side	42
Fig 6.23	Smart Contract DoRequest uploaded on thirdweb	43
Fig 6.24	Block generated for each request being filed	43

LIST OF ABBREVIATIONS

1. BCT - Blockchain Technology
2. ETH - Ethereum Currency
3. npm- Node Package Manager
4. SC - Smart Contract
5. IoT - Internet of Things
6. AI - Artificial Engineering
7. NGO - Non-governmental Organization

TOOLS USED

1. MacBook Air (M1, 2020) 8 GB/256GB
2. Asus VivoBook 16GB/512GB
3. macOS 13.3.1
4. Windows 11
5. Hardhat
6. ThirdWeb
7. Yarn - v1.22.19
8. npm - v9.5.0
9. Ethers.js
10. Node - v18.15.0
11. Hardhat -v2.9.3
12. Solidity ^0.8.7

CHAPTER 1 - BASE PAPER DETAILS

Title: BeSW: Utilizing Blockchain to Streamline Social Welfare

Journal name: IEEE TRANSACTIONS ON COMPUTATIONAL SOCIAL SYSTEMS

Authors: Mandeep Kaur, Pankaj Deep Kaur, and Sandeep Kumar Sood

Url: <https://ieeexplore.ieee.org/abstract/document/10004889>

Year: 2022

Summary of the base paper:

Donation websites nowadays pretend to be transparent but we know that this is not the case for many of the websites as more and more scams are coming to light. One such solution to this problem can be achieved with BCT so that the needy can still get the required relief. With the ability to track every transaction on the blockchain, contributors can see exactly where and how their money is being spent. The paper's primary goal is to encourage the usage of BCT. The proposed architecture is implemented using the Ethereum blockchain, which also offers a practical platform for donations. Donations made with SC enhance trust between donors, recipients, and non-profit organisations, preventing shady campaigns and scams.

The research paper has considered numerous BCT applications that already exist in a variety of fields, such as maintaining medical records, logistics, supply chain management, the IoT, security, and monetary systems, as well as how it is integrated with various technologies, such as IoT, ML, AI and cloud computing, and found a new application in the donation sector. As of now, there are not yet adequate frameworks and implementations for the BCT in these fields. The main publication placed an emphasis on completing the gaps left by earlier studies with all necessary data and execution.

To utilise this BCT, an application is developed (management policy) so that only verified donors and beneficiaries can participate in the network, using the Ethereum technology.

The primary focus of the paper is:-

1. To make the donation platform more transparent
2. To eliminate fraud and dubious campaigns
3. To foster trust amongst different actors.
4. To see its worth when compared to already existing systems.

Three stakeholders are defined here:-

1. NGO: An NGO is a nonprofit organisation that provides the data about requests for assistance and donations to philanthropic cause on the suggested platform. Beneficiaries and donors are allowed to take part in campaigns thanks to the charity trust's administrator.
2. Contributor: Anyone who takes part in a charitable trust's fundraising efforts is known as a Contributor. To assist those in need, the contributor can sign up and contribute to the missions started by NGO on the platform.
3. Beneficiary: A beneficiary is a person who requires assistance during pandemics or disasters. The beneficiary may sign up and submit a request for financial assistance to the NGO.

The admin authorises entry into the blockchain network. Contributors can make contributions to the cause using their virtual wallets (MetaMask). The NGO's administrator verifies the beneficiaries' requests before transferring money to their accounts.

To implement this the SC is written in Solidity which works in tandem with the Ethereum BCT, NodeJS Web3js are some other technologies used to achieve this application. To replicate the application, one of the testnet Rinkeby test networks is employed.

Users will be able to track their transaction history, request donations, and view it on the platform. Requests for assistance can be sent by recipients. To prevent repeated gifts to the same beneficiary, the user's request for funds is first confirmed by looking up past transaction history. This helps to eliminate the fraud and improve the consistency of the system. After verification, the beneficiary's request is authorised, finalised, and marked with the finalised request code. NGO distributes payments to beneficiaries who have been verified. The transactions window displays information about the sender, receiver, prior transaction, and transaction hash.

CHAPTER 2 - MERITS AND DEMERITS OF THE BASE PAPER

Literature Survey By the base paper:-

Hu et al. provide a platform for charitable giving which helps in increasing the transparency of the charity operations which in turn helps in the boosting of the public trust in charity. Blockchain-based charity system has an Ethereum blockchain platform with solidity smart contract and has performance evaluation. There is no consensus mechanism, no algorithm, and no interface.

Saleh et al. proposed a tracking platform. This was proposed for charitable donations. It combined a centralized server which is included with the application programming interface key. It has an Ethereum blockchain platform with solidity smart contract. There is no consensus mechanism, no algorithm, no interface and there is no performance evaluation.

Lee et al. designed a model that improves security during transactions to provide a transparent environment for transactions using a one-off address system. This security is achieved by maintaining the stakeholders' privacy. It has an Ethereum blockchain platform with solidity smart contract and has performance evaluation. There is no consensus mechanism, no algorithm, and no interface.

Zwitter and Boisse - In their study, Despiaux notes that donations might be made by designating them digitally for particular programmes. This lessens the likelihood that any corruption or fraud will take place. It has no blockchain-based platform, no smart contract.

There is no consensus mechanism, no algorithm, no interface and there is no performance evaluation.

A decentralized charity tracking system was proposed by Singh et al. which was Ethereum-based. This offers transparent and direct funds to the expected beneficiaries. The tracking system for donations and aid has an Ethereum blockchain platform with a solidity smart contract. There is no consensus mechanism, no algorithm, and no performance evaluation but there is an interface.

Farooq et al. proposed an infrastructure for obtaining donations for charity that is open to both donors and the law enforcement so that audits may be carried out. A framework for making charitable donations transparent has an Ethereum blockchain platform with a solidity smart contract with a PoW consensus mechanism, no algorithm, and no interface but there is performance evaluation.

Creating a Trustworthy Service System for Charitable giving during COVID-19 was discussed by Wu et al. The use of blockchain technology for trustworthy charitable donations was discussed during the COVID-19 pandemic. It has a blockchain-based platform, no smart contract. There is no consensus mechanism, no algorithm, no interface and there is no performance evaluation.

A decentralized mechanism was proposed by Agarwal et al. aiming to increase transparency. The approach involves using bitcoins for charitable endeavours, increasing openness in the process. Without a blockchain-based platform or smart contract, it has a decentralised and financially sound approach to successful charity. There is no consensus mechanism, no algorithm, no interface and there is no performance evaluation.

Merits:-

The paper uses BCT to solve the problem of the untrustworthiness of donation platforms. Some of the problems are :

1. Elimination of middlemen: The BCT makes it possible to eliminate the middlemen in the charity donating process, which lowers transaction costs and ensures that more of the given monies are distributed to the intended beneficiaries.
2. Donations based on smart contracts: The paper suggests automating the donation process for charities using SC, which are self-executing contracts that may be programmed with specified criteria. This can improve accountability, lower the chance of fraud, and guarantee that donations are given out in accordance with established guidelines.
3. Tamper-proof records: The BCT can offer tamper-proof records of every transaction, which helps improve the process of charitable giving by increasing confidence and accountability. Further boosting transparency, the records can also be inspected and validated by outside parties.
4. International donations: The paper also addresses how BCT, by introducing a common currency ETH, can enable international donations without the requirement for a typical bank account, improving accessibility and lowering the obstacles to donating across borders.
5. Fast and efficient donations: The BCT can make donations that are fast and effective, allowing money to reach those in need right away and cutting down on the time required for paperwork.

6. **Better data management:** According to the paper, can manage donor and recipient data securely and effectively, lowering the risk of data breaches and enhancing data privacy.

Demerits:-

Even though the BCT is such an interesting solution, there are a few problems which are listed below:-

1. Use of the now deprecated testnet Rinkeby, during our project duration we lost the access to Rinkeby and GORELI testnet, eventually switching to Sepolia.
2. No clear-cut mention of the algorithms so that it works as a cohesive application.
3. NGOs helping physically by buying relief supplies with the donated money.
4. Unnecessary step for donors to register in order to participate in the donation process.

All the above demerits are addressed, which are mentioned throughout this paper.

CHAPTER 3 - TECHNOLOGY USED

1. HARDHAT

Hardhat is a development environment for building and testing SC on the Ethereum blockchain. It is an open-source tool that offers several functions and utilities to speed up the development process and make it simpler for programmers to create, test, and deploy their SC.

Developers can use Hardhat to create SC in Solidity or other languages, execute automated tests, publishing contracts to local or remote test networks, and communicate with published contracts via an integrated terminal. Hardhat is a flexible and effective solution for Ethereum development since it offers interaction with well-known development frameworks and tools like Truffle, Remix, and ether.js.

It has support for JavaScript and TypeScript Projects

2. THIRDWEB

With the help of the development framework thirdweb, we can create, launch, and administer web3 applications and games on any blockchain that is EVM-compatible. In order to make usable and accessible decentralised applications, thirdweb provides a full set of tools and technologies. This infrastructure offers choices for wallet integrations, gasless relayers, and fiat on-ramps, resulting in a smooth end-user experience.

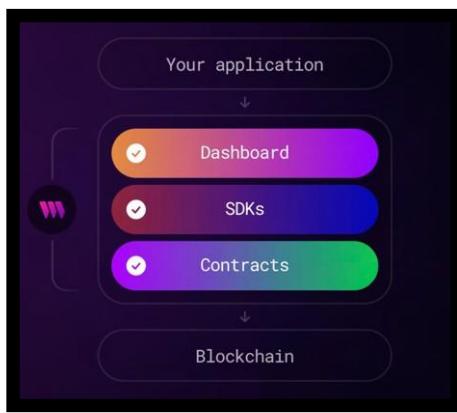


Fig 3.1 thirdweb architecture

3. YARN

Yarn is a package manager created to improve upon the existing npm system. It is used, like npm, to manage a project's dependencies and make sure that the required packages and libraries are set up and current.

Compared to npm, yarn manages dependencies more effectively. We can also share code with other developers from around the world thanks to it.

Code is shared through something called a package. A package contains all the code being shared as well as a package.json file that describes the product.

4. ETHER.JS

Creating a front-end to interact and a backend for Blockchain is ok, but how do the two come into one cohesive application? The answer is Ether.JS

It is a compact library for interacting with the Ethereum Blockchain and its ecosystem.

Be it connecting to the MetaMask wallet or handling contracts, it handles all. Thus it is also an important part of the project.

5. SOLIDITY

The programming language that we use to create our SC is called Solidity.

SC is a self-executing computer program that runs on a blockchain network. It is a kind of electronic contract intended to automatically enforce the conditions of a contract between two parties without the need for middlemen or reliable third parties.

It is a contract-oriented language intended for use in creating SC and decentralised apps (dApps) that can run on the Ethereum Virtual Machine (EVM).

It is easier to integrate with other tools.

6. ETERNAL

Similar to Etherscan, Ethernal is an open-source block explorer for private EVM-based networks.

We can browse blocks, transactions, accounts, and contracts with this. But also read contract variables, and decode function calls and events.

Whether you are running Ethernal locally or on a remote server, it can connect to any chain.

With the use of a CLI, Ethernal synchronises blocks and transactions with your dashboard.

CHAPTER 4 - IMPLEMENTED FRAMEWORK

In the Project we have made some improvements over the existing techniques, some of them are:-

1. We have used Hardhat and Thirdweb to deploy our SC and to provide a testnet of some sort which is stable.
2. All the money is transferred from the SC to the beneficiary at the end of the duration of the campaign.
3. Removed the registration process for donors so that everyone can see the content of the mission being run, thus promoting transparency and ease of donation

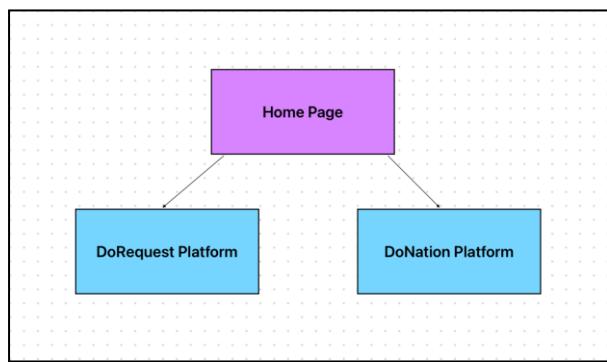


Fig 4.1 Flow of Home Page

HOME PAGE

The home page Fig 4.1 is directed to two websites.

These are:-

1. DoRequest: It handles the beneficiary request for the mission. Admin has the power to either accept or reject while giving suitable feedback. [Uses Blockchain]
2. DoNation Platform: - The main portal which is viewable by all the actors, only the admin has the select functionalities like adding a mission so that only legit missions which are accepted by the admin in the DoRequest Platform are mentioned here.

DONATION PLATFORM

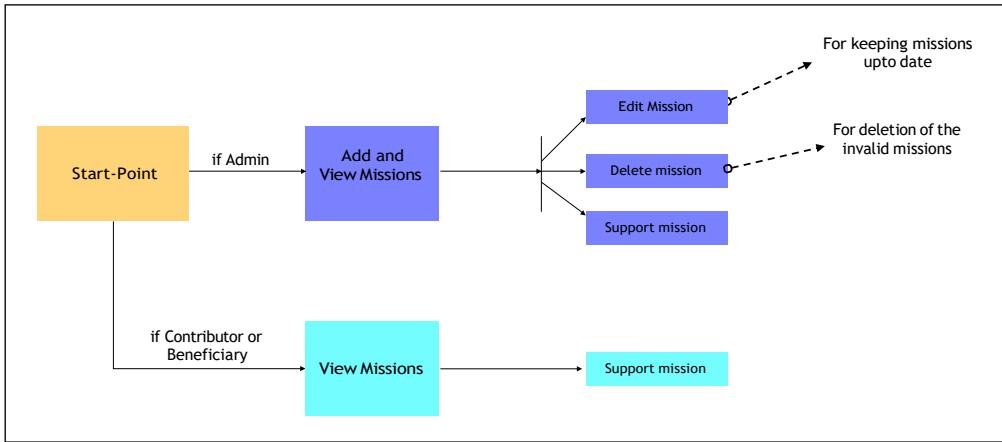


Fig 4.2 Flow of DoNation Platform usage by various actors

This part of the project is done in the Hardhat environment using JavaScript, Tailwind-CSS, Solidity and React.

There are three actors for this platform Admin, Contributor and Beneficiaries.

Admin alone has the permission to add a mission, it is done by adding the Admin's MetaMask account public address and storing it in the .env file, only the admin who has this account is able to access the + button to add new missions.

```

REACT_APP_BESW=0x70997970C51812dc3A010C7d01b50e0d17dc79C8
# Account #1 - Admin
#
# Account #0: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266 (10000 ETH)
  
```

Fig 4.3 The Admin's MetaMask account public address

```

src > views > Home.js > [x] Home > useEffect() callback
  1 import { useState, useEffect } from 'react'
  2 import AddButton from '../components/AddButton'
  3 import CreateMission from '../components/CreateMission'
  4 import Hero from '../components/Hero'
  5 import Missions from '../components/Missions'
  6 import { loadMissions } from '../services/blockchain'
  7 import { useGlobalState } from '../store'
  8
  9 const Home = () => {
10   const [missions] = useGlobalState('missions');
11   const [isAdmin, setAdmin] = useState(false);
12   const [account, setAccount] = useState('');
13   const admin = process.env.REACT_APP_BESW;
14   const adminToLower= admin.toLowerCase();
  
```

Fig 4.4 The use of the .env variable to allow only the admin to access the functionalities

The Beneficiary and the Donors have a common view and are able to see:

1. The Missions and their status (completed or still on)
2. The number of Donors that have contributed to the mission
3. The details about the mission

The basic flow of the Platform

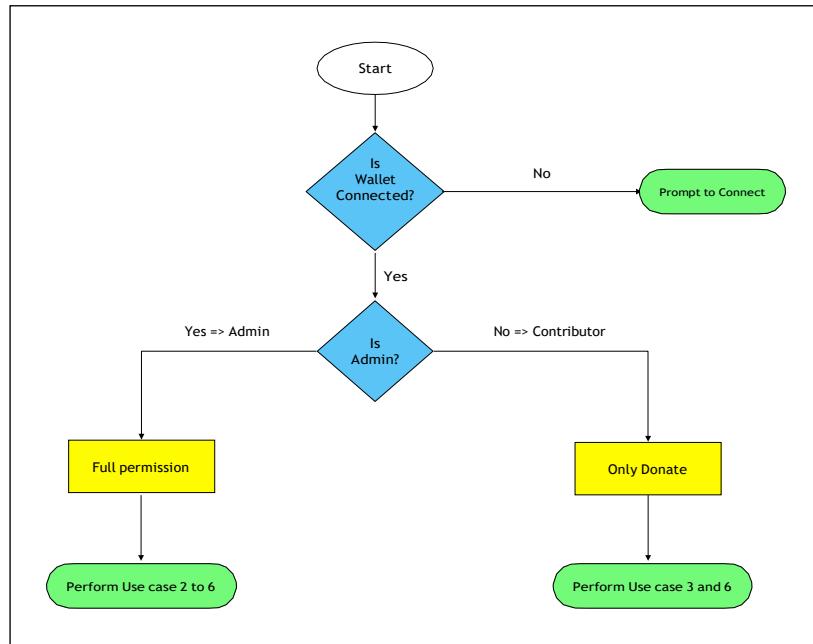


Fig 4.5 Flow of the DoNation Platform

Use Cases

1. Connect Wallet - admin + contributor
2. Create Mission - admin
3. Support Mission - admin + contributor
4. Delete Mission - admin
5. Update Mission - admin
6. View Mission Status - admin + contributor + beneficiary

Use Case Diagram

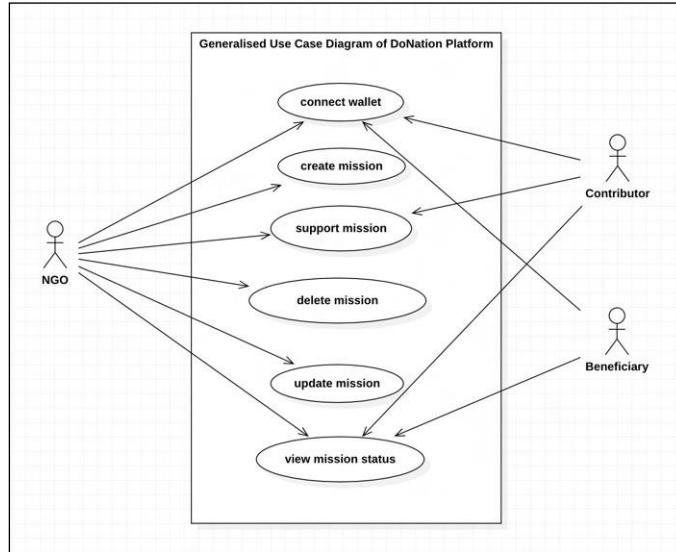


Fig 4.6 Use Case Diagram for DoNation Platform

Sequence Diagram

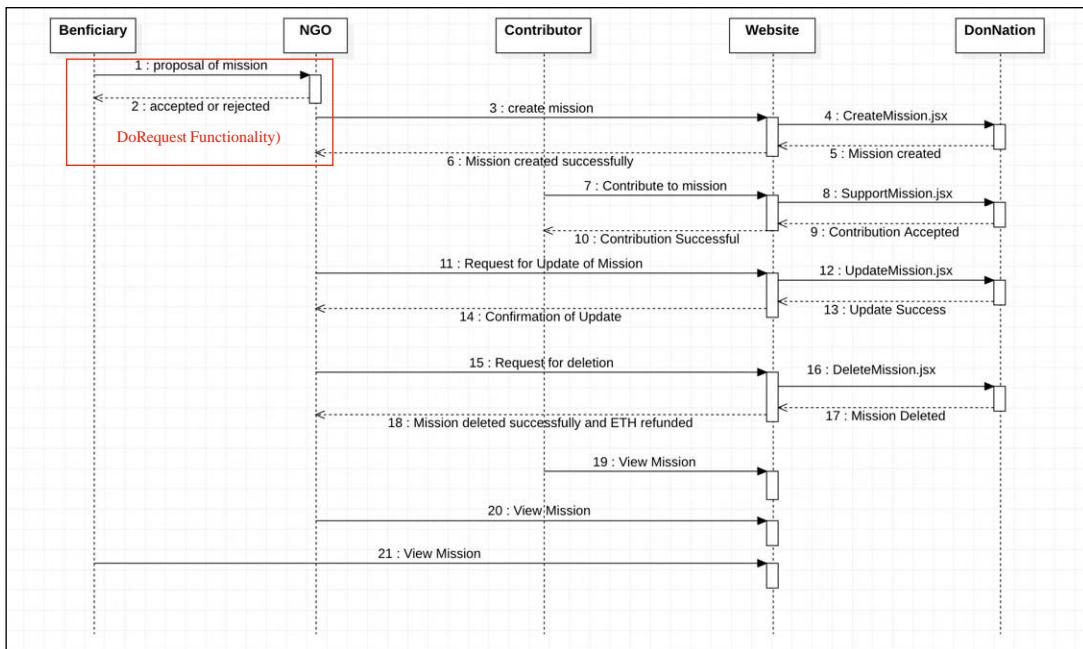


Fig 4.7 Sequence Diagram for the DoNation Platform

How does it all Connect?

The contract is written in Solidity. `Blockchain.jsx` utilises the SC and uses its various functionalities written in the SC to achieve this.

The SC is used to generate two parts:-

1. ABI:- It is the required component required for communicating with the SC, used in `Blockchain.jsx`
2. Byte Code:- The SC deployment is nothing but Byte Code deployment on the blockchain platform (Ethereum Platform).

Tackling Money Handling Problem

The ETH is sent to SC, therefore until the end of the mission validity, the funds are held by SC.

Deployment of the Project

```
1  {
2    "address": "0x5FbDB2315678afecb367f032d93F642f64180aa3"
3 }
```

Fig 4.8 DoNation SC address

The YARN handles all the installation of the dependencies.

Meanwhile, the Hardhat is the actual local development environment used, instead of the Rinkeby testnet because of the non-availability of testnet.

Hardhat provides a set of 20 accounts with dummy ethers.

```
○ ➔ BESW-DoNation-main yarn hardhat node
yarn run v1.22.19
$ /Users/santosh/Downloads/BESW-DoNation-main/node_modules/.bin/hardhat node
[Ethereal] Missing email to authenticate. Make sure you've either set ETERNAL_EMAIL in your environment or they key 'email' in your Eth
ernal config object & restart the node.
Started HTTP and WebSocket JSON-RPC server at http://127.0.0.1:8545/

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

Account #0: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266 (10000 ETH)
Private Key: 0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbcd5efcae784d7bf4f2ff80

Account #1: 0x70997970C51812dc3A010C7d01b50e0d17dc79C8 (10000 ETH)
Private Key: 0x59c6995e998f97a5a0e44966f0945389dc9e86dae88c7a8412f4603b6b78690d

Account #2: 0x3C4Cd0db6a900fa2b585dd299e03d12FA4293BC (10000 ETH)
Private Key: 0x5de4111afa1a4b94908f83103eb1f1706367c2e68ca870fc3fb9a804cdab365a

Account #3: 0x90F79bf6EB2c4f870365E785982E1f101E93b906 (10000 ETH)
Private Key: 0x7c852118294e51e653712a81e05800f4191a1751be58f605c371e15141b007a6

Account #4: 0x15d34AAf54267DB7Dc367839AAf71A00a2C6A65 (10000 ETH)
Private Key: 0x47e179ec197488593b187f80a00eb0da91f1b9d0b13f8733639f19c30a34926a

Account #5: 0x9965507D1a55bcC2695C58ba16FB37d819B0A4dc (10000 ETH)
Private Key: 0x8b3a350cf5c34c9194ca85829a2df0ec3153be0318b5e2d3348e872092edffba

Account #6: 0x976EA74026E726554dB657fA54763abd0C3a0aa9 (10000 ETH)
Private Key: 0x92db14e403b83df3df233f83dfa3a0d7095f21ca9b0dd6bb8d88b2b4ec1564e

Account #7: 0x14dC79964da2C08b23698B3D3cc7Ca32193d9955 (10000 ETH)
Private Key: 0x4bbbf85ce3377467afe5d46f804f221813b2bb87f24d81f60f1fcdbf7cbf4356
```

Fig 4.9 `yarn hardhat node` command used to initiate a node for local testing.

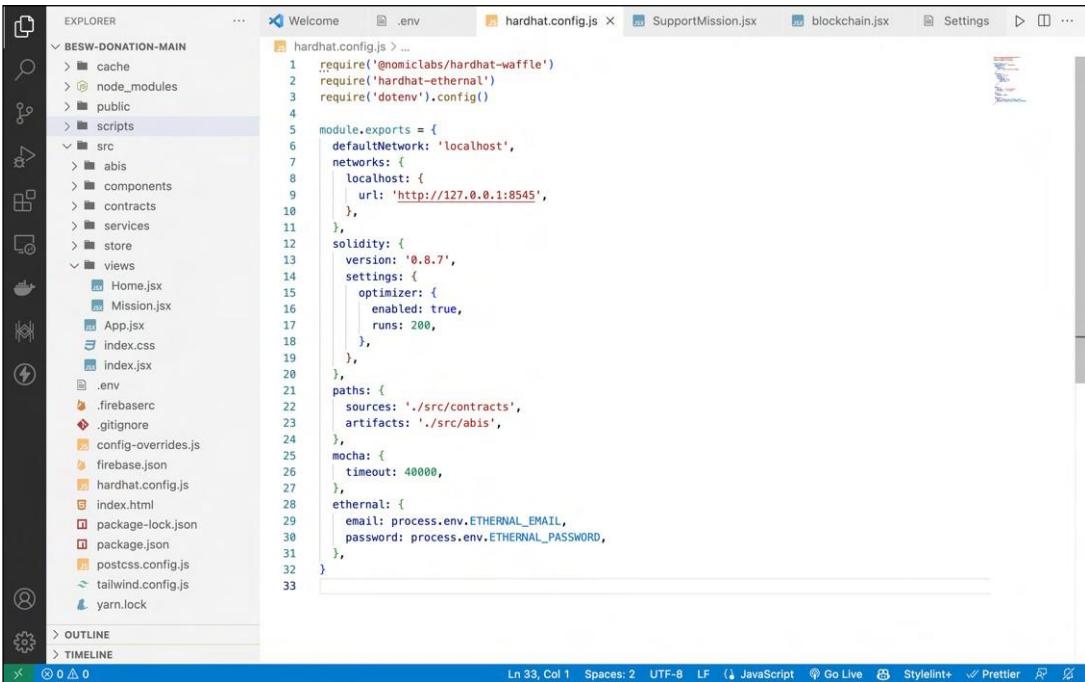
yarn hardhat run scripts/deploy.js is used to deploy the script on the hardhat local environment.

yarn start is used to start the whole project with all the components.



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE zsh +  
Account #16: 0x2546BcD3c84621e97608185a91A922aE77ECEc30 (10000 ETH)  
Private Key: 0xea6c44ac03bff858b476bba40716402b03e41b8e97e276d1baec7c37d42484a0  
Account #17: 0xbDA5747bfD65f08deb54cb465eB87D40e51B197e (10000 ETH)  
Private Key: 0x689af8efaf8c651a91ad287602527f3af2fe9f6501a7ac4b061667b5a93e037fd  
Account #18: 0xd02FD4581271e230360230F9337D5c430f44C0 (10000 ETH)  
Private Key: 0xde9be858da4a475276426320d5e9262ecfc3ba460bfa56360bfa6c4c28b4ee0  
Account #19: 0x8626f6940E2eb28930eFb4CeF49B2d1F2C9C1199 (10000 ETH)  
Private Key: 0xdf57089febbacf7ba0bc227dafbffa0fc08a93fd68e1e42411a14efcf23656e  
WARNING: These accounts, and their private keys, are publicly known.  
Any funds sent to them on Mainnet or any other live network WILL BE LOST.  
eth_chainId  
eth_accounts  
eth_blockNumber  
eth_chainId (2)  
eth_estimateGas  
eth_getBlockByNumber  
eth_feeHistory  
eth_sendTransaction  
  Contract deployment: DoNation  
  Contract address: 0x5fbdb2315678afebcb367f032d93f642f64180aa3  
  Transaction: 0xb56cf8f1e127fc68dab2b3b5e2adbaa39d274fca9a9041176ee3d3ef23ad89a1  
  From: 0xf39fd6e51aad88f6f4ce6ab8827279cffb92266  
  Value: 0 ETH  
  Gas used: 1979024 of 1979024  
  Block #1: 0x4178d85619953691cb24443ffec7f5411d70725cf0fc028af8e6588bcb64408a  
eth_chainId  
eth_getTransactionByHash  
eth_chainId  
eth_getTransactionReceipt
```

Fig 4.10 Block #1



```
EXPLORER .env hardhat.config.js SupportMission.js blockchain.js Settings ...  
BESW-DONATION-MAIN  
> cache  
> node_modules  
> public  
> scripts hardhat.config.js ...  
> src  
> abis  
> contracts  
> services  
> store  
> views  
  Home.jsx  
  Mission.jsx  
  App.jsx  
  index.css  
  index.jsx  
  .env  
  .firebaserc  
  .gitignore  
  config-overrides.js  
  firebase.json  
  hardhat.config.js  
  index.html  
  package-lock.json  
  package.json  
  postcss.config.js  
  tailwind.config.js  
  yarn.lock  
  
Ln 33, Col 1 Spaces: 2 UTF-8 LF ↵ JavaScript Go Live Stylelint+ Prettier ⌂ ⌂
```

Fig 4.11 hardhat.config.js

DOREQUEST PLATFORM

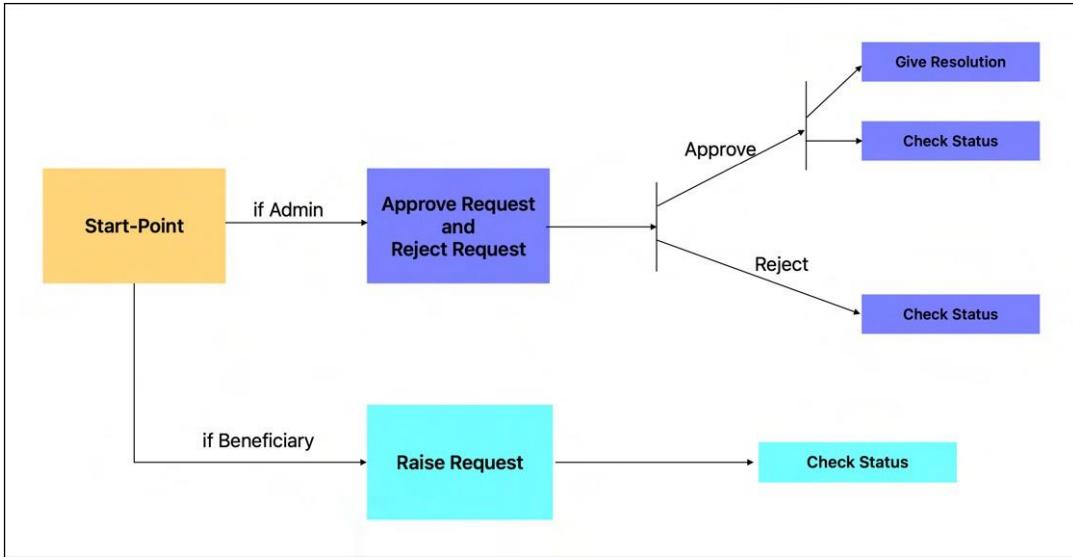


Fig 4.12 Flow of DoRequest Platform usage by various actors

This part of the project is done is using thirdweb React SDK, tailwind CSS and solidity for smart contract.

There are two actors of this platform- Admin and Beneficiary. Beneficiary raises the request and Admin can either approves it or declines it based on the information given by the beneficiary. Both admin and beneficiary can check the status of the request. Apart from these, admin can give resolution also in the end about the money that has been raised for the given cause.

For deploying the smart contract on thirdweb, a compiled contract artifact file is needed which can be generated by Hardhat during the development process. This artifact file contains the compiled bytecode, ABI, and other information needed to interact with the smart contract. Deployer metamask wallet address should be different from the NGO admin's as per thirdweb documentation for following the standards of good practice.

Different metamask wallet accounts and their roles

1. Deployer - the one who deploys the SC
2. NGO (admin) - only one account as per our smart contract
3. Beneficiary - can be more than 1

We are using Matic Mumbai Faucet to obtain testnet MATIC tokens for use on the Matic Mumbai network, which is a test network for the Polygon network. We are using these tokens to deploy and test SC and interact with our application on the network without having to use real MATIC tokens.

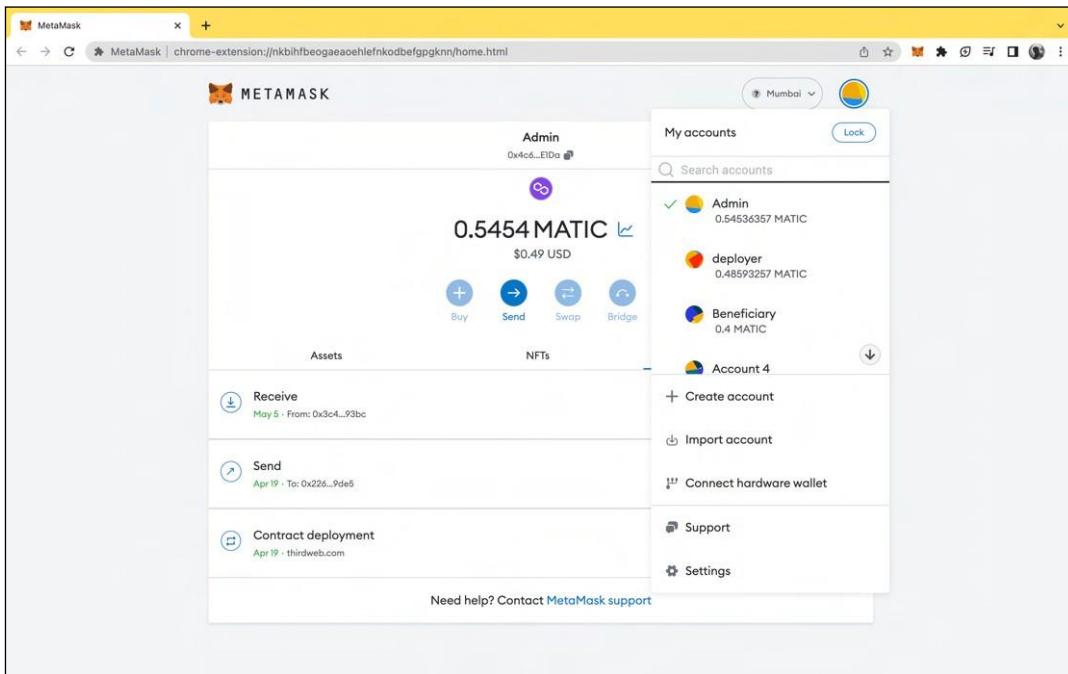


Fig 4.13 Different metamask wallet for different actors

Why Polygon?

This is a Layer 2 scaling solution built on top of the Ethereum blockchain network. To achieve faster transaction times and lower gas fees than Ethereum.

The screenshot shows the Visual Studio Code interface with the 'hardhat.config.js' file open in the editor. The code defines a Hardhat configuration object with the following properties:

```

module.exports = {
  solidity: {
    version: '0.8.9',
    defaultNetwork: 'mumbai',
  },
  networks: {
    hardhat: {},
    mumbai: {
      url: 'https://mumbai.rpc.thirdweb.com',
    },
  },
  settings: {
    optimizer: {
      enabled: true,
      runs: 200,
    },
  },
};

```

The 'EXPLORER' sidebar on the left shows the project structure, including files like 'artifacts', 'cache', 'contracts', 'node_modules', '.gitignore', 'deployArgs.json', 'hardhat.config.js', 'package-lock.json', 'package.json', 'README.md', and 'yarn.lock'. The 'OUTLINE' and 'TIMELINE' panels are also visible at the bottom of the sidebar.

Fig 4.14 hardhat.config.js

Deployment of SC

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a project structure under "DO_REQUEST" with "artifacts", "cache", and "contracts". "DoRequest.sol" is selected.
- Code Editor:** Displays the Solidity code for "DoRequest.sol".
- Terminal:** Shows the command line output:
 - A PowerShell prompt: PS C:\Users\Amal Prakash\Desktop\do_request>
 - The command: npm run deploy
 - Output:
 - > deploy
 - > npx thirdweb@latest deploy
- Status Bar:** Shows the current file is "DoRequest.sol", line 2, column 24, with 4 spaces, using UTF-8 encoding, and the Solidity language.
- Bottom Icons:** Includes icons for search, file operations, and developer tools like Prettier and ESLint.

Fig 4.15 `npm run deploy` – to deploy the smart contract.

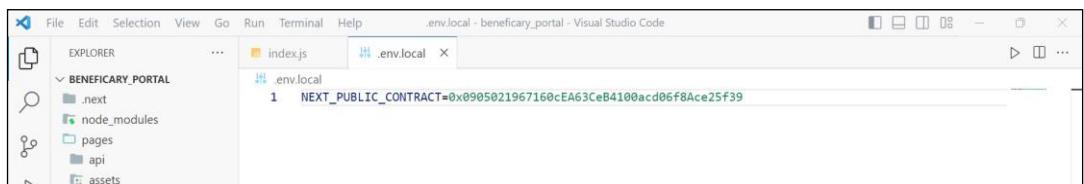


Fig 4.16 Deployed smart contract address stored in an environment variable.

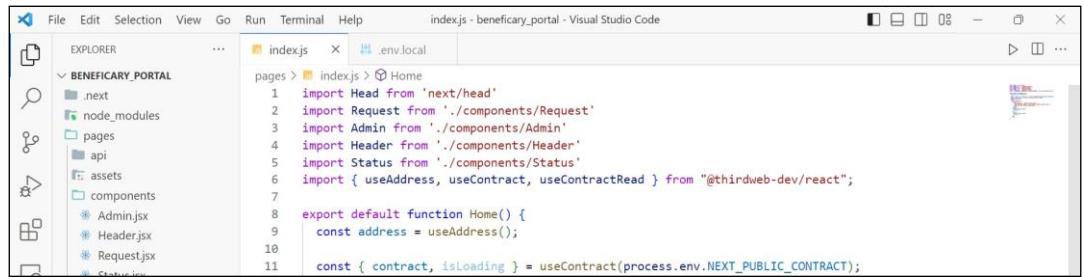


Fig 4.17 Use of environment variable to access the deployed smart contract.

The basic flow of the Platform

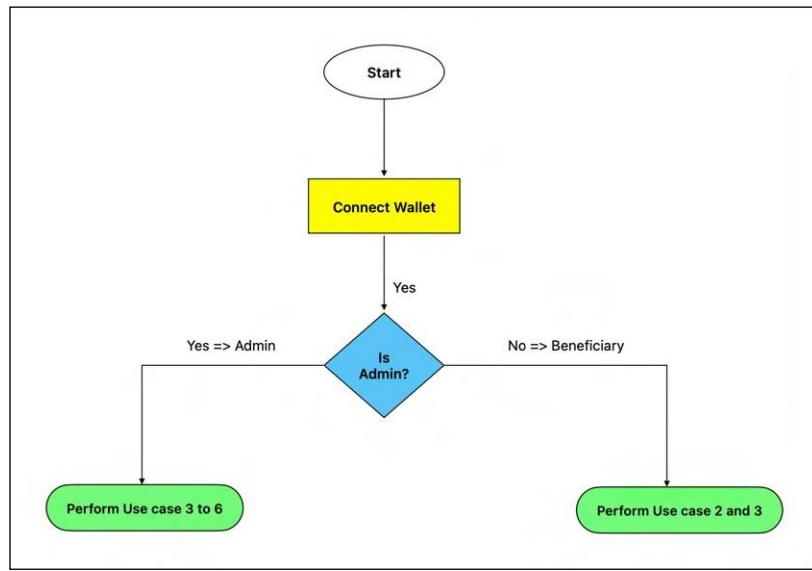


Fig 4.18 Flow of DoRequest

Use Case

1. Connect Wallet - admin + beneficiary
2. Raise Request - beneficiary
3. Check Status- admin + beneficiary
4. Approve Request - admin
5. Decline Request - admin
6. Give Resolution - admin

Use Case Diagram

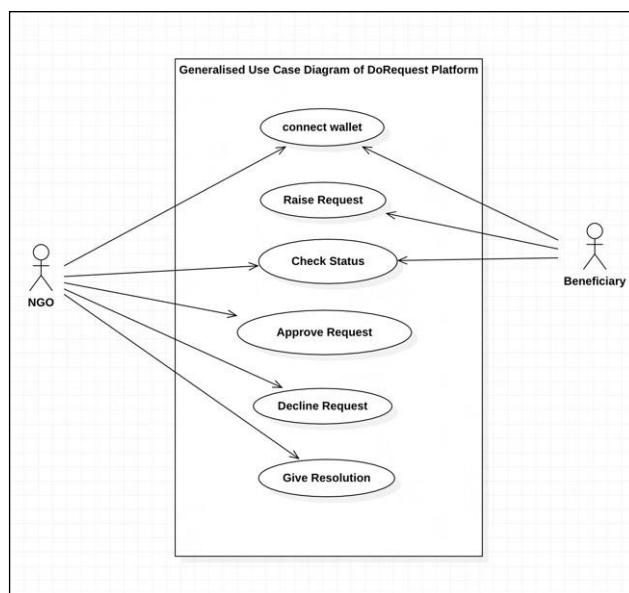


Fig 4.19 Use Case Diagram for DoRequest Platform

Sequence Diagram

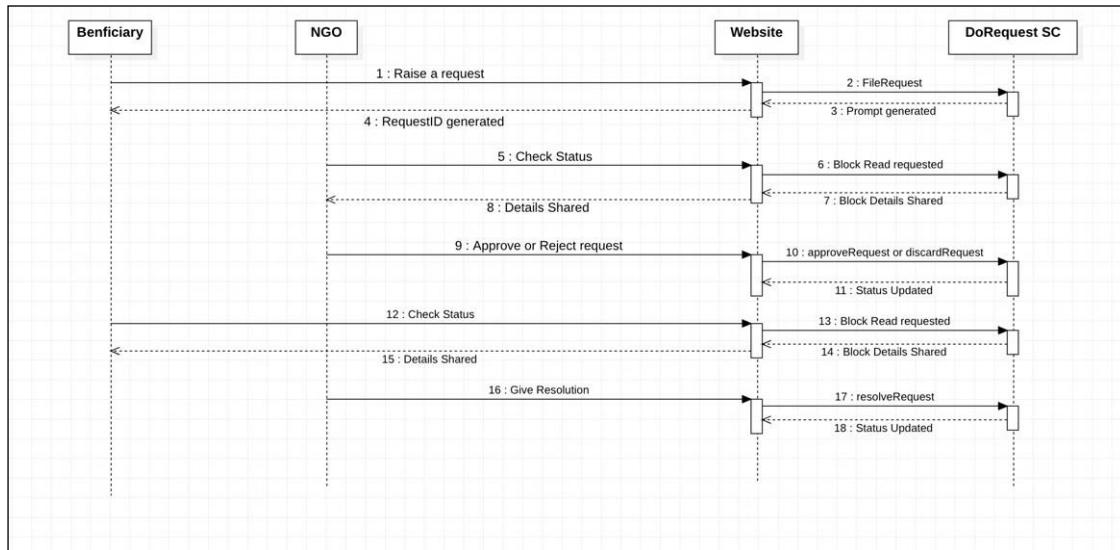


Fig 4.20 Sequence Diagram for DoRequest Platform

To Run the Application

Yarn is a package manager for Node.js that is used to manage dependencies and packages for a project. Yarn is generally faster than npm because it uses a caching mechanism that allows it to reuse previously downloaded packages.

A screenshot of the Visual Studio Code interface. The Explorer sidebar shows a project structure for 'BENEFICIARY_PORTAL' containing 'pages', 'api', 'assets', and 'components' folders, with 'Header.jsx' selected. The 'Header.jsx' file is open in the editor, showing its code. The Terminal tab at the bottom shows the command 'yarn dev' being run in a PowerShell window, with the output indicating that a development server is started on port 3000.

```
PS C:\Users\Amal Prakash\Desktop\DoRequest\frontend\beneficiary_portal> yarn dev
yarn run v1.22.19
$ next dev
  ready - started server on 0.0.0.0:3000, url: http://localhost:3000
  info - Loaded env from C:\Users\Amal Prakash\Desktop\DoRequest\frontend\beneficiary_portal\.env.local
```

Fig 4.21 `yarn dev` – to run the application

CHAPTER 5 - SOURCE CODE

ALGORITHMS IN THE BASE PAPER

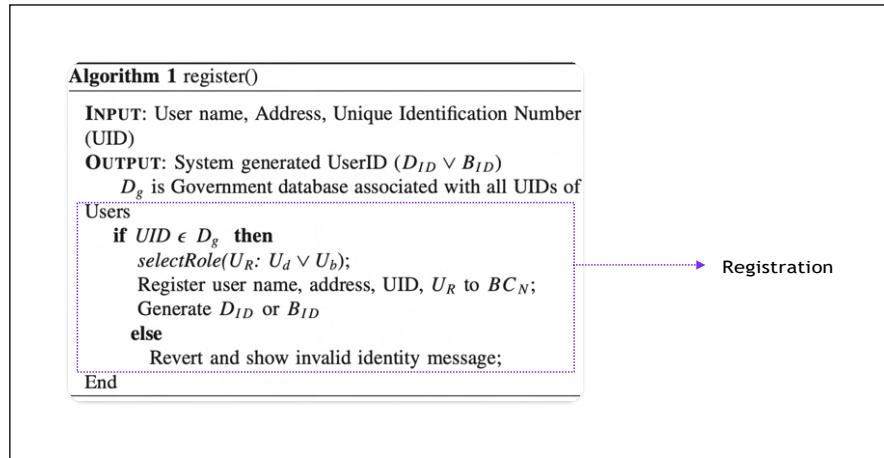


Fig 5.1 register()

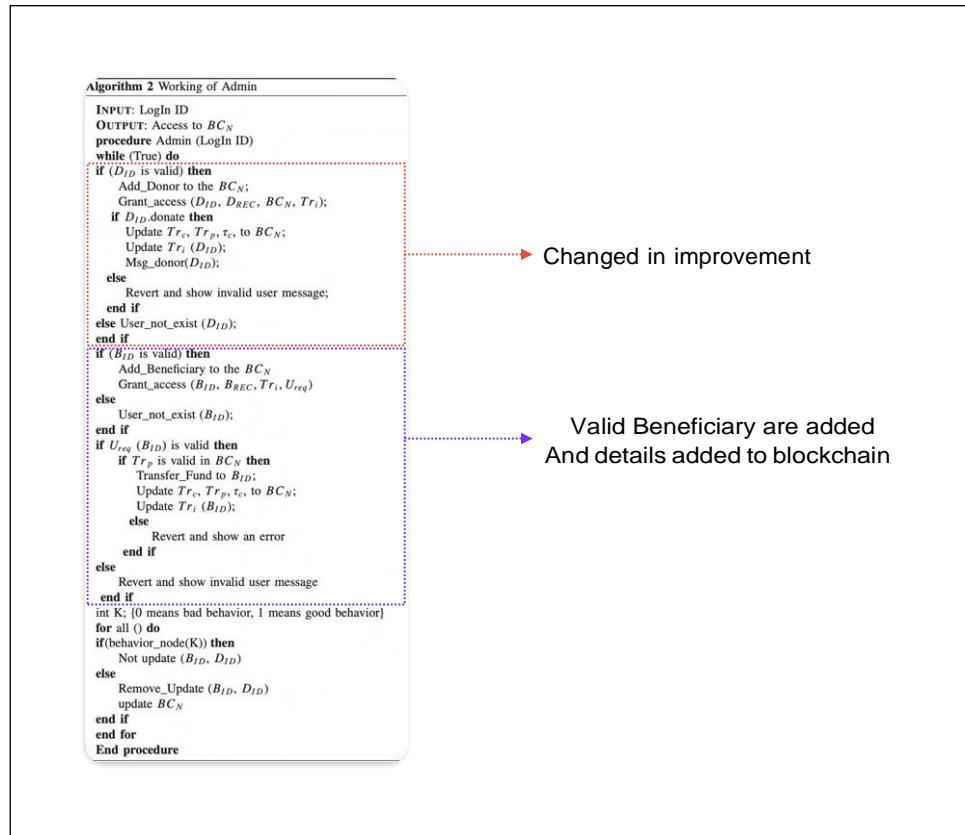


Fig 5.2 Working of admin()

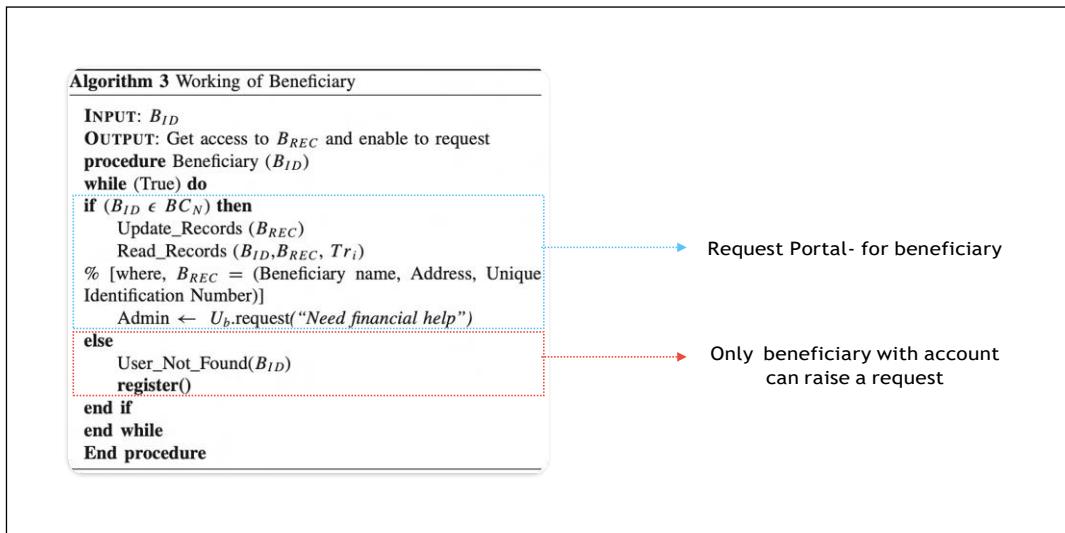


Fig 5.3 Working of Beneficiary()

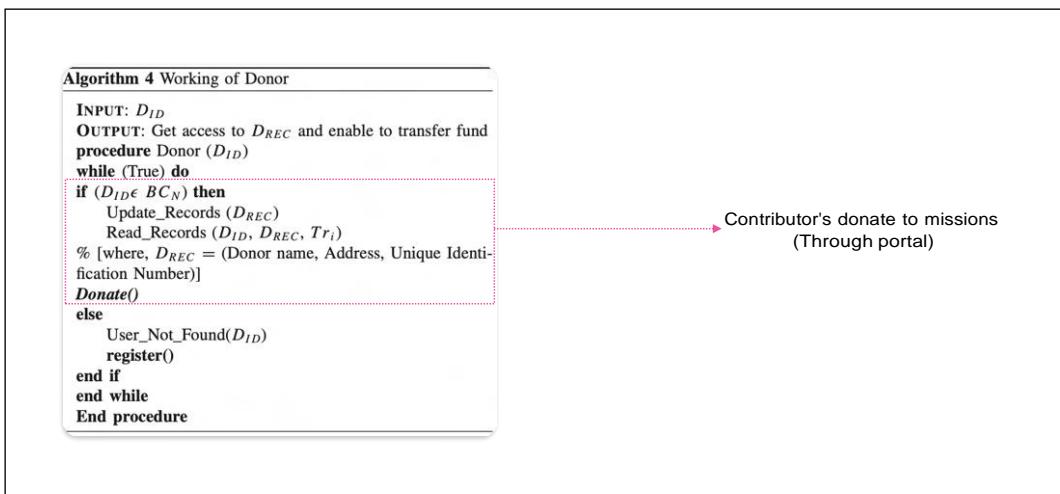


Fig 5.4 Working of Donor()

Registration of the stakeholders

As seen in Fig 5.1 the registration has 2 main steps:-

1. To see if the user is valid
2. To select their roles(donor or beneficiaries)

And thus allowing the respective users to register their accounts.

EXPLANATION OF SOLIDITY FUNCTIONS

1. DoNation.sol

The main functions of the DoNation.sol contract are:-

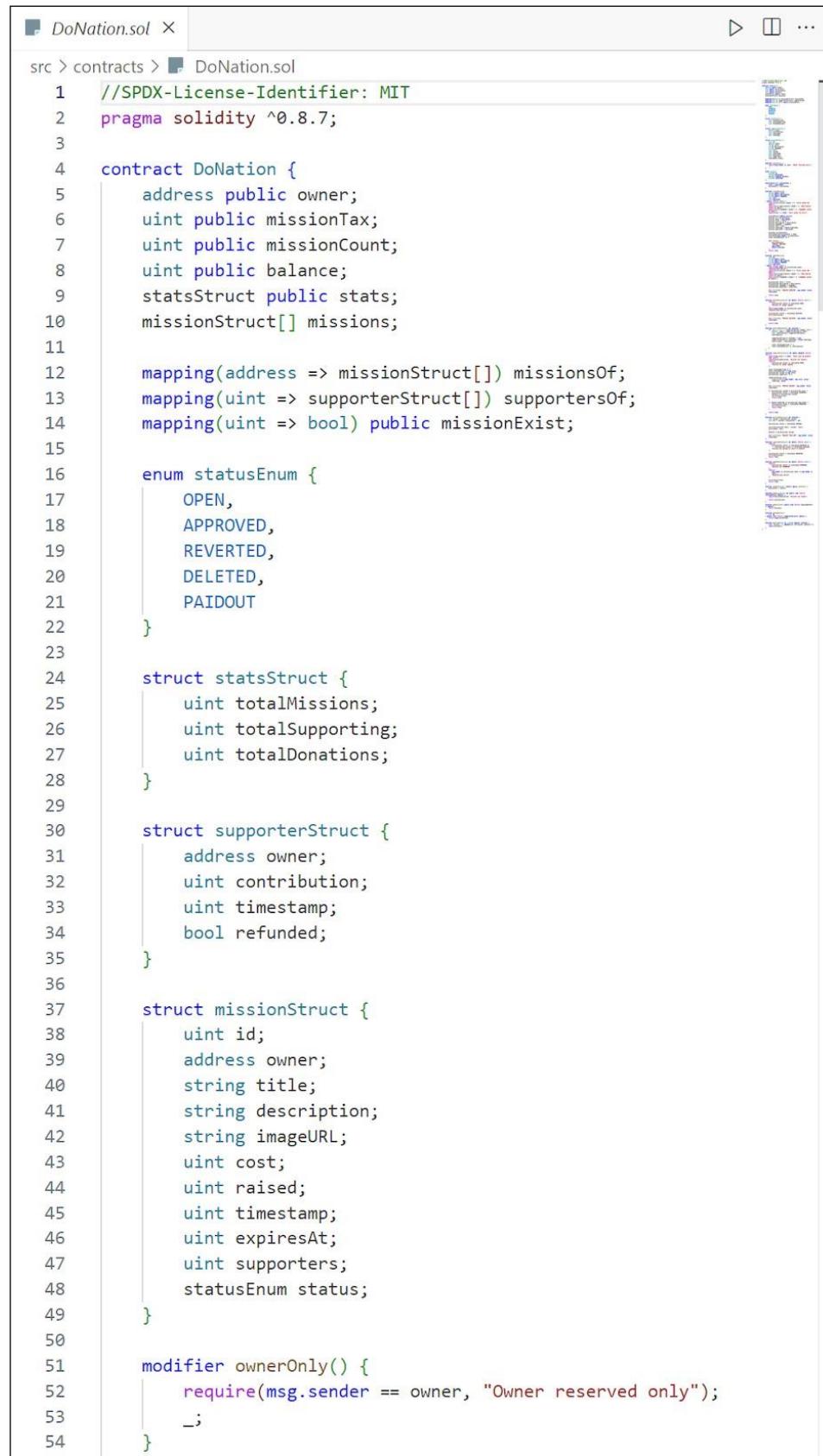
1. `createMission`:- only for admin, it handles the logic to add a new mission for the beneficiary
2. `updateMission`:- only for admin, as time passes some new changes are necessary to be made to the mission such as Date change.
It also handles the expiration date of the mission.
3. `performRefund`:- when in future the mission is not required, the admin deletes the project, thus the funds need to be sent back to the contributors, this function handles that logic
4. `supportMission`:- contributor donates the ETH to the mission
5. `getSupporters`:- lists all the contributors to a mission at the bottom of the page, promoting transparency.

2. DoRequest.sol

The main functions of DoRequest.sol contract are:-

1. `fileRequest` – only for the beneficiary, it allows a user to raise a new request and emit an event indicating that a new request has been filed. It also generates a unique request id.
2. `approveRequest` – only for admin, to approve the pending request
3. `discardRequest` - only for admin, to decline the pending request
4. `resolveRequest` - allows the admin to give final comments and mark a request as resolved.

CODE



The screenshot shows a code editor window with the file `DoNation.sol` open. The code is a Solidity smart contract named `DoNation`. The contract defines several variables, including `owner`, `missionTax`, `missionCount`, `balance`, `stats`, and `missions`. It also includes mapping variables for missions, supporters, and mission existence. The contract uses an enum `statusEnum` with values `OPEN`, `APPROVED`, `REVERTED`, `DELETED`, and `PAIDOUT`. It defines three structs: `statsStruct`, `supporterStruct`, and `missionStruct`. The `missionStruct` contains fields like `id`, `owner`, `title`, `description`, `imageURL`, `cost`, `raised`, `timestamp`, `expiresAt`, `supporters`, and `status`. A modifier `ownerOnly()` is defined to ensure only the owner can call certain functions.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

contract DoNation {
    address public owner;
    uint public missionTax;
    uint public missionCount;
    uint public balance;
    statsStruct public stats;
    missionStruct[] missions;

    mapping(address => missionStruct[]) missionsOf;
    mapping(uint => supporterStruct[]) supportersOf;
    mapping(uint => bool) public missionExist;

    enum statusEnum {
        OPEN,
        APPROVED,
        REVERTED,
        DELETED,
        PAIDOUT
    }

    struct statsStruct {
        uint totalMissions;
        uint totalSupporting;
        uint totalDonations;
    }

    struct supporterStruct {
        address owner;
        uint contribution;
        uint timestamp;
        bool refunded;
    }

    struct missionStruct {
        uint id;
        address owner;
        string title;
        string description;
        string imageURL;
        uint cost;
        uint raised;
        uint timestamp;
        uint expiresAt;
        uint supporters;
        statusEnum status;
    }

    modifier ownerOnly() {
        require(msg.sender == owner, "Owner reserved only");
        _;
    }
}
```

```
DoNation.sol X
src > contracts > DoNation.sol
  56     event Action(
  57         uint256 id,
  58         string actionType,
  59         address indexed executor,
  60         uint256 timestamp
  61     );
  62
  63     constructor(uint _missionTax) {
  64         owner = msg.sender;
  65         missionTax = _missionTax;
  66     }
  67
  68     function createMission(
  69         string memory title,
  70         string memory description,
  71         string memory imageURL,
  72         uint cost,
  73         uint expiresAt
  74     ) public returns (bool) {
  75         require(bytes(title).length > 0, "Title cannot be empty");
  76         require(bytes(description).length > 0, "Description cannot be empty");
  77         require(bytes(imageURL).length > 0, "ImageURL cannot be empty");
  78         require(cost > 0 ether, "Cost cannot be zero");
  79
  80         missionStruct memory mission;
  81         mission.id = missionCount;
  82         mission.owner = msg.sender;
  83         mission.title = title;
  84         mission.description = description;
  85         mission.imageURL = imageURL;
  86         mission.cost = cost;
  87         mission.timestamp = block.timestamp;
  88         mission.expiresAt = expiresAt;
  89
  90         missions.push(mission);
  91         missionExist[missionCount] = true;
  92         missionsOf[msg.sender].push(mission);
  93         stats.totalMissions += 1;
  94
  95         emit Action(
  96             missionCount++,
  97             "PROJECT CREATED",
  98             msg.sender,
  99             block.timestamp
 100         );
 101         return true;
 102     }
 103
 104     function updateMission(
 105         uint id,
 106         string memory title,
```

```
DoNation.sol ×
```

src > contracts > DoNation.sol

```
104     function updateMission(
105         uint id,
106         string memory title,
107         string memory description,
108         string memory imageURL,
109         uint expiresAt
110     ) public returns (bool) {
111         require(msg.sender == missions[id].owner,
112             "Unauthorized Entity");
113         require(bytes(title).length > 0, "Title cannot be
empty");
114         require(bytes(description).length > 0, "Description
cannot be empty");
115         require(bytes(imageURL).length > 0, "ImageURL cannot
be empty");
116
117         missions[id].title = title;
118         missions[id].description = description;
119         missions[id].imageURL = imageURL;
120         missions[id].expiresAt = expiresAt;
121
122         emit Action(id, "PROJECT UPDATED", msg.sender, block.
timestamp);
123
124         return true;
125     }
126
127     function deleteMission(uint id) public returns (bool) {
128         require(
129             missions[id].status == statusEnum.OPEN,
130             "Mission no longer opened"
131         );
132         require(msg.sender == missions[id].owner,
133             "Unauthorized Entity");
134
135         missions[id].status = statusEnum.DELETED;
136         performRefund(id);
137
138         emit Action(id, "PROJECT DELETED", msg.sender, block.
timestamp);
139
140         return true;
141     }
142     function performRefund(uint id) internal {
143         for (uint i = 0; i < supportersOf[id].length; i++) {
144             address _owner = supportersOf[id][i].owner;
145             uint _contribution = supportersOf[id][i].
contribution;
146
147             supportersOf[id][i].refunded = true;
148             supportersOf[id][i].timestamp = block.timestamp;
149             payTo(_owner, _contribution);
150         }
151     }
152 }
```

The screenshot shows a code editor window with the file 'DoNation.sol' open. The code is written in Solidity, a programming language for Ethereum smart contracts. The editor interface includes tabs for 'DoNation.sol' and 'src > contracts > DoNation.sol'. The code itself is a large function that handles various operations related to missions and donations.

```
function performRefund(uint id) internal {
    for (uint i = 0; i < supportersOf[id].length; i++) {
        address _owner = supportersOf[id][i].owner;
        uint _contribution = supportersOf[id][i].
            contribution;

        supportersOf[id][i].refunded = true;
        supportersOf[id][i].timestamp = block.timestamp;
        payTo(_owner, _contribution);

        stats.totalSupporting -= 1;
        stats.totalDonations -= _contribution;
    }
}

function supportMission(uint id) public payable returns
(bool) {
    require(msg.value > 0 ether, "Ether must be greater
than zero");
    require(missionExist[id], "Mission not found");
    require(
        missions[id].status == statusEnum.OPEN,
        "Mission no longer opened"
    );

    stats.totalSupporting += 1;
    stats.totalDonations += msg.value;
    missions[id].raised += msg.value;
    missions[id].supporters += 1;

    supportersOf[id].push(
        supporterStruct(msg.sender, msg.value, block.
            timestamp, false)
    );

    emit Action(id, "PROJECT BACKED", msg.sender, block.
        timestamp);

    if (missions[id].raised >= missions[id].cost) {
        missions[id].status = statusEnum.APPROVED;
        balance += missions[id].raised;
        performPayout(id);
        return true;
    }

    if (block.timestamp >= missions[id].expiresAt) {
        missions[id].status = statusEnum.REVERTED;
        performRefund(id);
        return true;
    }

    return true;
}
```

```
DoNation.sol X
src > contracts > DoNation.sol
107
190     function performPayout(uint id) internal {
191         uint raised = missions[id].raised;
192         uint tax = (raised * missionTax) / 100;
193
194         missions[id].status = statusEnum.PAIDOUT;
195
196         payTo(missions[id].owner, (raised - tax));
197         payTo(owner, tax);
198
199         balance -= missions[id].raised;
200
201         emit Action(id, "PROJECT PAID OUT", msg.sender, block.timestamp);
202     }
203
204     function requestRefund(uint id) public returns (bool) {
205         require(
206             missions[id].status != statusEnum.REVERTED ||
207             missions[id].status != statusEnum.DELETED,
208             "Mission not marked as revert or delete"
209         );
210
211         missions[id].status = statusEnum.REVERTED;
212         performRefund(id);
213         return true;
214     }
215
216     function payOutMission(uint id) public returns (bool) {
217         require(
218             missions[id].status == statusEnum.APPROVED,
219             "Mission not APPROVED"
220         );
221         require(
222             msg.sender == missions[id].owner || msg.sender ==
223             owner,
224             "Unauthorized Entity"
225         );
226
227         performPayout(id);
228         return true;
229     }
230
231     function changeTax(uint _taxPct) public ownerOnly {
232         missionTax = _taxPct;
233     }
234
235     function getMission(uint id) public view returns
236     (missionStruct memory) {
237         require(missionExist[id], "Mission not found");
238
239         return missions[id];
240     }
241
242     function getMissions() public view returns (missionStruct
```

```
DoNation.sol X
src > contracts > DoNation.sol
205     require(
206         missions[id].status != statusEnum.REVERTED ||
207         missions[id].status != statusEnum.DELETED,
208         "Mission not marked as revert or delete"
209     );
210
211     missions[id].status = statusEnum.REVERTED;
212     performRefund(id);
213     return true;
214 }
215
216     function payOutMission(uint id) public returns (bool) {
217         require(
218             missions[id].status == statusEnum.APPROVED,
219             "Mission not APPROVED"
220         );
221         require(
222             msg.sender == missions[id].owner || msg.sender ==
223             owner,
224             "Unauthorized Entity"
225         );
226
227         performPayout(id);
228         return true;
229     }
230
231     function changeTax(uint _taxPct) public ownerOnly {
232         missionTax = _taxPct;
233     }
234
235     function getMission(uint id) public view returns
236     (missionStruct memory) {
237         require(missionExist[id], "Mission not found");
238
239         return missions[id];
240     }
241
242     function getMissions() public view returns (missionStruct
243     [] memory) {
244         return missions;
245     }
246
247     function getSupporters(
248         uint id
249     ) public view returns (supporterStruct[] memory) {
250         return supportersOf[id];
251     }
252
253     function payTo(address to, uint256 amount) internal {
254         (bool success, ) = payable(to).call{value: amount}("");
255         require(success);
256     }
257 }
```

```
hardhat.config.js  DoRequest.sol X ...  
contracts > DoRequest.sol  
1 // SPDX-License-Identifier: MIT  
2 pragma solidity ^0.8.9;  
3  
4 contract DoRequest {  
5     address public admin;  
6     address public owner;  
7     uint256 public nextId;  
8     uint256[] public pendingApprovals;  
9     uint256[] public pendingResolutions;  
10    uint256[] public resolvedMission;  
11  
12    constructor(address _admin) {  
13        owner = msg.sender;  
14        admin = _admin;  
15        nextId = 1;  
16    }  
17  
18    modifier onlyOwner() {  
19        require(  
20            msg.sender == owner,  
21            "You are not the owner of this smart contract"  
22        );  
23        _;  
24    }  
25  
26    modifier onlyAdmin() {  
27        require(  
28            msg.sender == admin,  
29            "You are not registered admin of this smart  
30            contract"  
31        );  
32        _;  
33    }  
34  
35    struct request {  
36        uint256 id;  
37        address requestRegisteredBy;  
38        string title;  
39        string description;  
40        string beneficiaryName;  
41        uint256 beneficiaryPhone;  
42        uint256 beneficiaryAadhar;  
43        string approvalRemark;  
44        string resolutionRemark;  
45        bool isApproved;  
46        bool isResolved;  
47        bool exists;  
48    }  
49    mapping(uint256 => request) public Requests;  
50  
51    event requestFiled(  
52        uint256 id,  
53        address requestRegisteredBy,  
54        string title,
```

```
hardhat.config.js  DoRequest.sol X ...  
contracts > DoRequest.sol  
53     string title,  
54     string description,  
55     string beneficiaryName,  
56     uint256 beneficiaryPhone,  
57     uint256 beneficiaryAadhar  
58 );  
59  
60     function fileRequest(  
61         string memory _title,  
62         string memory _description,  
63         string memory _beneficiaryName,  
64         uint256 _beneficiaryPhone,  
65         uint256 _beneficiaryAadhar  
66     ) public {  
67         request storage newRequest = Requests[nextId];  
68         newRequest.id = nextId;  
69         newRequest.requestRegisteredBy = msg.sender;  
70         newRequest.title = _title;  
71         newRequest.description = _description;  
72         newRequest.beneficiaryName = _beneficiaryName;  
73         newRequest.beneficiaryPhone = _beneficiaryPhone;  
74         newRequest.beneficiaryAadhar = _beneficiaryAadhar;  
75         newRequest.approvalRemark = "Pending Approval";  
76         newRequest.resolutionRemark = "Pending Resolution";  
77         newRequest.isApproved = false;  
78         newRequest.isResolved = false;  
79         newRequest.exists = true;  
80         emit requestFiled(  
81             nextId,  
82             msg.sender,  
83             _title,  
84             _description,  
85             _beneficiaryName,  
86             _beneficiaryPhone,  
87             _beneficiaryAadhar  
88         );  
89         nextId++;  
90     }  
91  
92     function approveRequest(  
93         uint256 _id,  
94         string memory _approvalRemark  
95     ) public onlyAdmin {  
96         require(Requests[_id].exists == true, "This request id  
97         does not exist");  
98         require(  
99             Requests[_id].isApproved == false,  
100            "Request is already approved"  
101        );  
102         Requests[_id].isApproved = true;  
103         Requests[_id].approvalRemark = _approvalRemark;  
104     }  
105    function discardRequest(  
106        uint256 _id,
```

hardhat.config.js DoRequest.sol

contracts > DoRequest.sol

```

105     function discardRequest(
106         uint256 _id,
107         string memory _approvalRemark
108     ) public onlyAdmin {
109         require(Requests[_id].exists == true, "This request id
110             does not exist");
111         require(
112             Requests[_id].isApproved == false,
113             "Request is already approved"
114         );
115         Requests[_id].exists = false;
116         Requests[_id].approvalRemark = _approvalRemark;
117     }
118
119     function resolveRequest(
120         uint256 _id,
121         string memory _resolutionRemark
122     ) public onlyAdmin {
123         require(Requests[_id].exists == true, "This request id
124             does not exist");
125         require(
126             Requests[_id].isApproved == true,
127             "Request is not yet approved"
128         );
129         require(
130             Requests[_id].isResolved == false,
131             "Request is already resolved"
132         );
133         Requests[_id].isResolved = true;
134         Requests[_id].resolutionRemark = _resolutionRemark;
135     }
136
137     function calcPendingApprovalIds() public {
138         delete pendingApprovals;
139         for (uint256 i = 1; i < nextId; i++) {
140             if (Requests[i].isApproved == false && Requests[i].
141                 exists == true) {
142                 pendingApprovals.push(Requests[i].id);
143             }
144         }
145
146         function calcPendingResolutionIds() public {
147             delete pendingResolutions;
148             for (uint256 i = 1; i < nextId; i++) {
149                 if (
150                     Requests[i].isResolved == false &&
151                     Requests[i].isApproved == true &&
152                     Requests[i].exists == true
153                 ) {
154                     pendingResolutions.push(Requests[i].id);
155                 }
156             }
157         }
158     }

```

```
hardhat.config.js  DoRequest.sol X ...
contracts > DoRequest.sol
115
116
117
118
119
120     string memory _resolutionRemark
121 } public onlyAdmin {
122     require(Requests[_id].exists == true, "This request id
123 does not exist");
124     require(
125         Requests[_id].isApproved == true,
126         "Request is not yet approved"
127     );
128     require(
129         Requests[_id].isResolved == false,
130         "Request is already resolved"
131     );
132     Requests[_id].isResolved = true;
133     Requests[_id].resolutionRemark = _resolutionRemark;
134 }
135
136 function calcPendingApprovalIds() public {
137     delete pendingApprovals;
138     for (uint256 i = 1; i < nextId; i++) {
139         if (Requests[i].isApproved == false && Requests[i].
140             exists == true) {
141             pendingApprovals.push(Requests[i].id);
142         }
143     }
144
145 function calcPendingResolutionIds() public {
146     delete pendingResolutions;
147     for (uint256 i = 1; i < nextId; i++) {
148         if (
149             Requests[i].isResolved == false &&
150             Requests[i].isApproved == true &&
151             Requests[i].exists == true
152         ) {
153             pendingResolutions.push(Requests[i].id);
154         }
155     }
156
157 function calcResolvedIds() public {
158     delete resolvedMission;
159     for (uint256 i = 1; i < nextId; i++) {
160         if (Requests[i].isResolved == true) {
161             resolvedMission.push(Requests[i].id);
162         }
163     }
164
165
166 function setAdminAddress(address _admin) public onlyOwner {
167     admin = _admin;
168 }
169
170 }
```

CHAPTER 6 - SNAPSHOTS IN DIFFERENT SCENARIOS

HOME PAGE

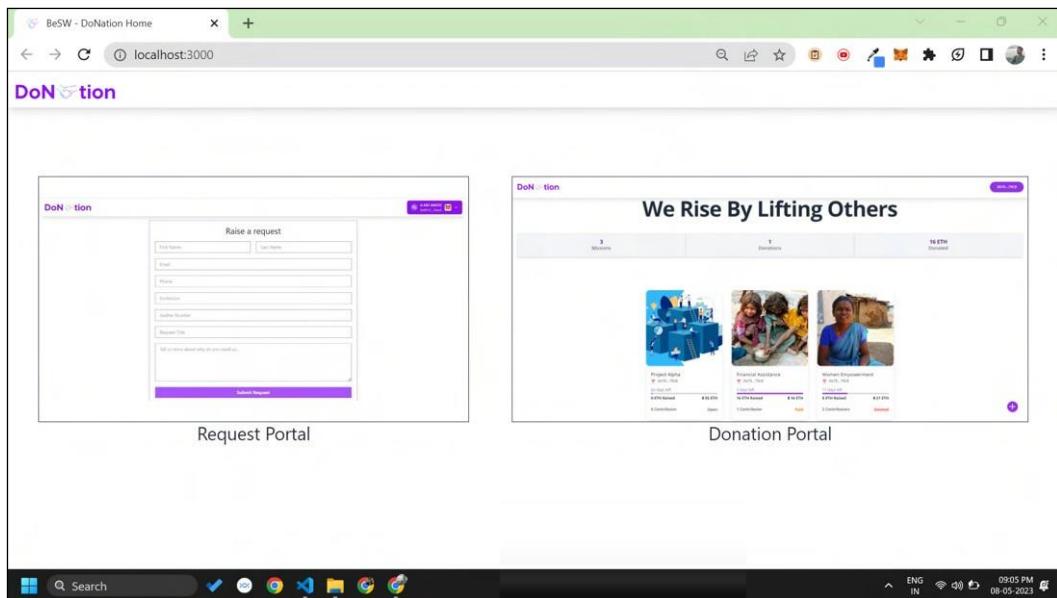


Fig 6.1 Home Page

The Home Page directs to two pages DoNation Platform and DoRequest Platform

DONATION WORKING

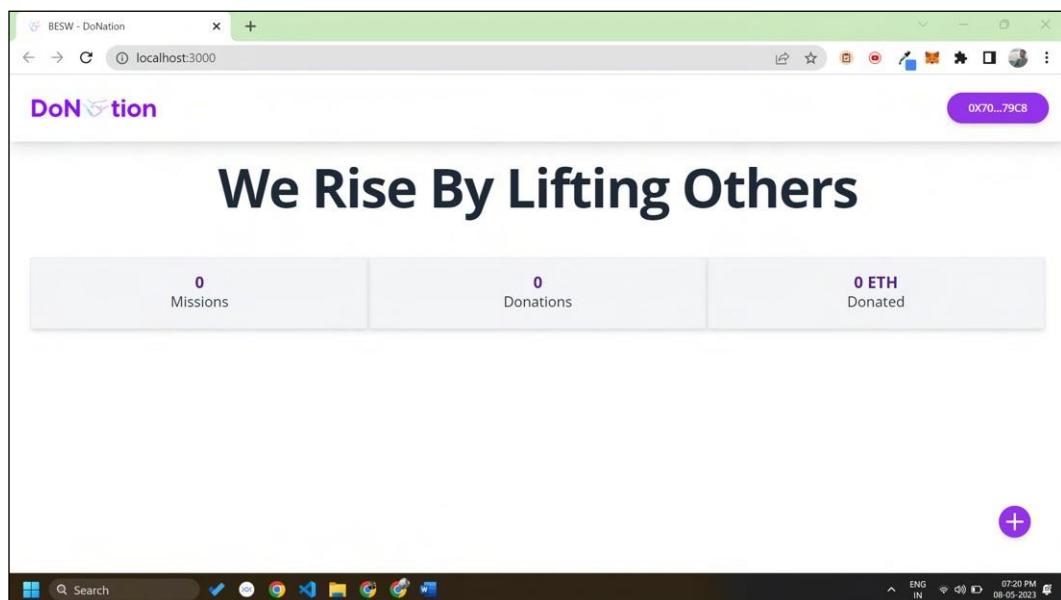


Fig 6.2 Admin View

Admin has the access to + button to add missions

```

20      },
21      paths: {
22        sources: './src/contracts',
23        artifacts: './src/abis'.
24
25      eth_sendTransaction
26        Contract deployment: DoNation
27        Contract address: 0x5fbdb2315678afebcb367f032d93f642f64180aa3
28        Transaction: 0x5fcf81e127fc68ab2b3b5e2adbaa39d74fcfa9a9041176ee3d3ef23ad89a1
29        From: 0x739fdfe51aad88f614ce6ab8827279cfffb92266
30        Value: 0 ETH
31        Gas used: 1979024 of 1979024
32        Block #: 0x4493e9a8a74066ac57771e573a154111463694efca2d8a3583489ef0784e4ef2
33
34      eth_chainId
35      eth_getTransactionByHash
36      eth_chainId
37      eth_getTransactionReceipt
38      eth_chainId
39      eth_accounts
40      eth_blockNumber
41      eth_chainId (2)
42      eth_estimateGas
43      eth_getBlockByNumber
44      eth_feeHistory
45      eth_sendTransaction
46        Contract deployment: DoNation
47        Contract address: 0xe771725e7734cc288f8367e1bb143e90bb3f0512
48        Transaction: 0x48feabe779ec94f1e049cc77cadede024d3ac2f299814fb9f73814819900669
49        From: 0xf39fdfe51aad88f614ce6ab8827279cfffb92266
50        Value: 0 ETH
51        Gas used: 1979024 of 1979024
52        Block #: 0xd0913d5d9491f0bc496bc37ff4658f68e64fd3913572cddcc156366a2b76a013
53
54      eth_chainId
55      eth_getTransactionByHash
56      eth_chainId
57      eth_getTransactionReceipt
58      eth_blockNumber
59      net_version
60      eth_getBlockByNumber

```

Fig 6.3 Block #2

After adding a mission a block is generated, here in this case the block is Block #2

The contributor uses his Metamask wallet to do the transaction.

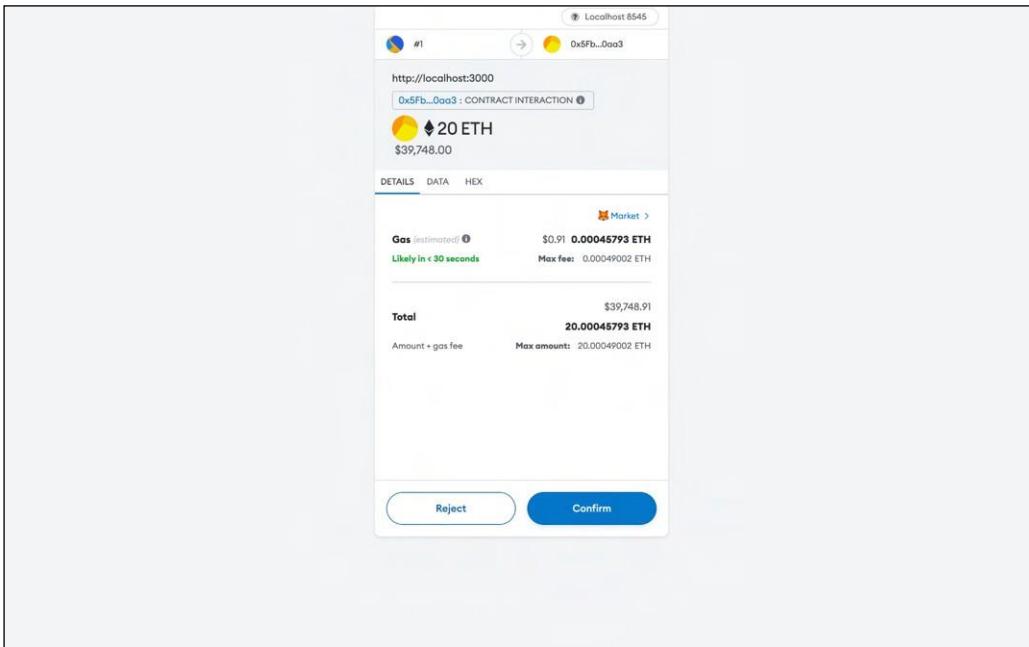


Fig 6.4 Contributor Payment

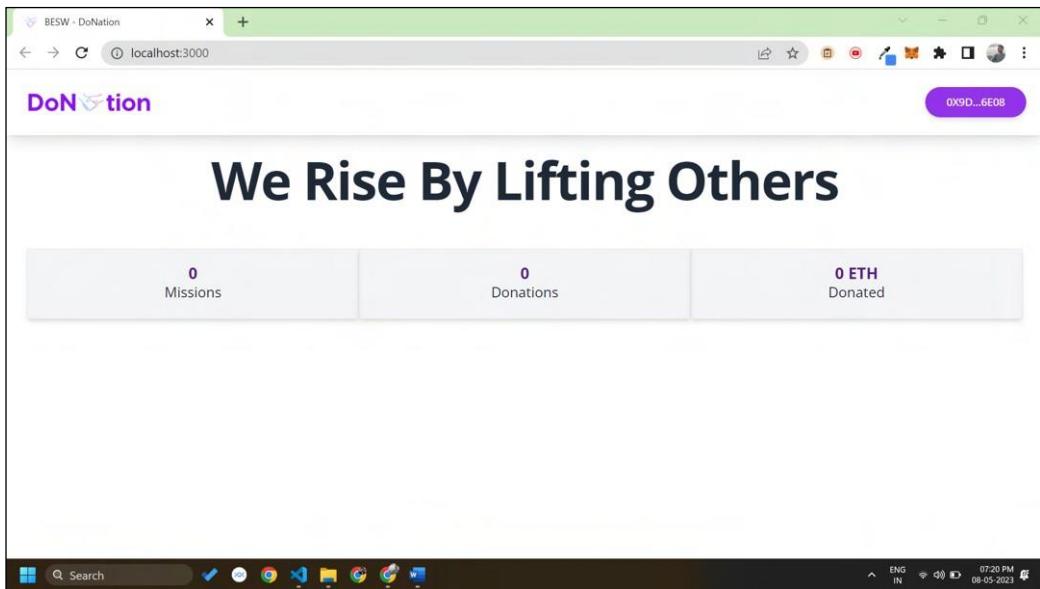


Fig 6.5 Contributor View

Note that the contributor doesn't have the + button to add missions.

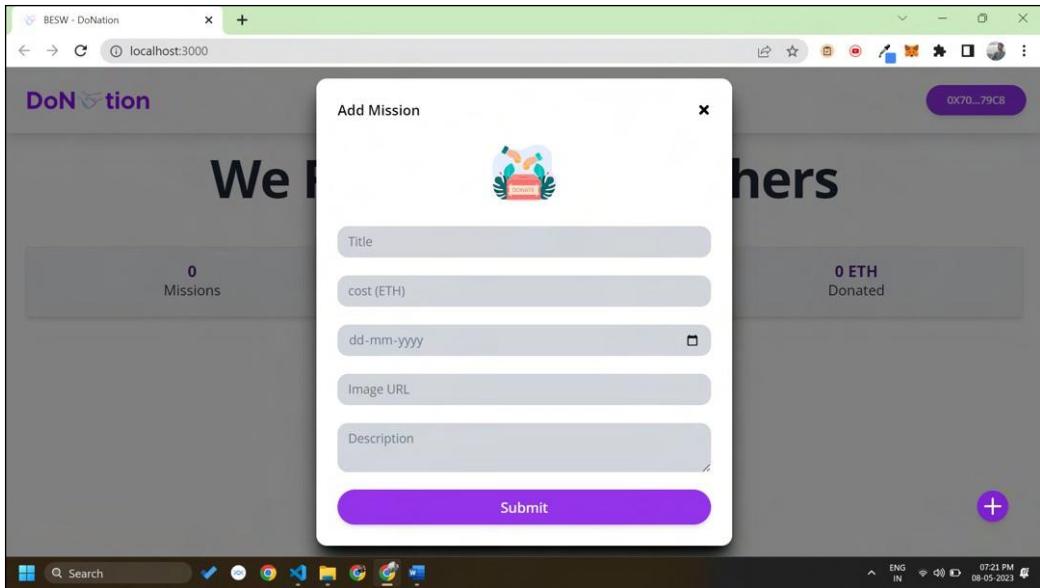


Fig 6.6 Admin adding Misson

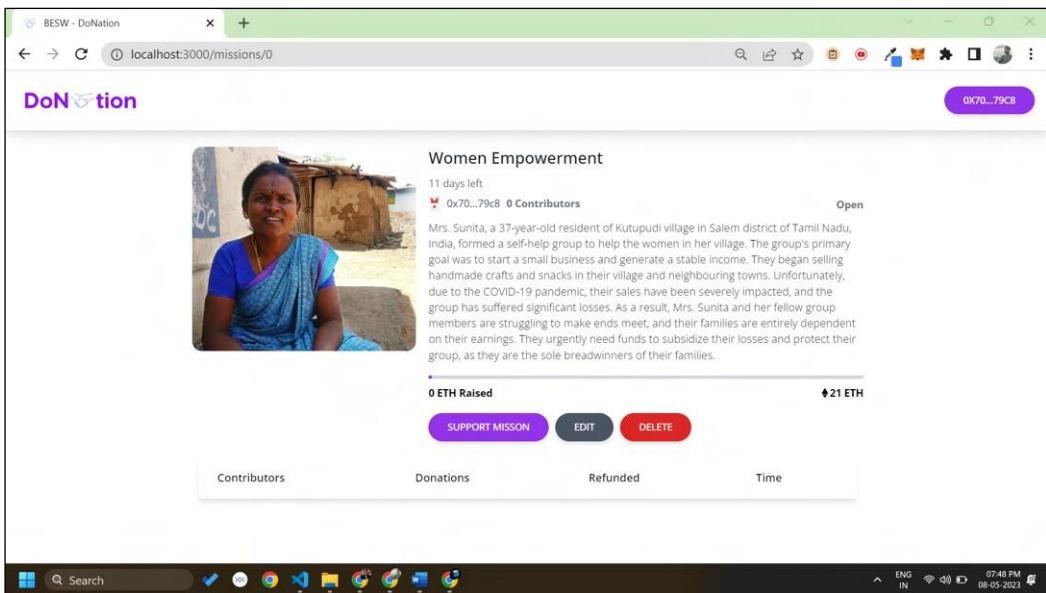


Fig 6.7 Mission's View (for admin)

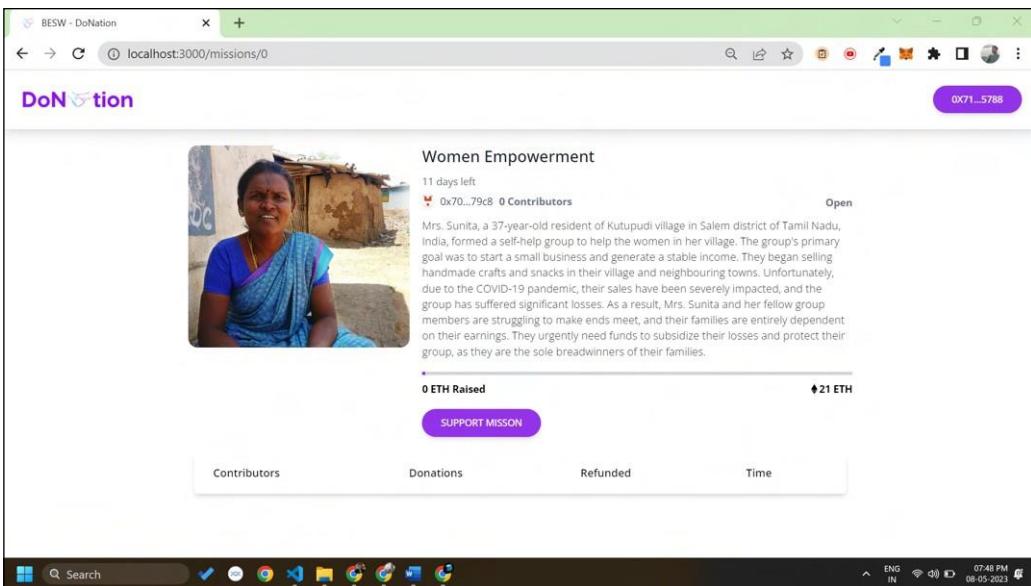


Fig 6.8 Mission's View (for contributor)

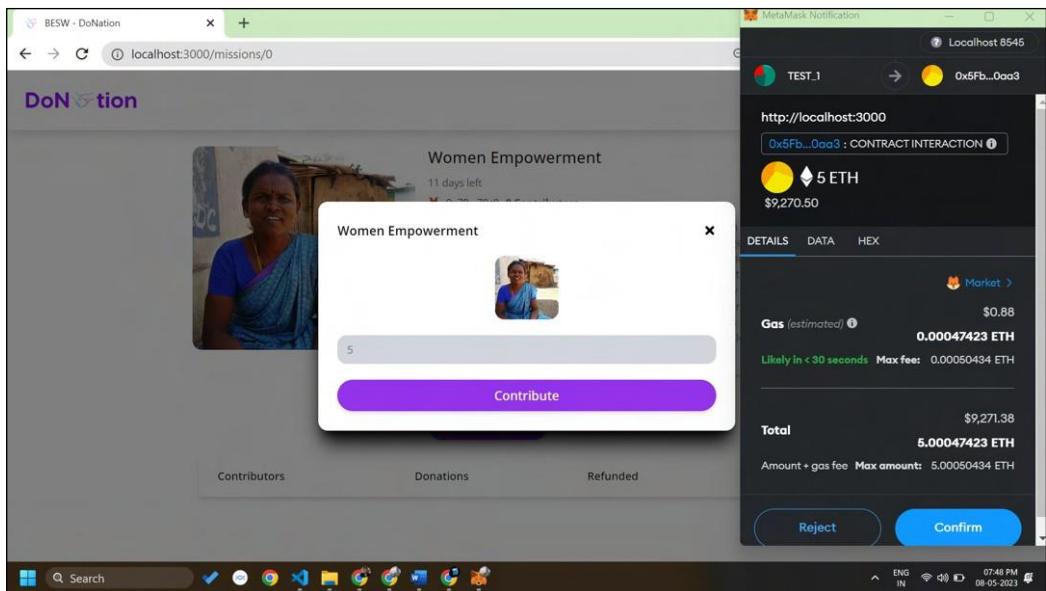


Fig 6.9 Contributing to the Mission

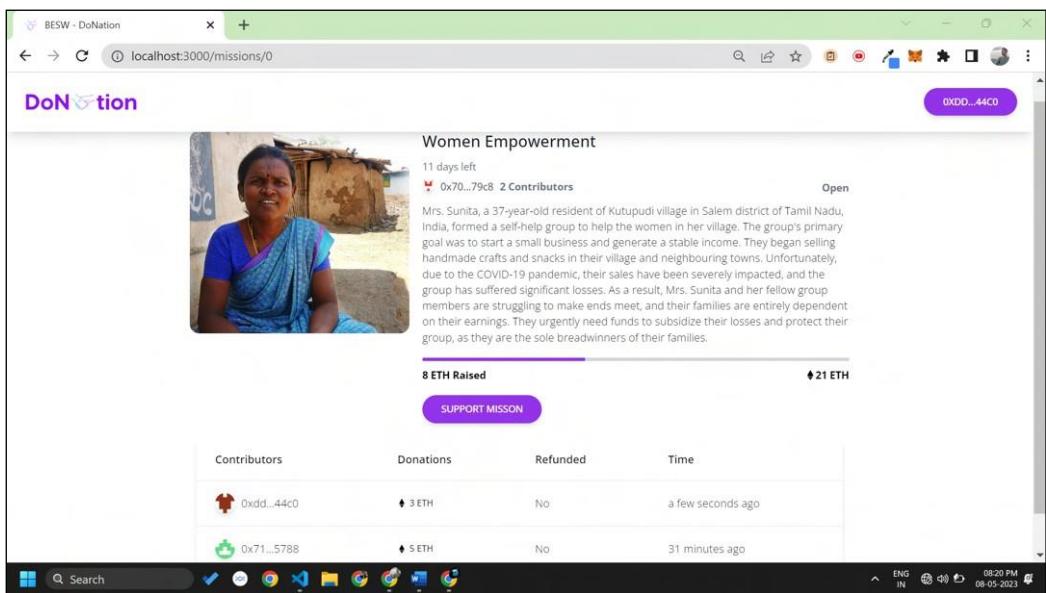


Fig 6.10 List of contributors

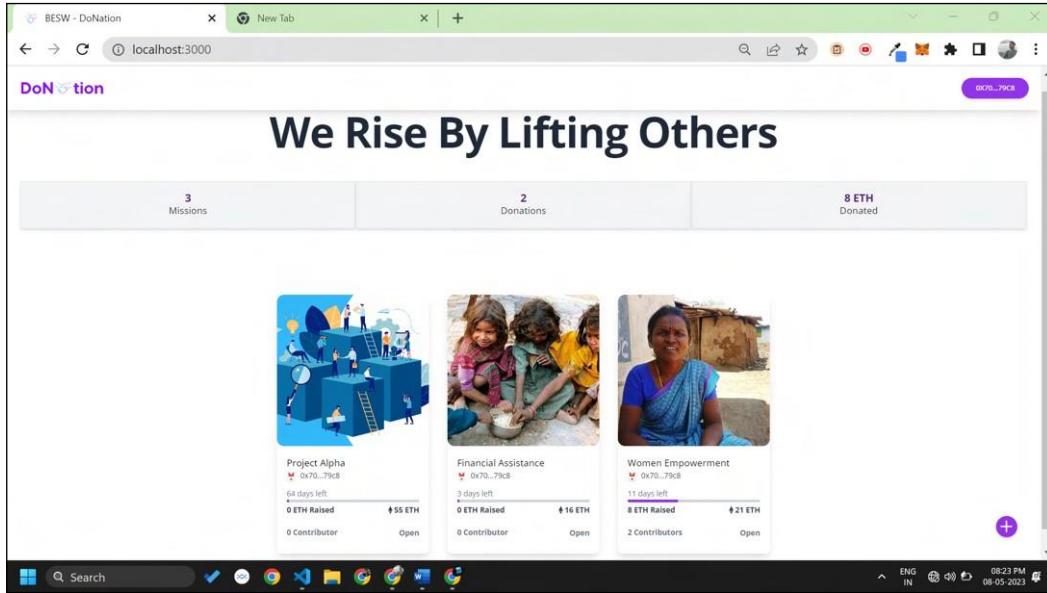


Fig 6.11 DoNation Dashboard View

Txn Hash	Method	Block	Mined On	From	To	Value	Fee
0x69599a58...82cab	0x634fb230	12	03/19 7:41:34 PM 5 hours ago	0x70997970...c79c8	0x5fdbdb231...80aa3	10 Ether	0.000392560053186315 Ether
0xc7557c3c...9823c	0x899d0e67	11	03/19 7:40:30 PM 5 hours ago	0x70997970...c79c8	0x5fdbdb231...80aa3	0 Ether	0.000100178355823414 Ether
0xbc044758...dcaff	0xf77b58db	10	03/19 7:39:53 PM 5 hours ago	0x70997970...c79c8	0x5fdbdb231...80aa3	0 Ether	0.001019162522005651 Ether
0xde5fea4...12f2b	0x634fb230	9	03/19 7:37:00 PM 5 hours ago	0x71be63f3...c5788	0x5fdbdb231...80aa3	1 Ether	0.000327403449158391 Ether
0x73421d77...16f00	0x634fb230	8	03/19 7:36:05 PM 5 hours ago	0x71be63f3...c5788	0x5fdbdb231...80aa3	18 Ether	0.000400631020675444 Ether
0x0ce802ad...8729f	0xf77b58db	7	03/19 7:35:43 PM	0x70997970...c79c8	0x5fdbdb231...80aa3	0 Ether	0.001179704965795785 Ether

Fig 6.12 Ethernal Transaction's View

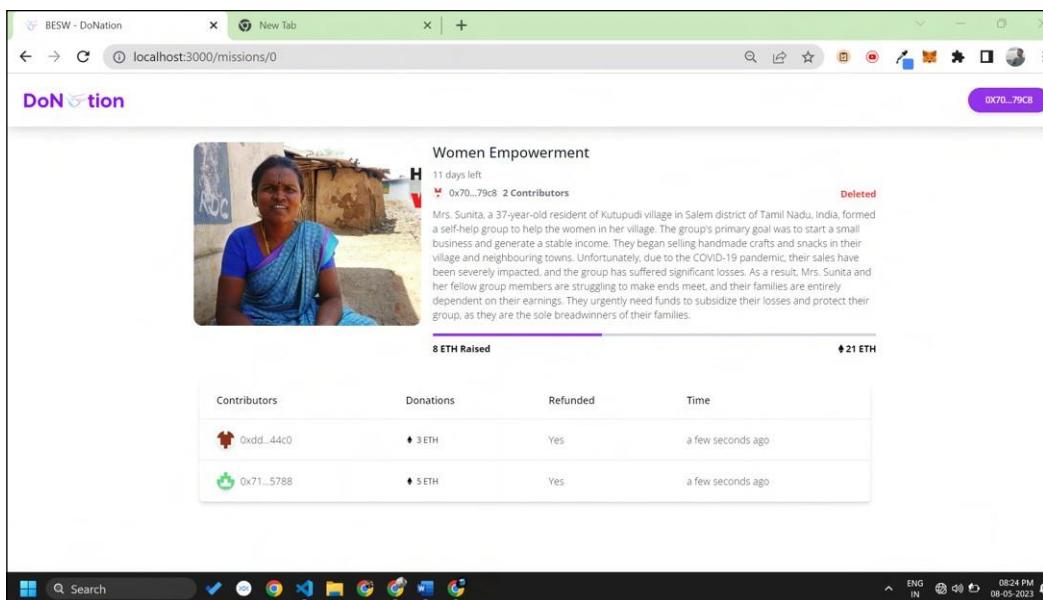


Fig 6.13 Refunded Status

After the mission is deleted/expired the contributors' money is refunded back.

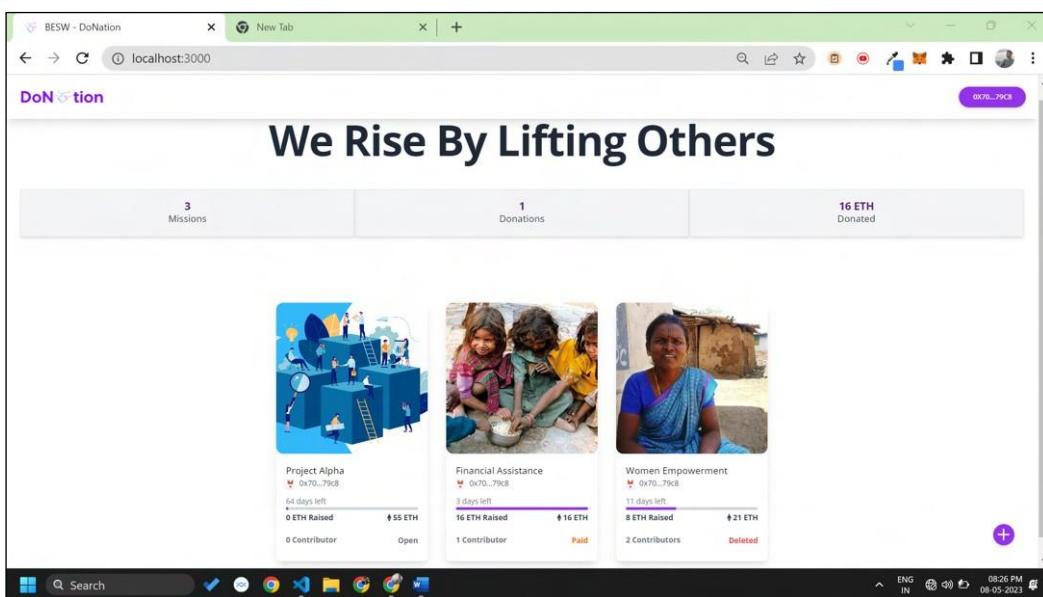


Fig 6.14 Normal Use Case View

DOREQUEST WORKING

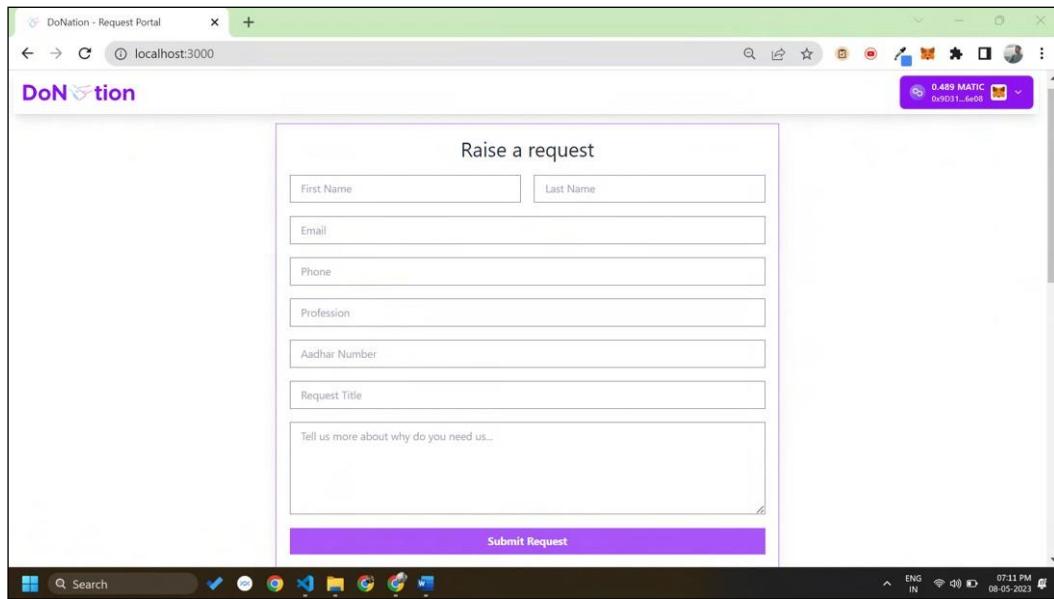


Fig 6.15 Beneficiary View

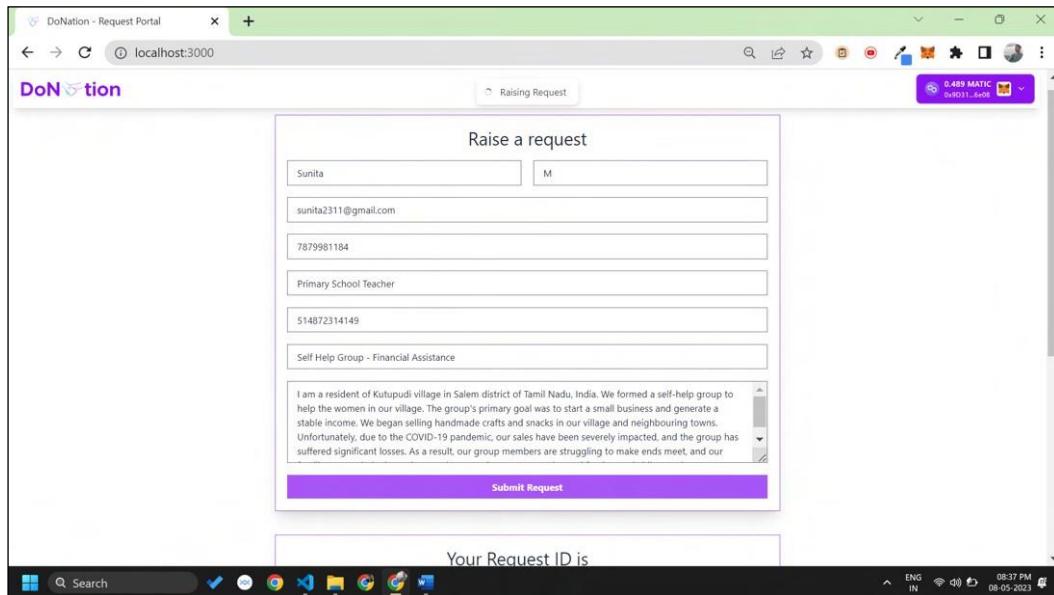


Fig 6.16 Beneficiary Filing the request

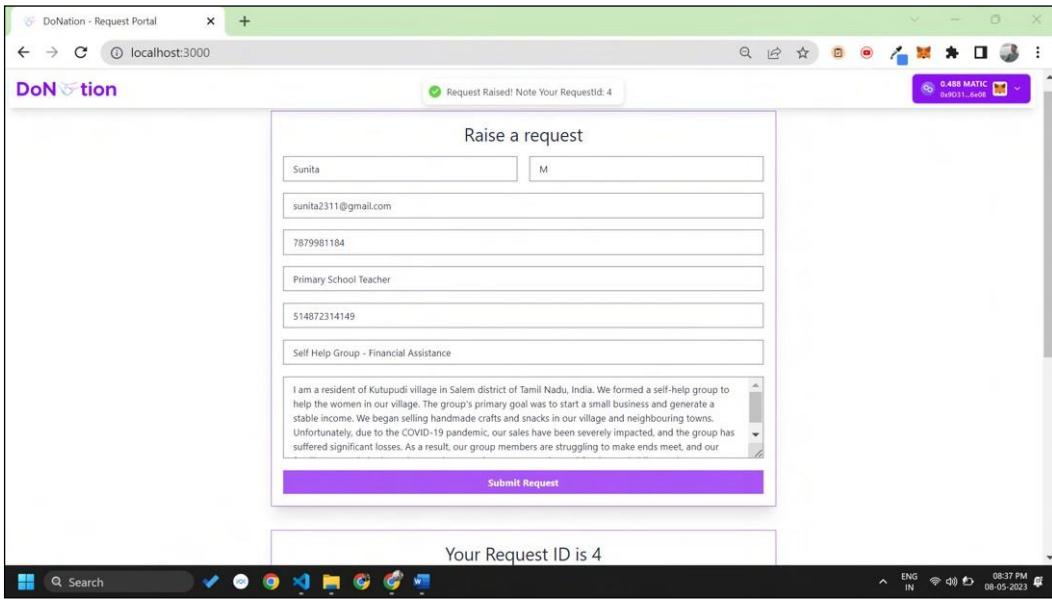


Fig 6.17 RequestID generated

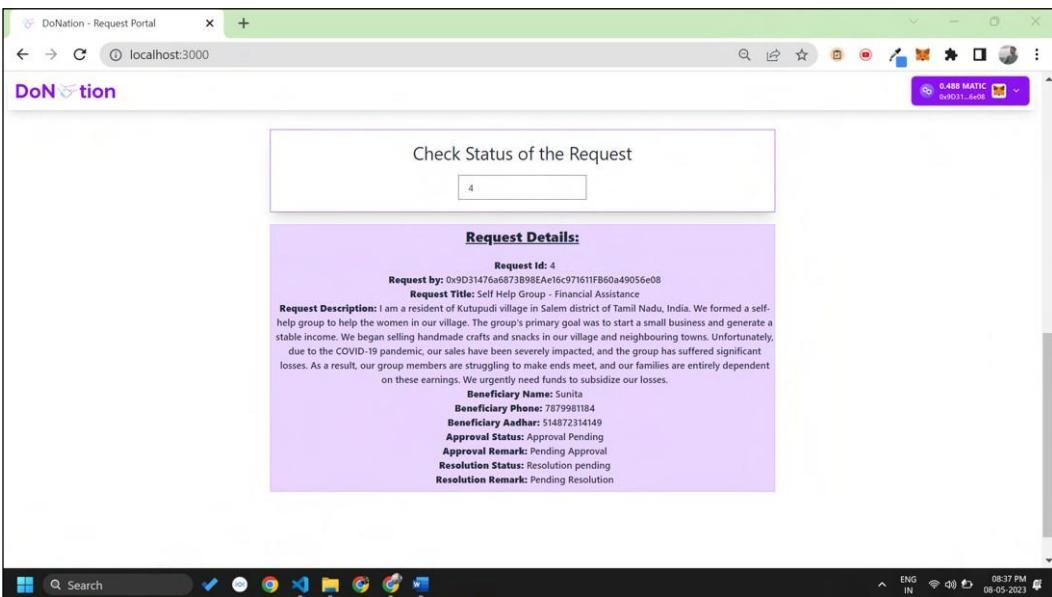


Fig 6.18 Checking status using RequestID - pending request

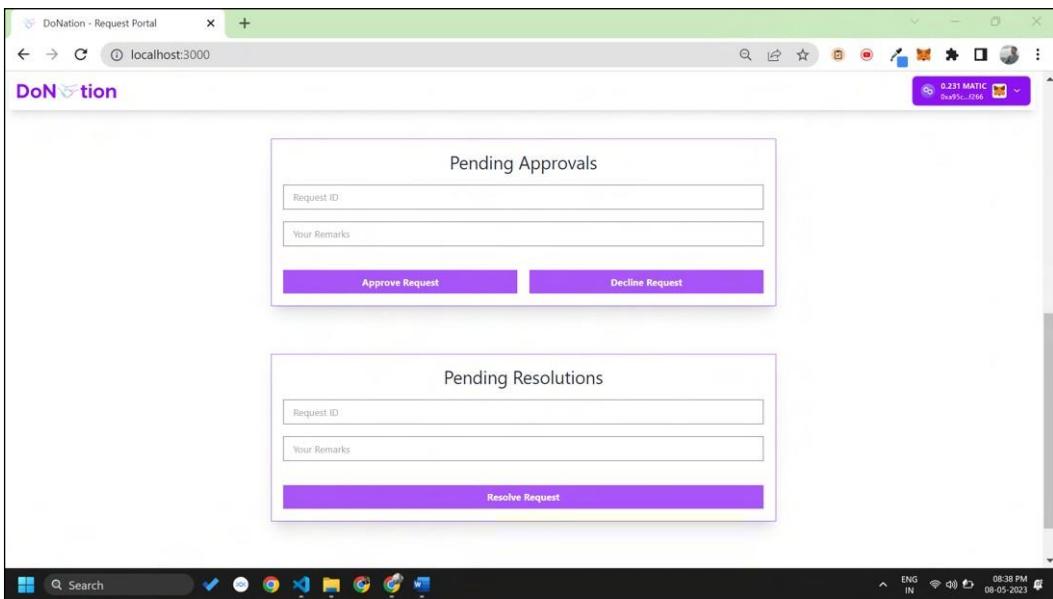


Fig 6.19 Admin View

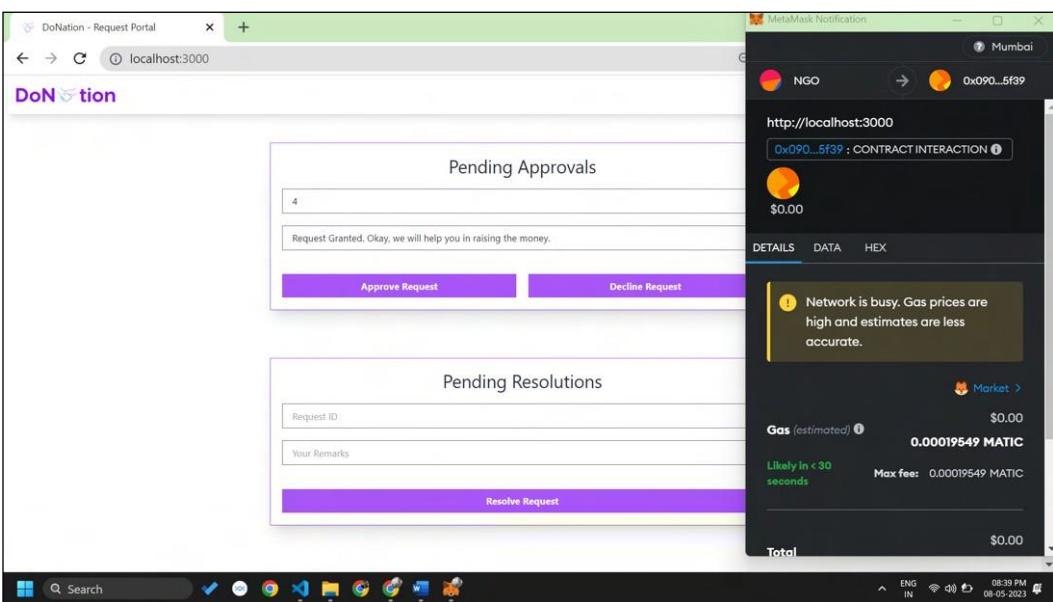


Fig 6.20 Admin View approving Request

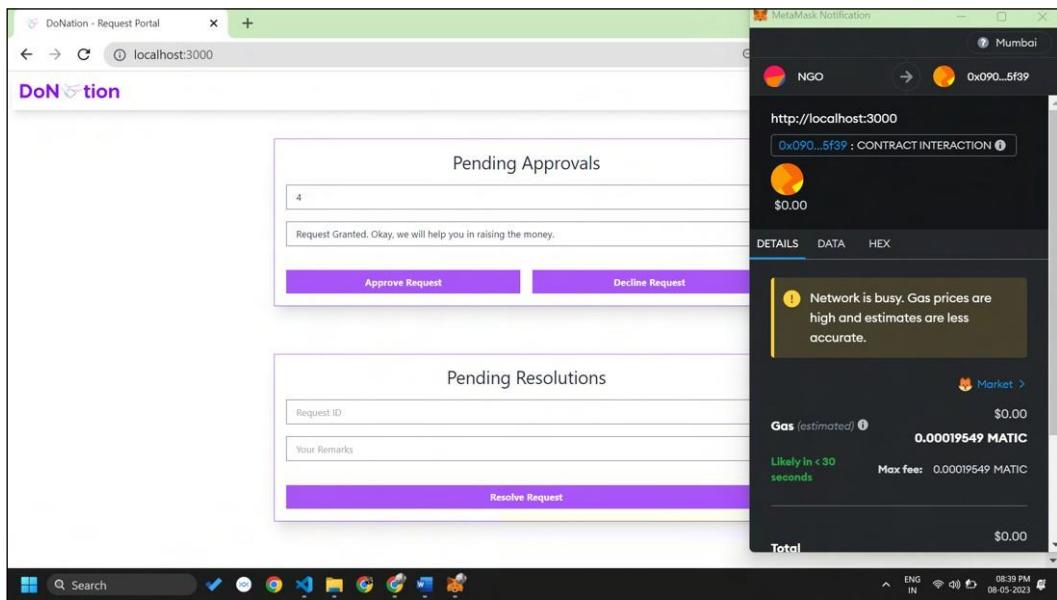


Fig 6.21 Admin View approving Request

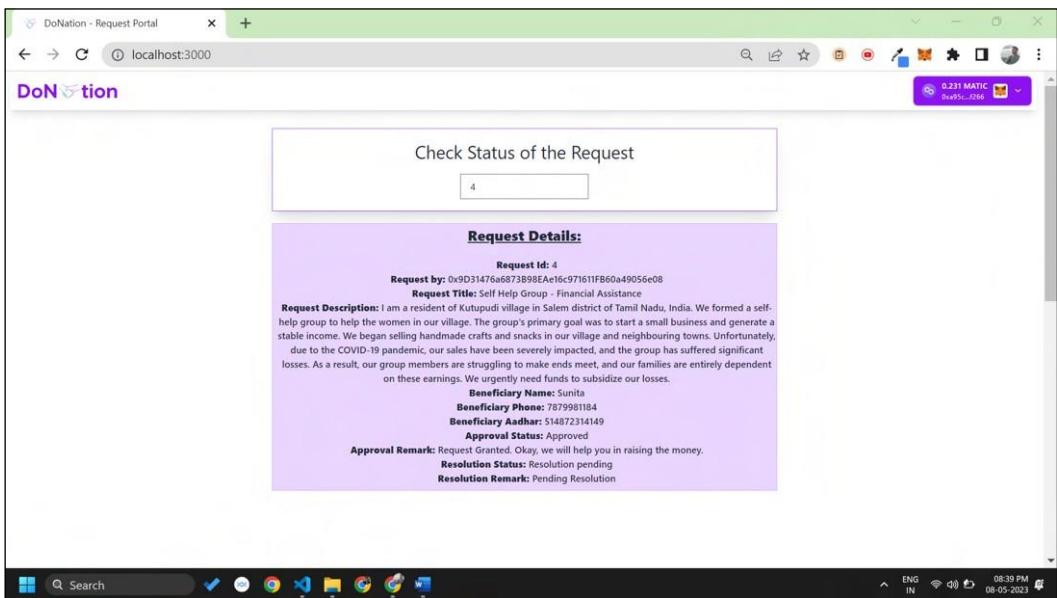


Fig 6.22 Request approved refelected on Beneficiary Side

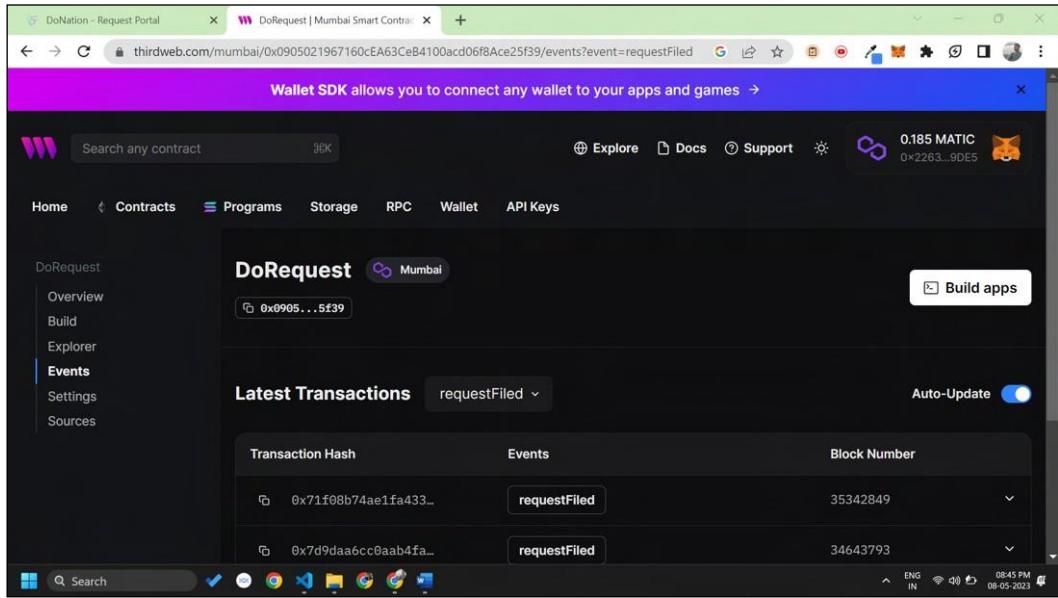


Fig 6.23 Smart Contract DoRequest uploaded on thirdweb

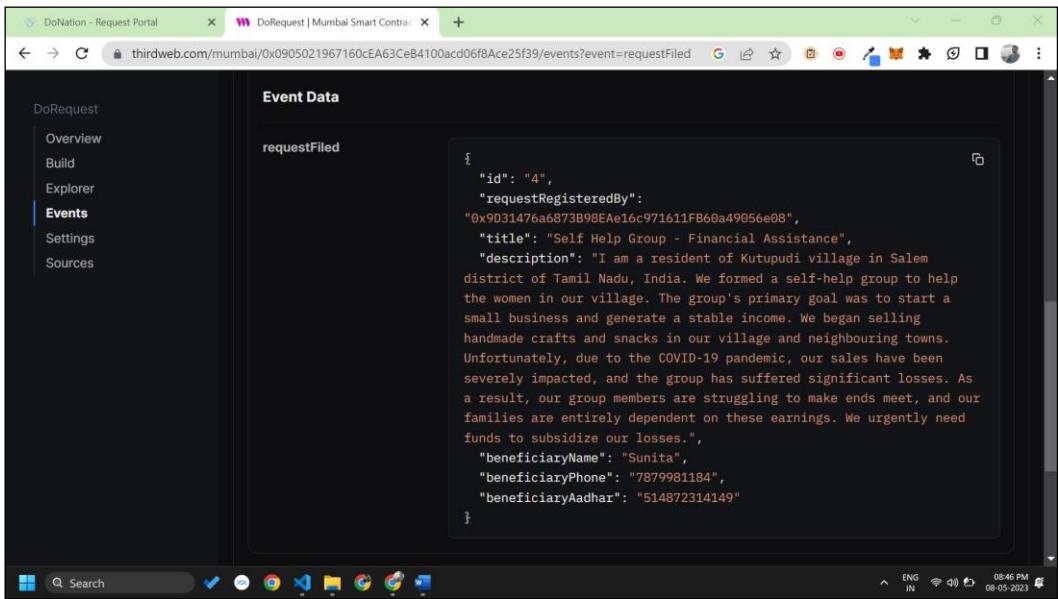


Fig 6.24 Block generated for each request being filed

CHAPTER 7 - CONCLUSION AND FUTURE WORKS

In conclusion, this project is an innovative solution that uses the power of BCT to connect donors with beneficiaries in need. The DoNation platform provides a secure and trustworthy mechanism for donors to contribute to the missions and the beneficiary request portal offers a straightforward and user-friendly interface for beneficiaries to raise requests, while the admin can review and approve or reject these requests. The resolution provided by the admin at the end ensures transparency and fairness in the process.

The platforms are built using React, Tailwind CSS, Hardhat, Thirdweb, and Ethereum blockchain, with two smart contracts in Solidity – one for the DoNation platform and another for the beneficiary request portal. This technology combination made transactions secure and quick while enabling a smooth user experience.

In the future, the platform could be further developed to include additional features and functionalities, like the ability to upload documents on beneficiary request portal, store them on IPFS, and use machine learning and image processing to verify them. Further we can integrate the DoNation platform with social media platforms to increase our reach.

We are utilizing the Ethereum Blockchain at the moment, but we intend to switch to Hyperledger Fabric 2.5, which will enable us to go at Enterprise Level. We can actually deploy this project to the real world by adopting DevOps techniques, which Hyperledger Fabric requires.

Overall, the fundraising platform with a beneficiary request portal is an initiative for promoting charitable giving and helping individuals in need. This blockchain-based social welfare initiatives has the potential to make a significant impact on communities around the world.

CHAPTER 8 - REFERENCES

1. Mandeep Kaur , Pankaj Deep Kaur, and Sandeep Kumar Sood (2022)
Blockchain Oriented Effective Charity Process During Pandemics and Emergencies
2. Hardhat Documentation
<https://hardhat.org/docs>
3. thirdweb Documentation
<https://blog.thirdweb.com/guides/>
<https://portal.thirdweb.com/getting-started/contracts>
4. Solidity Documentation
<https://docs.soliditylang.org/en/v0.8.19/>
5. Yarn Documentation
<https://classic.yarnpkg.com/lang/en/docs/>
6. React documentation
<https://react.dev/reference/react>
7. Tailwind CSS Documentation
<https://v2.tailwindcss.com/docs>
8. Ethers.js documentation
<https://docs.ethers.org/v5/>
9. B. Hu and H. Li, “Research on charity system based on blockchain,” IOP Conf. Mater. Sci. Eng., vol. 768, no. 7, Mar. 2020, Art. no. 072020, doi: 10.1088/1757-899X/768/7/072020.
10. H. Saleh, S. Avdoshin, and A. Dzhonov, “Platform for tracking donations of charitable foundations based on blockchain technology,” in Proc. Actual Problems Syst. Softw. Eng. (APSSE), 2019, pp. 182–187, doi: 10.1109/APSSE47353.2019.00031.

11. H. Wu and X. Zhu, “Developing a reliable service system of charity donation during the COVID-19 outbreak,” IEEE Access, vol. 8, pp. 154848–154860, 2020, doi: 10.1109/ACCESS.2020.3017654.
12. J. Lee, A. Seo, Y. Kim, and J. Jeong, “Blockchain-based one-off address system to guarantee transparency and privacy for a sustainable donation environment,” Sustainability, vol. 10, no. 12, p. 4422, 2018, doi: 10.3390/su10124422.
13. P. Agarwal, S. Jalan, and A. Mustafi, “Decentralized and financial approach to effective charity,” in Proc. Int. Conf. Soft-Comput. Netw. Secur. (ICSNS), 2018, pp. 1–3, doi: 10.1109/ICSNS.2018.8573644.
14. A. Zwitter and M. Boisse-Despiaux, “Blockchain for humanitarian action and development aid,” J. Int. Humanitarian Action, vol. 3, no. 1, p. 16, Dec. 2018, doi: 10.1186/s41018-018-0044-5.
15. A. Singh, R. Rajak, H. Mistry, and P. Raut, “Aid, charity and donation tracking system using blockchain,” in Proc. 4th Int. Conf. Trends Electron. Informat. (ICOEI), Jun. 2020, pp. 457–462, doi: 10.1109/ICOEI48184.2020.9143001.
16. M. S. Farooq, M. Khan, and A. Abid, “A framework to make charity collection transparent and auditable using blockchain technology,” Comput. Electr. Eng., vol. 83, May 2020, Art. no. 106588, doi: 10.1016/j.compeleceng.2020.106588.