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

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. Karthi

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. Anushiadevi 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

LIST OF FIGURES

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 uses 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.

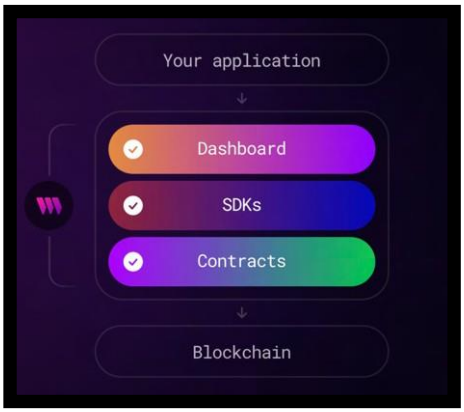


Fig 3.1 thirdweb architecture

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. ETHERNAL

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

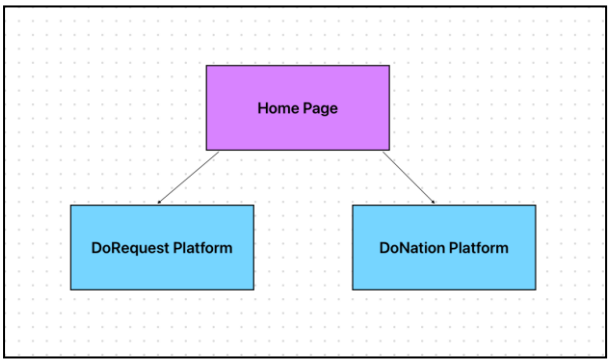


Fig 4.1 Flow of Home Page

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

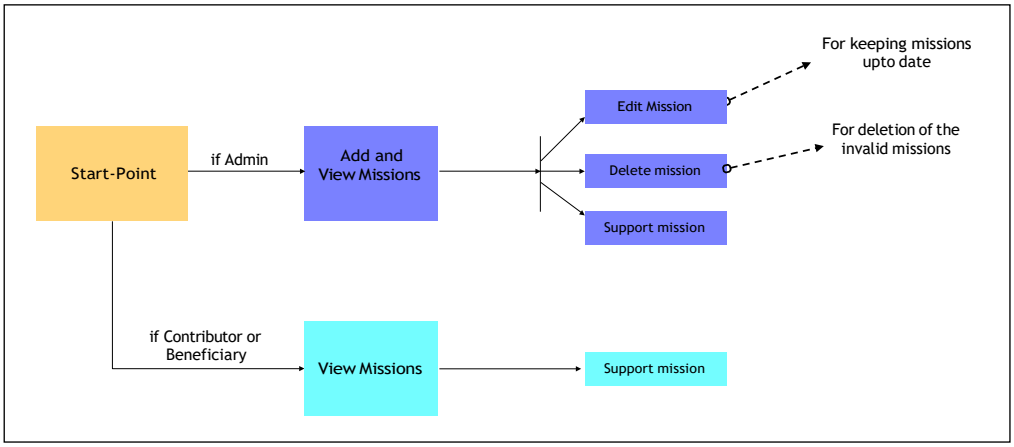


Fig 4.2 Flow of DoNation Platform usage by various actors

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.

Fig 4.3 The Admin's MetaMask account public address

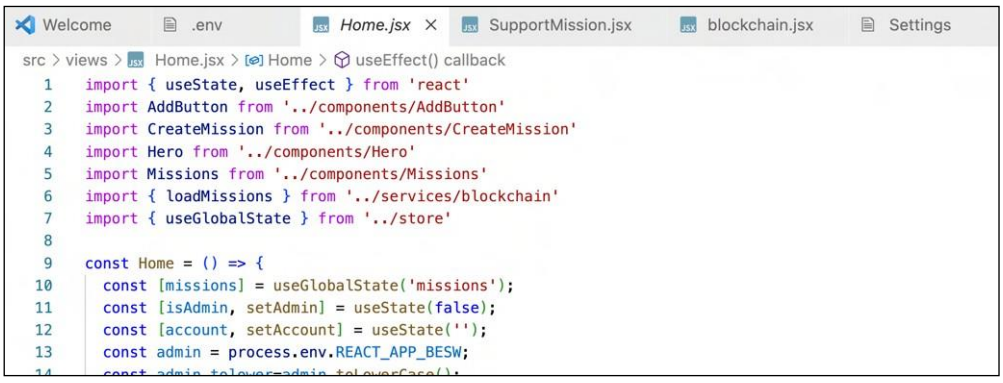


Fig 4.4 The use of the .env variable to allow only the admin to access the functionalities

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

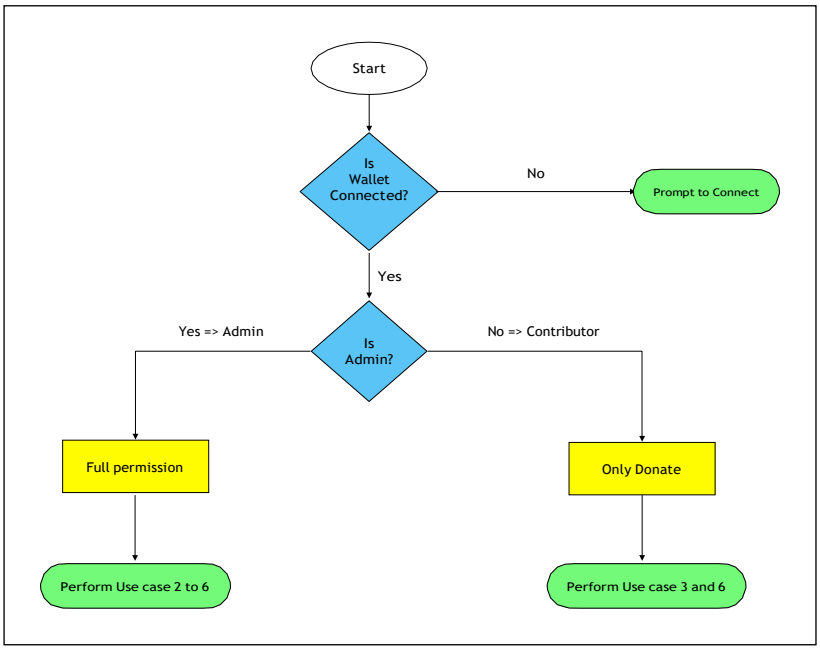


Fig 4.5 Flow of the DoNation Platform

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

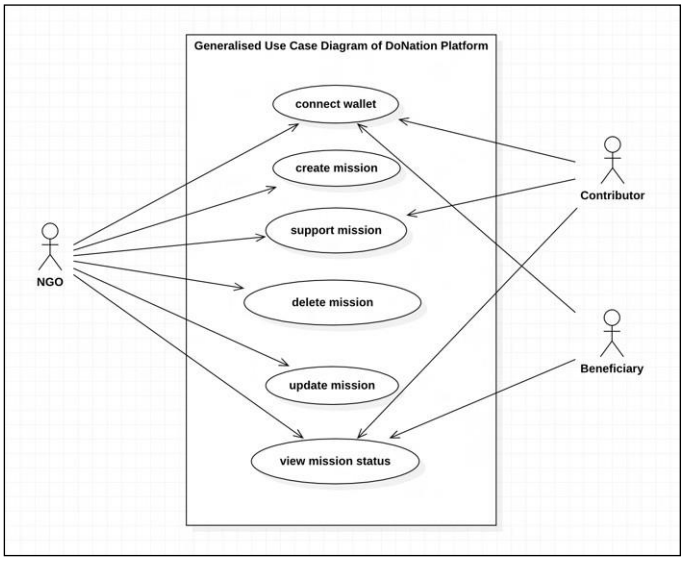


Fig 4.6 Use Case Diagram for DoNation Platform

Sequence Diagram

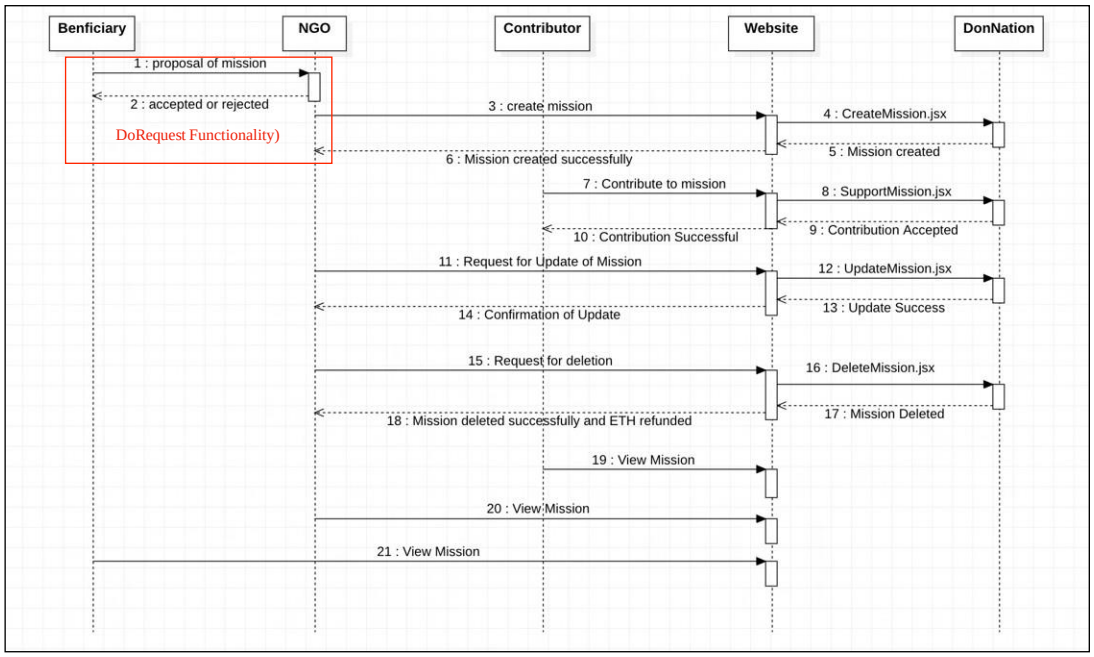


Fig 4.7 Seqeunce Diagram for the DoNation Platform

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

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.

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.

Fig 4.10 Block #1

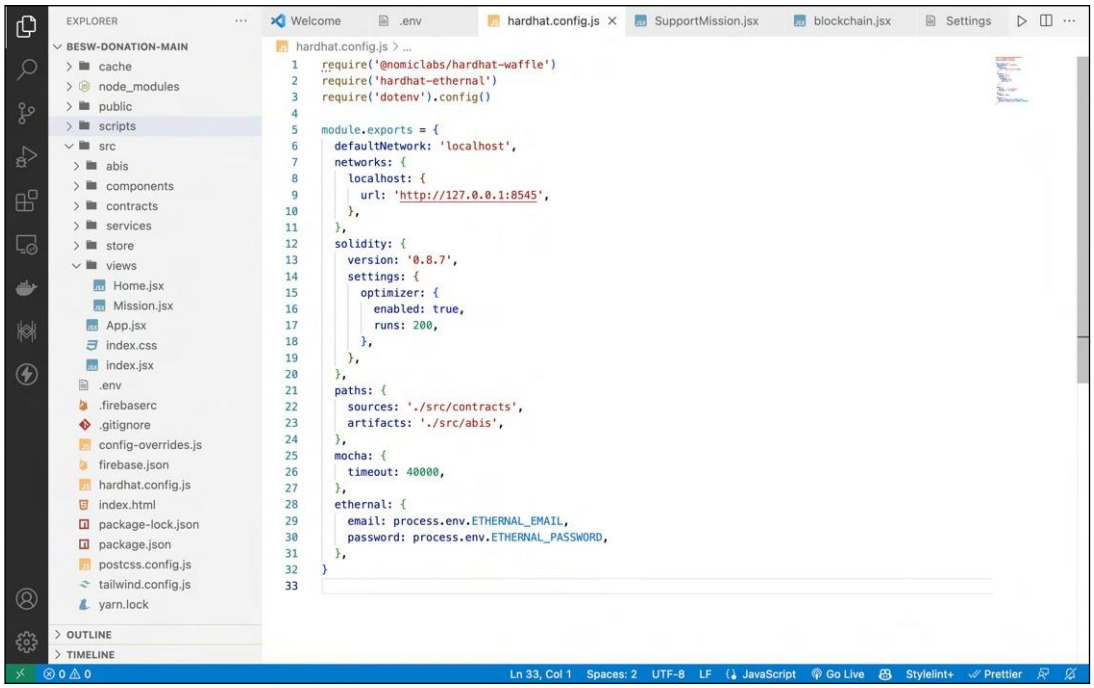


Fig 4.11 hardhat.config,js

DOREQUEST PLATFORM

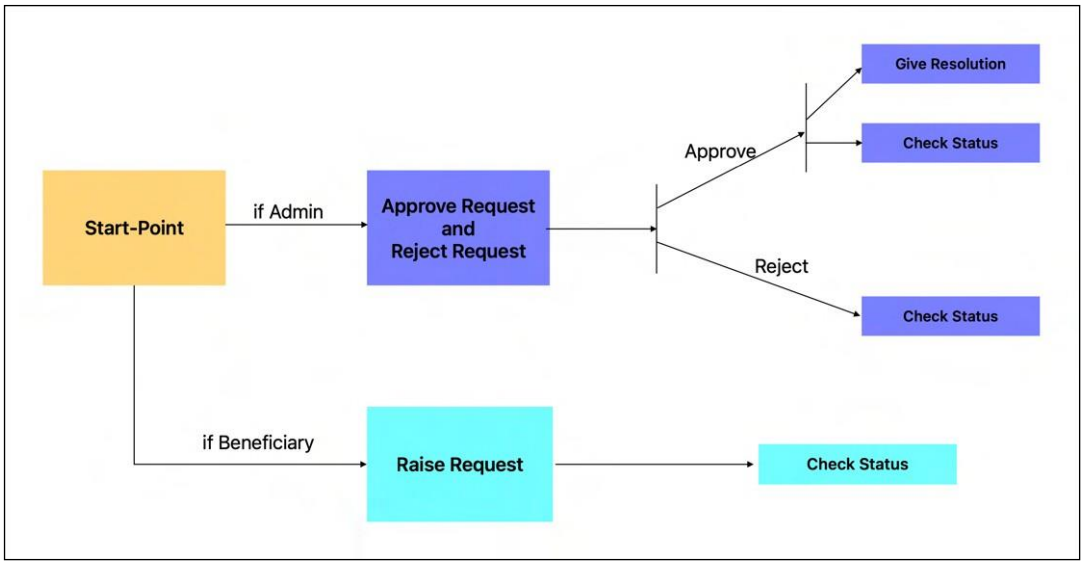


Fig 4.12 Flow of DoRequest Platform usage by various actors

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.

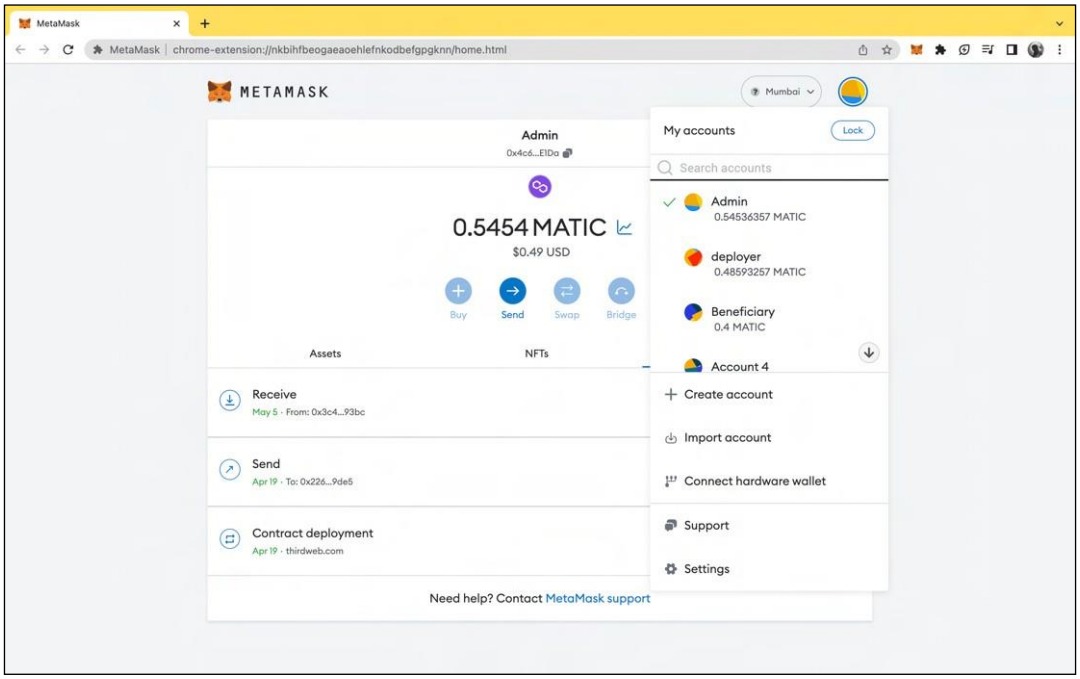


Fig 4.13 Different metamask wallet for different actors

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.

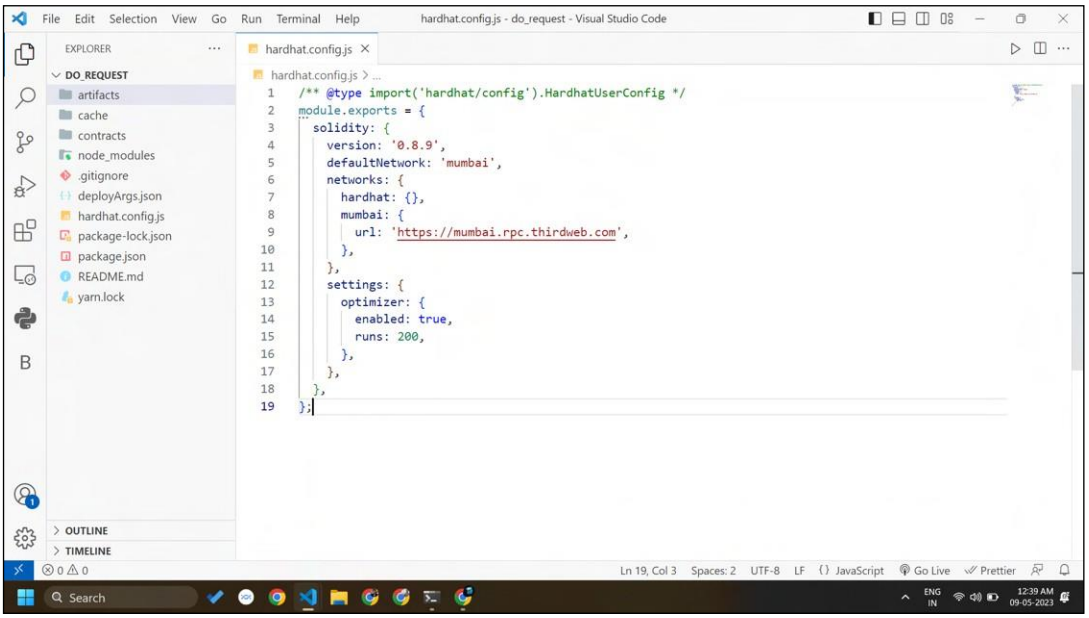


Fig 4.14 hardhat.config.js

Deployment of SC

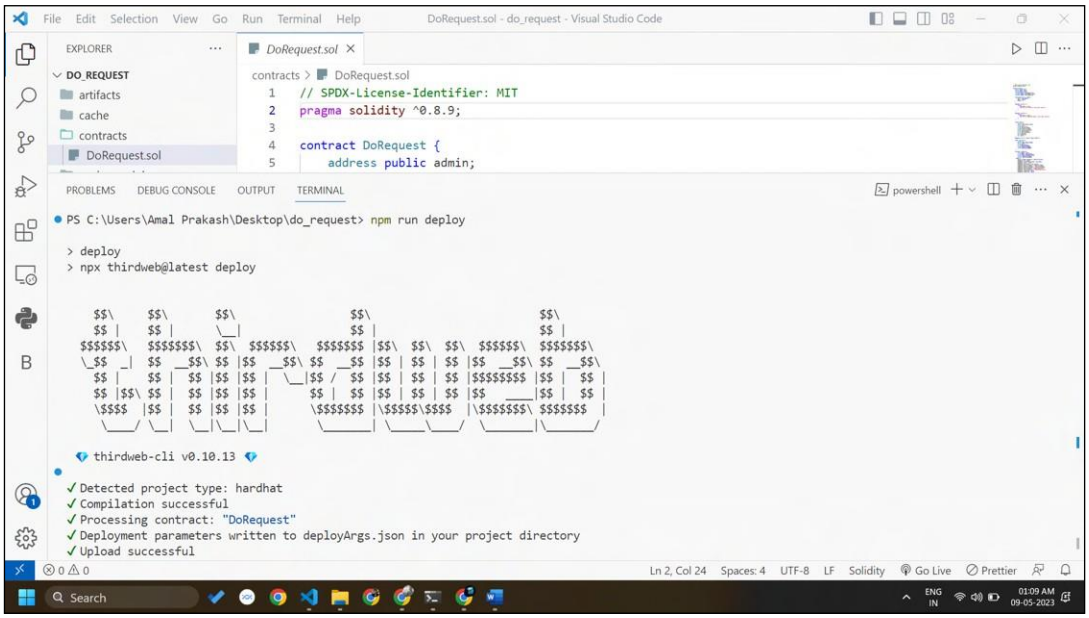


Fig 4.15 `npm run deploy` – to deploy the smart contract.

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

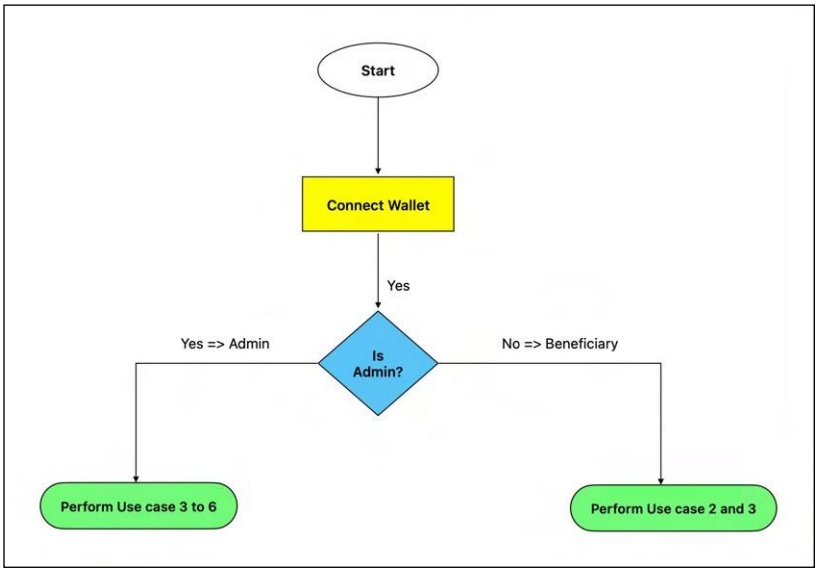


Fig 4,18 Flow of DoRequest

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

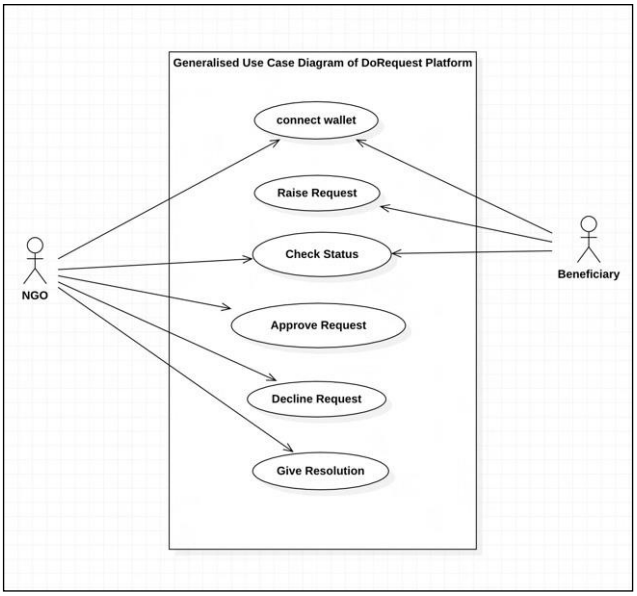


Fig 4.19 Use Case Diagram for DoRequest Platform

Sequence Diagram

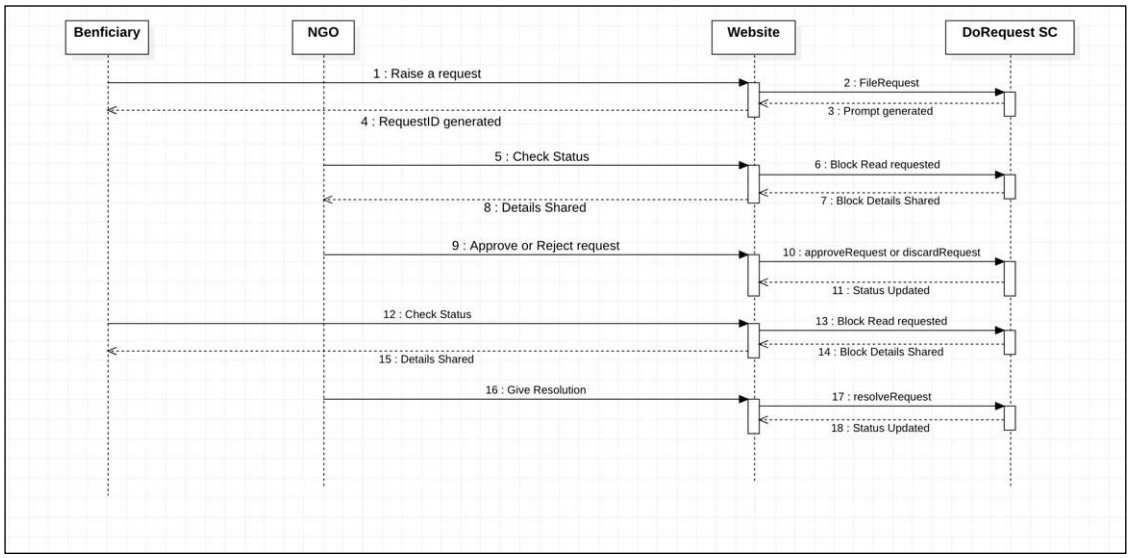


Fig 4.20 Sequence Diagram for DoRequest Platform

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.

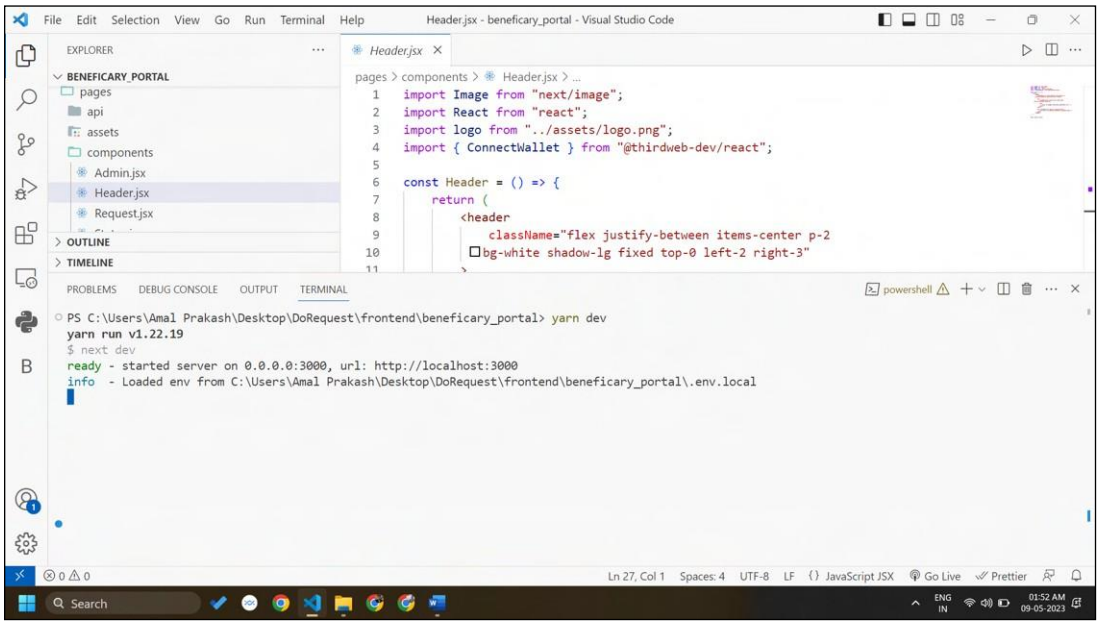


Fig 4.21 `yarn dev` – to run the application

CHAPTER 5 - SOURCE CODE

ALGORITHMS IN THE BASE PAPER

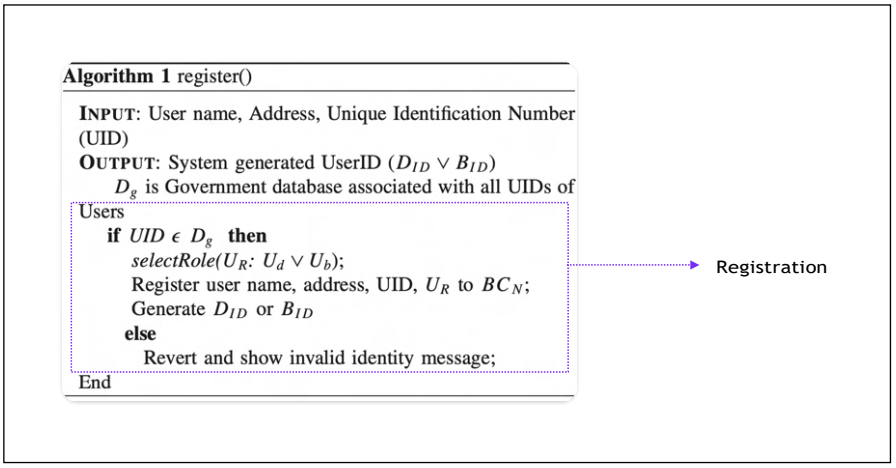


Fig 5.1 register()

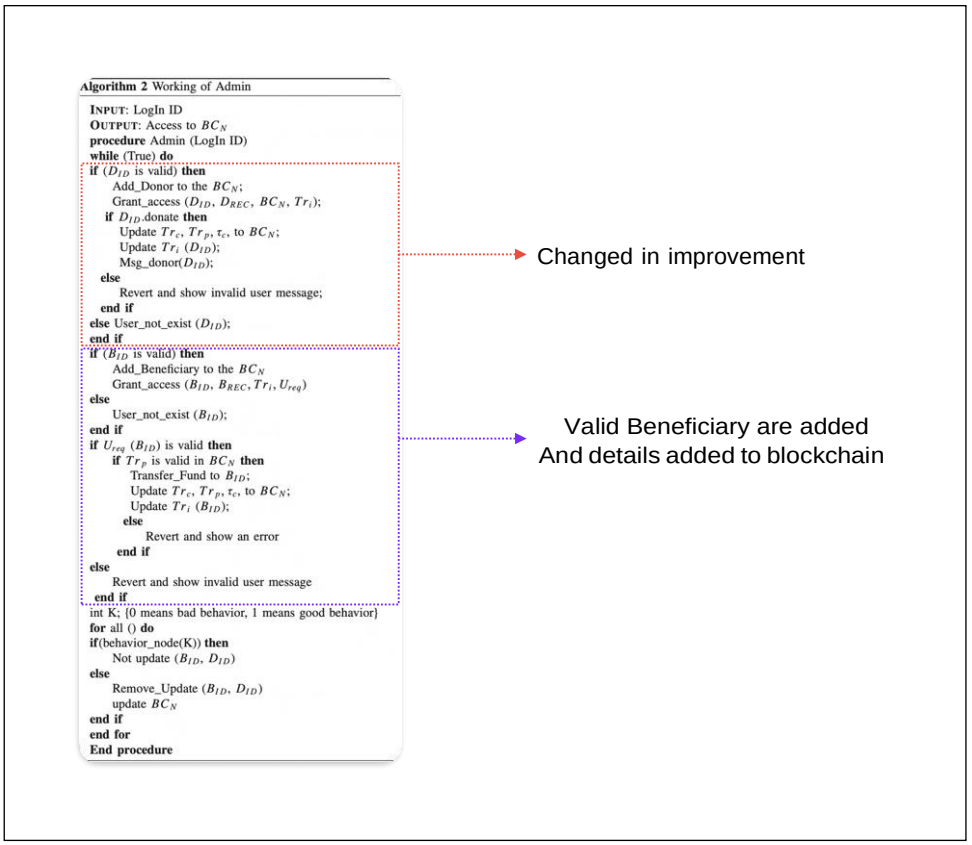


FIg 5.2 Working of admin()

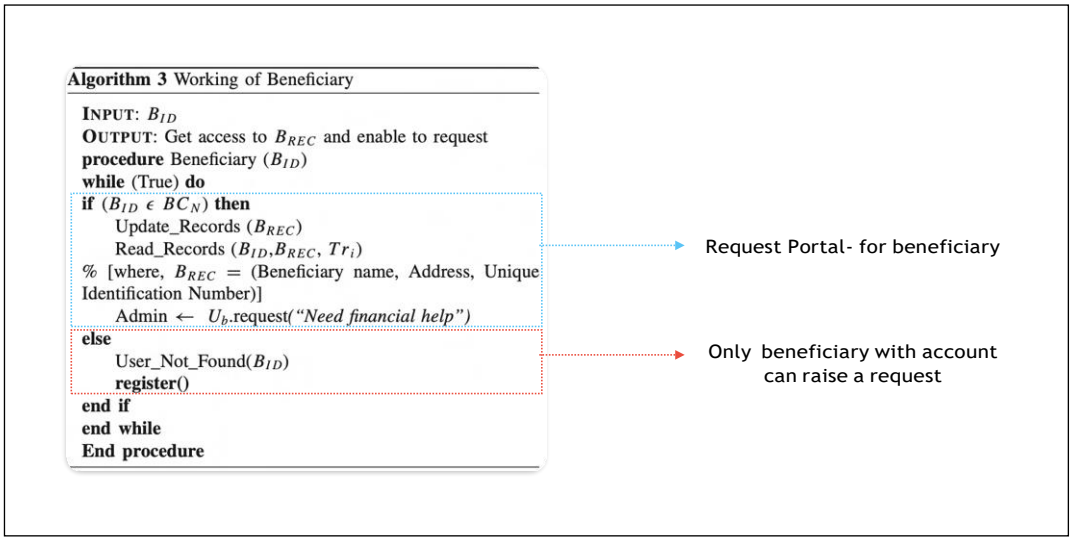


Fig 5.3 Working of Beneficiary()

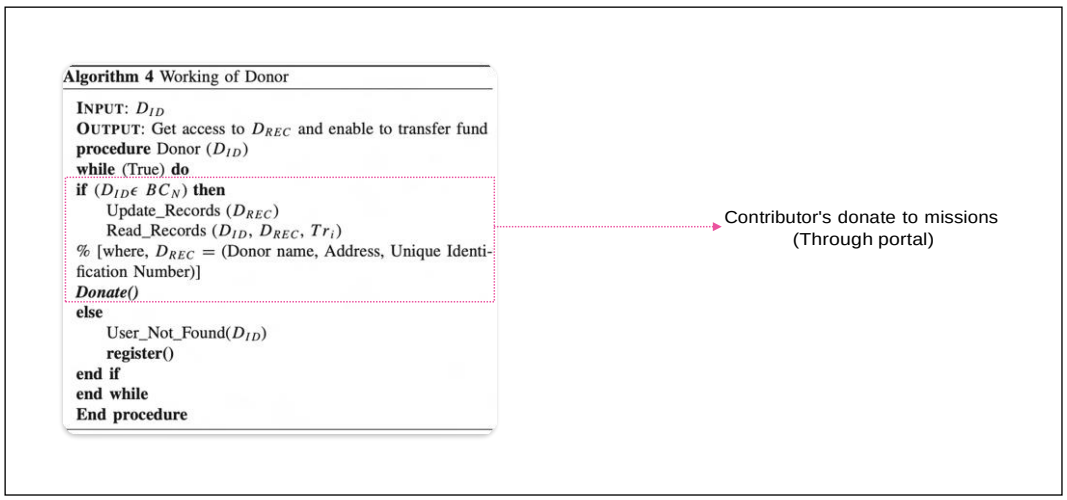


Fig 5.4 Working of Donor()

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:-

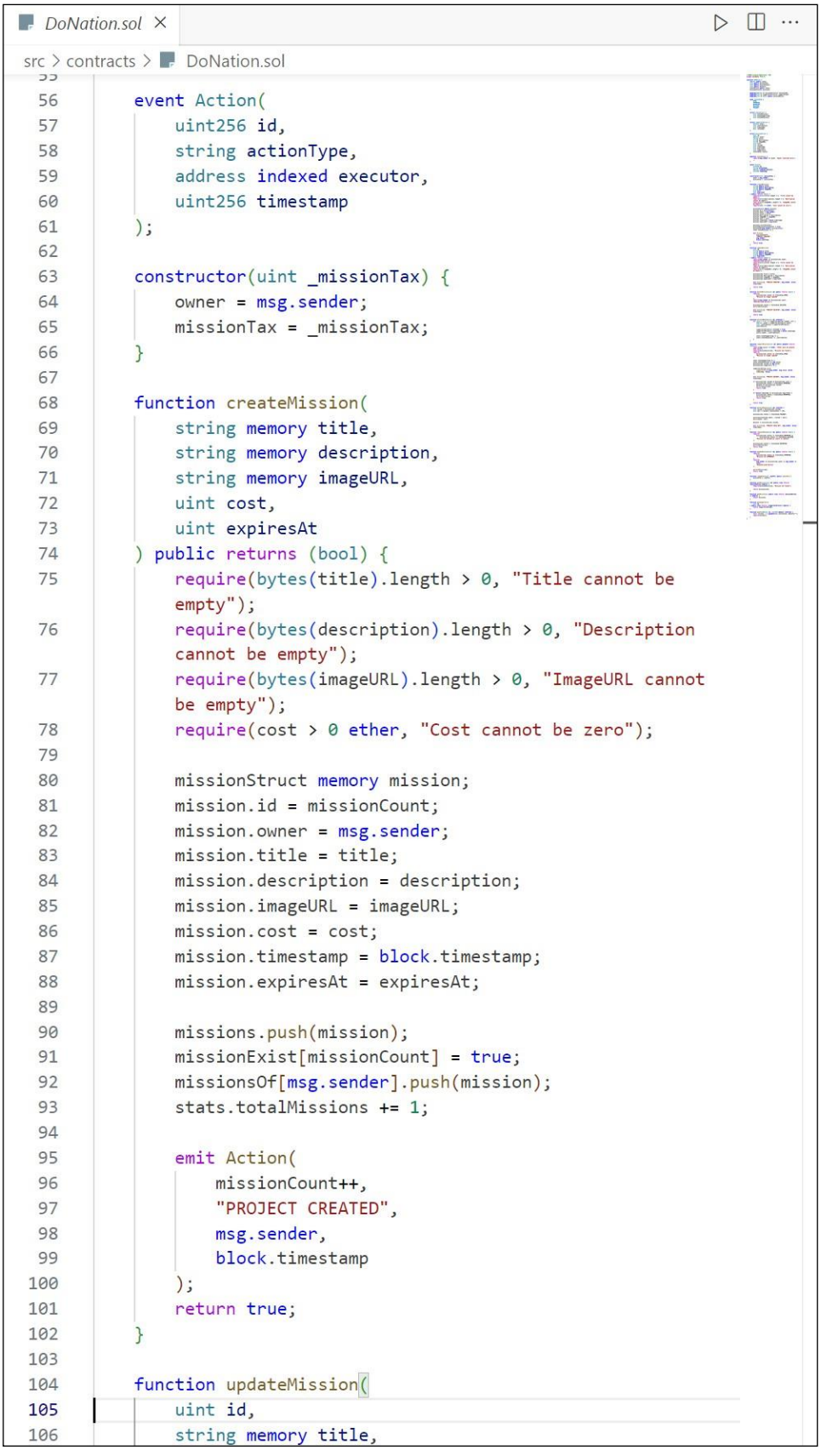
1. `fileRequest` – only for the beneficiary, it allows a user to raise a new request and emit an event indicating that a new request has been filed. It also generates a unique request id.
2. `approveRequest` – only for admin, to approve the pending request
3. `discardRequest` - only for admin, to decline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

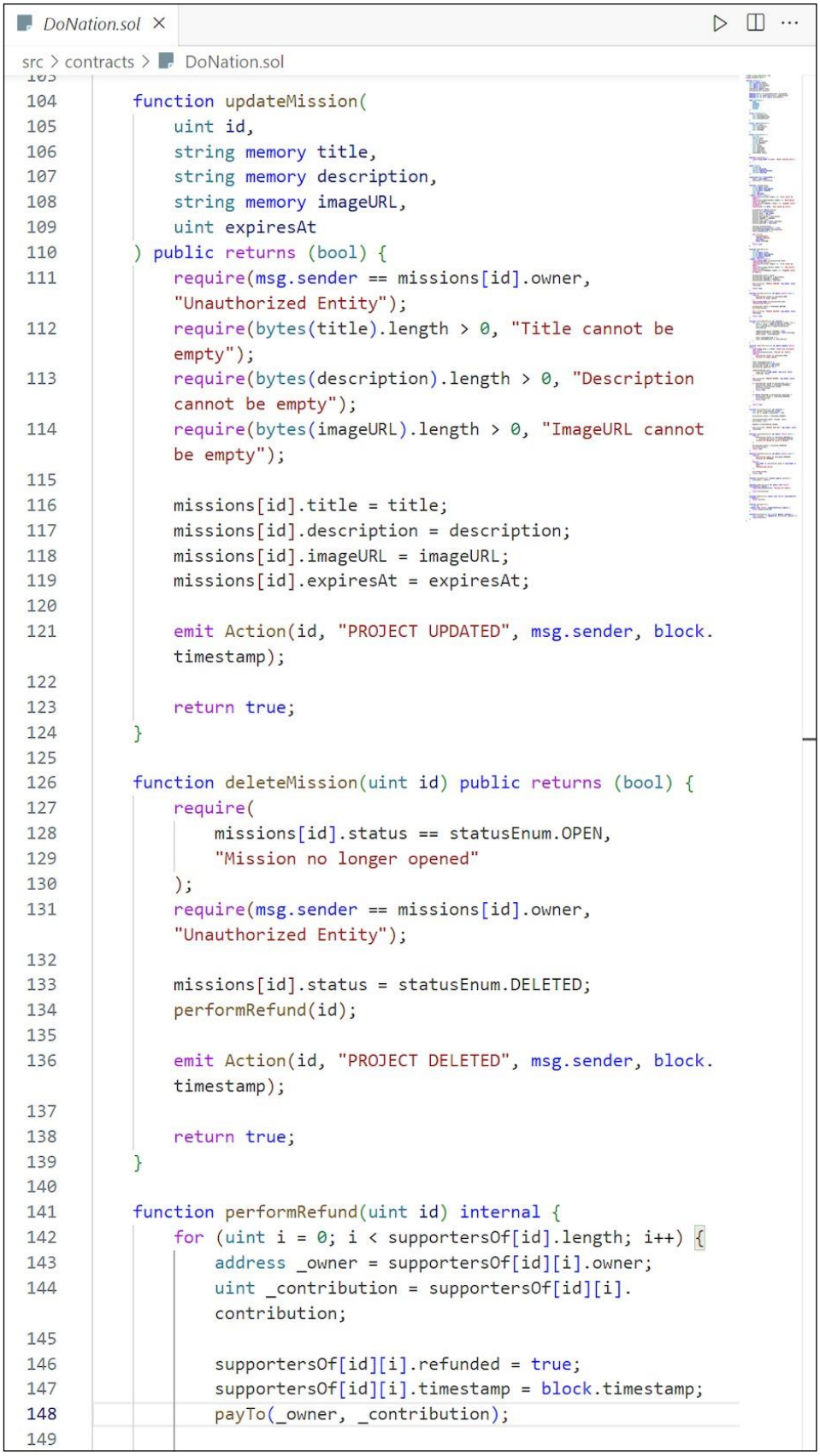
CODE

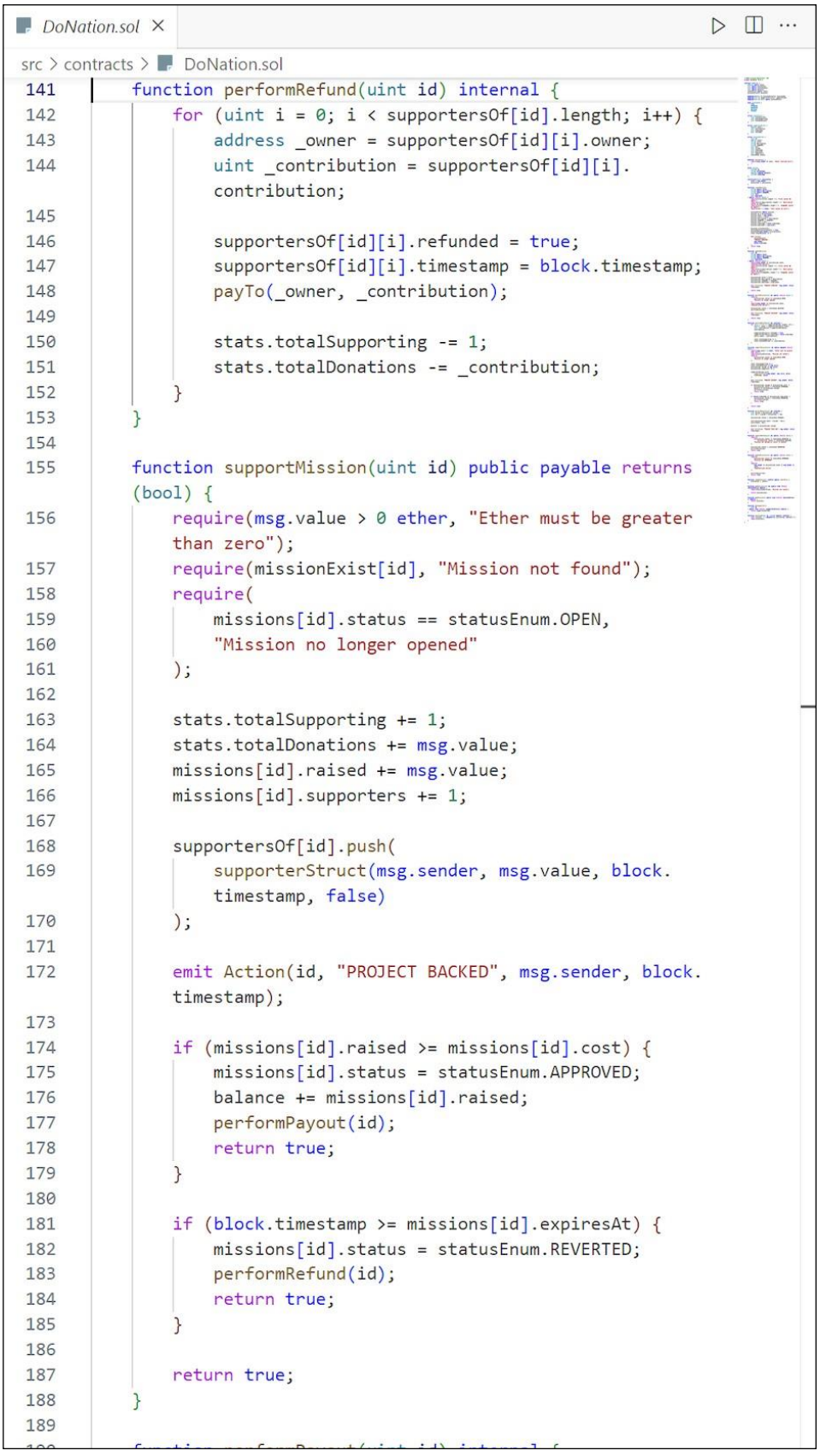
DoNation.sol

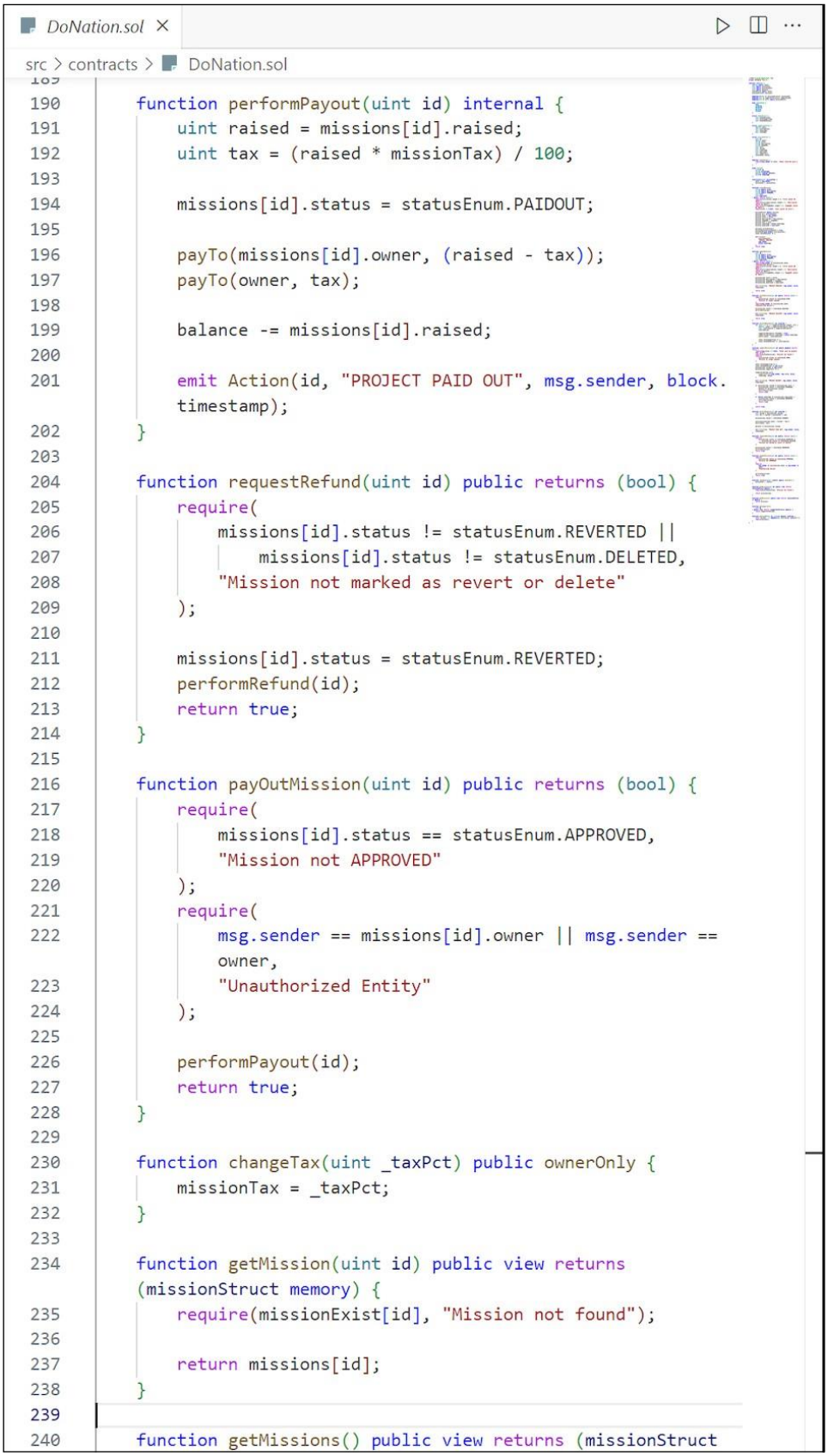
src > contracts > DoNation.sol

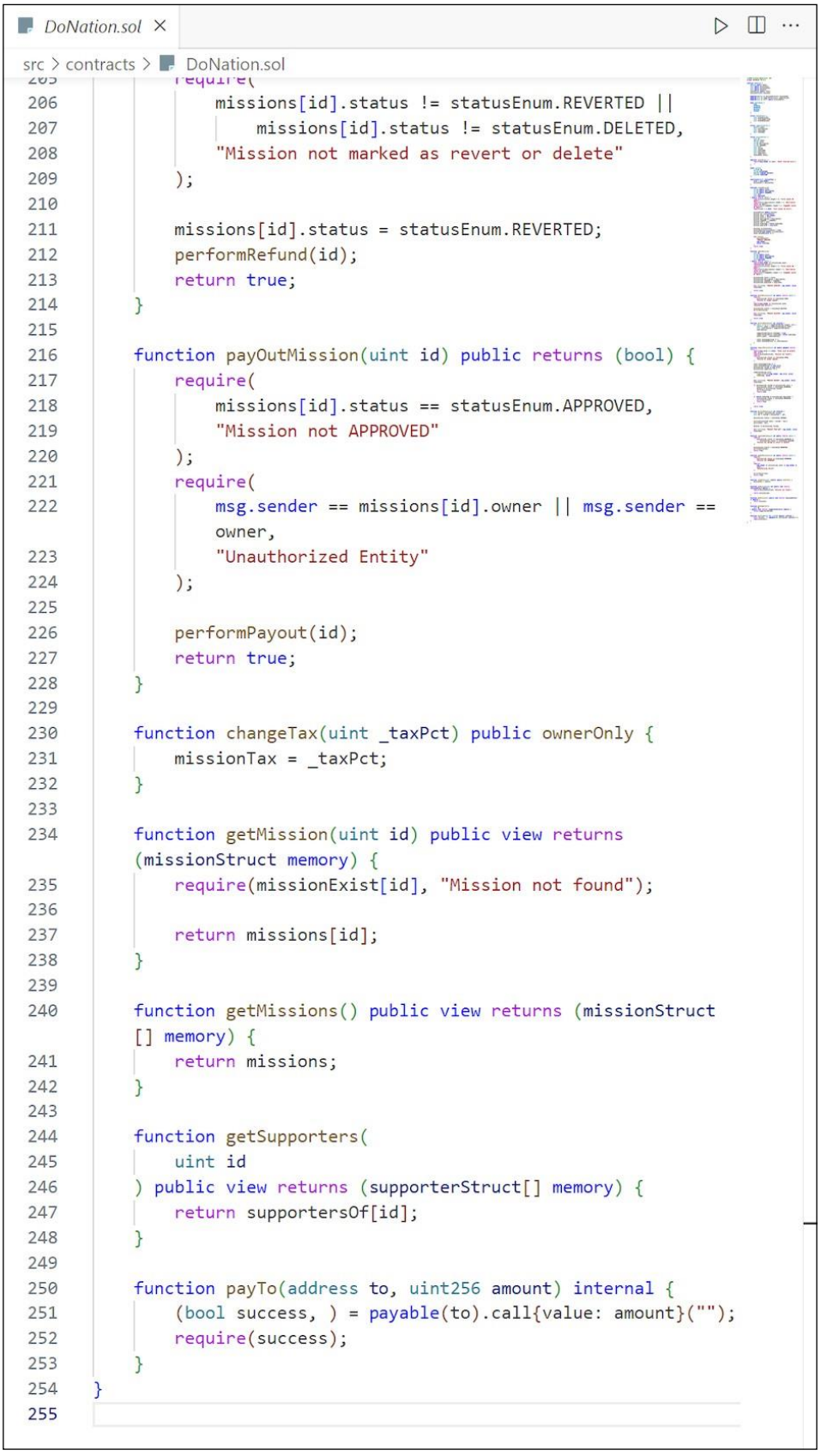
```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

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


DoNation.sol
src > contracts > DoNation.sol
141 function performRefund(uint id) internal {
142 for (uint i = 0; i < supportersOf[id].length; i++) {
143 address _owner = supportersOf[id][i].owner;
144 uint _contribution = supportersOf[id][i].
contribution;
145
146 supportersOf[id][i].refunded = true;
147 supportersOf[id][i].timestamp = block.timestamp;
148 payTo(_owner, _contribution);
149
150 stats.totalSupporting -= 1;
151 stats.totalDonations -= _contribution;
152 }
153 }
154
155 function supportMission(uint id) public payable returns
(bool) {
156 require(msg.value > 0 ether, "Ether must be greater
than zero");
157 require(missionExist[id], "Mission not found");
158 require(
159 missions[id].status == statusEnum.OPEN,
160 "Mission no longer opened"
161);
162
163 stats.totalSupporting += 1;
164 stats.totalDonations += msg.value;
165 missions[id].raised += msg.value;
166 missions[id].supporters += 1;
167
168 supportersOf[id].push(
169 supporterStruct(msg.sender, msg.value, block.
timestamp, false)
170);
171
172 emit Action(id, "PROJECT BACKED", msg.sender, block.
timestamp);
173
174 if (missions[id].raised >= missions[id].cost) {
175 missions[id].status = statusEnum.APPROVED;
176 balance += missions[id].raised;
177 performPayout(id);
178 return true;
179 }
180
181 if (block.timestamp >= missions[id].expiresAt) {
182 missions[id].status = statusEnum.REVERTED;
183 performRefund(id);
184 return true;
185 }
186
187 return true;
188 }
189


DoNation.sol
src > contracts > DoNation.sol
function performPayout(uint id) internal {
uint raised = missions[id].raised;
uint tax = (raised * missionTax) / 100;
missions[id].status = statusEnum.PAIDOUT;
payTo(missions[id].owner, (raised - tax));
payTo(owner, tax);
balance -= missions[id].raised;
emit Action(id, "PROJECT PAID OUT", msg.sender, block.
timestamp);
}
function requestRefund(uint id) public returns (bool) {
require(
missions[id].status != statusEnum.REVERTED ||
missions[id].status != statusEnum.DELETED,
"Mission not marked as revert or delete"
);
missions[id].status = statusEnum.REVERTED;
performRefund(id);
return true;
}
function payOutMission(uint id) public returns (bool) {
require(
missions[id].status == statusEnum.APPROVED,
"Mission not APPROVED"
);
require(
msg.sender == missions[id].owner || msg.sender ==
owner,
"Unauthorized Entity"
);
performPayout(id);
return true;
}
function changeTax(uint _taxPct) public ownerOnly {
missionTax = _taxPct;
}
function getMission(uint id) public view returns
(missionStruct memory) {
require(missionExist[id], "Mission not found");
return missions[id];
}
function getMissions() public view returns (missionStruct

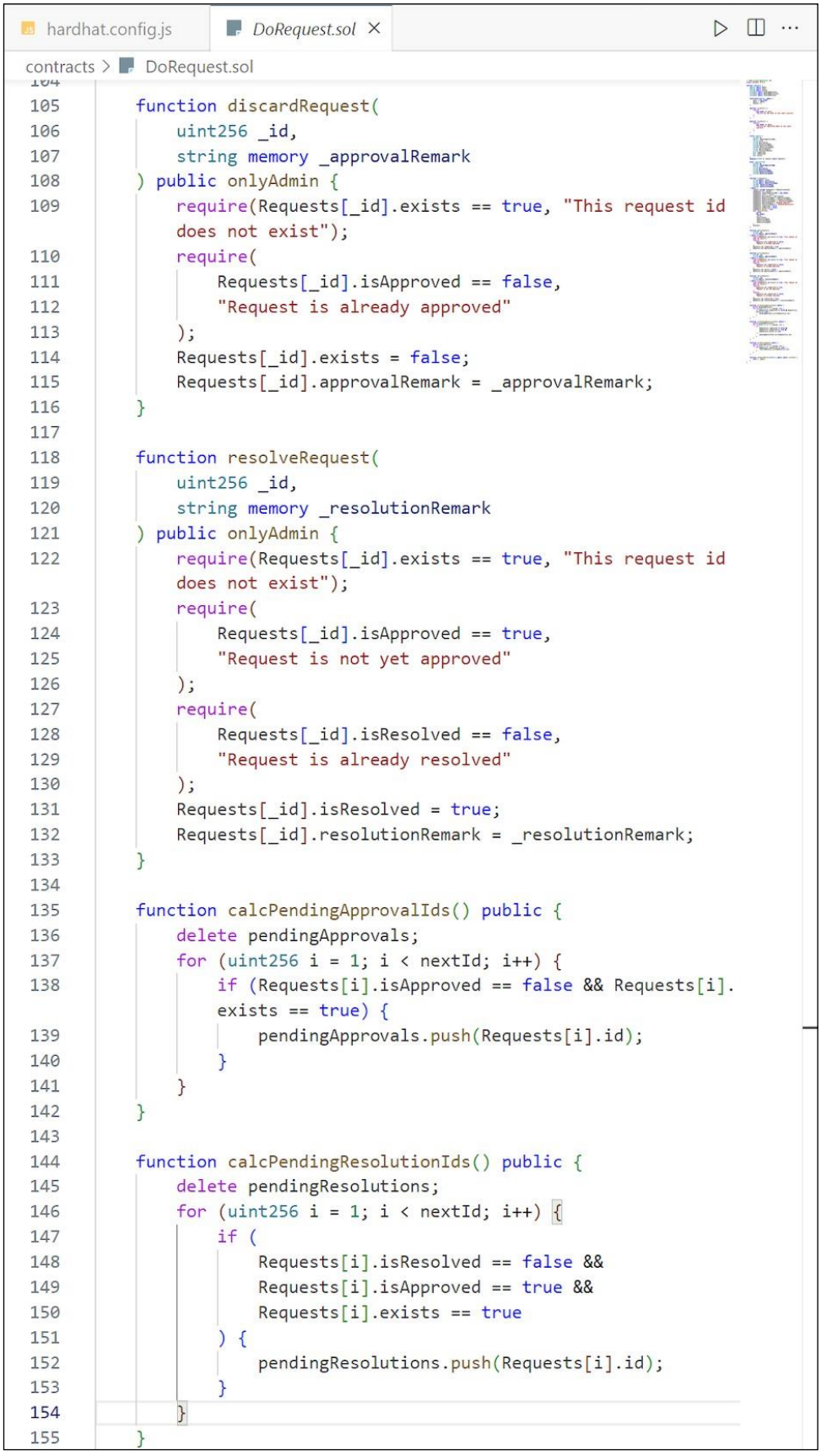

DoNation.sol
src > contracts > DoNation.sol
206 missions[id].status != statusEnum.REVERTED ||
207 missions[id].status != statusEnum.DELETED,
208 "Mission not marked as revert or delete"
209);
210
211 missions[id].status = statusEnum.REVERTED;
212 performRefund(id);
213 return true;
214 }
215
216 function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

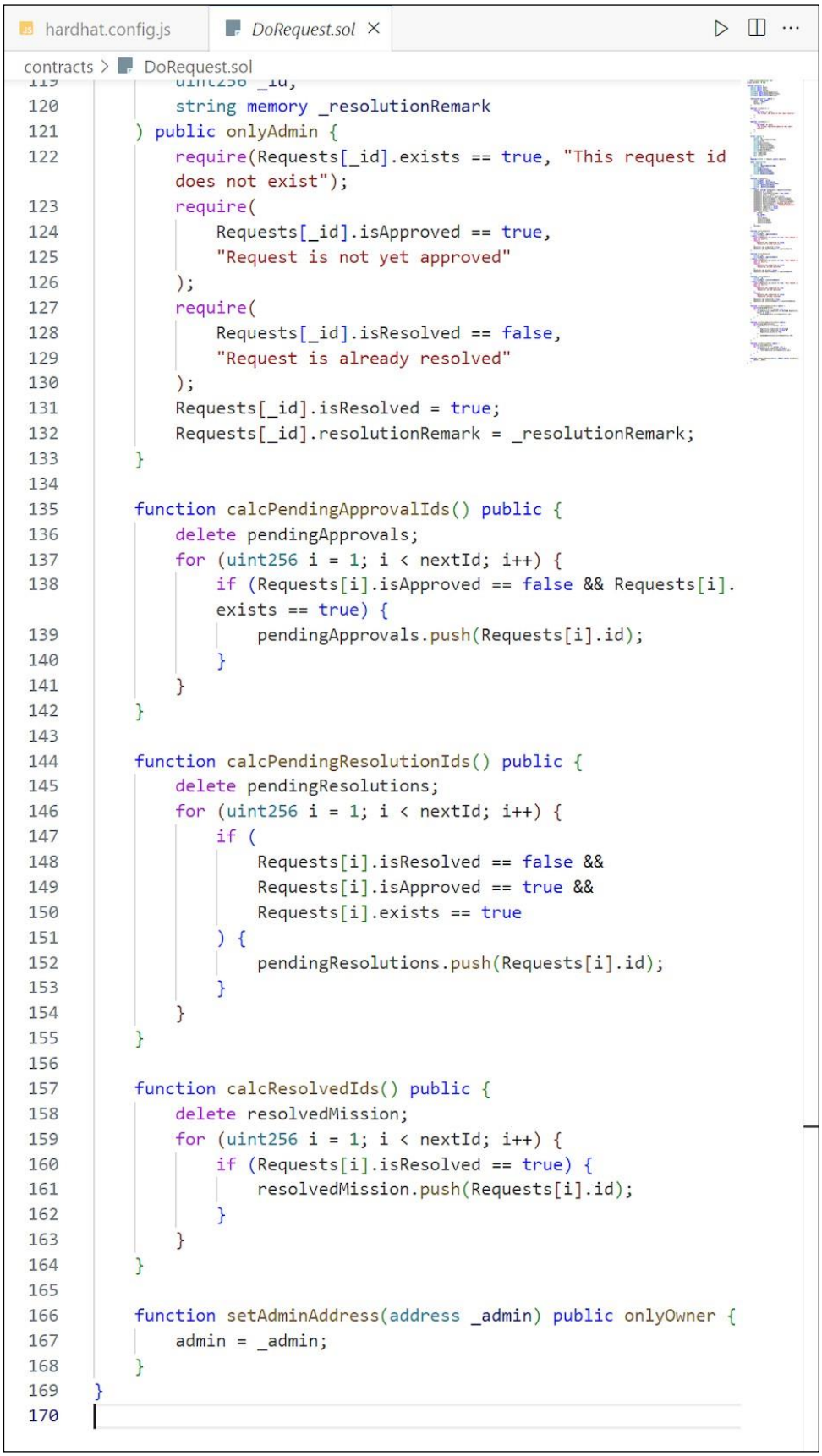
hardhat.config.js | DoRequest.sol

contracts > DoRequest.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

contract DoRequest {
    address public admin;
    address public owner;
    uint256 public nextId;
    uint256[] public pendingApprovals;
    uint256[] public pendingResolutions;
    uint256[] public resolvedMission;

    constructor(address _admin) {
        owner = msg.sender;
        admin = _admin;
        nextId = 1;
    }

    modifier onlyOwner() {
        require(
            msg.sender == owner,
            "You are not the owner of this smart contract"
        );
        _;
    }

    modifier onlyAdmin() {
        require(
            msg.sender == admin,
            "You are not registered admin of this smart
            contract"
        );
        _;
    }

    struct request {
        uint256 id;
        address requestRegisteredBy;
        string title;
        string description;
        string beneficiaryName;
        uint256 beneficiaryPhone;
        uint256 beneficiaryAadhar;
        string approvalRemark;
        string resolutionRemark;
        bool isApproved;
        bool isResolved;
        bool exists;
    }
    mapping(uint256 => request) public Requests;

    event requestFiled(
        uint256 id,
        address requestRegisteredBy,
        string title,
```


hardhat.config.js
DoRequest.sol
contracts > DoRequest.sol
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
does not exist");
97 require(
98 Requests[_id].isApproved == false,
99 "Request is already approved"
100);
101 Requests[_id].isApproved = true;
102 Requests[_id].approvalRemark = _approvalRemark;
103 }
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 does not exist");
110 require(
111 Requests[_id].isApproved == false,
112 "Request is already approved"
113);
114 Requests[_id].exists = false;
115 Requests[_id].approvalRemark = _approvalRemark;
116 }
117
118 function resolveRequest(
119 uint256 _id,
120 string memory _resolutionRemark
121) public onlyAdmin {
122 require(Requests[_id].exists == true, "This request id does not exist");
123 require(
124 Requests[_id].isApproved == true,
125 "Request is not yet approved"
126);
127 require(
128 Requests[_id].isResolved == false,
129 "Request is already resolved"
130);
131 Requests[_id].isResolved = true;
132 Requests[_id].resolutionRemark = _resolutionRemark;
133 }
134
135 function calcPendingApprovalIds() public {
136 delete pendingApprovals;
137 for (uint256 i = 1; i < nextId; i++) {
138 if (Requests[i].isApproved == false && Requests[i].exists == true) {
139 pendingApprovals.push(Requests[i].id);
140 }
141 }
142 }
143
144 function calcPendingResolutionIds() public {
145 delete pendingResolutions;
146 for (uint256 i = 1; i < nextId; i++) {
147 if (
148 Requests[i].isResolved == false &&
149 Requests[i].isApproved == true &&
150 Requests[i].exists == true
151) {
152 pendingResolutions.push(Requests[i].id);
153 }
154 }
155 }


hardhat.config.js
DoRequest.sol
contracts > DoRequest.sol
uint256 _id,
string memory _resolutionRemark
) public onlyAdmin {
require(Requests[_id].exists == true, "This request id does not exist");
require(
Requests[_id].isApproved == true,
"Request is not yet approved"
);
require(
Requests[_id].isResolved == false,
"Request is already resolved"
);
Requests[_id].isResolved = true;
Requests[_id].resolutionRemark = _resolutionRemark;
}
function calcPendingApprovalIds() public {
delete pendingApprovals;
for (uint256 i = 1; i < nextId; i++) {
if (Requests[i].isApproved == false && Requests[i].exists == true) {
pendingApprovals.push(Requests[i].id);
}
}
}
function calcPendingResolutionIds() public {
delete pendingResolutions;
for (uint256 i = 1; i < nextId; i++) {
if (
Requests[i].isResolved == false &&
Requests[i].isApproved == true &&
Requests[i].exists == true
) {
pendingResolutions.push(Requests[i].id);
}
}
}
function calcResolvedIds() public {
delete resolvedMission;
for (uint256 i = 1; i < nextId; i++) {
if (Requests[i].isResolved == true) {
resolvedMission.push(Requests[i].id);
}
}
}
function setAdminAddress(address _admin) public onlyOwner {
admin = _admin;
}
}

CHAPTER 6 - SNAPSHOTS IN DIFFERENT SCENARIOS

HOME PAGE

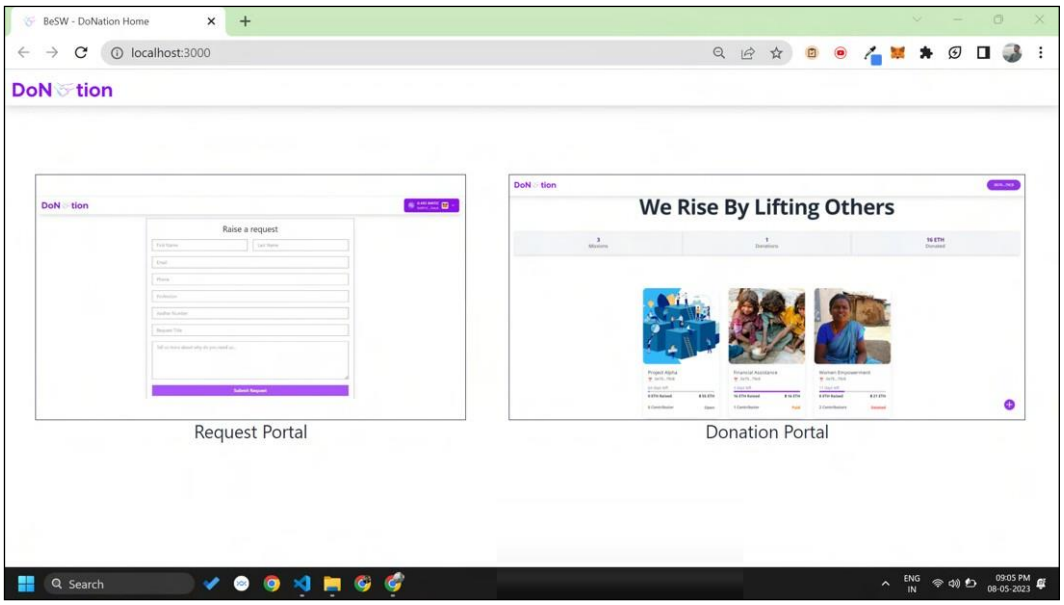


Fig 6.1 Home Page

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

DONATION WORKING

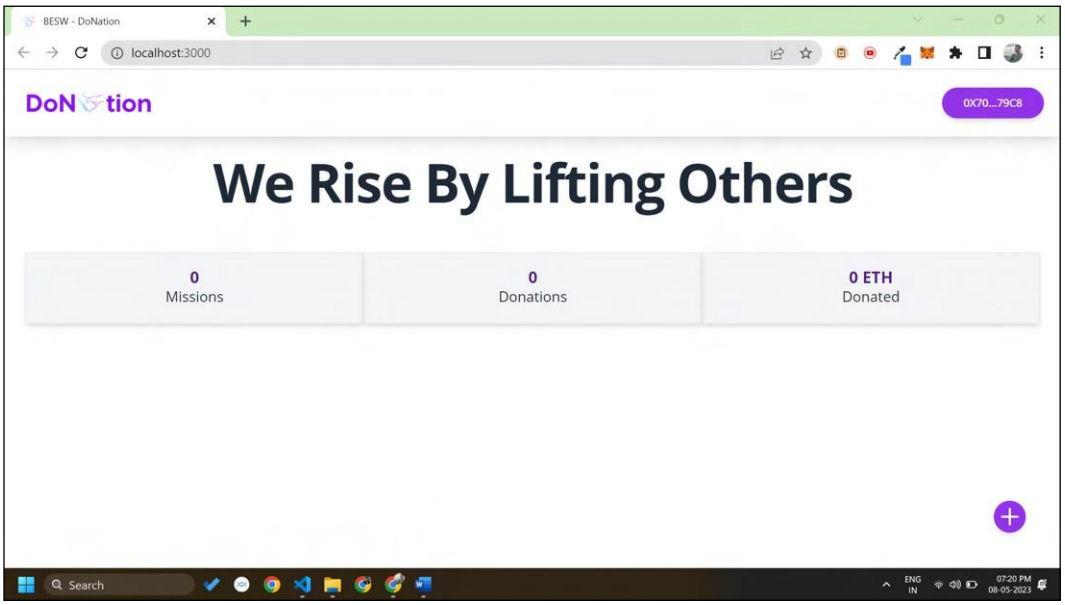


Fig 6.2 Admin View

Admin has the access to + button to add missions

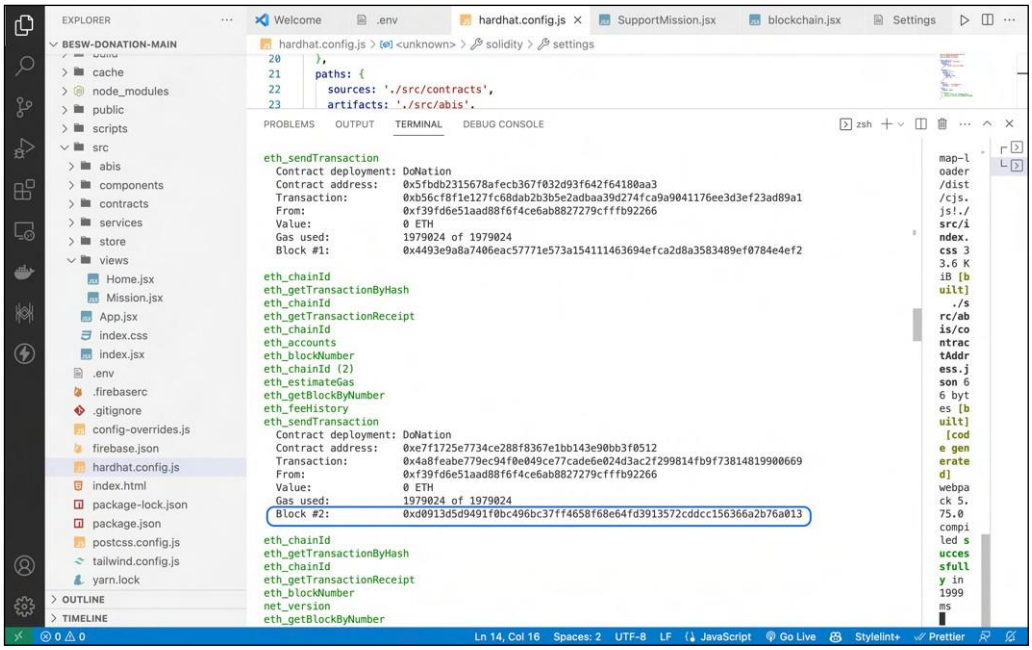


Fig 6.3 Block #2

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.

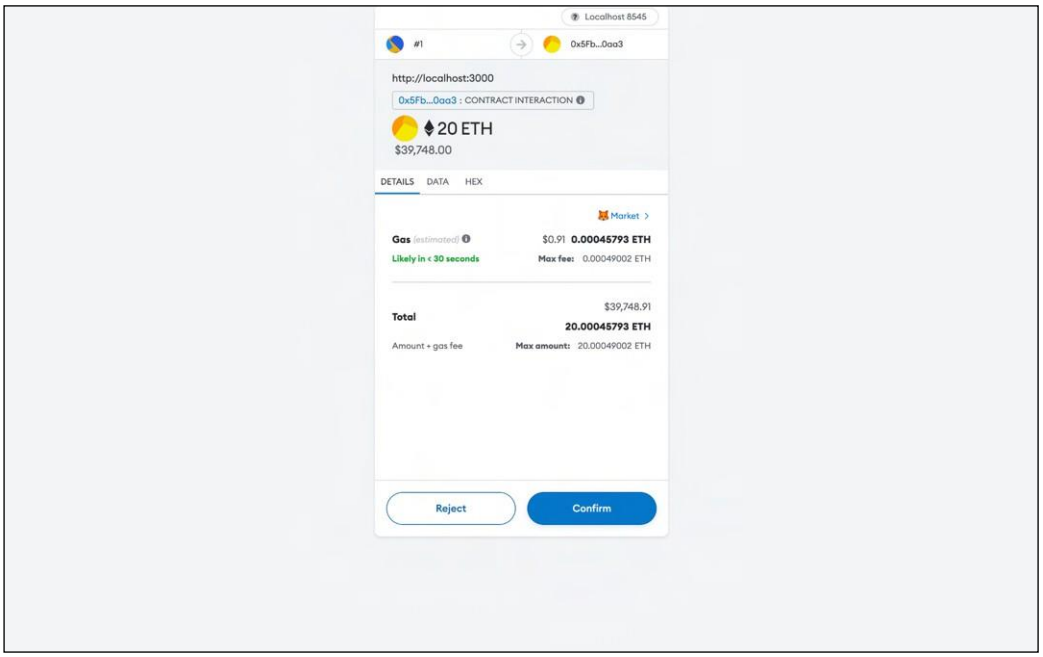


Fig 6.4 Contributor Payment

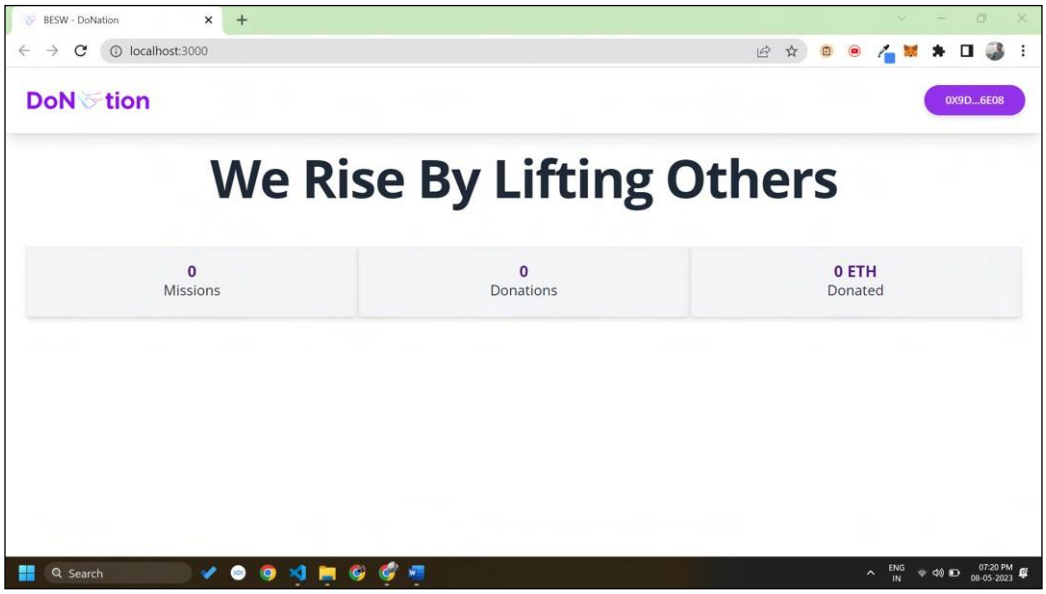


Fig 6.5 Contributor View

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

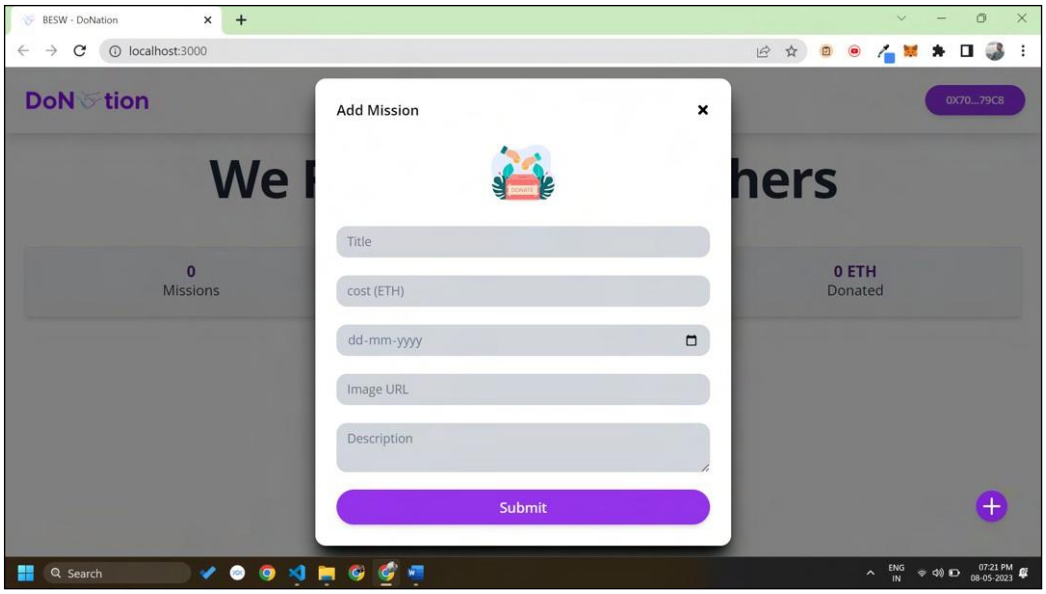


Fig 6.6 Admin adding Misson

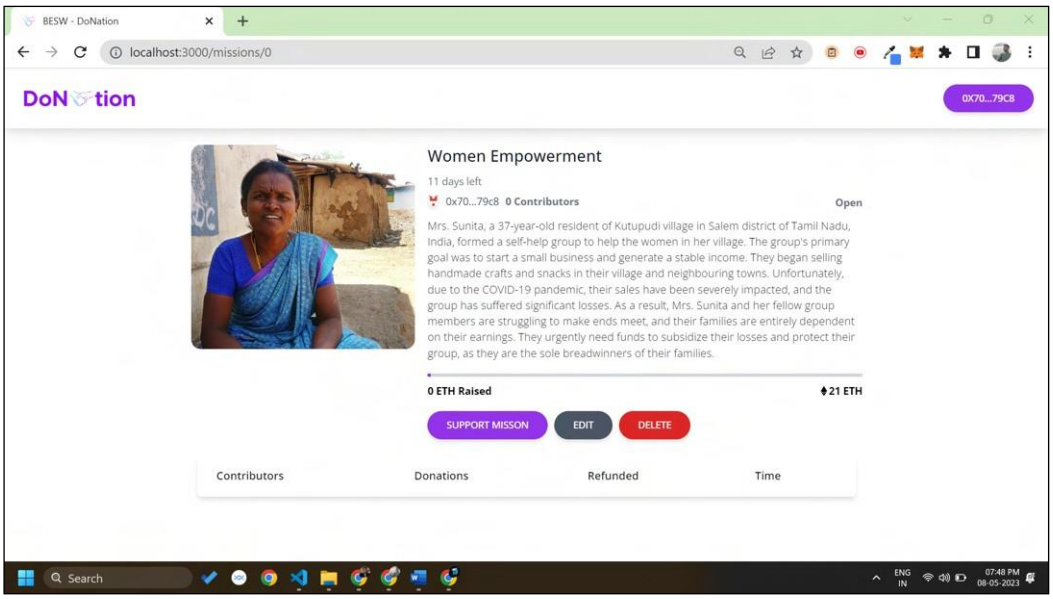


Fig 6.7 Mission's View (for admin)

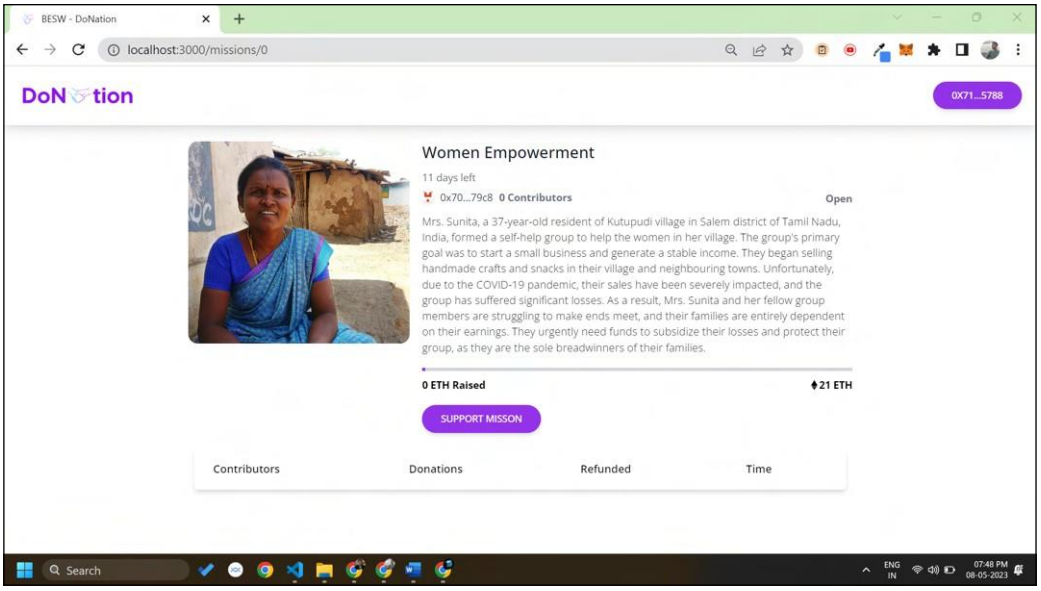


Fig 6.8 Mission's View (for contributor)

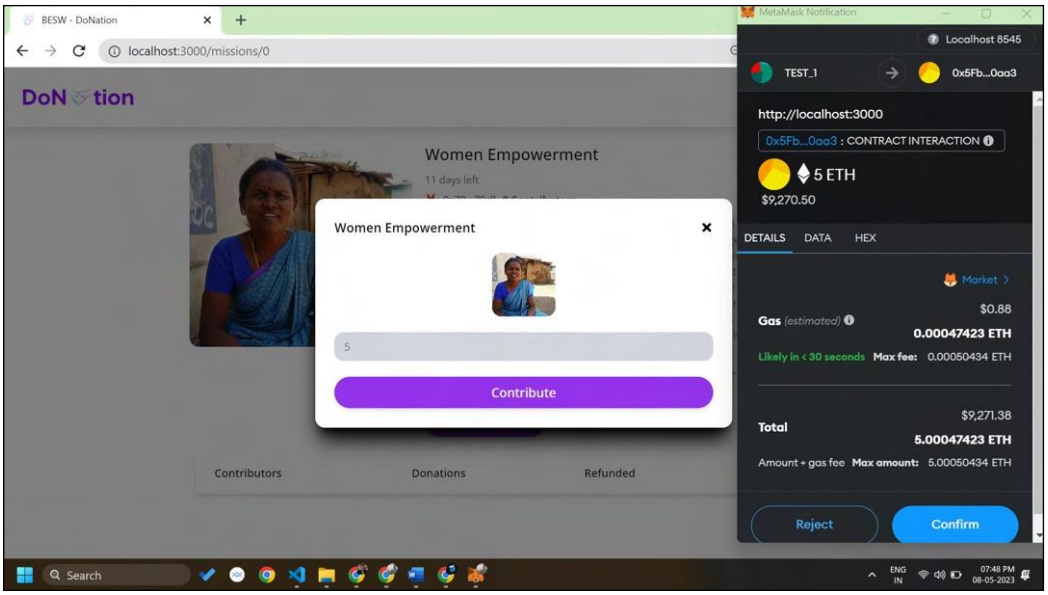


Fig 6.9 Contributing to the Mission

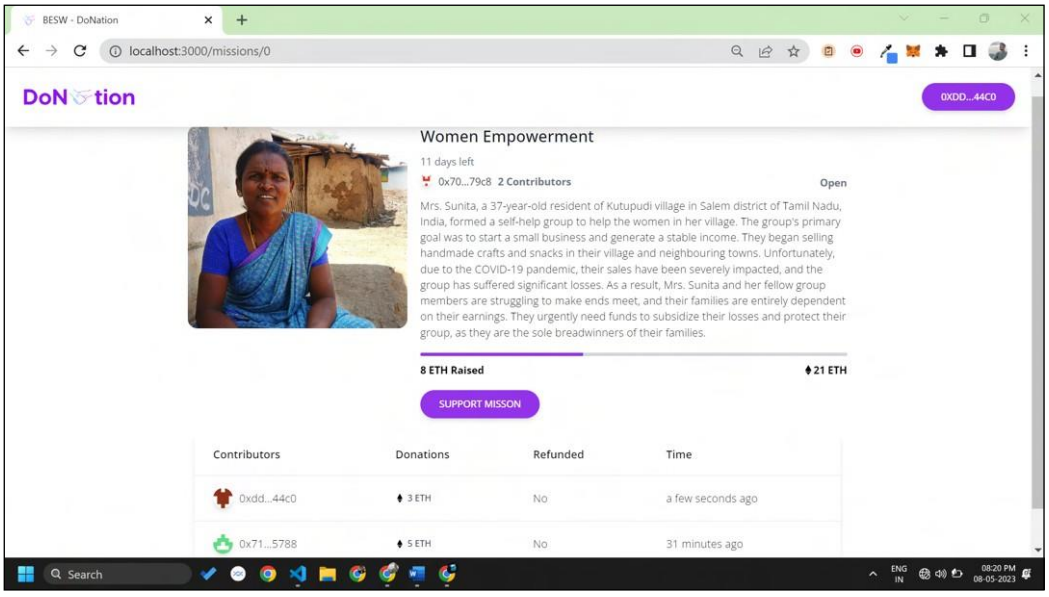


Fig 6.10 List of contributors

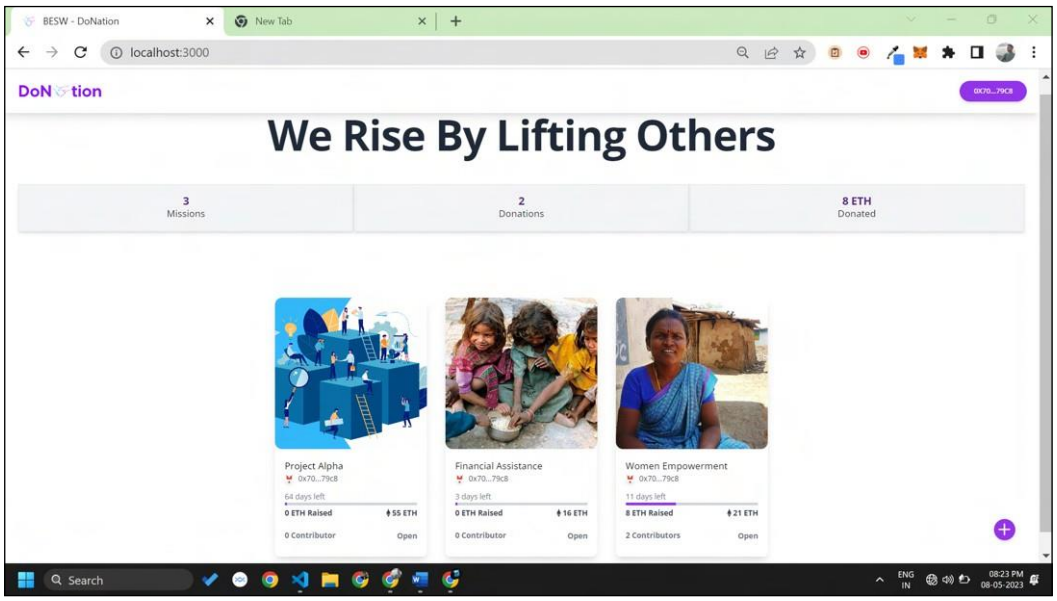


Fig 6.11 DoNation Dashboard View

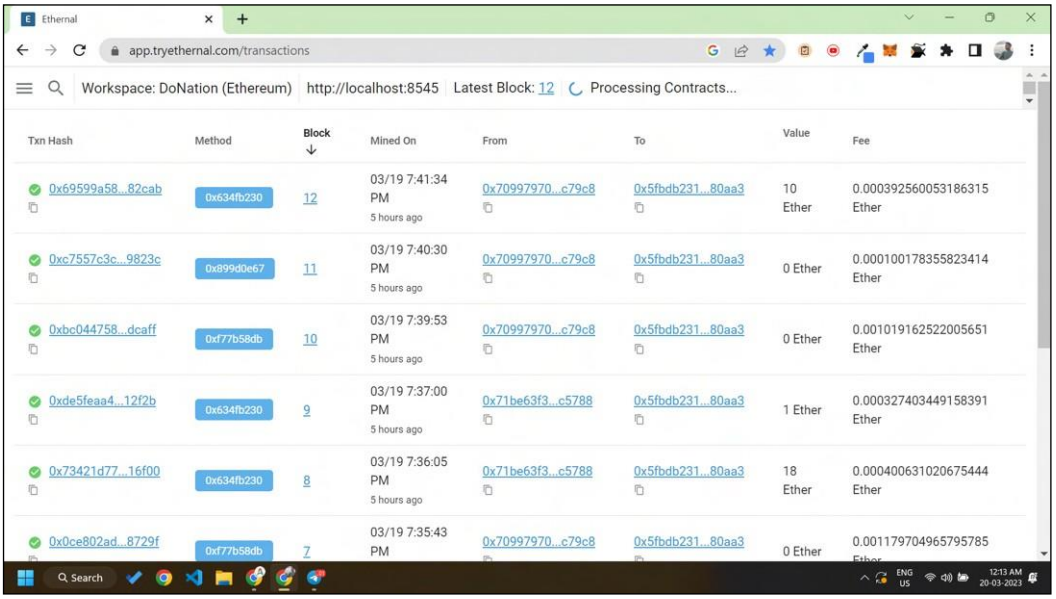


Fig 6.12 Ethernal Transaction's View

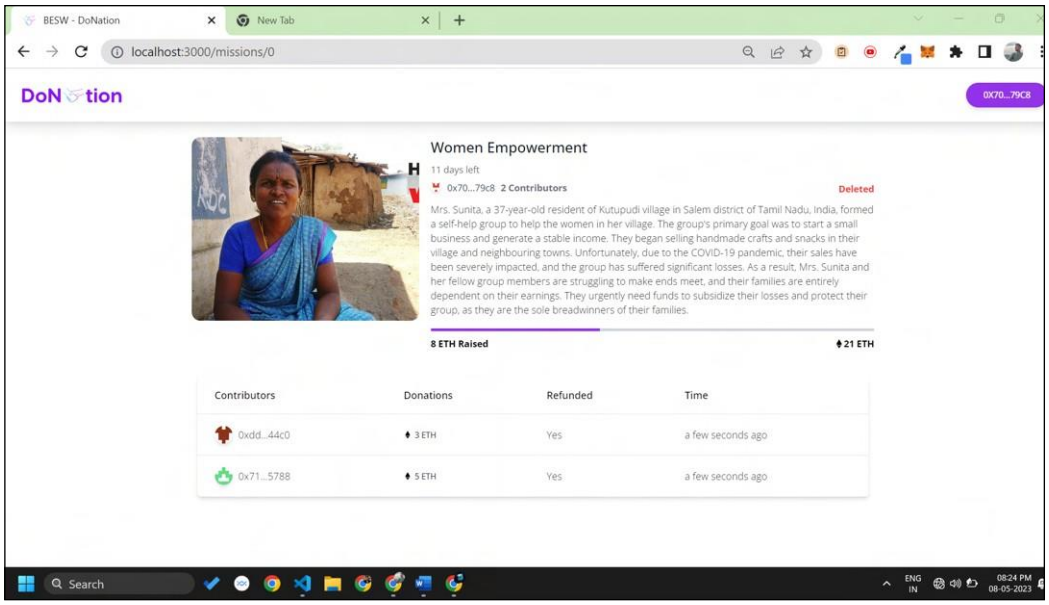


Fig 6.13 Refunded Status

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

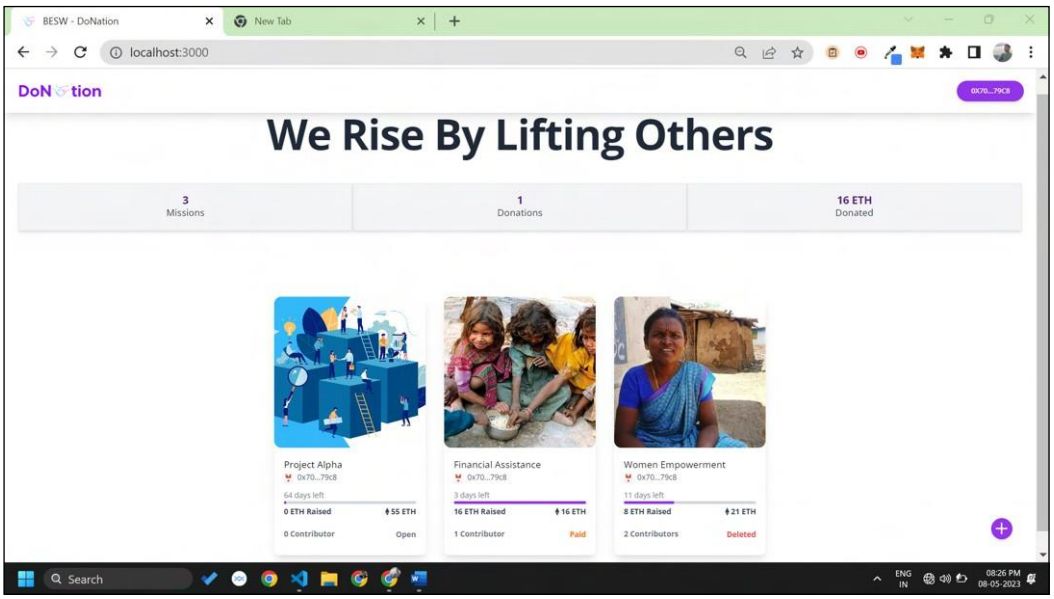


Fig 6.14 Normal Use Case View

DOREQUEST WORKING

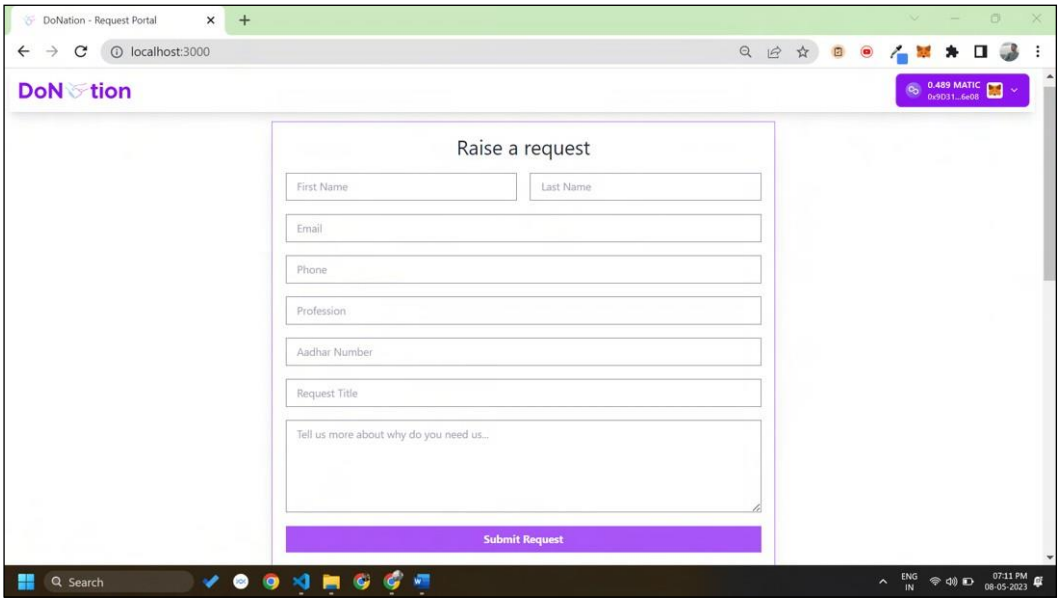


Fig 6.15 Beneficiary View

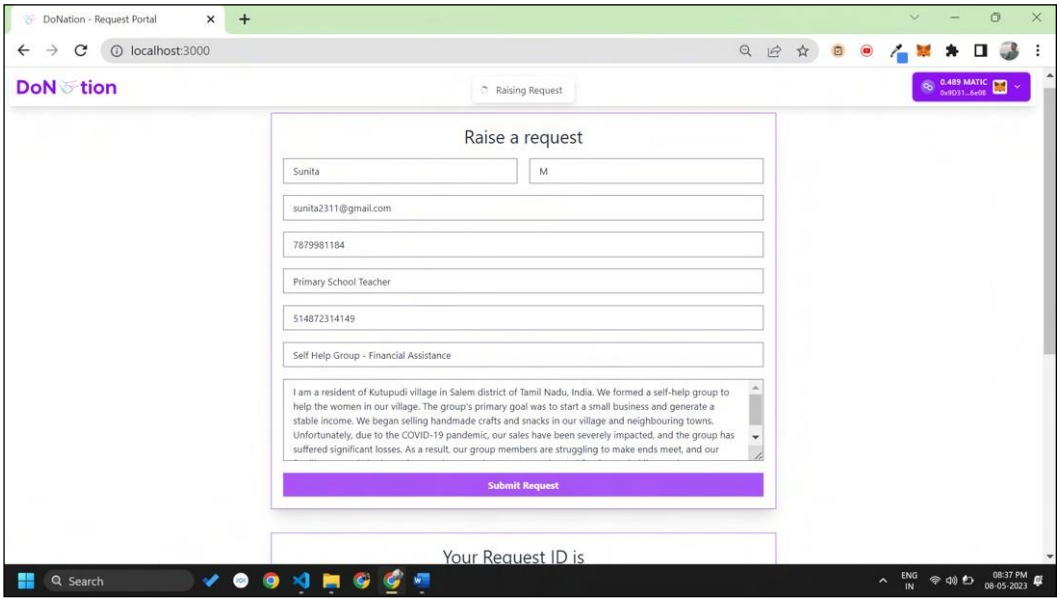


Fig 6.16 Beneficiary Filing the request

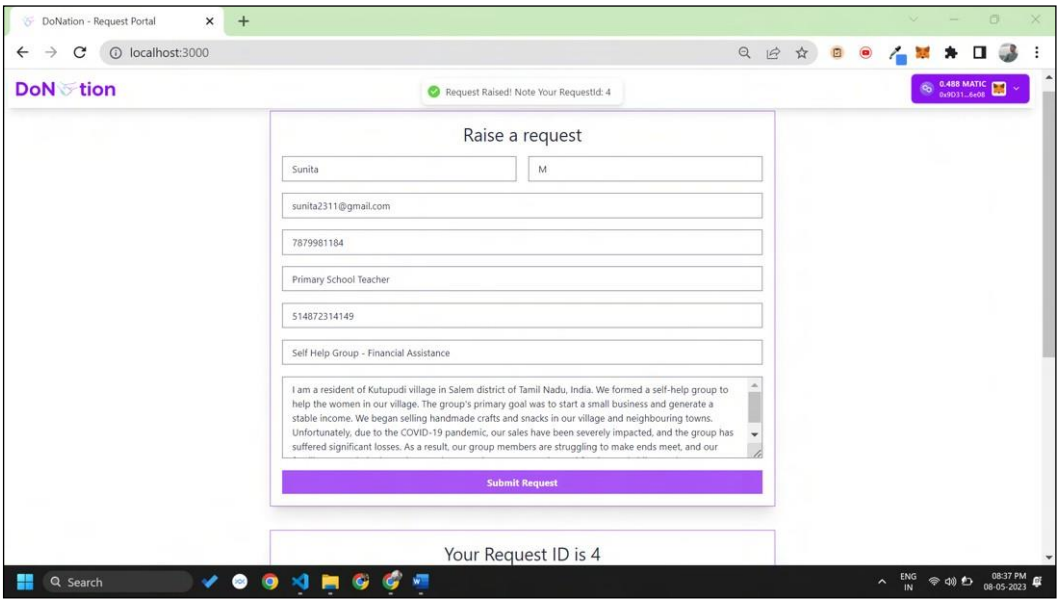


Fig 6.17 RequestID generated

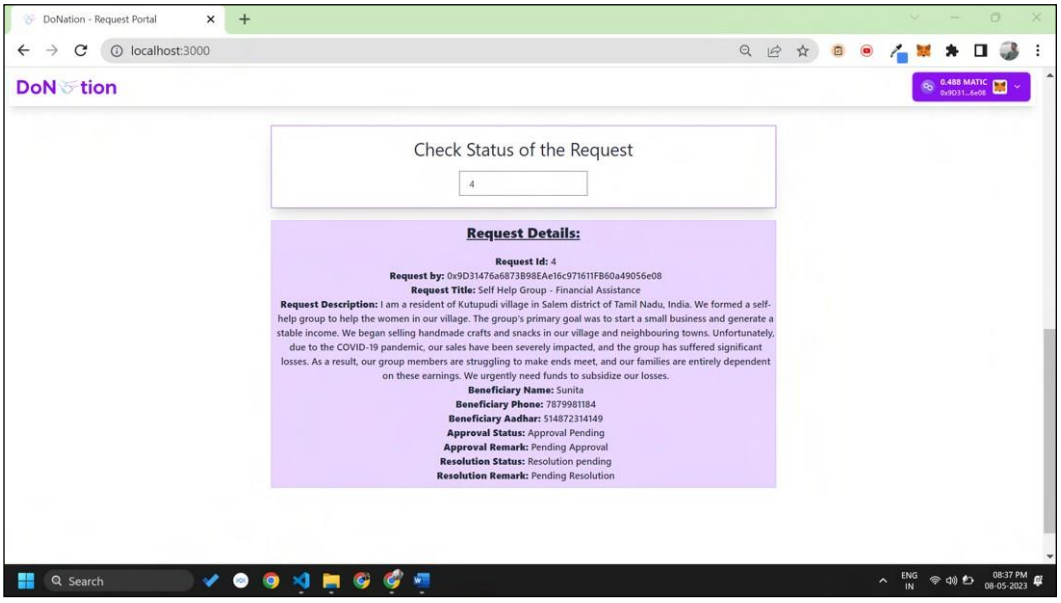


Fig 6.18 Checking status using RequestID - pending request

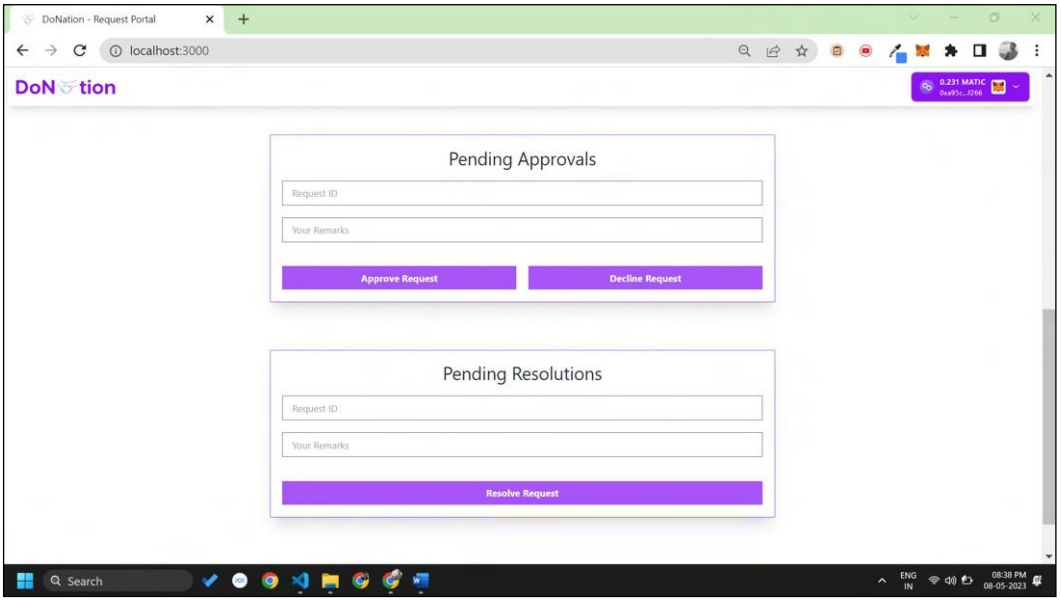


Fig 6.19 Admin View

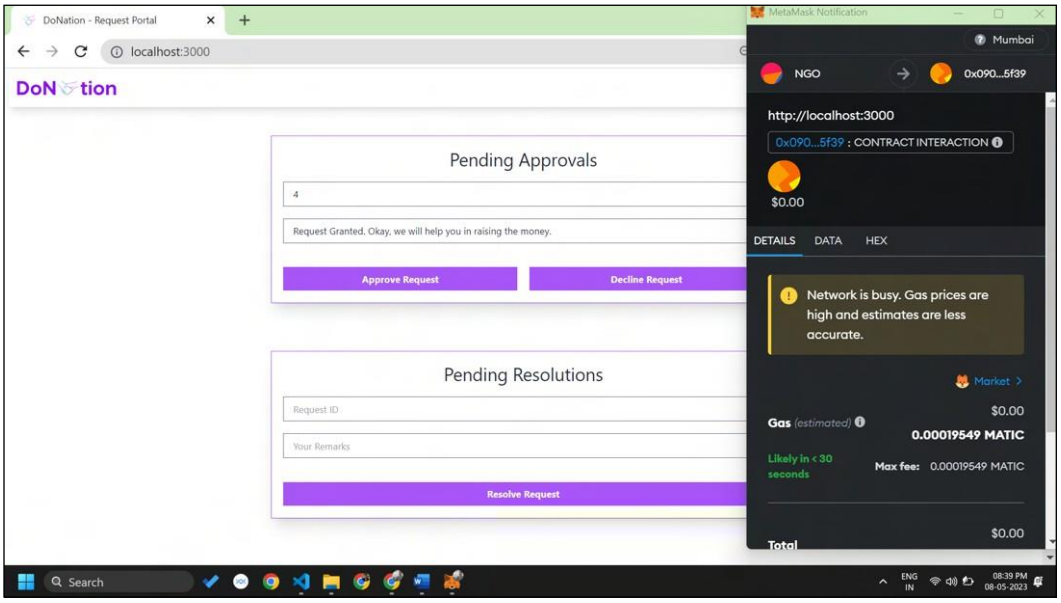


Fig 6.20 Admin View approving Request

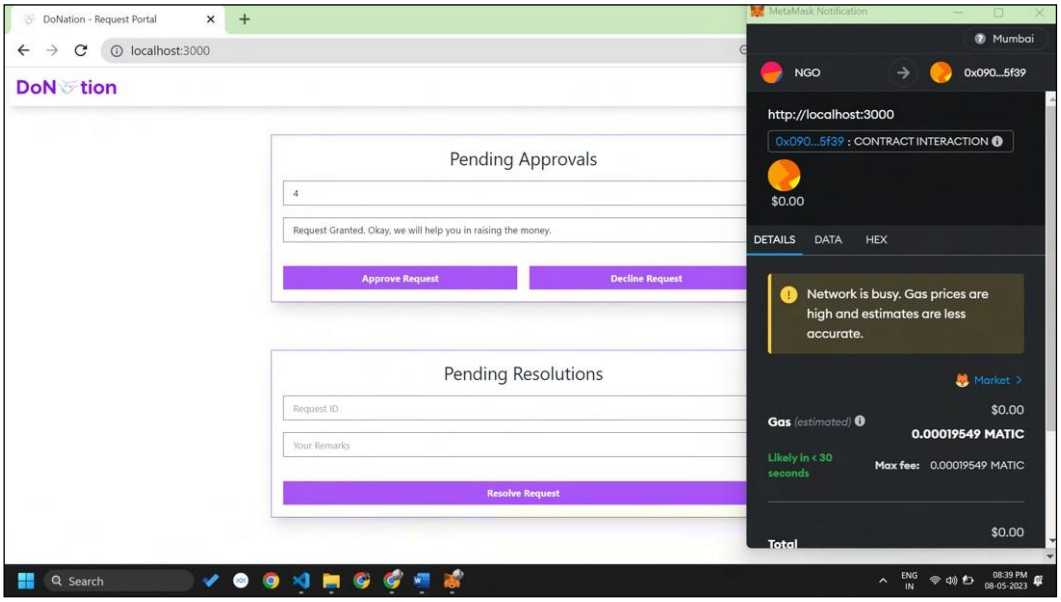


Fig 6.21 Admin View approving Request

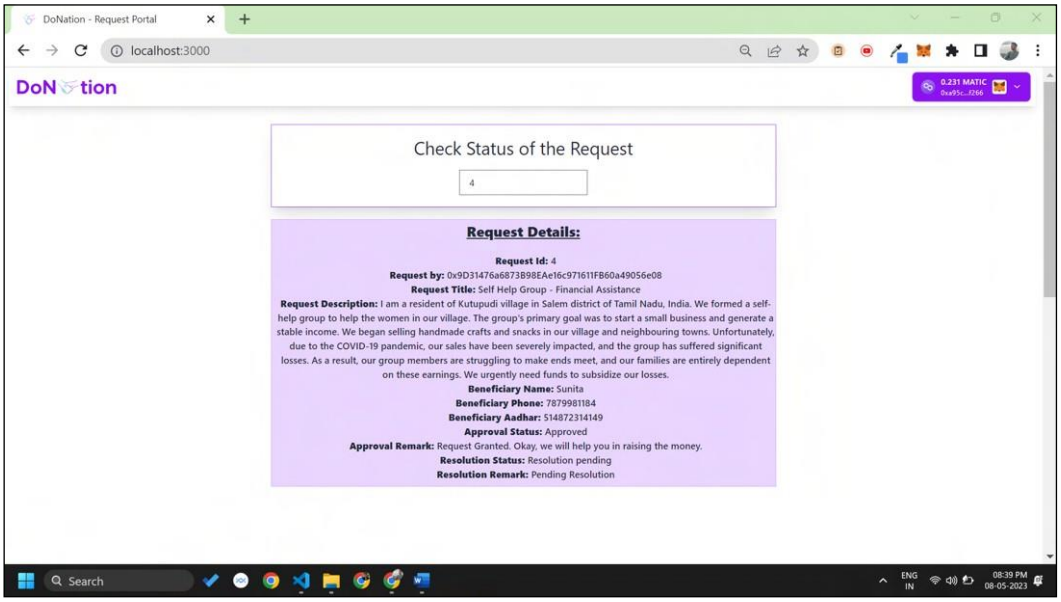


Fig 6.22 Request approved refelected on Beneficiary Side

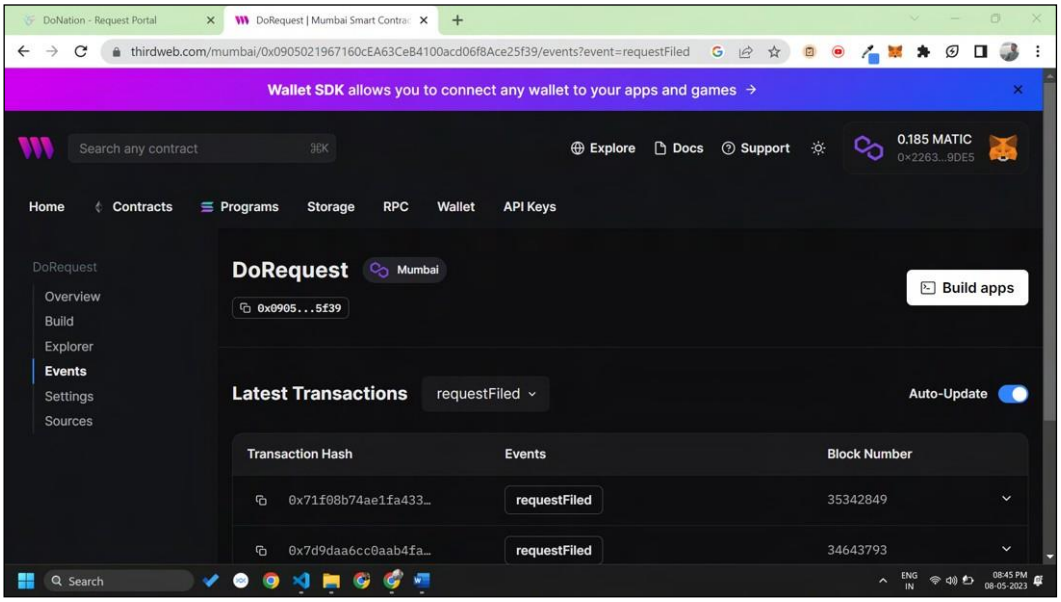


Fig 6.23 Smart Contract DoRequest uploaded on thirdweb

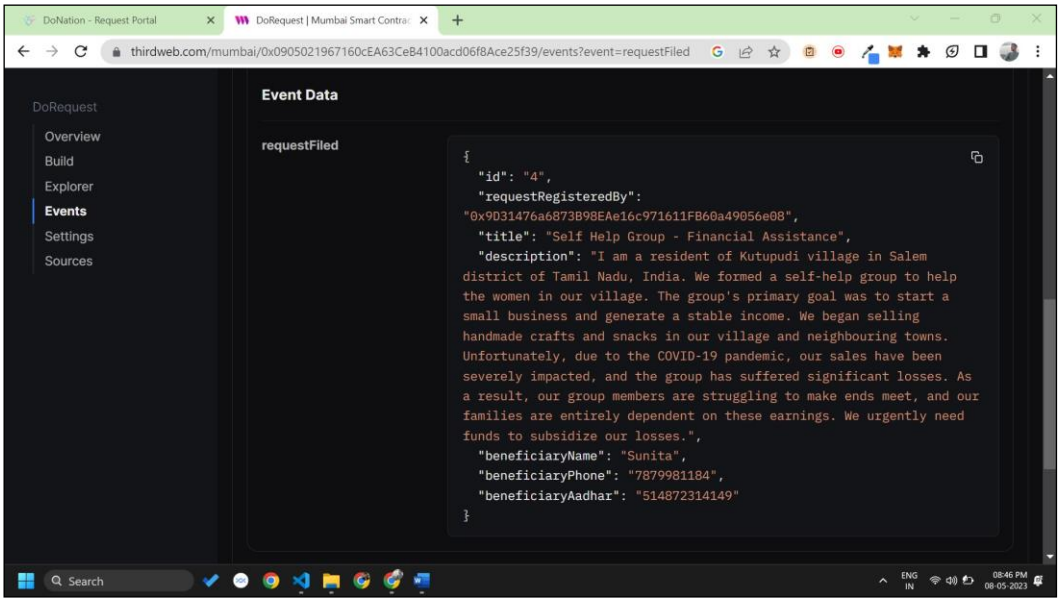


Fig 6.24 Block generated for each request being filed

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 char- ity 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.