CROWDFUNDING WITH SMART CONTRACTS

SUKLESH S RAO (16Z355)
HAREESHWAR K (17Z317)
SHIVANI B (17Z348)
TEJAS H BADANI (17Z354)
PRAVIN PRABHU S (18Z471)

15Z820 INTERNSHIP – III & PROJECT WORK II

Dissertation submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF ENGINEERING

Branch: COMPUTER SCIENCE AND ENGINEERING

of Anna University

April 2021

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE – 641 004

PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE – 641 004

CROWDFUNDING WITH SMART CONTRACTS

Bona fide record of work done by

SUKLESH S RAO	(16Z355)
HAREESHWAR K	(17Z317)
SHIVANI B	(17Z348)
TEJAS H BADANI	(17Z354)
PRAVIN PRABHU S	(18Z471)

Dissertation submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF ENGINEERING

Branch: COMPUTER SCIENCE AND ENGINEERING

of Anna University

APRIL 2021

K. Sathiyapriya

Dr. K. Sathiyapriya
Faculty guide

G. Sudha

.....

Dr. G. Sudha Sadasivam
Head of the Department

Certified that the candidate was examined in the viva-voce examination held on 29.4.21.

K. Sathiyapriya

(Internal Examiner)

.....

(External Examiner)

CERTIFICATE

Certified that this report titled "**CROWD FUNDING WITH SMART CONTRACTS**," for the INTERNSHIP – III & Project Work II (15Z820) is a bonafide work of **SUKLESH S RAO(16Z355), HAREESHWAR K(17Z317), SHIVANI B(17Z348), TEJAS H BADANI (17Z354), PRAVIN PRABHU S (18Z471)** who have carried out the work under my supervision for the partial fulfillment of the requirements for the award of the degree of Bachelor of Engineering in Computer Science and Engineering. Certified further that to the best of my knowledge and belief, the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or an award was conferred on an earlier occasion.

Place: Coimbatore

Date: 22.4.21

Dr. K. Sathiyapriya

Department of Computer Science and Engineering

PSG College of Technology

Coimbatore - 641004

COUNTERSIGNED

HEAD

Department of Computer Science and Engineering

PSG College of Technology, Coimbatore - 641004

ACKNOWLEDGEMENT

We would like to extend our sincere thanks to our Principal In-charge**, Dr. K. Prakasan**, for providing us with this opportunity to develop and complete our project in our field of study.

We sincerely express our sincere thanks to our Head of the Department, Dr**. G.Sudha Sadasivam,** for encouraging and allowing us to present the project at our department premises for the partial fulfillment of the requirements leading to the award of BE degree.

We take immense pleasure in conveying our thanks and a deep sense of gratitude to the Programme Coordinator**, Dr. Engels Rajangam**, Associate Professor, Department of Computer Science and Engineering, for his valuable suggestions and motivation all through the course of the project.

It is our privilege to express our sincere regards to our project guide, **Dr. K.Sathiyapriya**, Assistant Professor (SG), Department of Computer Science and Engineering, for her valuable inputs, guidance, encouragement, whole-hearted cooperation, and constructive criticism throughout the duration of our project.

We express our sincere thanks to all the panel members and faculty members of the department for helping us in resolving the problems incurred during the project work. We take this opportunity to thank all staff members of the department.

SYNOPSIS

Crowdfunding with smart contracts is a project powered by Blockchain technology to help people build projects and crowdfund them. The projects can be anything from community service, civil structure restoration to movie making and gadget building. In a centralized crowdfunding management system, the mutable records of the transactions could be exploited for unethical purposes. With Blockchain, the whole environment is decentralized and the transactional details are immutable. The activities can be tracked publicly by anyone in the network and the possibility of fraudulent activities can be greatly reduced.

This project is implemented using Ethereum smart contracts. In Ethereum smart contracts the projects can receive donations from anybody on the internet with an Ethereum wallet. The proper usage of these funds is then verified by volunteers called Moderators who could be random people on the public website. The Moderators validate the withdrawals completed by the projects and award reputation tokens to the respective project creators. The project creators are incentivized by these tokens to keep submitting proof and perform valid transactions so that they are able to do more withdrawals of larger amounts. In return, the moderators also receive Moderator tokens which are worth certain ether (transactional token of Ethereum) units as compensation for their moderation work.

On the whole, this application help organizations and project leads to maintain accountability and increase efficiency, while providing the necessary transparency to the donors about the transactional activities done by the organization.

CONTENTS

CHAPTER **Page No.**

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Crowdfunding is a process of raising funds for a project or venture from the masses, where people pool in smaller contributions/investments to create a significantly larger investment to meet the requirements. The burden of investment is distributed across the masses. This is so that no single person has to spend more than they can spare. This concept has been around for ages now. There are several instances where artists, authors, philosophers have crowdfunded their works.

Blockchain technology and digital currencies have proven to be capable of being retrofitted to meet the needs of any industry, including the time-tested ones. Crowdfunding is one such segment into which the technology is capable of blending in thoroughly. Blockchain technology is perfectly capable of managing value exchange under contract. The features provided by Blockchain will come in handy when it comes to managing rewards-based and equity-based crowdfunding campaigns. The process of assigning relevant rewards and equity against their contributions can be automated, using Blockchain-based smart contracts for crowdfunding.

The crowdfunding platform is a peer-to-peer fundraising model that provide different possibilities for the start-ups by raising funds to create their digital currency, and it. Some of the famous crowdfunding cryptocurrencies are coin space, swarm, judobaby, etc. Crowdfunding has offers for creators and other consumers. Anyone can participate in this crowdfunding if they have invested any new cryptocurrency (e.g., Ethereum) and can contribute as much as possible.

By programming a set of pre-defined crowdfunding conditions on smart contracts, the system can be automated to execute the smart contract to issue certain rewards or proof of ownership of a certain percentage of equity based on the amount contributed towards the campaign.

1.1 BLOCKCHAIN TECHNOLOGY

The Blockchain is a distributed database of records of all transactions or digital events that have been executed and shared among participating parties. Each transaction is verified by the majority of participants of the system. It contains every single record of each transaction. Blockchain technology records transactions in a digital ledger, which is distributed over the network, thus making it incorruptible. Anything of value, like Land Assets, Cars, etc., can be recorded on the Blockchain as a Transaction.

1.1.1 WORKING OF BLOCKCHAIN

A Blockchain is nothing but a chain of blocks that holds some severe properties utilised to enable decentralisation over the internet. Decentralisation means nobody has full authority or control over the network, but rather the authority is distributed to the users who use it, in the case of Blockchain, to the miners, the users.

Distributed Ledger

A ledger is a system containing all the records of the input and output of a process. A distributed ledger is a data structure that is spread across different computing devices. DLT (Distributed Ledger Technology) is the technology that distribute records across all the users. DLT consists of 3 components – the Data Model (current state of ledger), Language of transactions (which changes ledger state), and Protocol (used to build consensus). Blockchain is a type of DLT. This way, the data is shared among all its users, increasing transparency and avoiding corruption.

Consensus

The consensus is a process of ensuring that all the different users in a Blockchain come to an agreement regarding the current state of the Blockchain. Different Blockchains use several consensus mechanisms to achieve consensus. For example, Bitcoin uses Proof-of-Work while Ethereum is moving from Proof-of-Work to Proof-of-Stake algorithm.

A Block in a Blockchain

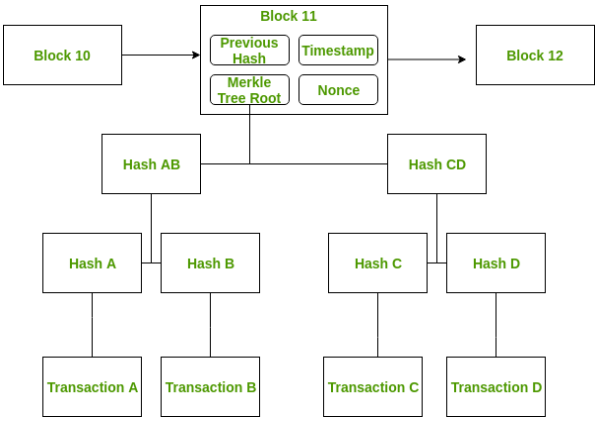

Fig 1.1 A block in a Blockchain

A block as shown in figure 1.1 consists of four parts:

- Previous Hash
- The timestamp
- Nonce
- Merkle tree root

Each block contains a cryptographic hash of the data of the previous block. The miners calculate the nonce by solving a cryptographic puzzle to propose the next block in the chain. It is known as proof of work. The Blockchain is said to be immutable because of its cryptographic properties. However, this does not mean that changing the data is impossible. It means that it is tough to change the data, and any change can be easily detected. A merkle tree is a binary tree with hash pointers. A merkle tree is a structure that allows for efficient and secure verification of content in a large body of data. The advantage of using merkle trees is that proving membership requires O(logn) steps. Also, in a sorted merkle tree, non-membership can be proved in O(logn) steps. The first block is known as the genesis block.

1.1.2 PROOF OF WORK ALGORITHM

The purpose of the Proof of Work consensus mechanism is to bring all the nodes in agreement, that is, trust one another, in an environment where the nodes do not trust each other. All the transactions in the new block are then validated, and the new block is then added to the Blockchain. The miners bundle up a group of transactions into a block and try

to mine. To mine it, a challenging mathematical problem has to be solved. This problem is called the proof of work problem that has to be solved to show that the miner has done some work to find out the solution to the problem, and hence the mined block must be valid.

In the Hashcash Proof of Work System, the answer to the problem needs to be a lower number than the hash of the block for it to be accepted, known as the **'**target hash**.'** A target hash is a number that the header of a hashed block must be equal to or less than for a new block, along with the reward, to be awarded to a miner. The lower a target is, the more difficult it is to generate a block. A miner continues testing different unique values (known as nonce(s)) until a suitable one is produced. When a miner finally finds the right solution, the node broadcasts it to the whole network simultaneously, receiving a cryptocurrency prize (the reward) provided by the PoW protocol.

1.1.3 SMART CONTRACTS IN BLOCKCHAIN

A smart contract is just a digital contract with the security coding of the Blockchain. A smart contract has details and permissions written in code that require an exact sequence of events to take place to trigger the agreement of the terms mentioned in the smart contract. It can also include the time constraints that can introduce deadlines in the contract.

This contract is embedded in the Blockchain, making it transparent, immutable, inexpensive, and decentralized. Every smart contract has its address in the Blockchain. The contract can be interacted with by using its address, presuming the contract has been broadcasted in the network.

The idea behind smart contracts is pretty simple. They are executed based on simple logic, IF-THEN for example:

- **IF** Person A sends Person B the object X, **THEN** the sum (of money, in cryptocurrency) will be transferred to Person B
- **IF** Person A transfer a certain amount of digital assets (cryptocurrency, for example, ether, bitcoin), **THEN** the X object will be transferred to Person A
- **IF** Person A finishes the work, **THEN** the digital assets mentioned in the contract will be transferred to Person A

1.2 FEATURES OF BLOCKCHAIN

1. **Trust:** The smart contract cannot be lost as it is embedded in the Blockchain itself.
2. **Accuracy:** Smart contracts are accurate to the limit a programmer has accurately coded them for execution.
3. **Speed:** Smart contracts use software code to automate tasks, reducing the time it takes to maneuver through all the human interaction-related processes.

4. **Backup:** Every node in the Blockchain maintains the shared ledger, providing probably the best backup facility.
5. **Autonomy:** No intermediaries involved, which minimizes bullying and grants full authority to the dealing parties. The smart contract is also maintained and executed by all the nodes on the network, thus removing all the controlling power from any one party's hand.
6. **Safety:** Cryptography makes sure that the assets are safe and sound. Even if someone breaks the encryption, the hacker will have to modify all the blocks that come after the block which has been modified. This is a challenging and computation-intensive task and is practically impossible for a small or medium-sized organisation to do.
7. **Savings:** Smart contracts save money as they eliminate the presence of intermediaries in the process. Also, the money spent on the paperwork is minimal to zero.

1.3 APPLICATIONS OF BLOCKCHAIN

1. **Real Estate:** A smart contract to transfer ownership of an asset once a certain number of resources have been transferred to the seller's account (or wallet).
2. **Maintenance and Ownership**: The smart contract can, for example, enforce vehicle maintenance service every six months, failure of which will lead to suspension of driving license.
3. **Music Industry:** A smart contract embedded in the Blockchain and royalties can be credited to the owner's account when the song is used for commercial purposes. It can also work in resolving ownership disputes.
4. **Government elections:** Once the votes are logged in the Blockchain, it would be very hard to decrypt the voter address and modify the vote leading to more confidence against the ill practices.
5. **Supply Management:** A smart contract can be deployed to trigger the supply of raw materials when 10 tonnes of plastic bags are produced.
6. **Automating healthcare payment:** Every treatment is registered on the ledger, and in the end, the smart contract can calculate the sum of all the transactions. The patient cannot be discharged from the hospital until the bill has been paid, and it can be coded in the smart contract.

1.4 MOTIVATION

Crowdfunding allows businesses with great products and service ideas to raise funds from regular people in small investment amounts. Companies like Kickstarter, Indiegogo, and Crowdfunder were among the earliest to make it popular. According to Kickstarter, 78% of campaigns that raise 20% of their goal ultimately become fully funded, while 11% of projects finish having never received any funding at all. Blockchain can change the crowdfunding landscape by making the funding process safe. It offers completely transparent access from anywhere in the world. Crowdfunding platforms that use Blockchain can help maximize a project's success, and it is about time a decentralised platform revolutionises the crowdfunding sector.

1.5 PROBLEM DEFINITION

In a centralized crowdfunding management system, the transactions' mutable records could be exploited for unethical purposes. Most of the current platforms are run by a company, and once the donor has parted with her cash, it is up to the company to secure this money and transfer it. Crowdfunding has become a trendy way to democratize fundraising for many new-age start-ups. However, the current crowdfunding platforms are centralized, which means the platform controls the fund and can provide preferential treatment to specific campaigns. The activities are not fully transparent and are not publicly tracked, potentially leading to scams and loss of funds.

1.6 OBJECTIVE OF THE PROJECT

This crowdfunding project powered by Blockchain aims to

- Allow anybody on the network to start a project for funding
- Enable the projects to receive donations from anyone with an Ethereum wallet
- Verify proper usage of funds by Moderators who could be random people on the network
- Provides incentives for the Moderators and the Organizations to compensate their work
- Provide transparency to the donors about the transactional activities with their donations

1.7 SCOPE OF THE PROJECT

This project's scope extends to the funding and transactional activities being tracked publicly by anyone on the network and brings down the possibility of fraudulent activities. Some of the applications to which this project can be extended are as follows.

- Initial Coin Offerings (ICO) of organisations offering their cryptocurrency
- Charity campaigns by governmental and non-governmental organisations
- Relief fund donations during period of pandemics or natural disasters
- Funding for auctioning assets

CHAPTER 2

SYSTEM STUDY

This chapter details the knowledge gained from the materials in literature, which include substantive findings and theoretical contributions on crowdfunding using blockchain.

Kangana et al. [1] developed a web application using the Ethereum platform's smart contracts to ensure the security for crowdfunding's contribution amount. They provided interactive forms for campaign creation, monetary contribution, viewing campaigns, request approval and finalizing requests through which both campaign creators and contributors can easily create and fund the campaigns.

Vikas et al. [2] proposed a global crowdfunding platform called BitFund in which Investors and developers act as different nodes of the network. The investors can request a specific project, and they can give their initial bid value in terms of time, cost and maintenance required. Different developers can bid with different values of the same parameters to get the project ownership. A smart contract is deployed between the investors and the developers to reach an optimal solution for the investors. Multiple iterations of bidding are carried out between the developers until the optimal solution or equilibrium is reached. The proposed model motivate the developers to iteratively change their bids to ultimately match the investor's needs and only present the most suitable option to the investor.

Pledge camp et al. [3] discuss a new crowdfunding ecosystem that introduces a Vendor Network for hiring services and a Knowledge Center for crowdfunding research. The paper discusses a two-token economy with Pledge Coins and Camp Shares.This powers the platform and compensate the users for their contributions to the ecosystem.

Hasan Barber [4] studied WHIRL, a blockchain-based crowdfunding platform founded in 2018. Whirl worked on the principle of "Pay-it-forward," which means a project can only be started when its owner has already backed or contributed towards other projects. The platform believes in inspiring the positive return loop of generosity. It worked based on Karma points that a member has to collect to start its own campaign.

Table 2.1 Summary of system study

Paper Name	Author	Key Points	Year
Application of blockchain technology of crowdfunding using smart contract	Kangana W. M, Sowmya M, Namita Chougule, Sejal Patil, Soumya Kadadi	• Web-based application • Ensures security for the contribution amount • Uses a voting system to provide the money provided by the donors to the campaign creator.	2020
BitFund: A Blockchain-based Crowd Funding Platform for Future Smart and Connected Nation	Vikas Hassija, Vinay Chamola, Sherali Zeadally	• Proposed a unique and secure crowdfunding platform based on blockchain and ethereum smart contracts. • Applied an optimal-cost job assignment based on the Hungarian algorithm. • Implement iterative auction algorithms that allow developers to vary their bid amounts over iterations to increase their winning chances. • Eliminated the need for any manual negotiation between investors and developers for the project's parameters.	2020
Pledgecamp the next generation of crowdfunding	https://pledgecamp.com/	• Introduced the use of tokens • Parties are incentivized towards shared goals. ○ Creators and platforms should be equally invested as backers in the successful delivery of products. • Proposes to fix long-standing crowdfunding problems by introducing accountability and transparency with smart contracts and aligning user interests through an inclusive and rewarding token economy.	2020
Based Crowdfunding: A'Pay-it-Forward' Model of WHIRL	Hasnan Baber	• Worked on the principle of "Pay-it-forward ○ The project can only be started when its owner has already backed or contributed towards other projects. • Works based on Karma points	2019

From the system study, the various existing aspects of implementing a decentralised crowdfunding platform as well as token incentivization are analysed. The current models include incentives for the moderating parties and the removal of intermediaries. Based on this, our proposed solution is to include an additional incentive for the project creator called the reputation token, motivating them to work towards the shared goal without any misappropriation of the funds.

CHAPTER 3

SYSTEM ANALYSIS

This chapter describes the system analysis, which is conducted to study a system and its parts to identify its objectives.

3.1 FUNCTIONAL REQUIREMENTS

The system should be able to,

1. Accept payment in Ether
2. Distribute tokens efficiently
3. Allow users based on their roles (Donor, Moderator, Project Creator)
4. Transfer Ether in a short duration
5. Track the Ether and the token distribution

3.2 NON-FUNCTIONAL REQUIREMENTS

The non-functional requirements are as follows:

3.2.1 Usability Requirements

The system should be user-friendly and does not require any guidance. In other words, the application has to be as simple as possible, so its users shall use it easily.

3.2.2 Reliability Requirements

The system should not have any unexpected failure and must be reliable at least 98% of the time. In order to avoid any failure's occurrence, the specifications have been respected and followed correctly.

3.2.3 Efficiency Requirements

The system response time should be adequate and sufficient to increase the system's efficiency. The application should be compatible with different versions of the libraries used.

3.3 EXPERIMENTAL SETUP

3.3.1 Hardware Requirements

The following are the details of the minimum requirements of the workstation needed for implementation and evaluation.

- PC with x86 compatible processor

- Windows 7 or later
- OS X El Capitan 10.11 or later (If Mac)
- 4GB RAM

3.3.2 Software Requirements

The following are the software requirements of the system:

1. Ganache installed for local development
2. Google Chrome for smart contract development and testing on Remix IDE
3. Visual Studio to edit code
4. Libraries used:
    a. Truffle
    b. NodeJS
    c. Web3.js
    d. ReactJS
    e. Chai and Mocha for JS Testing

3.4 FEASIBILITY ANALYSIS

A feasibility study is used to determine the practicality of an idea or proposed plan.

3.4.1 Economic Feasibility

As this project uses only open-source distributions like Ganache, NodeJS, etc., it is economically feasible.

3.4.2 Technical Feasibility

Ethereum has numerous features like third-party modules, extensive support for libraries, simplicity, and enhanced processing capabilities, making it a viable option for speed and productivity.

3.4.3 Operational Feasibility

All the functions performed by the system are valid and without conflict. All the functions and constraints specified in the requirements are operational.

CHAPTER 4
SYSTEM DESIGN

This chapter describes the design of the blockchain-based crowdfunding platform. It outlines the functionalities of the crowdfunding platform after analyzing the requirements.

4.1. USE CASE DIAGRAM

Fig.4.1 shows the use case diagram for the Blockchain-based crowdfunding platform. The proposed decentralised crowdfunding solution consists of three actors who interact with the system.

- Project creator:
  - The project creator could be any person on the network who wants to initiate a new project and request for funding. The project creator can withdraw the donated fund received and submit proof of proper usage to receive reputation tokens.
- Donor:
  - A Donor could also be anyone on the network who wants to donate to a certain project. The donor can view the desired project which was created by a project creator and donate funds from an ethereum wallet.
- Moderator:
  - The moderator is a randomly chosen person on the internet who verifies and validates the transactions done in the network - donations and withdrawals - and receives moderations tokens as compensation.

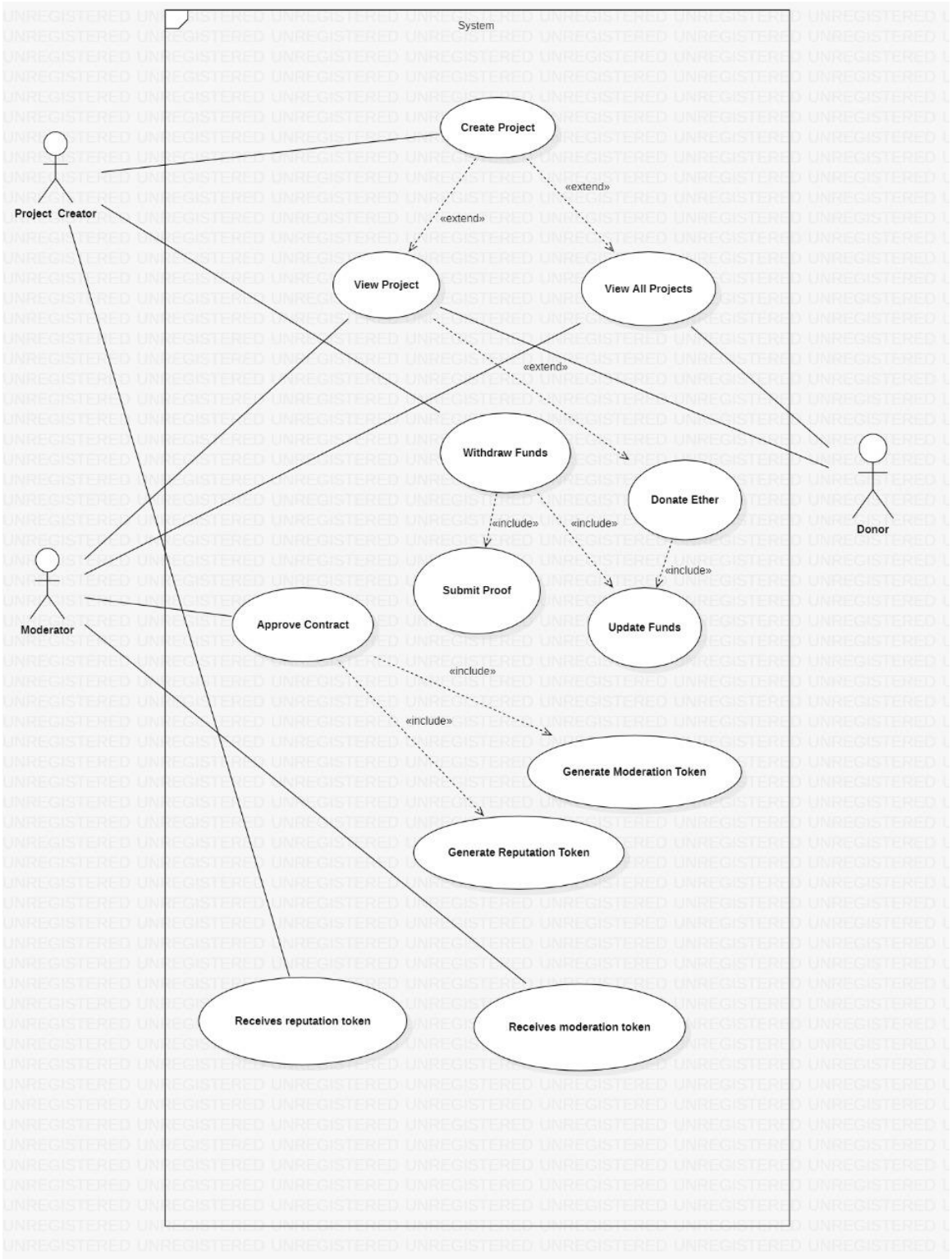

Fig 4. 1 Use case diagram

4.2. ACTIVITY DIAGRAM

Fig.4.2 shows the activity diagram for the Blockchain-based crowdfunding platform.

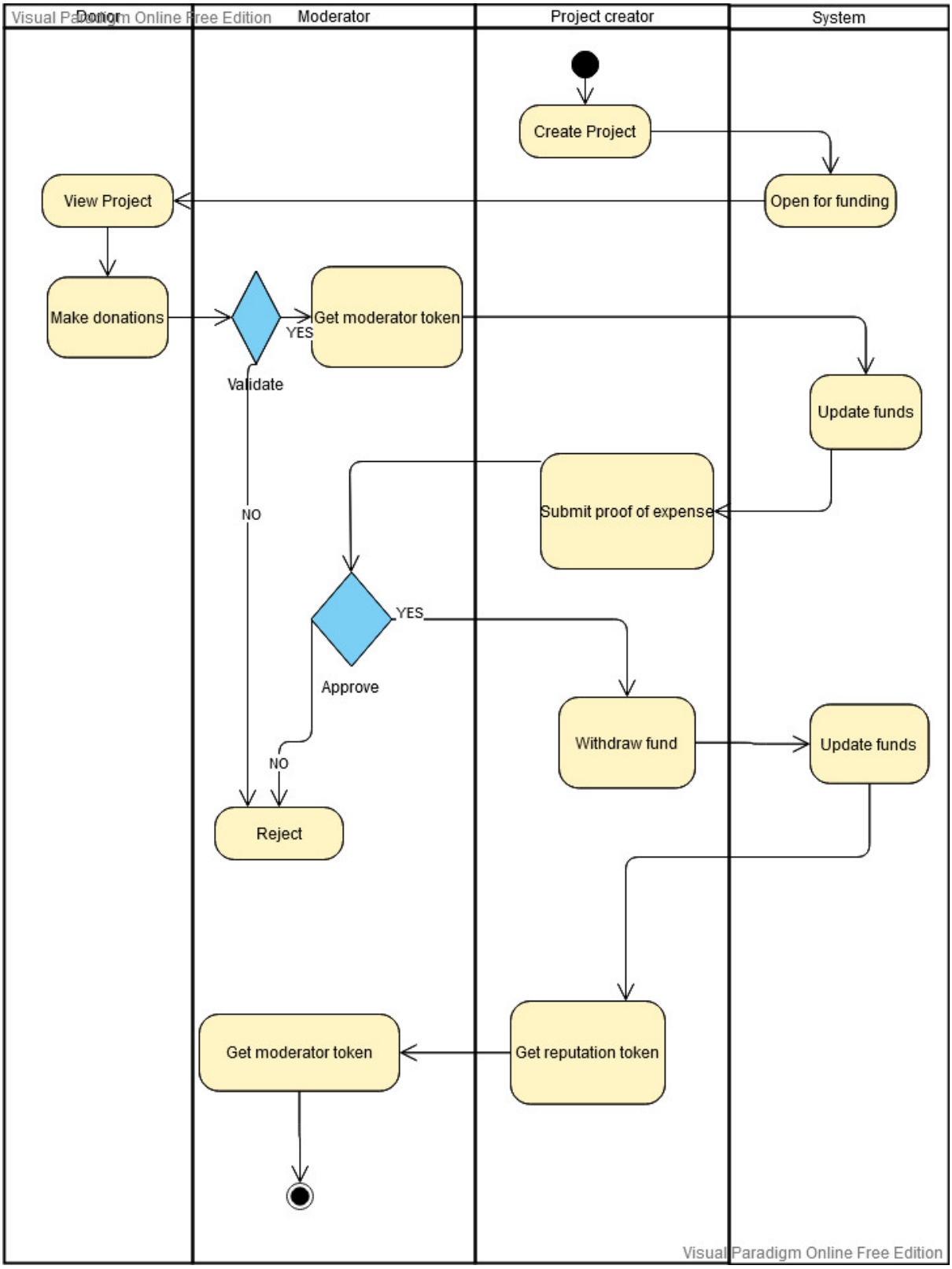

Fig 4. 2 Activity diagram

The flow of events in this setup are as follows:

1. First, a project has to be created for the crowdfunding to be initiated. The project creator creates a new project which goes into the blockchain and becomes open for funding.
2. The willing donors can view the projects that are open for funding and make donations to the desired project.
3. The moderators validate the donations, and the funds are updated. Upon successful updation, the moderators receive moderation token.
4. The donated funds for a project can be withdrawn by the respective project creator who then has to submit a proof of useful expense.
5. These withdrawals are verified again by the moderators which upon successful completion allows for the project creators to receive reputation tokens and the moderators to receive moderation tokens.

4.3. ARCHITECTURE DIAGRAM

Fig.4.3 shows the architecture diagram for the Blockchain-based crowdfunding platform.

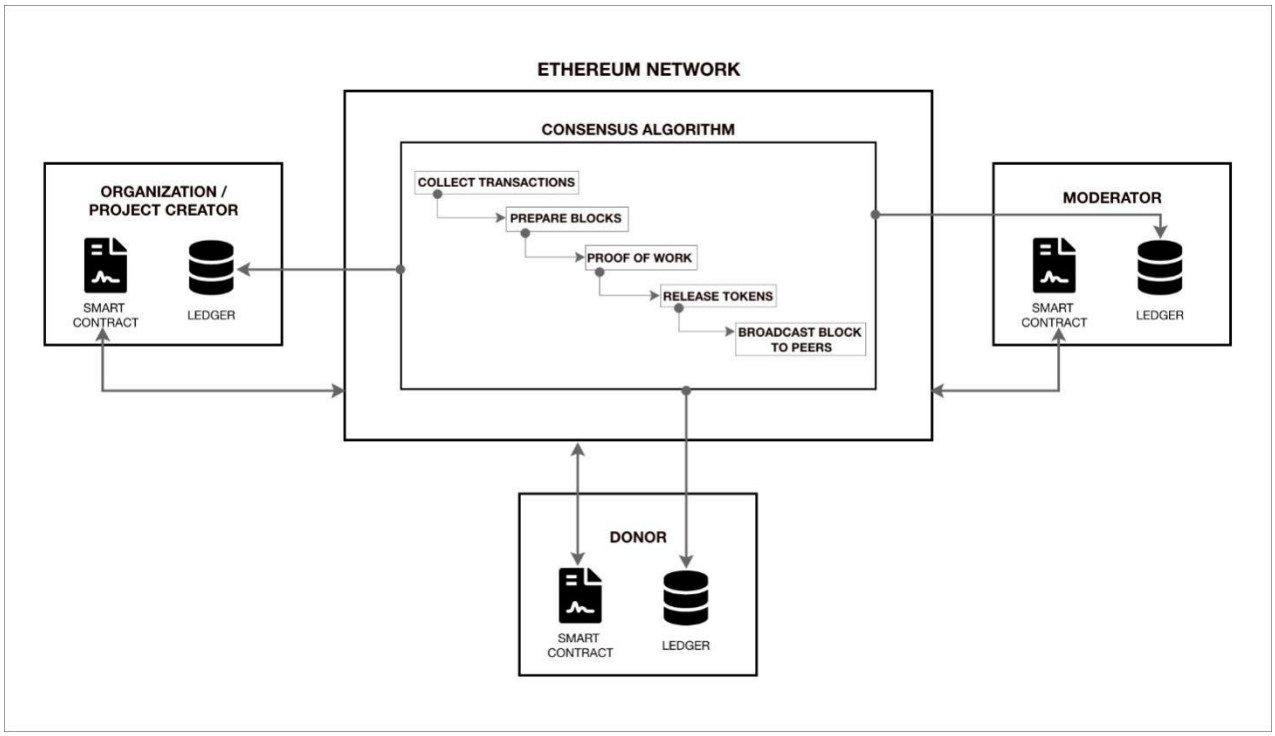

Fig 4. 3 Architecture diagram

4.4 MODULES

4.4.1 BACK-END

- **Project Creator:**

1. The Project Creator Module allows for the creation of a new project by the project creator with various specifications such as description and total donation required.
2. It enables storing and viewing of the projects that the user has created.
3. The list of projects and the project creators address are mapped to form a global index to access the projects.
4. Once a project is created, project creator index is checked for duplicate entries before adding the new project creator address.
5. The project creator can make withdrawals of the donated funds by submitting proof which should be accessible by the Moderator module for validation.
6. All the projects created by the user are stored in the blockchain along with the details of every transaction done in each project.
7. A reputation token when distributed after a validated transaction, is assigned to the address of the project creator pertaining to the specific project.
8. The balances of the project's fund are updated after each transaction.
9. The funds of the project can be viewed in ether value.

- **Donor:**

1. All the projects open for funding are accessible from the global index.
2. The project address is obtained from the Project Creator module, to which the donations can be made.
3. If an invalid amount is donated or if the donation limit is crossed, a message is thrown saying 'Limit crossed/Invalid Amount'.
4. The donated funds are verified and once that is done, the values are updated.

- **Moderator:**

1. The moderator verifies the transaction between the Project Creator and the Donor and so the module is functional once the transaction is started.
2. The module decides which moderator approves the transaction, randomly in the network.
3. The module ensures that the moderator does not validate his own transactions.
4. The reputation tokens are distributed for the owner of project upon successful approval of the transaction by the moderator.

5. For every moderated transaction, irrespective of approval or rejection, the moderator gets a moderation token.

4.4.2 FRONTEND

After writing the smart contract and deploying it in the local test blockchain, the functions of the smart contract have to be tested via the console. For ease of use a client-side user interface is created. Using JavaScript, the user interface interacts with the Ethereum blockchain. Node.js, a JavaScript runtime environment is used to execute JavaScript code outside of a browser. It can retrieve user accounts, send transactions, interact with smart contracts, and more. The website is designed using HTML and CSS.

CHAPTER 5
SYSTEM IMPLEMENTATION

This chapter gives a brief explanation of the implementation of the crowdfunding system using blockchain.

5.1 CREATING THE SMART CONTRACTS

The smart contracts were coded using solidity which helps create complicated contracts in a clearly defined and coded way. A contract in the sense of Solidity is a collection of code (its *functions*) and data (its *state*) that resides at a specific address on the Ethereum blockchain.

5.1.1 Project creator

- Declare the variables necessary for storing the details of the project and project creator

```
struct Project{
        string name;
        string description;
        uint limit;
        address admin;
        uint donatedValue;
        mapping ( uint=> Transaction) transactions;
        uint transactionSize;
        uint spentAmount;
        uint index;
    }
 struct Transaction{
        Project p;
        string purpose;
        uint value;
        string proofLink;
        address admin;
        uint index;
        bool verified;
        uint transactionIndex;
    }
struct globalIndexToAdmin{
        address admin;
        uint adminIndex;
    }
```

- Functions in the project creator smart contract:
  - showBalance():
    - To view the balance of the project.
  - showEtherBalance():
    - To view the balance of the project in ether value.
  - createProject():
    - To create a project with a name, description and a value
    - Here the ratio between transactionCount and reputation matters.
      - If the ratio is too low, the project will not be created.
    - The project is added to a list of projects mapped for the creator's address
    - Add project to project map with admin address and his project count
  - viewProject():
    - To view the details of a particular project
  - returnProjectCountForCreator():
    - Returns the number of projects created under a creator address
  - viewProjects():
    - To view the details of a the list of projects under a particular creator
  - viewAllProjectAdmins():
    - Returns a list of all the owners of a project.
  - submitProof():
    - To check whether an admin is able to submit proof.
    - Checks whether the user is an admin
    - Adds transaction link to project
    - Adds transaction to the transactions queue
  - requestedRandomTransactionFromMod():
    - Takes the first pending transaction from the queue and returns it.
  - Withdraw():
    - To withdraw the amount based on reputation or based on Ratio of transactionCount and reputation.
  - checkLimit():
    - To find whether the amount being withdrawn or donated is within the limit.
  - viewReputationTokenBalance():
    - Returns the balance of the reputation toke

- donatedAmount():
  - Updates the donated value and the balance.
- viewDonatedAmount():
  - Returns the value donated for a particular project.
- viewSpentAmount():
  - Returns the value spent by the project creator on a particular project.
- makeTransactionValid():
  - Verifies a single transaction.
- viewAllTransactionsForProject():
  - Verifies all the transactions for a particular project.
- viewSingleTransaction():
  - Returns the details of a particular transaction.

5.1.2 Donor

- Declare the variables necessary for storing the details of the donor.

```
address payable public donor;
ProjectCreator p;
address payable projectCreatorAdd;
address public temp;
// Get project creator address and initialise projectcreator contract
constructor(address payable _pCreatorToken) public{
projectCreatorAdd = _pCreatorToken;
p = ProjectCreator(projectCreatorAdd);
temp = projectCreatorAdd;}
```

- Functions in the donor smart contract
  - donate()
    - Donates money to a particular project
    - Does not allow self donations
    - Calls functions in ProjectCreator contract to update values
    - If the donation amount limit is exceeded it displays a message and reverts the transactions.

5.1.3 Moderator

- Declare the variables necessary for storing the details of the moderator.

```
struct Transaction{
string purpose;
    uint value;
    string proofLink;
    address admin;
    uint index;
    bool verified;
    uint transactionIndex;
    bool initialised;
  }
  Transaction t;
  constructor(address _modToken, address _repToken, address payable _pCreator)
public{
    mod = ModeratorToken(_modToken);
    rep = ReputationToken(_repToken);
    project = ProjectCreator(_pCreator);
  }
```

- Several functions are implemented to cater to all possible activities that can be done by the moderator.
  - validateTransaction():
    - Allows moderator to validate the transaction.
    - Does not allow the validation of own transaction.
    - Increases the moderator tokens and increases the reputation tokens for the owner of project(admin of the project/project creator)
  - rejectTransaction():
    - Allows moderator to reject the transaction.
    - Increases the moderator tokens of the moderator.
  - viewModeratorTokenBalance():
    - Allows the moderator to view their own moderator tokens.

5.1.4 Moderator token

- Declare the moderator token as an ERC20 token.

```
contract ModeratorToken is ERC20{
  address owner;
  //minting reputation token with name 'ModeratorToken' and Symbol 'MOD'
```

```
constructor() ERC20("ModeratorToken","MOD") public {
    _mint(msg.sender, 10000 * 10 ** uint(decimals()));
    owner = msg.sender;
}
```

- Functions in the moderator token smart contract:
    - approveContract():
        - Approves the transfer of moderator token to a certain recipient address
    - returnOwner():
        - returns address of the owner of the moderator token.

5.1.5 Reputation tokens

- Declare the reputation token as an ERC20 token.

```
contract ReputationToken is ERC20{
    address owner;
    //minting reputation token with name 'Reputation' and Symbol 'REP'
    constructor() ERC20("Reputation","REP") public {
        _mint(msg.sender, 100000 * 10 ** uint(decimals()));
        owner = msg.sender;


    }
```

- Functions in the reputation token smart contract:
    - approveContract():
        - Approves the transfer of reputation token to a certain recipient address
    - returnOwner():
        - returns address of the owner of the reputation token.

5.2 RUNNING THE MIGRATIONS

Migrations are JavaScript files that help to deploy contracts to the Ethereum network. These files are responsible for staging the deployment tasks, and they're written under the assumption that the deployment needs will change over time. A history of previously run migrations is recorded on-chain through a special migrations contract.

```
const Migrations = artifacts.require("Migrations");
module.exports = function (deployer) {
  deployer.deploy(Migrations);
};
```

The migration files will use the deployer to stage deployment tasks. The deployment tasks can be written synchronously and they'll be executed in the correct order.

```
// Importing Contracts
const ReputationToken = artifacts.require("ReputationToken");
const ModeratorToken = artifacts.require("ModeratorToken");
const ProjectCreator = artifacts.require("ProjectCreator");
const Moderator = artifacts.require("Moderator");
const Donor = artifacts.require("Donor");
module.exports = function(deployer) {
  deployer.then(
    // SO that one happens after the other in the required order of initialisation
    async () => {
    // address of the REP token contract
    const repTokenInst = await deployer.deploy(ReputationToken);
    console.log('ReputationToken contract deployed at', repTokenInst.address);
    const modTokenInst = await deployer.deploy(ModeratorToken);
    console.log('ModeratorToken contract deployed at', modTokenInst.address);
    const projInst = await deployer.deploy(ProjectCreator, repTokenInst.address);
    console.log('ProjCreator contract deployed at', projInst.address);
    const modInst = await deployer.deploy(Moderator, modTokenInst.address,
repTokenInst.address, projInst.address);
    console.log ('Moderator contract deployed at', modInst.address);
    // Allowing the usage of tokens
    await repTokenInst.approveContract(modInst.address,100000);
    await modTokenInst.approveContract(modInst.address,100000);
    const donorInst = await deployer.deploy(Donor, projInst.address);
    console.log('Donor contract deployed at', donorInst.address);
  })
};
```

CHAPTER 6
TESTING

This chapter describes the result of testing the modules after implementation.

6.1 UNIT TESTING

The tabulations below contain the summaries of the testing of the individual modules.

6.1.1 PROJECT CREATOR

Table 6.1 Summary of unit testing for creating a project

Test case No.	Inputs	Expected Output	Obtained Output	Test case Acceptance
1	Name == Empty String	Invalid input	Invalid input	Yes
2	Name == Not Empty String	Project Created	Project Created	Yes
3	Description == Empty String	Invalid input	Invalid input	Yes
4	Description == Not Empty String	Project Created	Project Created	Yes
5	Value == String	Invalid input	Invalid input	Yes
6	Value == Valid Number	Project Created	Project Created	Yes
7	Ratio <= 2	Reputation token too low to create project	Creation failed	Yes
8	Ratio >2	Project Created	Project Created	Yes

Table 6.2 Summary of unit testing for viewing a project

Test case No.	Inputs	Expected Output	Obtained Output	Test case Acceptance
1	Admin address == not valid hex Ethereum address	Invalid input	Invalid input	Yes
2	Admin address == valid hex Ethereum address	Project details	Project Details	Yes
3	Index < 0	Invalid input	Invalid input	Yes
4	Index >=0	Project Details	Project Details	Yes

Table 6.3 Summary of unit testing for viewing all projects of admin

Test case No.	Inputs	Expected Output	Obtained Output	Test case Acceptance
1	Admin address == not valid hex Ethereum address	Invalid input	Invalid input	Yes
2	Admin address == valid hex Ethereum address	List of Projects of admin	List of Projects of admin	Yes

Table 6.4 Summary of unit testing for submitting proof for a transaction

Test case No.	Inputs	Expected Output	Obtained Output	Test case Acceptance
1	Admin address == not valid hex Ethereum address	Invalid input	Invalid input	Yes
2	Admin address == not valid hex Ethereum address	Transaction added to queue	Transaction added to queue	Yes
3	Index < 0	Invalid input	Invalid input	Yes
4	Index >=0	Transaction added to queue	Transaction added to queue	Yes
5	Transaction Index < 0	Invalid input	Invalid input	No

6	Transaction Index >=0	Transaction added to queue	Transaction added to queue	Yes

Table 6.5 Summary of unit testing withdrawing funds

Test case No.	Inputs	Expected Output	Obtained Output	Test case Acceptance
1	Amount < 0	Invalid input	Invalid input	Yes
2	Amount >=0	Funds withdrawn successfully	Funds withdrawn successfully	Yes
3	Index < 0	Invalid input	Invalid input	Yes
4	Index >=0	Funds withdrawn successfully	Funds withdrawn successfully	Yes
5	Ratio <= 2	Reputation tokens too low to create withdraw funds	Withdraw failed	Yes
6	Ratio >2	Funds withdrawn successfully	Funds withdrawn successfully	Yes

Table 6.6 Summary of unit testing for viewing all transactions of project

Test case No.	Inputs	Expected Output	Obtained Output	Test case Acceptance
1	Admin address == not valid hex Ethereum address	Invalid input	Invalid input	Yes
2	Admin address == valid hex Ethereum address	Project Transactions	Project Transactions	Yes
3	Index < 0	Invalid input	Invalid input	Yes
4	Index >=0	Project Transactions	Project Transactions	Yes

Table 6.7 Summary of unit testing for viewing a transaction of a project

Test case No.	Inputs	Expected Output	Obtained Output	Test case Acceptance
1	Admin address == not valid hex Ethereum address	Invalid input	Invalid input	Yes
2	Admin address == valid hex Ethereum address	Project Transaction	Project Transaction	Yes
3	Index < 0	Invalid input	Invalid input	Yes
4	Index >=0	Project Transaction	Project Transaction	Yes
5	Transaction Index < 0	Invalid input	Invalid input	Yes
6	Transaction Index >=0	Project Transaction	Project Transaction	Yes

6.1.2 DONOR

Table 6.8 Summary of unit testing for donating to a project

Test case No.	Inputs	Expected Output	Obtained Output	Test case Acceptance
1	Admin address == not valid hex Ethereum address	Invalid input	Invalid input	Yes
2	Admin address == valid hex Ethereum address	Donation successs	Donation successs	Yes
3	Index < 0	Invalid input	Invalid input	Yes
4	Index >=0	Project Transactions	Project Transactions	Yes
5	Amount <= 0	Invalid input	Invalid input	Yes
6	Amount >0	Donation successs	Donation successs	Yes

7	Donor != admin	Donation successs	Donation successs	Yes
8	Donor == Admin	Cannot donate to self	Cannot donate to self	Yes

6.1.3 MODERATOR

Table 6.9 Summary of unit testing for validating a transaction

Test case No.	Inputs	Expected Output	Obtained Output	Test case Acceptance
1	Moderator address != Admin	Cannot initialise own transaction	Cannot initialise own transaction	Yes
2	Moderator address == Admin	Transaction Validated and one reputation token earned by Project Creator and one moderator Token earned by Moderator	Transaction Validated and one reputation token earned by Project Creator and one moderator Token earned by Moderator	Yes
3	Index < 0	Invalid input	Invalid input	Yes
4	Index >=0	Transaction Validated and one reputation token earned by Project Creator and one moderator Token earned by Moderator	Transaction Validated and one reputation token earned by Project Creator and one moderator Token earned by Moderator	Yes

Table 6.10 Summary of unit testing for rejecting a transaction

Test case No.	Inputs	Expected Output	Obtained Output	Test case Acceptance
1	Moderator address != Admin	Cannot initialise own transaction	Cannot initialise own transaction	Yes
2	Moderator address == Admin	Transaction Rejected and one moderator Token earned by Moderator	Transaction Rejected and one moderator Token earned by Moderator	Yes
3	Index < 0	Invalid input	Invalid input	Yes

4	Index >=0	Transaction Rejected and one moderator Token earned by Moderator	Transaction Rejected and one moderator Token earned by Moderator	Yes

INTEGRATION TESTING

The tabulations below contain the testing of a combination of modules.

6.2.1 PRIMARY SCREEN

Table 6.11 Summary of integration testing for primary screen of the Web App

Test case No.	Inputs	Expected Output	Obtained Output	Test case Acceptance
1	Automatic prompt to connect Metamask and application	Connects successfully	Connected successfully	Yes
2	No automatic prompt to connect Metamask and application	Metamask Not installed in Browser	Metamask Not installed in Browser	Yes
3	Click on Project Creator	Web app shows project creator tab	Web app shows project creator tab	Yes
4	Click on Donor	Web app shows donor tab	Web app shows donor tab	Yes
5	Click on Moderator	Web app shows moderator tab	Web app shows moderator tab	Yes

6.2.2 PROJECT CREATOR SCREEN

Table 6.12 Summary of integration testing for the Project Creator Screen

Test case No.	Inputs	Expected Output	Obtained Output	Test case Acceptance
1	Click on Create a Project	Three input fields and a button show	Three input fields and a button show	Yes
2	Click on View All Projects	Two input fields and details of project shows	Two input fields and details of project shows	Yes
3	Click on View All Admins	One input fields and address of all admins show	One input fields and address of all admins show	Yes

4	Click on Withdraw from a project	Three input fields and a button show	Three input fields and a button show	Yes
5	Click on view all transactions	Three input fields and details of transaction show	Three input fields and details of transaction show	Yes
6	Click on submit proof	Three input fields and a button show	Three input fields and a button show	Yes
7	Click on view reputation tokens balance	Shows reputation token balance	Shows reputation token balance	Yes

6.2.3 DONOR SCREEN

Table 6.13 Summary of integration testing for the Donor Screen

Test case No.	Inputs	Expected Output	Obtained Output	Test case Acceptance
1	Click on View All Projects	Two input fields and details of project shows	Two input fields and details of project shows	Yes
2	Click on View All Admins	One input fields and address of all admins show	One input fields and address of all admins show	Yes
3	Click on Donate	Value input and project list will show	Value input and project list will show	Yes

6.2.4 MODERATOR SCREEN

Table 6.14 Summary of integration testing for the Moderator Screen

Test case No.	Inputs	Expected Output	Obtained Output	Test case Acceptance
1	Click on view Moderator tokens balance	Shows moderator token balance	Shows moderator token balance	Yes
2	Click on Validate Transaction	Details of a transaction show along with a field to enter if validated or not	Details of a transaction show along with a field to enter if validated or not	Yes

CHAPTER 7
RESULTS

This chapter contains a summary of the results achieved while implementing the system.

7.1 BACKEND SETUP

Figures 7.1,7.2,7.3 and 7.4 depict the compilation and migration of the smart contracts in the backend.

```
harry@Harrys-MacBook HelpAnybody-Ethereum-CrowdFunding-main % truffle migrate --reset

Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.


Starting migrations...
======================
> Network name:    'development'
> Network id:      5777
> Block gas limit: 6721975 (0x6691b7)


1_initial_migration.js
======================

   Replacing 'Migrations'
   ----------------------
   > transaction hash:    0xe932793666f0dd9221e64056a242c83d6256d36ec035ef86b8949d761d0483c3
   > Blocks: 0            Seconds: 0
   > contract address:    0x7C09042bf33C58551bb81Ed078bF24846960e9d0
   > block number:        1
   > block timestamp:     1616424289
   > account:             0x2f1Bf80F310C289fe72a3c34d04742d9cA7D6252
   > balance:             99.99692588
   > gas used:            153706 (0x2586a)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.00307412 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.00307412 ETH
```

Fig 7. 1:Initial migration

```
3_deploy_projects.js
====================

   Replacing 'ReputationToken'
   ---------------------------
   > transaction hash:    0x89d0da5b6973ec23e02b63fe29f46cc4a4ce8e7a040131f6e51533e0756aa1a0
   > Blocks: 0            Seconds: 0
   > contract address:    0x63dE166e8A1a7Ab2E396a258178d6Ae6e8C57892
   > block number:        3
   > block timestamp:     1616424291
   > account:             0x2f1Bf80F310C289fe72a3c34d04742d9cA7D6252
   > balance:             99.9808169
   > gas used:            763194 (0xba53a)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.01526388 ETH

ReputationToken contract deployed at 0x63dE166e8A1a7Ab2E396a258178d6Ae6e8C57892

   Replacing 'ModeratorToken'
   --------------------------
   > transaction hash:    0x3174dcf53205767550d5d714ae24bb9a0aa55d2cafcb9a4a907820d469897aa9
   > Blocks: 0            Seconds: 0
   > contract address:    0x868f2074498248f4ad8F79d63cCcCDDbBb2AEdF5
   > block number:        4
   > block timestamp:     1616424292
   > account:             0x2f1Bf80F310C289fe72a3c34d04742d9cA7D6252
   > balance:             99.96554864
   > gas used:            763413 (0xba615)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.01526826 ETH

ModeratorToken contract deployed at 0x868f2074498248f4ad8F79d63cCcCDDbBb2AEdF5
```

Fig 7. 2 Smart contract deployment

```
   Replacing 'Donor'
   -----------------
   > transaction hash:    0xdf88af2032b4d4881a8573848b84b9d26df0384573a1c362d6b1c226f2cb8481
   > Blocks: 0            Seconds: 0
   > contract address:    0xC4407c4cdF60a56FeDf086bC0364140cd3A1D977
   > block number:        9
   > block timestamp:     1616424295
   > account:             0x2f1Bf80F310C289fe72a3c34d04742d9cA7D6252
   > balance:             99.88125092
   > gas used:            354075 (0x5671b)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.0070815 ETH

Donor contract deployed at 0xC4407c4cdF60a56FeDf086bC0364140cd3A1D977

   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.11306894 ETH


Summary
=======
> Total deployments:   6
> Final cost:          0.11614306 ETH
```

Fig 7. 3 Migration summary

```
harry@Harrys-MacBook HelpAnybody-Ethereum-CrowdFunding-main % truffle test
Using network 'development'.


Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.

ReputationToken contract deployed at 0x2Cd13eDDb95fA20f49c72e461189b2889F247a4a
ModeratorToken contract deployed at 0x34912688cC84513f0E7651b9c8cD4bee7189b2C5
ProjCreator contract deployed at 0x958BdE7074601451FF34a1F534C70B0C2206A22D
Moderator contract deployed at 0x11ce5daF85A35131239FBB7b605470a54BCA1678
Donor contract deployed at 0x879A064e901e899A5c1365C278566e3BD76d4593


  Contract: ProjectCreator
    ✓ Create and View a Project (1118ms)
    ✓ Donating to a project (489ms)
    ✓ Invalid Donation Test Case (1801ms)
    ✓ Withdrawing Test Case (1711ms)
    ✓ Submitting Proof Test (1515ms)
    ✓ Getting a transaction for a Moderator (659ms)
    ✓ Validating a Transaction in Moderator (891ms)
    ✓ Rejecting a Transaction in Moderator (1028ms)


  8 passing (11s)
```

Fig 7. 4 Testing of smart contracts

7.2 FRONTEND

Figure 7.5 depicts the home page of the application.

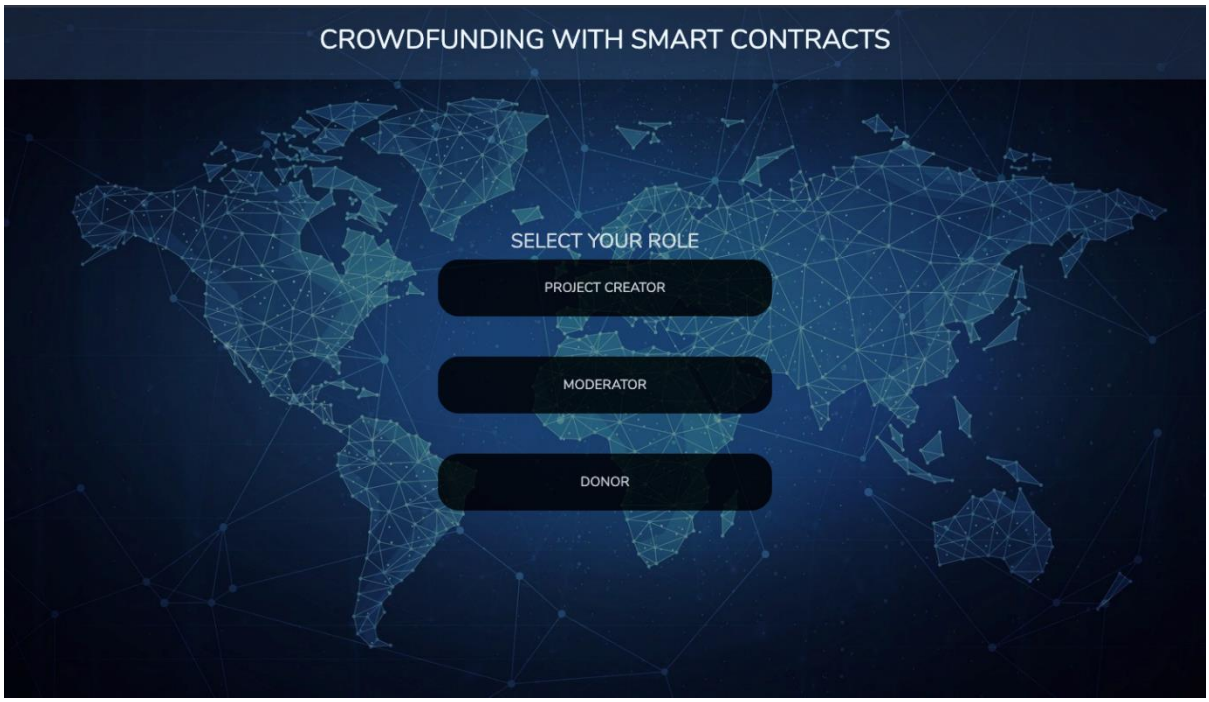

Fig 7. 5 Home page of the application

Figure 7.6 depicts the project creator's home page of the application.

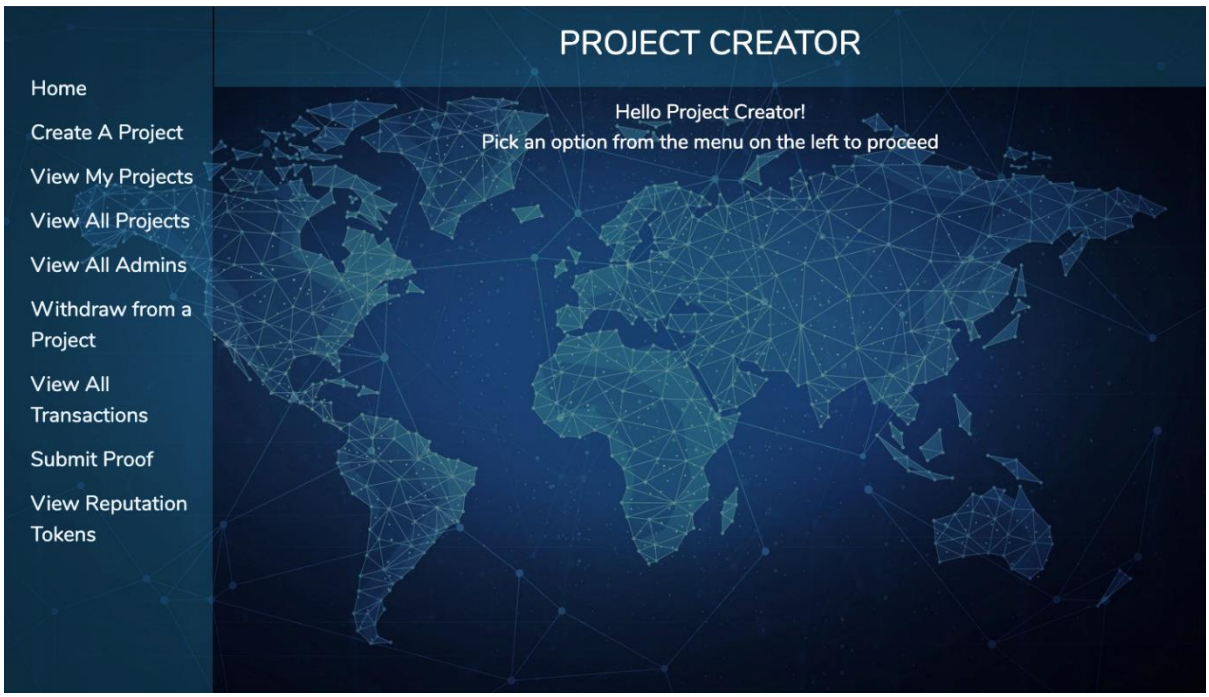

Fig 7. 6 Project creator's home page

Figure 7.7 depicts the moderator‘s home page of the application.

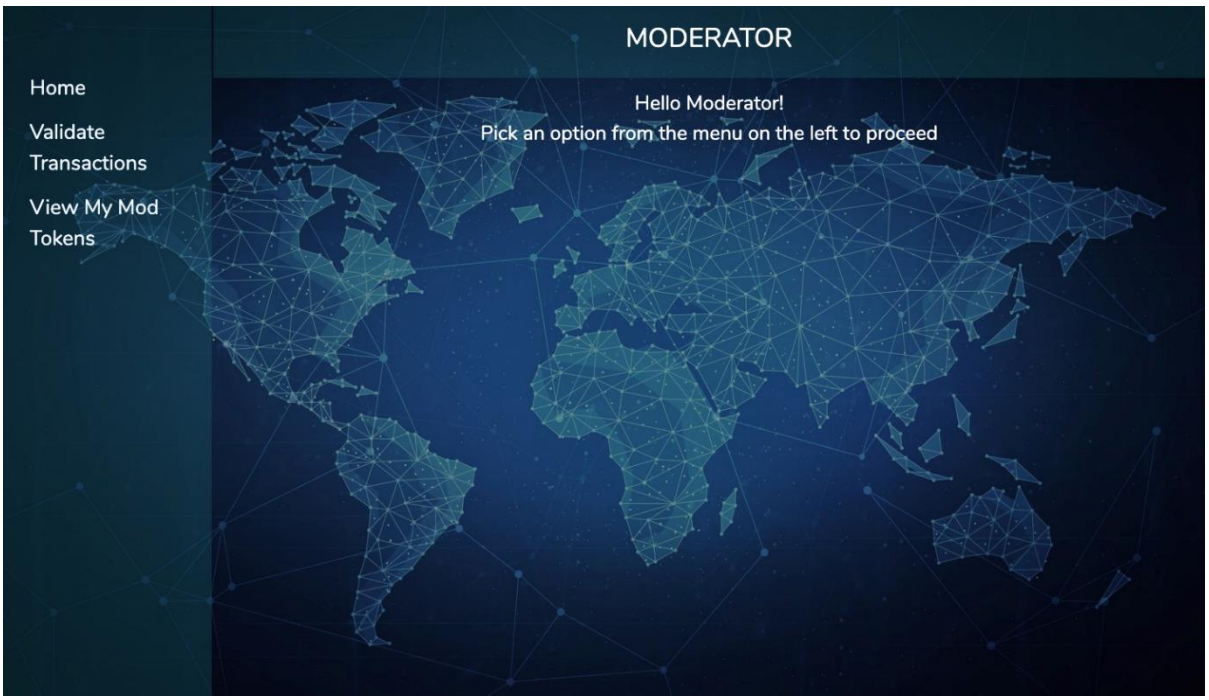

Fig 7. 7 Moderator’s home page

Figure 7.8 depicts the donor‘s home page of the application.

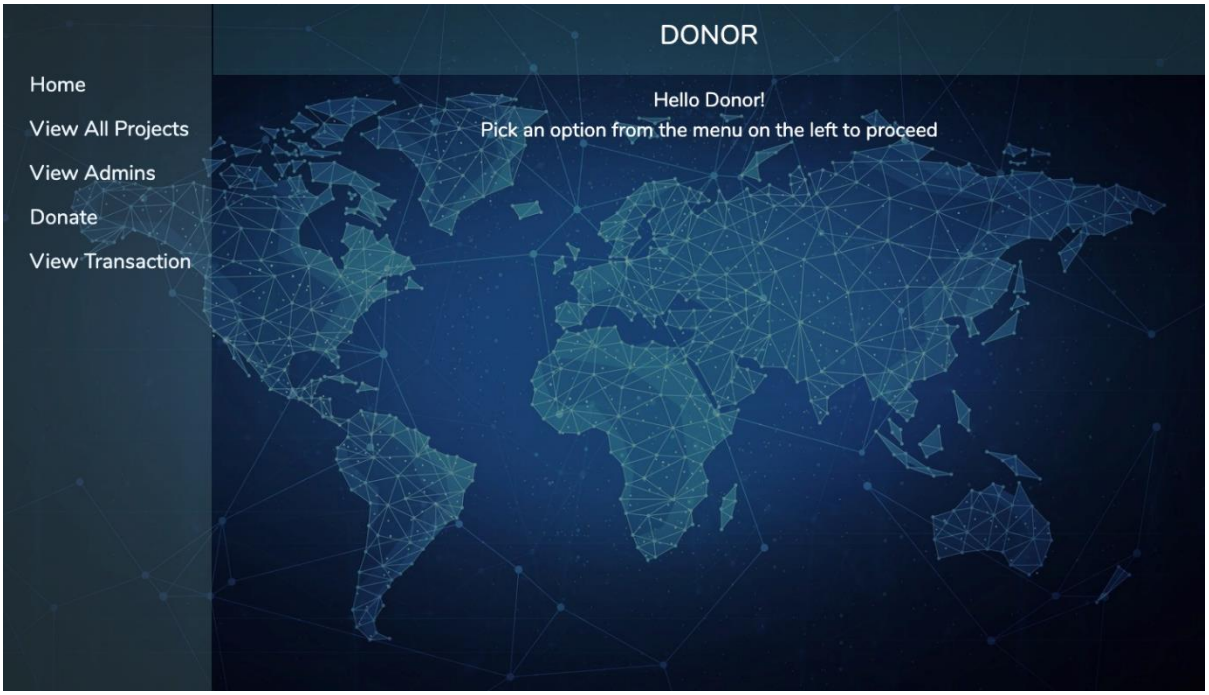

Fig 7. 8 Donor’s home page

Figure 7.9 depicts the form for project creation by the project creator.

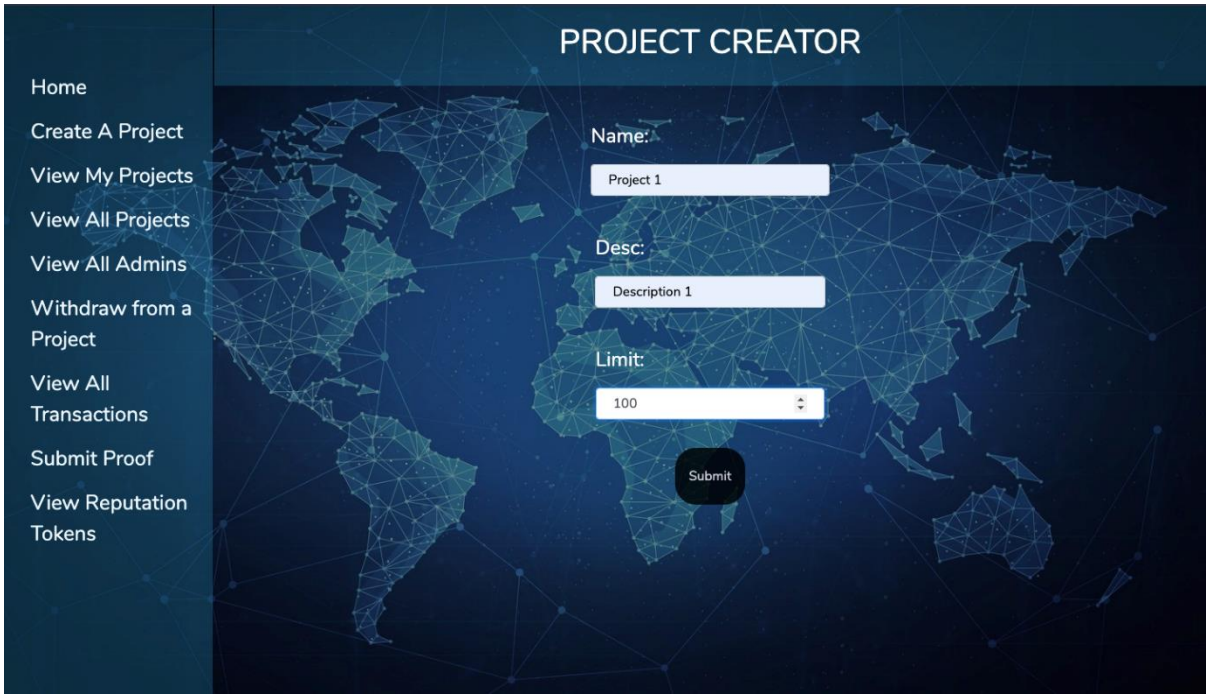

Fig 7. 9 Project creation

Figure 7.10 depicts the page to view the all the projects created in the network.

Fig 7. 10 Viewing all projects

Figure 7.11 depicts the page to view the projects created by the project creator..

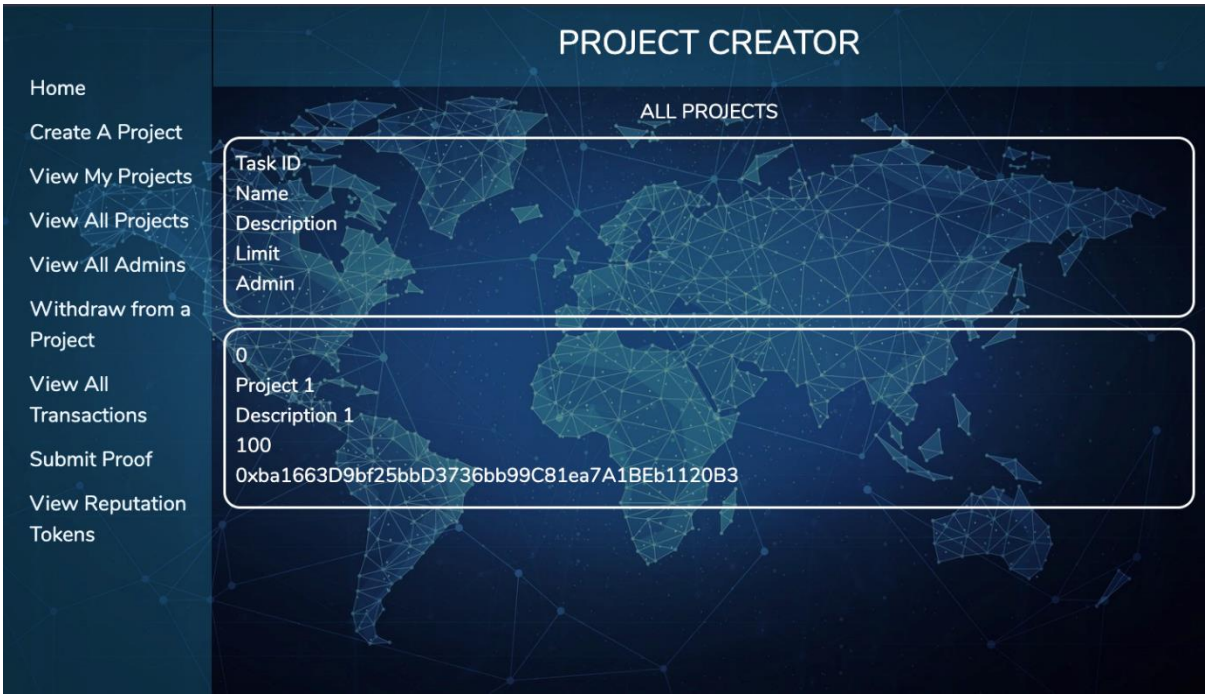

Fig 7. 11 Viewing all projects pertinent to project creator

Figure 7.12 depicts the page to view all the admins of a particular project.

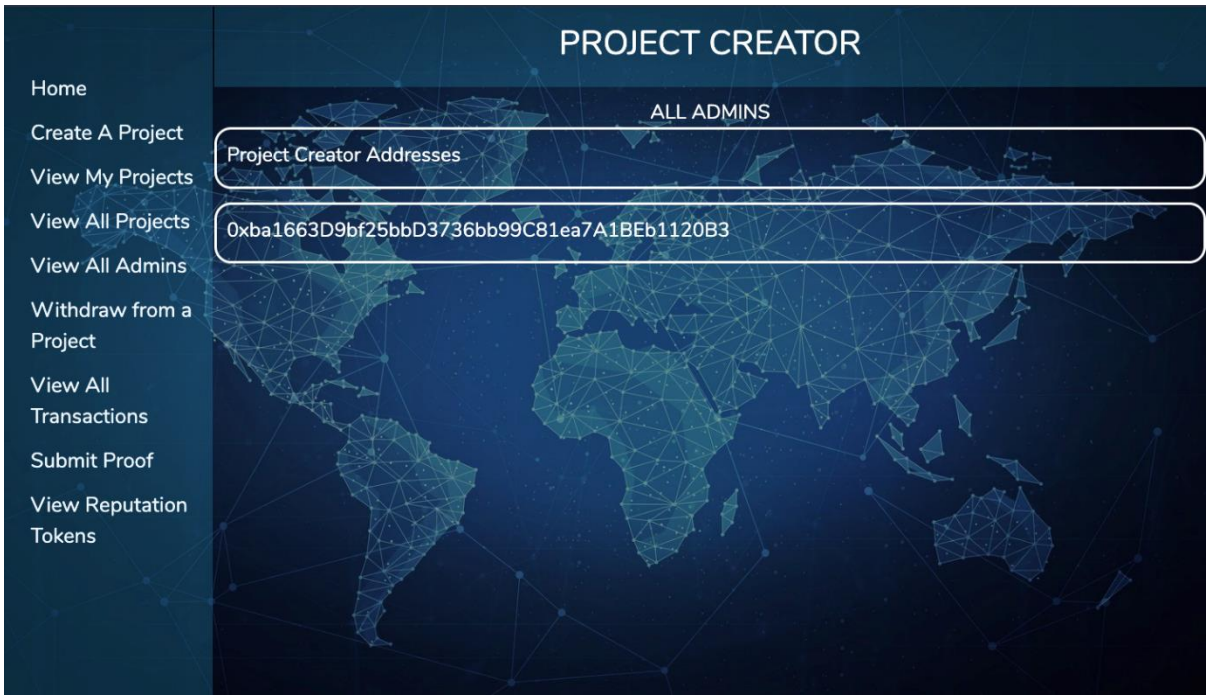

Fig 7. 12 Viewing the admins of a project

Figure 7.13 depicts the page to view using which the donor can look in to projects in the network.

Fig 7. 13 View projects-donor

Figure 7.14 depicts the page to view using which the donor can check the admin of a project in the network.

Fig 7. 14 View admin-donor

Figure 7.15 depicts the page to view using which the donor can check the transactions of a project .

Fig 7. 15 View transactions-donor

Figure 7.16 depicts the page that allows the donor to donate funds.

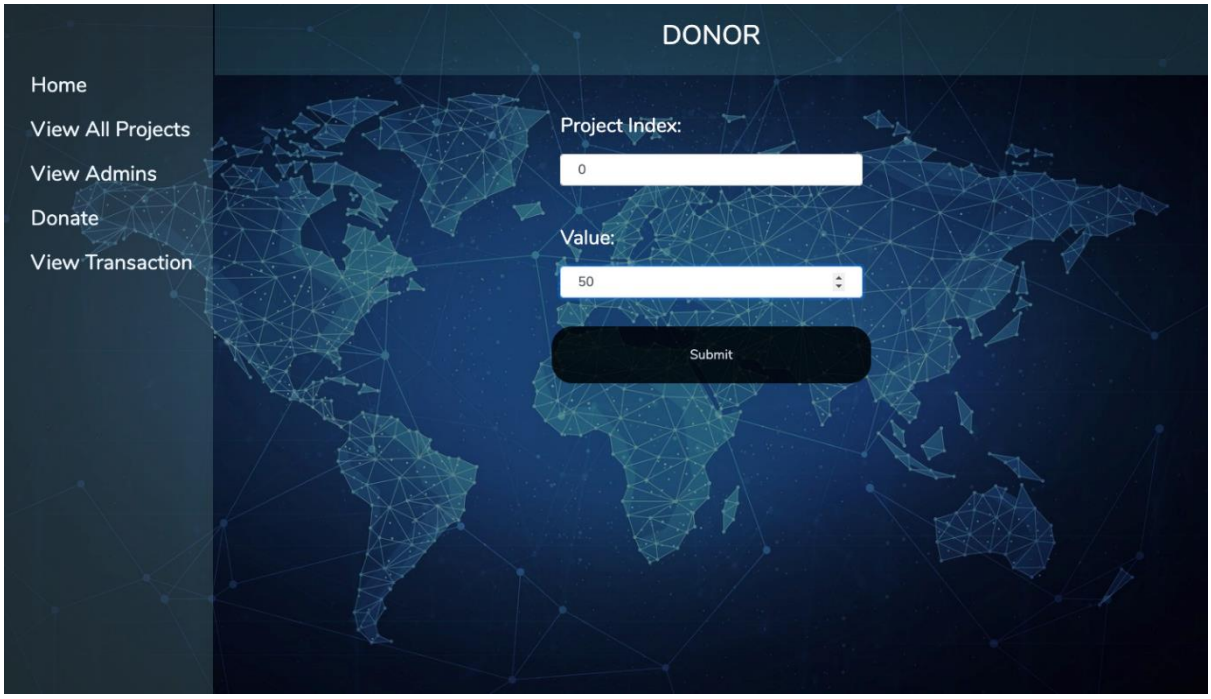

Fig 7. 16 Donate funds

Figure 7.17 depicts the page that allows the project creator to withdraw funds.

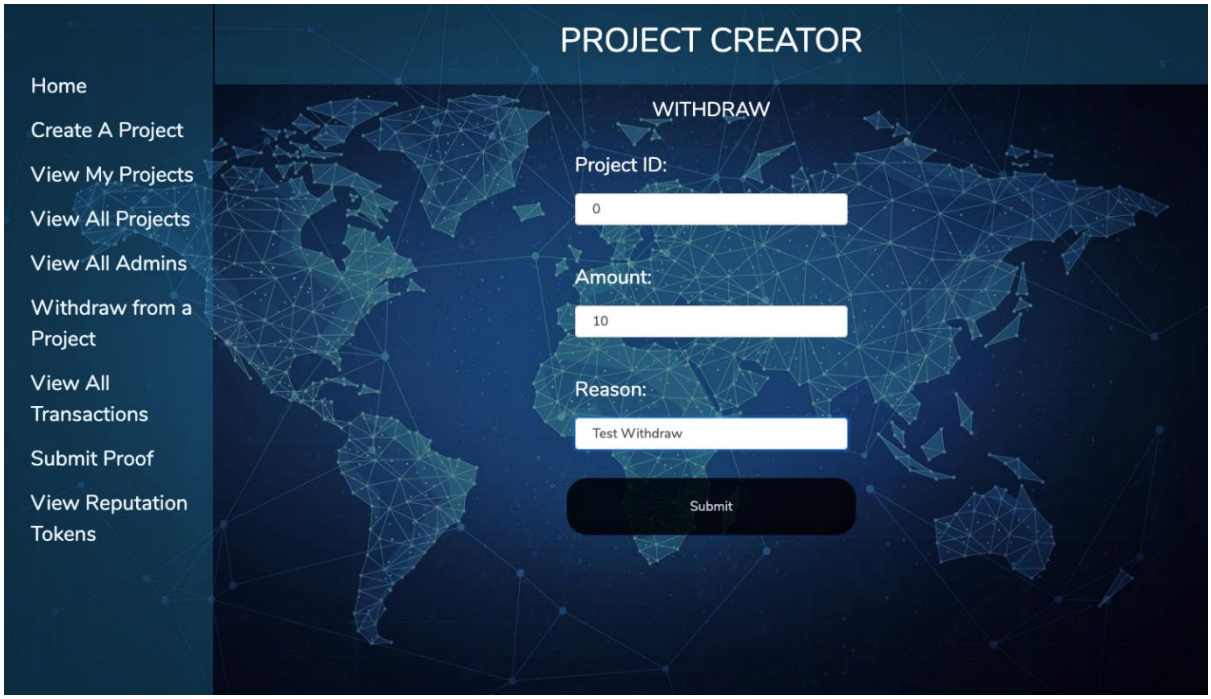

Fig 7. 17 Withdraw funds

Figure 7.18 and figure 7.19 depicts the page which allows the project creator to view the transactions of the project.

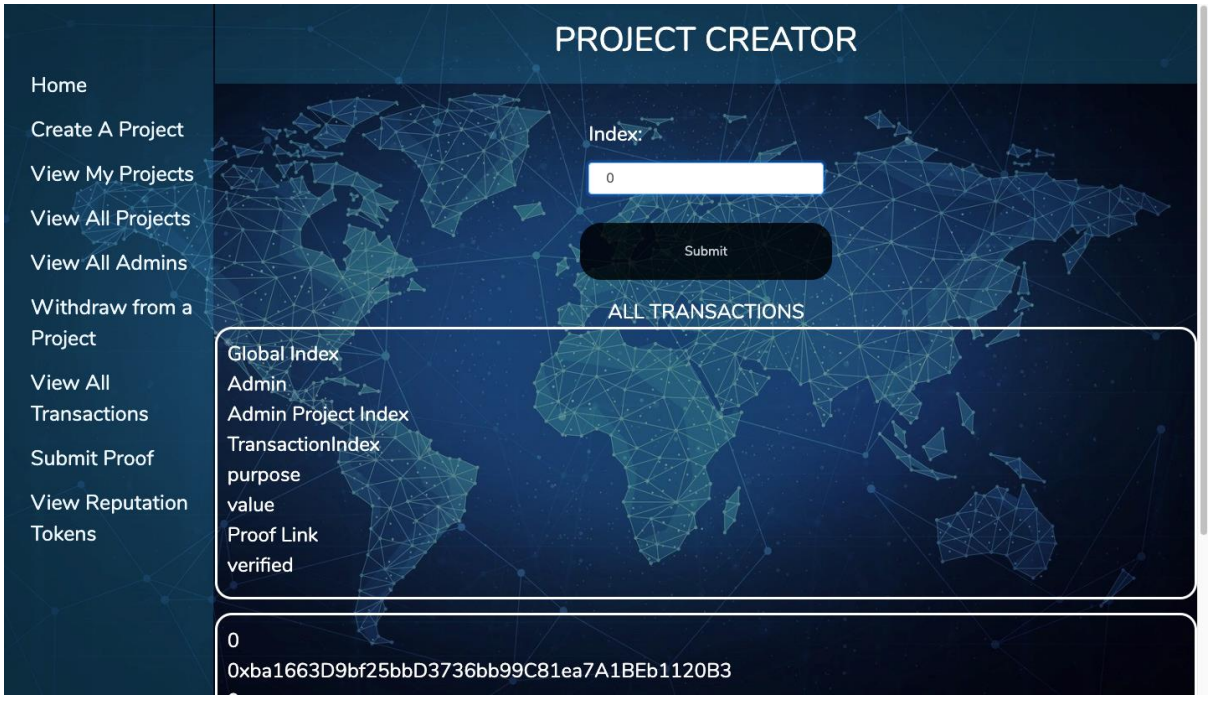

Fig 7. 18 View transactions 1-Project creator

Fig 7. 19 View transactions 2-Project creator

Figure 7.20 depicts the page that allows the project creator to submit proof for a transaction.

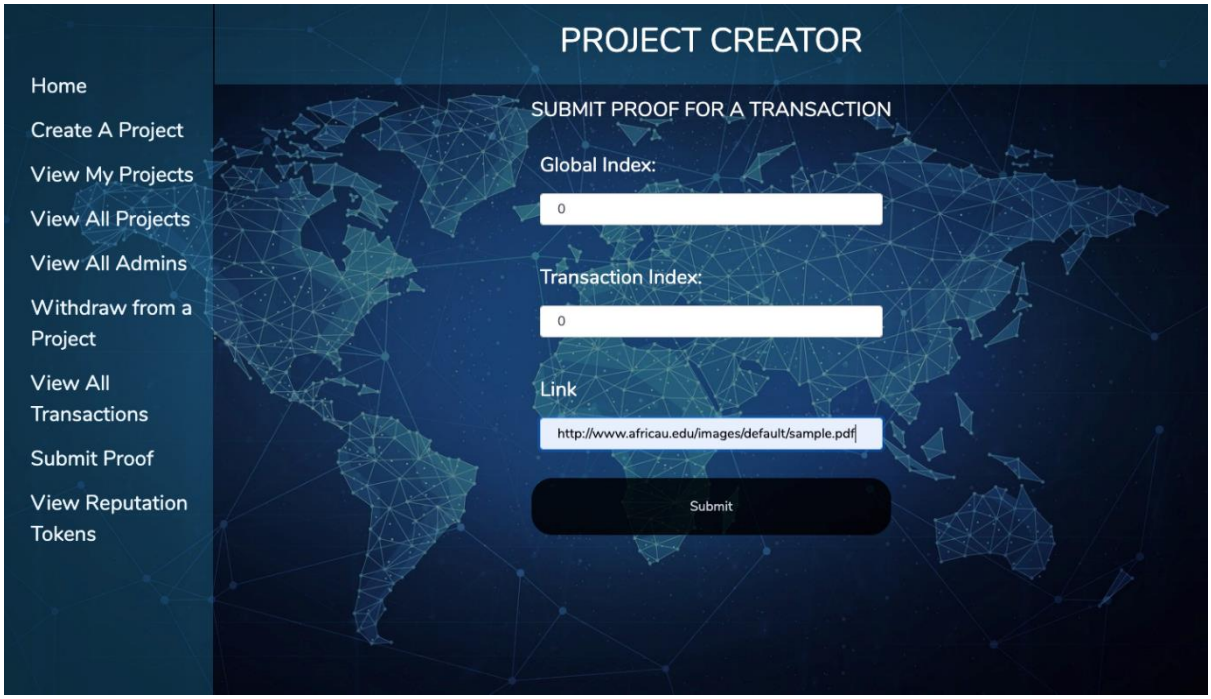

Fig 7. 20 Submit proof

Figure 7.21 depicts the page that allows the moderator to validate the transactions.

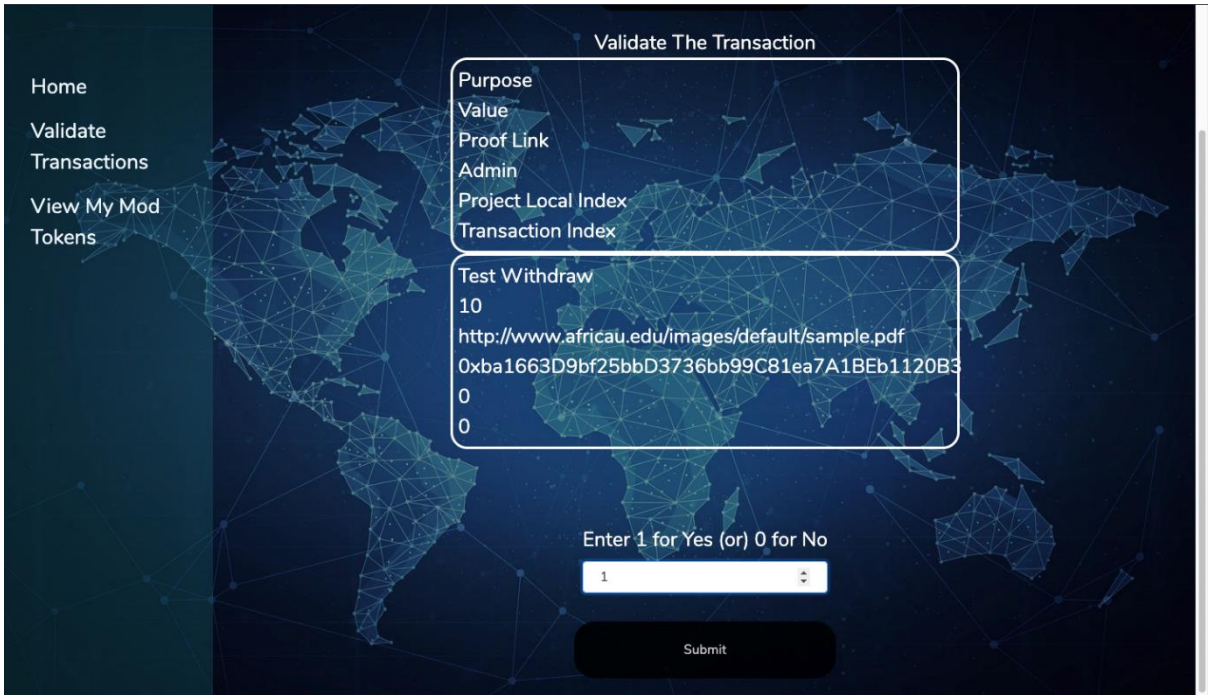

Fig 7. 21 Validate transactions

Figure 7.22 depicts the page to view the total reputation tokens held by the project creator.

Fig 7. 22 View reputation tokens

Figure 7.23 depicts the page which allows the moderator to view his/her total moderator tokens.

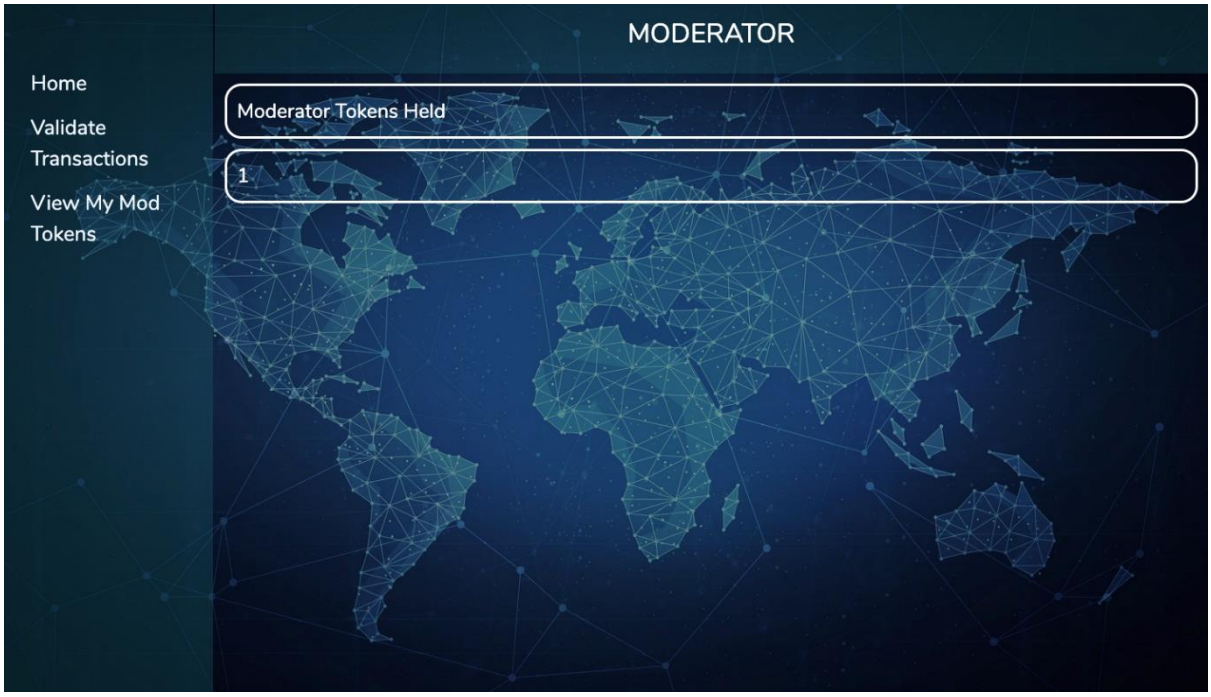

Fig 7. 23 View moderation tokens

Figure 7.24 and figure 7.25 shows proof that the transactions were sucessful and were executed on the blockchain

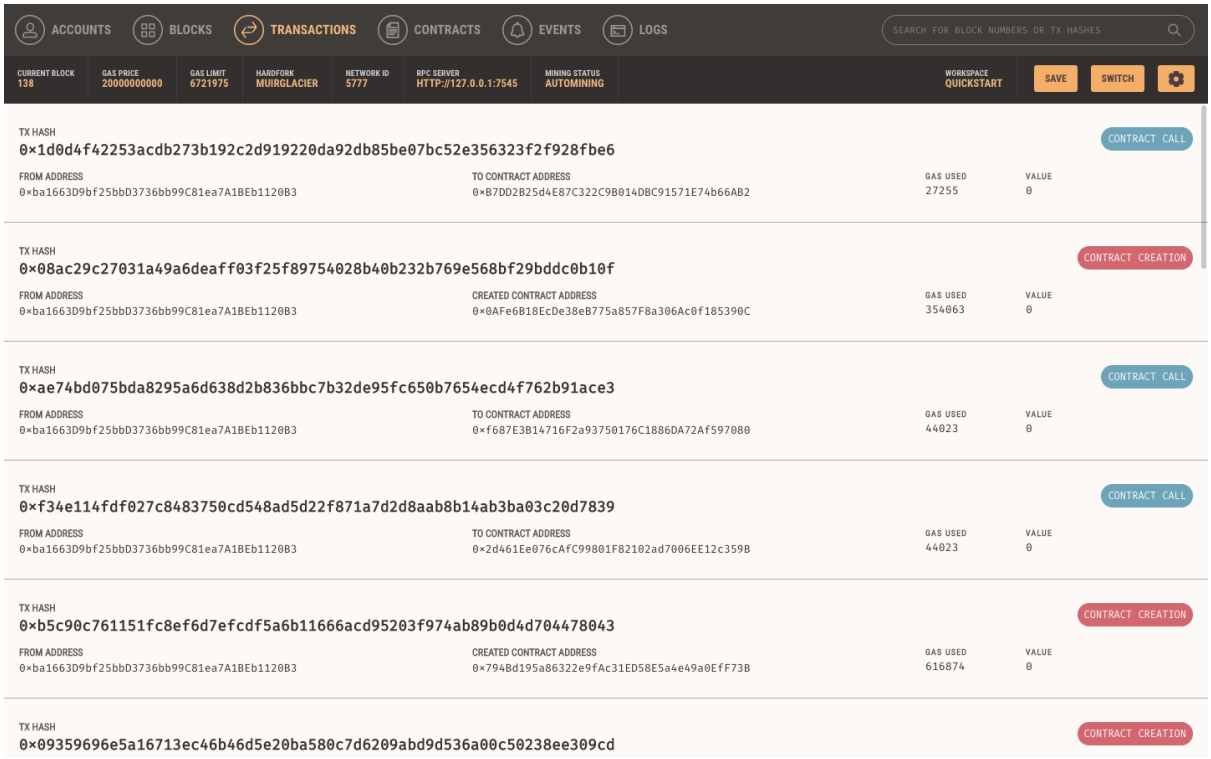

Fig 7. 24 Transactions recorded by ganache

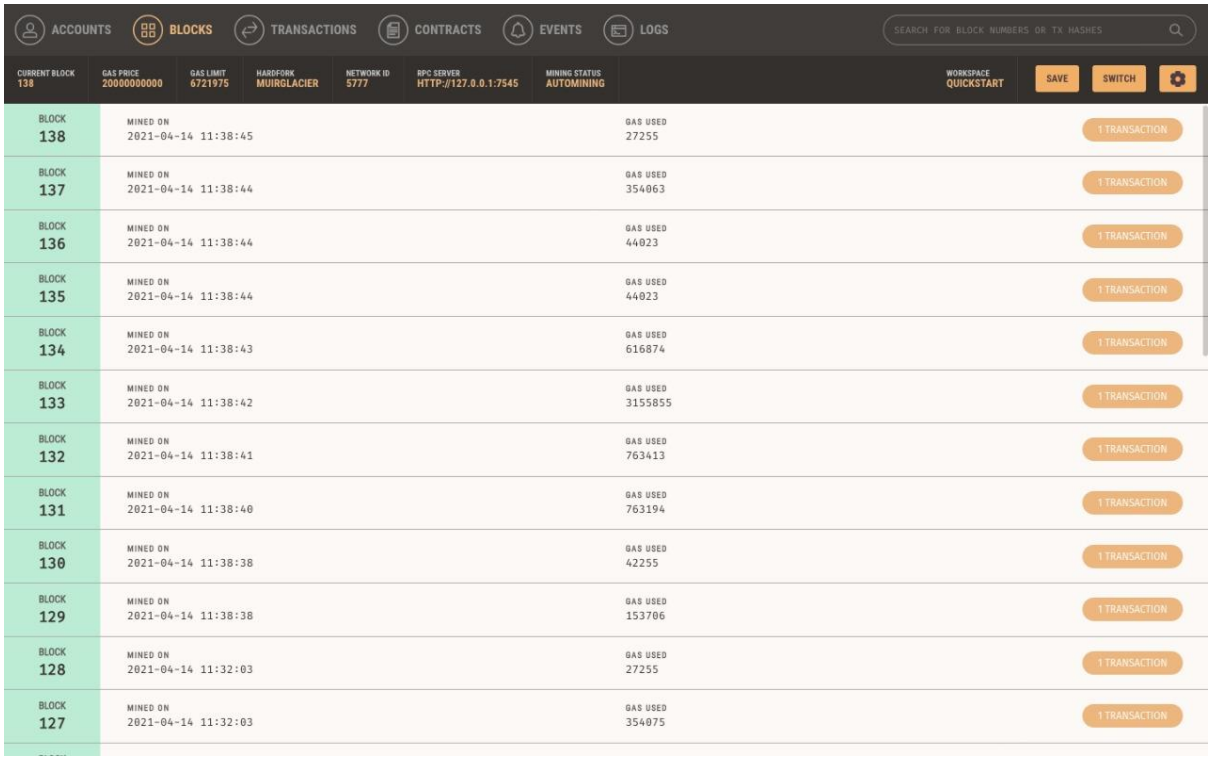

Fig 7. 25 Blocks recorded by ganache

CHAPTER 8

CONCLUSION

Crowdfunding using smart contracts intend to bring an easy and trust worthy way for Project Creators to kickstart their projects and to allow Donors to make safe donations for valuable causes. The three actors in the project namely Donor, Project Creator and Moderator all play crucial parts in maintaining the decentralised nature of the project. There is no central entity controlling funds which prevents corruption and improves efficiency. Using smart contracts, rules are enforced in the decentralised system and the smart contracts handle all the funds with code that is tested and proven to be strong against any kind of attack. This project can bring about real change in the world of non-governmental organisations and any community serving projects.

But a computer software is like fashion, it never ends. Some of the future enhancements that can be added are,

1. An incentive for the Donor. In the current version, the donor does not receive anything in return for their investment. This change can be done by introducing NFTs (Non-Fungible Tokens using ERC 721 in the Ethereum Blockchain). An NFT can be awarded each time a user makes a donation. This NFT can further carry value in the coming years that can be redeemed by the Donor for the respective fiat currency.
2. Improve coin circulation. With the current version of the project, the circulation of Moderator token and reputation token is not perfect. To bring immense value to these tokens, a burn event can be done.
3. The most common problem of blockchain based applications is the lack of ease of understanding for the layman. The interface can be made simpler and easier to understand based on the needs of the layman.
4. A better way of performing transactions. Currently Metamask is used to perform any blockchain related transactions. This can be changed to make donating easier using credit/debit card methods along with UPI.

BIBLIOGRAPHY

1. Kangana W. M, Sowmya M, Namita Chougule, Sejal Patil, Soumya Kadadi, “The Applications of Blockchain Technology of Crowdfunding using Smart Contract”, International Research Journal of Modernization in Engineering Technology and Science Volume 02, Issue 09 ,September 2020.
2. Vikas Hassija, Vinay Chamola, Sherali Zeadally, “BitFund: A Blockchain-based Crowd Funding Platform for Future Smart and Connected Nation”,CDATA [Sustainable Cities and Society], (2020)
3. Pledgecamp, “Pledgecamp the next generation of crowd-funding,” https://pledgecamp.com/, online; accessed 11 January 2020.
4. Hasnan Baber,”Blockchain-Based Crowdfunding: A ‘Pay-it-Forward’ Model of WHIRL”,International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878, Volume 8 Issue 3, September 2019.
5. “Blockchain: Smart Contracts,” GeeksforGeeks, 02-Jan-2020. [Online]. Available: https://www.geeksforgeeks.org/smart-contracts/. [Accessed: 3-Feb-2021].
6. “Introduction to Blockchain technology: Set 2,” GeeksforGeeks, 02-Dec-2019. [Online]. Available: https://www.geeksforgeeks.org/introduction-to-blockchain/. [Accessed: 3-Feb-2021].
7. “Proof of Work (PoW) Consensus,” GeeksforGeeks, 09-Jan-2019. [Online]. Available: https://www.geeksforgeeks.org/proof-of-work-pow-consensus/. [Accessed: 3-Feb-2021].
8. T. M. Team, “An overview of blockchain consensus protocols,” Medium, 05-Sep-2018. [Online]. Available: https://blog.modex.tech/an-overview-of-blockchain-consensus-protocols-9e9350b6c329. [Accessed: 3-Feb- 2021].

APPENDIX

SOLIDITY CONTRACTS

Donor.sol

```
pragma solidity >=0.4.22 <=0.7.0;
import './ReputationToken.sol';
import './ProjectCreator.sol';
contract Donor{

    address payable public donor;
    ProjectCreator p;
    address payable projectCreatorAdd;
    address public temp;
    // Get project creator address and initialise projectcreator contract
    constructor(address payable _pCreatorToken) public{
        projectCreatorAdd = _pCreatorToken;
        p = ProjectCreator(projectCreatorAdd);
        temp = projectCreatorAdd;
    }
    event DonatedMoney(address donor, uint amount,string message,address
projectAdmin, uint index);
    function donate(uint donation, address admin, uint index,string memory
message)public payable{
        donor = msg.sender;
        require(donor != admin, "Cannot donate to self");
        //Call function in ProjectCreator to update values
        if(p.checkLimit(donation,admin,index) == false){
            revert('Limt crossed/Invalid Amount');
            //Message saying not possible to donate since limit has been
crossed/ Invalid amount
        }else{
            //projectCreatorAdd.transfer(donation);
            p.donatedAmount(donation,admin,index);
            //Emit event after this
            emit DonatedMoney(donor,donation,message,admin,index);

            //Access transaction using web3.js

            //Give a reward badge for donating money
        }

    }

}
```

Moderator.sol

```
pragma solidity >=0.4.22 <0.7.0;
import './ReputationToken.sol';
import './ModeratorToken.sol';
import './ProjectCreator.sol';
contract Moderator{
    ModeratorToken mod;
    ReputationToken rep;
    ProjectCreator project;

      struct Transaction{
        string purpose;
        uint value;
        string proofLink;
        address admin;
```

```
        uint index; // number of the transaction in the particular project
        bool verified;
        uint transactionIndex;
        // true is for the successful retrieval of the transaction - end of
delay
        bool initialised;
    }
    Transaction t;
    constructor(address _modToken, address _repToken, address payable
_pCreator) public{

        mod = ModeratorToken(_modToken);
        rep = ReputationToken(_repToken);
        project = ProjectCreator(_pCreator);

    }
    // function generateRandomTransactionFromReturn() public{
    //   (string memory purpose, uint value, string memory
proofLink,address admin,uint index,bool verified,uint transactionIndex) =
project.requestedRandomTransactionFromMod();
    //   t =
Transaction(purpose,value,proofLink,admin,index,verified,transactionIndex,t
rue);
    // }

     function generateRandomTransactionFromEmit(string memory purpose, uint
value, string memory proofLink,address admin,uint index,bool verified,uint
transactionIndex) public{
        // Call this function after retrieveing it from the emit
       t =
Transaction(purpose,value,proofLink,admin,index,verified,transactionIndex,t
rue);

    }

    // For which moderator approves reputation for which receiver
    event transferReputation(address receiver, address moderator);
    // For which moderator gets the moderation token
    event transferModeratorToken(address receiver);

    function validateTransaction() public{
        //TODO Should not be able to validate own transaction
        require(msg.sender != t.admin);
        require(t.initialised == true,"Not initialised yet!");
        //Replace with transaction index
        project.makeTransactionValid(t.admin,t.index,t.transactionIndex);
        //Increase mod tokens and increase reputation tokens for the owner
of project
        //Rep tokens to the admin of the project/project creator
        rep.transferFrom(rep.returnOwner(),t.admin,1);
        //Moderator tokens to the moderator and REP goes to the project
creator
        mod.transferFrom(mod.returnOwner(),msg.sender,1);
        emit transferReputation(t.admin,msg.sender);
        emit transferModeratorToken(msg.sender);
    }

    function rejectTransaction() public{
        // Cannot approve your own transaction
        require(msg.sender != t.admin);
        require(t.initialised == true,"Not initialised yet!");
```

```
        // Moderator gets MOD for rejecting also
        mod.transferFrom(mod.returnOwner(),msg.sender,1);
        emit transferModeratorToken(msg.sender);
    }

    function viewModeratorTokenBalance()public view returns(uint){
        return mod.balanceOf(msg.sender);
    }

}
```

Projectcreator.sol

```
pragma solidity >=0.4.22 <0.7.0;
// ABIEncoder enables to use structs as parameters to functions
pragma experimental ABIEncoderV2;
import './ReputationToken.sol';
contract ProjectCreator{
    // stores details of a project
    struct Project{
        string name;
        string description;
        // max donation needed by the project
        uint limit;
        address admin;
        uint donatedValue;
        // limit minus donated value - can receive this much still
        uint balance;
        // mapping every transaction to an integer value
        mapping ( uint=> Transaction) transactions;
        // number of transactions done in this project
        uint transactionSize;
        uint spentAmount;
        // number of the project created by the particular project owner
        uint index;
    }
    // stores details and status of a project
    struct Transaction{
        Project p;
        string purpose;
        uint value;
        string proofLink;
        address admin;
        uint index;
        bool verified;
        uint transactionIndex;
    }
    // for the global list of admins
    struct globalIndexToAdmin{
        address admin;
        uint adminIndex;
    }
    ReputationToken rep;
    Project p;
    // number of projects
    uint public pCount;
    // mapping a value to every admin which can be seen globally
    mapping (uint => globalIndexToAdmin) public projectMap;
    // List of Projects mapped against an address (Project Creators'
addresses)
    mapping (address => Project[]) public projects;
    // Count of Projects mapped against an address (Project Creators'
addresses)
```

```
    mapping (address => uint) public projectCount;
    // For the transactions pending approval by the moderators
    mapping(uint256 => Transaction) queue;
    uint256 first = 1;
    uint256 last = 0;

    // true when the address is an admin (creator of a project)
    // to avoid adding the same address twice if he creates a new project
    mapping (address => bool) adminCheck;
    // an admin address goes into the keyList
    address[] public keyList;
    // total number of admins - total number of project creators
    uint public adminCount = 0;

    // Mapping of number of transactions for an address (Project Creators'
addresses)
    mapping (address => uint) overallTransactionCount;

    // Initiating reputation token contract with the address of the project
    constructor(address _repToken) public{
        rep = ReputationToken(_repToken);
    }
    // Viewing balance of the project
    function showBalance()public view returns(uint){
        return address(this).balance;
    }
    // Viewing balance of the project in ether value
    function showEtherBalance() public view returns(uint){
        return address(this).balance / (10**18);
    }

    receive() external payable{
        //Receive Ether
    }
    fallback() external{
        //In case of an error in receiving ether
    }

    // Creating a project with a name, description and a value
    function createProject(string memory _name, string memory _desc, uint
_value) public{
        //Ratio between transactionCount count and reputation matters here
        //If ratio is too low, no project creation
        uint transactionCount = overallTransactionCount[msg.sender];
        uint repTokensBalance = rep.balanceOf(msg.sender);
        if(repTokensBalance == 0){
           repTokensBalance = 1; // to avoid division by 0
        }
        uint ratio = transactionCount/repTokensBalance;
        require(ratio <=2,"Reputation too low to create new project!");
        p = Project({name: _name, description: _desc, limit: _value,admin:
msg.sender,donatedValue: 0, balance: _value,transactionSize:
0,spentAmount:0,index: projectCount[msg.sender]});
        // Add project to list of projects mapped for the creator's address
        projects[msg.sender].push(p);
        // Add project to project map with admin address and his project
count
        projectMap[pCount] = globalIndexToAdmin({admin: msg.sender,
adminIndex: projectCount[msg.sender] });
        //Remove Duplicates
        // checking and adding new admin address to keyList
```

```
        if(adminCheck[msg.sender] == false){
            adminCheck[msg.sender] = true;
            keyList.push(msg.sender);
            adminCount ++;
        }
        projectCount[msg.sender]++;
        pCount++;
    }

    event ViewProject(string  name, string  desc,uint value,address admin);
    function viewProject(address owner,uint index) public returns(string
memory name, string memory desc,uint value,address admin, uint
donatedValue, uint balance, uint transactionSize, uint spentAmount, uint
index1){
        Project memory pr = projects[owner][index];
        emit ViewProject(pr.name,pr.description,pr.limit,pr.admin);
        return (pr.name,pr.description,pr.limit,pr.admin, pr.donatedValue,
pr.balance, pr.transactionSize, pr.spentAmount, pr.index );
    }

    // Returns the number of projects created under a creator address
    function returnProjectCountForCreator(address admin)external view
returns (uint count){
        return projectCount[admin];
    }

    // Event to emit list of projects
    event ViewAllProjects(string[]  names, string[]  desc, uint[]
limits,address[]  owners);

    function viewProjects(address admin) public returns (string[] memory
names, string[] memory desc, uint[] memory limits,address[] memory owners){
        uint length = projects[admin].length;
        string[] memory n = new string[](length);
        string[] memory d = new string[](length);
        uint[] memory l = new uint[](length);
        address[] memory o = new address[](length);
        for (uint i=0;i<length;i++){
            n[i] =  projects[admin][i].name;
            d[i] =  projects[admin][i].description;
            l[i] =  projects[admin][i].limit;
            o[i] =  projects[admin][i].admin;
        }
        emit ViewAllProjects(n,d,l,o);
        return (n,d,l,o);
    }

     //Use this list to retreive all the owners and based on that, use the
view projects function.
    function viewAllProjectAdmins() public view returns(address[] memory
arrayOfProjectOwners){
        return keyList;
    }

    function submitProof(string memory link, address admin, uint index,
uint transactionIndex) public{
        // check for only admin being able to submit proof
        require(admin == msg.sender,"Invalid User");
        // add transaction link to project
        projects[admin][index].transactions[transactionIndex].proofLink =
link;
```

```
        // add transaction to the transactions queue
        enqueue(projects[admin][index].transactions[transactionIndex]);
    }

    event generateRandomTransaction(string  purpose,uint value, string
proofLink,address admin,uint index,bool verified,uint transactionIndex);
    function requestedRandomTransactionFromMod()public returns( string
memory purpose,uint value, string memory proofLink,address admin,uint
index,bool verified,uint transactionIndex){
        // Take the first pending transaction from the queue
        Transaction memory review = dequeue();
        // Process the emit here and pass it on the moderator function
        // tb implemented
        emit
generateRandomTransaction(review.purpose,review.value,review.proofLink,revi
ew.admin,review.index,review.verified,review.transactionIndex);
        return
(review.purpose,review.value,review.proofLink,review.admin,review.index,rev
iew.verified,review.transactionIndex);
    }

    function enqueue(Transaction memory data) internal {
        last += 1;
        queue[last] = data;
    }

    function dequeue() internal returns (Transaction memory data) {
        require(last >= first,'Err');  // non-empty queue
        data = queue[first];
        delete queue[first];
        first += 1;
    }

    function withdraw(uint amount, uint index,string memory purpose)
public{
        //Withdraw based on reputation
        //Based on Ratio of transactionCount and reputation.
        //Require statement not needed because i am using msg.sender
        uint transactionCount =
projects[msg.sender][index].transactionSize;
        uint repTokensBalance = rep.balanceOf(msg.sender);
        if(repTokensBalance == 0){
           repTokensBalance = 1;
        }
        uint ratio = transactionCount/repTokensBalance;
        require(ratio<=2,"Reputation Balance Low to Withdraw!");
        uint remainingValue = projects[msg.sender][index].donatedValue -
projects[msg.sender][index].spentAmount;
        uint amtInEther = amount / (10**18);
        require(remainingValue > amtInEther,"Invalid Withdrawal Amount -
Higher than available balance");
        msg.sender.transfer(amount);
        amount = amount / (10**18);
        //Add transaction details
        uint transactionSize = projects[msg.sender][index].transactionSize;
        // New unapproved transaction. Hence verified value is false
        Transaction memory t =
Transaction(projects[msg.sender][index],purpose,amount,"",msg.sender,index,
false,transactionSize);
        projects[msg.sender][index].transactions[transactionSize] = t;
        projects[msg.sender][index].transactionSize++;
```

```
        overallTransactionCount[msg.sender]++;
        // updating the spent amount
        projects[msg.sender][index].spentAmount =
projects[msg.sender][index].spentAmount + amount;
    }
    function checkLimit(uint donatedAmt, address admin, uint index) public
view returns (bool isLimitReached){
        uint currentDonated = projects[admin][index].donatedValue;
        uint limit = projects[admin][index].limit;
        if(currentDonated+ donatedAmt < limit){
            return true; // within limit
        }else{
            return false; // limit crossed
        }

    }
    function viewReputationTokenBalance() public view returns(uint){
        return rep.balanceOf(msg.sender);
    }
    function donatedAmount(uint donatedAmt, address admin, uint index)
public {
        projects[admin][index].donatedValue =
projects[admin][index].donatedValue + donatedAmt;
        projects[admin][index].balance = projects[admin][index].limit -
projects[admin][index].donatedValue;
    }

    function viewDonatedAmount(address admin, uint index)public view
returns(uint value){
        return projects[admin][index].donatedValue;
    }
    function viewSpentAmount(address admin, uint index)public view
returns(uint value){
        return projects[admin][index].spentAmount;
    }
    function makeTransactionValid(address admin, uint index, uint
transactionIndex)public{
        projects[admin][index].transactions[transactionIndex].verified =
true;
    }
    event ViewAllTransactions(string[]  _purpose, uint[]  _value,string[]
_proofLink,address _admin,bool[] _verified, uint[]  _transactionIndex);
    function viewAllTransactionsForProject(address admin, uint
index)public{
        uint length = projects[admin][index].transactionSize;
        string[] memory purpose = new string[](length);
        uint[] memory value = new uint[](length);
        string[] memory proofLink = new string[](length);
        bool[] memory verified = new bool[](length);
        uint[]  memory transactionIndex = new uint[](length);
        for(uint i=0;i<length;i++){
            purpose[i] = projects[admin][index].transactions[i].purpose;
            value[i] = projects[admin][index].transactions[i].value;
            proofLink[i] =
projects[admin][index].transactions[i].proofLink;
            verified[i] = projects[admin][index].transactions[i].verified;
            transactionIndex[i] =
projects[admin][index].transactions[i].transactionIndex;
        }
        emit
```

```
ViewAllTransactions(purpose,value,proofLink,admin,verified,transactionIndex
);
    }
    function viewSingleTransaction(address admin, uint index, uint tindex)
public view returns (string memory purpose,uint value, string memory
proofLink,bool verified) {
        Transaction memory temp =
projects[admin][index].transactions[tindex];
        return(temp.purpose,temp.value, temp.proofLink,temp.verified);
    }
}
```

Moderator tokens.sol

```
pragma solidity >=0.4.22 <0.7.0;
import '@openzeppelin/contracts/token/ERC20/ERC20.sol';

contract ModeratorToken is ERC20{
    address owner;
    //minting reputation token with name 'ModeratorToken' and Symbol 'MOD'
    constructor() ERC20("ModeratorToken","MOD") public {
        _mint(msg.sender, 10000 * 10 ** uint(decimals()));
        owner = msg.sender;

    }
    // approving the transfer of MOD to a certain recipient address
    function approveContract(address recipient,uint value)external{
        approve(recipient,value);
    }
    // returns address of the owner of the MOD
    function returnOwner() public view returns(address _owner){
        return owner;
    }


}
```

Reputation tokens.sol

```
pragma solidity >=0.4.22 <=0.7.0;
import '@openzeppelin/contracts/token/ERC20/ERC20.sol';

contract ReputationToken is ERC20{
    address owner;
    //minting reputation token with name 'Reputation' and Symbol 'REP'
    constructor() ERC20("Reputation","REP") public {
        _mint(msg.sender, 100000 * 10 ** uint(decimals()));
        owner = msg.sender;

    }
    // approving the transfer of REP to a certain recipient address
    function approveContract(address recipient,uint value)external{
        approve(recipient,value);
    }
    // returns address of the owner of the REP
      function returnOwner() public view returns(address _owner){
        return owner;
    }
```

}**Migrations.sol**

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.8.0;

contract Migrations {
  address public owner = msg.sender;
```

```
  uint public last_completed_migration;

  modifier restricted() {
    require(
      msg.sender == owner,
      "This function is restricted to the contract's owner"
    );
    _;
  }

  function setCompleted(uint completed) public restricted {
    last_completed_migration = completed;
  }
}
```

MIGRATION
Initial migration.js

```
const Migrations = artifacts.require("Migrations");

module.exports = function (deployer) {
  deployer.deploy(Migrations);
};
```

Deploy projects.js

```
// Importing Contracts
const ReputationToken = artifacts.require("ReputationToken");
const ModeratorToken = artifacts.require("ModeratorToken");
const ProjectCreator = artifacts.require("ProjectCreator");
const Moderator = artifacts.require("Moderator");
const Donor = artifacts.require("Donor");
module.exports = function(deployer) {
  deployer.then(
    // SO that one happens after the other in the required order of
initialisation
    async () => {

    // address of the REP token contract
    const repTokenInst = await deployer.deploy(ReputationToken);

    console.log('ReputationToken contract deployed at',
repTokenInst.address);

    const modTokenInst = await deployer.deploy(ModeratorToken);
    console.log('ModeratorToken contract deployed at',
modTokenInst.address);

    const projInst = await deployer.deploy(ProjectCreator,
repTokenInst.address);
    console.log('ProjCreator contract deployed at', projInst.address);

    const modInst = await deployer.deploy(Moderator, modTokenInst.address,
repTokenInst.address, projInst.address);
    console.log('Moderator contract deployed at', modInst.address);

    // Allowing the usage of tokens
    await repTokenInst.approveContract(modInst.address,100000);
    await modTokenInst.approveContract(modInst.address,100000);

    const donorInst = await deployer.deploy(Donor, projInst.address);
    console.log('Donor contract deployed at', donorInst.address);
```

```
  })
};
```

TESTING

Testfile.js

```
const ProjectCreator = artifacts.require('ProjectCreator');
const truffleAssert = require("truffle-assertions");
const Donor = artifacts.require('Donor');
const Moderator = artifacts.require('Moderator');
contract('ProjectCreator',async accounts=>{
    let projectCreator = null;
    let creator = null;
    let acc2 = null;
    let donor = null;
    let moderator = null;
    before(async()=>{
        projectCreator = await ProjectCreator.deployed();
        donor = await Donor.deployed();
        moderator = await Moderator.deployed();
        creator = accounts[1];
        acc2 = accounts[2];
    });
    //Create Project And View Project
    it('Create and View a Project',async()=>{
        let tx = await
projectCreator.createProject("Project1","Description",100,{from: creator});
        const val = await projectCreator.viewProject.call(creator,0,{from:
creator});
        assert(val[3] === creator);
        assert(val[2].toNumber() === 100);
        assert(val[0] === "Project1" );
        assert(val[1] === "Description");
    });

    //Donating to a project
    it('Donating to a project',async()=>{
        let one_eth = web3.utils.toWei('10', "ether");
        await web3.eth.sendTransaction({from: acc2, to:
projectCreator.address, value: one_eth});
        let donationTx = await donor.donate(10,creator,0,"Donation From
Account 2",{from: acc2});
        const donatedVal = await
projectCreator.viewDonatedAmount.call(creator,0,{from: creator});
        assert(donatedVal.toNumber() === 10);
        truffleAssert.eventEmitted(donationTx, 'DonatedMoney', (ev) => {
            return ev.donor === acc2 && ev.amount.toNumber() === 10 &&
ev.projectAdmin === creator && ev.index.toNumber() == 0;
        });
    });

    //Invalid donation test case
    it('Invalid Donation Test Case',async()=>{
        //Donate 95 more
        try{
        let donationTx = await donor.donate(95,creator,0,"2nd Donation From
Account 2",{from: acc2});
        }catch(e){
            assert(e.message.includes('Limt crossed/Invalid Amount'));
            return;
        }
        assert(false);
```

```
    });

    //Withdraw Test case
    it('Withdrawing Test Case',async()=>{
        let five_eth = web3.utils.toWei('5','ether');
        let one_eth = web3.utils.toWei('1','ether');
        let withdrawTx = await
projectCreator.withdraw(five_eth,0,"Transaction 1",{from: creator});

        const spentAmount = await
projectCreator.viewSpentAmount.call(creator,0,{from: creator});
        assert(spentAmount.toNumber() === 5);
        const contractBalance = await
projectCreator.showEtherBalance.call();
        assert(contractBalance.toNumber() === 5);
        let withdrawTx2 = await
projectCreator.withdraw(one_eth,0,"Transaction 2",{from: creator});
    });

    //Submit proof test
    it('Submitting Proof Test',async()=>{
        const submitProofTx = await projectCreator.submitProof("Website
Link",creator,0,0,{from: creator});
        const viewSingleTx = await
projectCreator.viewSingleTransaction.call(creator,0,0,{from: creator});
        assert(viewSingleTx[0] === "Transaction 1" );
        assert(viewSingleTx[1].toNumber() === 5);
        assert(viewSingleTx[2] === "Website Link");
        assert(viewSingleTx[3] === false);
        const submitProofTx2 = await projectCreator.submitProof("Website
Link 2",creator,0,1,{from: creator});
    });

    //Generating transaction for Moderator
    it('Getting a transaction for a Moderator',async()=>{
        let generateRandomTx = await
projectCreator.requestedRandomTransactionFromMod({from: acc2});
        //console.log(generateRandomTx);
        let purpose;
        let value;
        let proofLink;
        let admin;
        let index;
        let verified;
        let txIndex;
        truffleAssert.eventEmitted(generateRandomTx,
'generateRandomTransaction', (ev) => {
            purpose = ev.purpose;
            value = ev.value.toNumber();
            proofLink = ev.proofLink;
            admin = ev.admin;
            index = ev.index.toNumber();
            verified = ev.index.verified;
            txIndex = ev.transactionIndex.toNumber();
            return ev.admin === creator && ev.transactionIndex.toNumber()
=== 0 && ev.index.toNumber() === 0;
        });
        const callModFunction = await
moderator.generateRandomTransactionFromEmit(purpose,value,proofLink,admin,i
ndex,verified,txIndex);
```

```
    });

    //Validating a transaction
    it('Validating a Transaction in Moderator',async()=>{

        let validateTx = await moderator.validateTransaction({from: acc2});
        truffleAssert.eventEmitted(validateTx,'transferReputation',(ev)=>{
            return ev.receiver == creator && ev.moderator == acc2;
        });
        truffleAssert.eventEmitted(validateTx,'transferModeratorToken',(ev)
=>{
            return ev.receiver == acc2;
        });
        //Check if validation successful
        let successCheck = await
projectCreator.viewSingleTransaction.call(creator,0,0);
        assert(successCheck.verified === true);
        //Check reputation and moderator Balance
        let repBalance = await
projectCreator.viewReputationTokenBalance.call({from: creator});
        assert(repBalance.toNumber() === 1);
        //console.log(repBalance);
        let modBalance = await
moderator.viewModeratorTokenBalance.call({from: acc2});
        //console.log(modBalance)
        assert(modBalance.toNumber() === 1);
    });

    //Reject a transaction
    it('Rejecting a Transaction in Moderator',async()=>{
        let generateRandomTx = await
projectCreator.requestedRandomTransactionFromMod({from: acc2});
        //console.log(generateRandomTx);
        let purpose;
        let value;
        let proofLink;
        let admin;
        let index;
        let verified;
        let txIndex;
        truffleAssert.eventEmitted(generateRandomTx,
'generateRandomTransaction', (ev) => {
            purpose = ev.purpose;
            value = ev.value.toNumber();
            proofLink = ev.proofLink;
            admin = ev.admin;
            index = ev.index.toNumber();
            verified = ev.index.verified;
            txIndex = ev.transactionIndex.toNumber();
            return ev.admin === creator && ev.transactionIndex.toNumber()
=== 1 && ev.index.toNumber() === 0;
        });
        const callModFunction = await
moderator.generateRandomTransactionFromEmit(purpose,value,proofLink,admin,i
ndex,verified,txIndex);

        let rejectTx = await moderator.rejectTransaction({from: acc2});
        truffleAssert.eventEmitted(rejectTx,'transferModeratorToken',(ev)
=>{
            return ev.receiver == acc2;
        });
```

```
        let successCheck = await
projectCreator.viewSingleTransaction.call(creator,0,1);
        assert(successCheck.verified === false);
        let modBalance = await
moderator.viewModeratorTokenBalance.call({from: acc2});
        //console.log(modBalance)
        assert(modBalance.toNumber() === 2);
    });
});
```

FRONTEND

App.js

```
App = {
  loading:false,
  contracts: {},
  mainState: 0,
  globalIndex: 0,
  globalAdmin: 0,
  adminIndex: 0,
init: async function() {
    return await App.initWeb3();
  },

  load: async () => {
    await App.loadWeb3()
    await App.loadAccount()
    await App.loadContract()
    await App.render()
  },


  // https://medium.com/metamask/https-medium-com-metamask-breaking-change-
injecting-web3-7722797916a8
  loadWeb3: async () => {
    if (typeof web3 !== 'undefined') {
        console.log("im here!!")
      const provider = await detectEthereumProvider();
      console.log(provider)
      // var version = web3.version.api;
      console.log(version); // "0.2.0"
      App.Web3Provider = provider
      console.log("Metamask is Installed!!")
      //web3 = new Web3(web3.currentProvider)
    } else {
      window.alert('Please connect to Metamask.')
    }
    Modern dapp browsers...
    if (window.ethereum) {
    window.web3 = new Web3(ethereum)
    try {
     Request account access if needed
     const accAccess = await window.ethereum.request({ method:
'eth_requestAccounts' });
       // Acccounts now exposed
        //:TODO

        // try {
        // const transactionHash = await ethereum.request({
        //   method: 'eth_sendTransaction',
        //   params: [
        //     {
```

```
        //       to: null,
        //       'from': null,
        //       value: null,
        //       // And so on...
        //     },
        //   ],
        // });
        //   // Handle the result
        //   console.log(transactionHash);
        // } catch (error) {
        //   console.error(error);
        // }
        // web3.eth.sendTransaction({
        //   /* ... */
        // })
        console.log("Given access to metamask accounts");
      } catch (error) {
        // User denied account access...
        if (error.code === 4001) {
          console.log("Denied access to metamask accounts");
        }

      }
    }
    // // Legacy dapp browsers...
    // else if (window.web3) {
    //   App.web3Provider = web3.currentProvider
    //   window.web3 = new Web3(web3.currentProvider)
    //   // Acccounts always exposed
    //   // web3.eth.sendTransaction({
    //   //   /* ... */
    //   // })
    // }
    // Non-dapp browsers...
    else {
      console.log(
        'Non-Ethereum browser detected. You should consider trying
MetaMask!',
      )
    }
  },

  loadAccount: async () => {
    // Set the current blockchain account
    const accounts = await ethereum.request({ method: 'eth_accounts' });
    App.account = accounts[0];
    console.log(App.account)
    console.log(App.account)
  },

  loadContract: async () => {
    // Create a JavaScript version of the smart contract
    const PC = await $.getJSON('ProjectCreator.json')
    App.contracts.PC = TruffleContract(PC)
    App.contracts.PC.setProvider(App.web3Provider)

    const M = await $.getJSON('Moderator.json')
    App.contracts.M = TruffleContract(M)
    App.contracts.M.setProvider(App.web3Provider)

    const D = await $.getJSON('Donor.json')
```

```
  App.contracts.D = TruffleContract(D)
  App.contracts.D.setProvider(App.web3Provider)

  // Hydrate the smart contract with values from the blockchain
  App.PC = await App.contracts.PC.deployed()
  App.M = await App.contracts.M.deployed()
  App.D = await App.contracts.D.deployed()
},


render: async () => {
  // Prevent double render
  if (App.loading) {
    return
  }

  // Update app loading state
  App.setLoading(1)

  // Render Account
  console.log(App.account)


  //Render All ProjectCreator
  await App.renderMyProjects()
  await App.renderAllProjects()
  await App.renderAllAdmins()
  await App.renderReputationBalance()

  //Render All Moderator
  await App.renderModTokenBalance()

  //Render All Donor
  await App.renderAllProjectsDonor()
  await App.renderAllAdminsDonor()

  // Update loading state
  if (App.mainState === 0) {
    App.setLoading(1)
  }
  else if (App.mainState === 1) {
    App.setLoading(110)
  }
  else if (App.mainState === 2) {
    App.setLoading(203)
  }
  else if (App.mainState === 3) {
    App.setLoading(305)
  }
},




SET_MAIN_PAGE: async () => {
  console.log('selected MAIN_PAGE')
  App.mainState = 0
  App.setLoading(110)
  //window.location.reload()
},
```

```
// MAIN PAGE FUNCTIONS
SET_MAIN_PAGE: async () => {
  console.log('selected main_page')
  App.mainState = 2
  App.setLoading(110)
},
SET_PROJECT_CREATOR: async () => {
  console.log('selected SET_PROJ_CREATOR')
  App.mainState = 1
  App.setLoading(110)
  //window.location.reload()
},
SET_MODERATOR: async () => {
  console.log('selected SET_MOD')
  App.mainState = 2
  App.setLoading(203)
},
SET_DONOR: async () => {
  console.log('selected SET_DONOR')
  App.mainState = 3
  App.setLoading(305)
},
// END OF MAIN PAGE FUNCTIONS
```

```
// PROJECT CREATOR FUNCTIONS

renderAllProjects: async () => {
  // Load the total task count from the blockchain
  const PCCount = await App.PC.pCount()
  const $allProjectsTemplate = $('.allProjectsTemplate')

  // Render out each task with a new task template
  for (var i = 0; i < PCCount; i++) {
    // Fetch the task data from the blockchain
    const indexMap = await App.PC.projectMap(i)
    //console.log(indexMap)
    const projAdmin = indexMap[0]
    const projAdminIndex = indexMap[1].toNumber()
    //console.log(projAdmin)
    //console.log(projAdminIndex)
```

```
    // Fetch the task data from the blockchain
    //console.log("GONNA FETCH")
    const proj = await App.PC.viewProject.call(projAdmin, projAdminIndex)
    //console.log("FETCHED PROJ")
    //console.log(proj)
    const name = proj[0]
    const desc = proj[1]
    const limit = proj[2].toNumber()
    const admin = proj[3]
    const donatedVal = proj[4].toNumber()
    const bal = proj[5].toNumber()

    // Create the html for the task
    const $newAllProjectsTemplate = $allProjectsTemplate.clone()
    $newAllProjectsTemplate.find('.pID').html(i)
    $newAllProjectsTemplate.find('.pName').html(name)
    $newAllProjectsTemplate.find('.pDesc').html(desc)
    $newAllProjectsTemplate.find('.pLimit').html(limit)
    $newAllProjectsTemplate.find('.pAdmin').html(admin)

    $('#allProjects').append($newAllProjectsTemplate)

    $newAllProjectsTemplate.show()
  }
},

renderMyProjects: async () => {
  // Load the total task count from the blockchain
  const PCCount = await App.PC.pCount()
  const $myProjectsTemplate = $('.myProjectsTemplate')
  //console.log("pCount")
  //console.log(PCCount)

  // Render out each task with a new task template
  for (var i = 0; i < PCCount; i++) {
    // Fetch the task data from the blockchain
    const indexMap = await App.PC.projectMap(i)
    //console.log(indexMap)
    const projAdmin = indexMap[0]
    const projAdminIndex = indexMap[1].toNumber()
    //console.log(projAdmin)
    //console.log(projAdminIndex)

    // Fetch the task data from the blockchain
    //console.log("GONNA FETCH")
    const proj = await App.PC.viewProject.call(projAdmin, projAdminIndex)
    //console.log("FETCHED PROJ")
    //console.log(proj)
    const name = proj[0]
    const desc = proj[1]
    const limit = proj[2].toNumber()
    const admin = proj[3]
    const donatedVal = proj[4].toNumber()
    const bal = proj[5].toNumber()
    const NOT = proj[6].toNumber()
    const spent = proj[7].toNumber()
    const adminIndex = proj[8].toNumber()

    if (admin === App.account) {
      // Create the html for the task
      const $newMyProjectsTemplate = $myProjectsTemplate.clone()
```

```
      $newMyProjectsTemplate.find('.pID').html(i)
      $newMyProjectsTemplate.find('.pName').html(name)
      $newMyProjectsTemplate.find('.pDesc').html(desc)
      $newMyProjectsTemplate.find('.pLimit').html(limit)
      $newMyProjectsTemplate.find('.pAdmin').html(admin)
      $newMyProjectsTemplate.find('.pDonatedVal').html(donatedVal)
      $newMyProjectsTemplate.find('.pBal').html(bal)
      $newMyProjectsTemplate.find('.pNOT').html(NOT)
      $newMyProjectsTemplate.find('.pSpent').html(spent)
      $newMyProjectsTemplate.find('.pAdminIndex').html(adminIndex)

      $('#myProjects').append($newMyProjectsTemplate)

      $newMyProjectsTemplate.show()
    }
  }
},
renderAllAdmins: async () => {
  // Load the total task count from the blockchain
  const AdminCount = await App.PC.adminCount()
  const $allAdminsTemplate = $('.allAdminsTemplate')
  //console.log("ADMIN COUNT BELOW")
  //console.log(AdminCount)
  // Render out each task with a new task template
  for (var i = 0; i < AdminCount; i++) {
    // Fetch the task data from the blockchain
    const admin = await App.PC.keyList(i)
    //console.log("admin here")
    //console.log(admin)

    // Create the html for the task
    const $newAllAdminsTemplate = $allAdminsTemplate.clone()
    $newAllAdminsTemplate.find('.adminAddr').html(admin)

    $('#allAdmins').append($newAllAdminsTemplate)

    $newAllAdminsTemplate.show()
  }
},

createProject: async () => {
  App.setLoading(110)
  const name = $('#name').val()
  const desc = $('#desc').val()
  const limit = $('#limit').val()
  console.log("I AM HERE NOW")
  await App.PC.createProject(name, desc, limit)
  window.location.reload()
},

withdrawFromProject: async () => {
  const globalIndex = $('#projID').val()
  const amount1 = $('#amount').val()
  const amount = amount1 * ((10) ** 18)
  const reason = $('#Reason').val()
  const indexMap = await App.PC.projectMap(globalIndex)
  console.log(indexMap)
  const projAdmin = indexMap[0]
  const projAdminIndex = indexMap[1].toNumber()
  const proj = await App.PC.viewProject.call(projAdmin, projAdminIndex)
  console.log(proj)
```

```
    console.log("gonna Withdraw now")
    await App.PC.withdraw(amount, projAdminIndex, reason)
  },

  renderReputationBalance: async () => {
    const bal1 = await App.PC.viewReputationTokenBalance.call()
    console.log("BALANCE")
    const bal = bal1.toNumber()
    console.log(bal)
    const $reputationTemplate = $('.reputationTemplate')
    const $newReputationTemplate = $reputationTemplate.clone()
    $newReputationTemplate.find('.repBal').html(bal)
    $('#reputation').append($newReputationTemplate)
    $newReputationTemplate.show()
  },

  transactionsOfProject: async () => {
    const globalIndex = $('#projID1').val()
    const indexMap = await App.PC.projectMap(globalIndex)
    console.log("BEGINNING TRANSACTION RENDER")

    const projAdmin = indexMap[0]
    const projAdminIndex = indexMap[1].toNumber()
    //console.log(projAdmin)
    //console.log(projAdminIndex)
    const proj = await App.PC.viewProject.call(projAdmin, projAdminIndex)
    //console.log(proj)
    const $viewAllTransactionsTemplate = $('.viewAllTransactionsTemplate')
    const tsize = proj[6].toNumber()
    for (var i = 0; i < tsize; i++) {

      const transaction = await
App.PC.viewSingleTransaction.call(projAdmin, projAdminIndex, i)
      console.log(transaction)
      const purpose = transaction[0]
      const value = transaction[1].toNumber()
      const proofLink = transaction[2]
      const verified = transaction[3]


      // Create the html for the task

      const $newViewAllTransactionsTemplate =
$viewAllTransactionsTemplate.clone()

$newViewAllTransactionsTemplate.find('.globalIndexTransaction').html(global
Index)

$newViewAllTransactionsTemplate.find('.adminTransaction').html(projAdmin)

$newViewAllTransactionsTemplate.find('.adminProjIndexTransaction').html(pro
jAdminIndex)
      $newViewAllTransactionsTemplate.find('.TransactionIndex').html(i)

$newViewAllTransactionsTemplate.find('.purposeTransaction').html(purpose)
      $newViewAllTransactionsTemplate.find('.valueTransaction').html(value)

$newViewAllTransactionsTemplate.find('.linkTransaction').html(proofLink)

$newViewAllTransactionsTemplate.find('.verifiedTransaction').html(verified)
```

```
    $('#allTransactions').append($newViewAllTransactionsTemplate)

    $newViewAllTransactionsTemplate.show()
  }
},

submitProof: async () => {
  const globalIndex = $('#globalIndex').val()
  const tindex = $('#TransactionIndex').val()
  const link = $('#link').val()
  console.log("SUBMITTING PROOF")
  const indexMap = await App.PC.projectMap(globalIndex)
  const projAdmin = indexMap[0]
  const projAdminIndex = indexMap[1].toNumber()
  console.log(link)
  console.log(projAdmin)
  console.log(projAdminIndex)
  console.log(tindex)
  await App.PC.submitProof(link, projAdmin, projAdminIndex, tindex)
},
// TRANSFERRING FUNCTIONS
goToCreate: async () => {
  console.log('selected goToCreate')
  App.setLoading(102)
},

goToViewMyProjects: async () => {
  console.log('selected gotToViewMyProjects')
  App.setLoading(103)
},

goToViewAllProjects: async () => {
  console.log('selected goToViewAllProjects')
  App.setLoading(104)
},

goToViewAllAdmins: async () => {
  console.log('selected goToViewAllAdmins')
  App.setLoading(105)
},

goToWithdraw: async () => {
  console.log('selected goToWithdraw')
  App.setLoading(106)
},

goToViewTransactions: async () => {
  console.log('selected goToViewTransactions')
  App.setLoading(107)
},

goToSubmitProof: async () => {
  console.log('selected goTosubmitProof')
  App.setLoading(108)
},

goToViewRepTokens: async () => {
  console.log('selected goToViewRepTokens')
  App.setLoading(109)
},
```

```
  goToHome: async () => {
    console.log('selected goToHome')
    App.setLoading(110)
  },
  // END OF PROJECT CREATOR FUNCTIONS









  // MODERATOR FUNCTIONS
  renderModTokenBalance: async () => {
    const $viewModTokenBalance = $('#viewModTokenHTML');
    const value = await App.M.viewModeratorTokenBalance.call();
    $('#balanceMod').html(value.toNumber());
  },


  genRandomTransactionMod: async () => {
    console.log("GEN RAND TRANSACTION")
    const randT = await App.PC.requestedRandomTransactionFromMod.call()
    await App.PC.requestedRandomTransactionFromMod()
    console.log(randT)

    const purposeTransaction = randT[0]
    const valueTransaction = randT[1].toNumber()
    const proofLinkTransaction = randT[2]
    const adminTransaction = randT[3]
    const indexTransaction = randT[4].toNumber()
    const verifiedTransaction = randT[5]
    const tindexTransaction = randT[6].toNumber()

    const $validateTransactionTemplate = $('.validateTransactionTemplate')

    // Create the html for the task

    const $newValidateTransactionTemplate =
$validateTransactionTemplate.clone()

$newValidateTransactionTemplate.find('.purposeTransactionValidate').html(pu
rposeTransaction)

$newValidateTransactionTemplate.find('.valueTransactionValidate').html(valu
eTransaction)

$newValidateTransactionTemplate.find('.proofLinkTransactionValidate').html(
proofLinkTransaction)

$newValidateTransactionTemplate.find('.adminTransactionValidate').html(admi
nTransaction)
```

```
$newValidateTransactionTemplate.find('.projIndexTransactionValidate').html(
indexTransaction)

$newValidateTransactionTemplate.find('.transactionIndexTransactionValidate'
).html(tindexTransaction)

    $('#validateTransaction').append($newValidateTransactionTemplate)

    $newValidateTransactionTemplate.show()
    console.log("WISH ME LUCK")
    await App.M.generateRandomTransactionFromEmit(purposeTransaction,
valueTransaction, proofLinkTransaction, adminTransaction, indexTransaction,
verifiedTransaction, tindexTransaction)
  },

  modValidateTransaction: async () => {
    const answer = $('#answerMod').val()
    console.log("GONNA VALIDATE")
    console.log(answer)
    if (answer == 1) {
      await App.M.validateTransaction()
      console.log("VALIDATED")
    }
    else if (answer == 0) {
      await App.M.rejectTransaction()
      console.log("REJECTED")
    }
  },
  //Transfer Functions
  goToModHome: async () => {
    console.log('selected goToModHome')
    App.setLoading(203)
  },

  goToValidateTransaction: async () => {
    console.log('selected goToValidateTransaction')
    App.setLoading(201)
  },

  goToViewModTokens: async () => {
    console.log('selected goToModTok')
    App.setLoading(202)
  },
  // END OF MOD FUNCS









  // DONOR FUNCS
  renderAllProjectsDonor: async () => {
```

```
    // Load the total task count from the blockchain
    const PCCount = await App.PC.pCount()
    const $allProjectsDonorTemplate = $('.allProjectsDonorTemplate')

    // Render out each task with a new task template
    for (var i = 0; i < PCCount; i++) {
      // Fetch the task data from the blockchain
      const indexMap = await App.PC.projectMap(i)
      //console.log(indexMap)
      const projAdmin = indexMap[0]
      const projAdminIndex = indexMap[1].toNumber()
      //console.log(projAdmin)
      //console.log(projAdminIndex)

      // Fetch the task data from the blockchain
      //console.log("GONNA FETCH")
      const proj = await App.PC.viewProject.call(projAdmin, projAdminIndex)
      //console.log("FETCHED PROJ")
      //console.log(proj)
      const name = proj[0]
      const desc = proj[1]
      const limit = proj[2].toNumber()
      const admin = proj[3]
      const donatedVal = proj[4].toNumber()
      const bal = proj[5].toNumber()
      const NOT = proj[6].toNumber()
      const spent = proj[7].toNumber()
      const adminIndex = proj[8].toNumber()

      // Create the html for the task
      const $newAllProjectsDonorTemplate =
$allProjectsDonorTemplate.clone()
      $newAllProjectsDonorTemplate.find('.pID').html(i)
      $newAllProjectsDonorTemplate.find('.pName').html(name)
      $newAllProjectsDonorTemplate.find('.pDesc').html(desc)
      $newAllProjectsDonorTemplate.find('.pLimit').html(limit)
      $newAllProjectsDonorTemplate.find('.pAdmin').html(admin)
      $newAllProjectsDonorTemplate.find('.pDonatedVal').html(donatedVal)
      $newAllProjectsDonorTemplate.find('.pBal').html(bal)
      $newAllProjectsDonorTemplate.find('.pNOT').html(NOT)
      $newAllProjectsDonorTemplate.find('.pSpent').html(spent)
      $newAllProjectsDonorTemplate.find('.pAdminIndex').html(adminIndex)

      $('#allProjectsDonor').append($newAllProjectsDonorTemplate)

      $newAllProjectsDonorTemplate.show()
    }
  },

  renderAllAdminsDonor: async () => {
    // Load the total task count from the blockchain
    const AdminCount = await App.PC.adminCount()
    const $allAdminsDonorTemplate = $('.allAdminsDonorTemplate')
    //console.log("ADMIN COUNT BELOW")
    //console.log(AdminCount)
    // Render out each task with a new task template
    for (var i = 0; i < AdminCount; i++) {
      // Fetch the task data from the blockchain
      const admin = await App.PC.keyList(i)
      //console.log("admin here")
      //console.log(admin)
```

```
    // Create the html for the task
    const $newAllAdminsDonorTemplate = $allAdminsDonorTemplate.clone()
    $newAllAdminsDonorTemplate.find('.adminAddr').html(admin)

    $('#allAdminsDonor').append($newAllAdminsDonorTemplate)

    $newAllAdminsDonorTemplate.show()
  }
},

donateToProject: async () => {
  const mainIndex = $('#mainIndex').val()
  const amount = $('#value').val()
  const indexMap = await App.PC.projectMap(mainIndex)
  console.log(indexMap)
  const admin = indexMap[0]
  const adminIndex = indexMap[1].toNumber()
  const msg = "hi"
  console.log("ALL SET TO GO")
  console.log(amount)
  console.log(admin)
  console.log(adminIndex)
  await App.D.donate(amount, admin, adminIndex, "Hi")
  console.log("DONATED IN FUNC")
  console.log("BEGINNING ACTUAL MONEY TRANSFER")

  try {
    const transactionHash = await ethereum.request({
      method: 'eth_sendTransaction',
      params: [
        {
          from: App.account,
          to: App.PC.address,
          value: amount * ((10) ** 18)
        },
      ],
    });
    // Handle the result
    console.log(transactionHash);
  } catch (error) {
    console.error(error);
  }
  // web3.eth.sendTransaction({
  //   from: App.account,
  //   to: App.PC.address,
  //   value: amount * ((10)**18)
  // }, function(error,hash){
  //   console.log(error)
  // })
  const balanceNow = await App.PC.showBalance.call()
  console.log("BALANCE BEFORE TRANSFER")
  console.log(balanceNow.toNumber())
  //window.location.reload()
},

transactionsOfProjectDonor: async () => {
  const globalIndex = $('#projID1').val()
  const indexMap = await App.PC.projectMap(globalIndex)
  console.log("BEGINNING TRANSACTION RENDER")
```

```
    const projAdmin = indexMap[0]
    const projAdminIndex = indexMap[1].toNumber()
    //console.log(projAdmin)
    //console.log(projAdminIndex)
    const proj = await App.PC.viewProject.call(projAdmin, projAdminIndex)
    //console.log(proj)
    const $viewAllTransactionsTemplateDonor =
$('.viewAllTransactionsTemplateDonor')
    const tsize = proj[6].toNumber()
    for (var i = 0; i < tsize; i++) {

      const transaction = await
App.PC.viewSingleTransaction.call(projAdmin, projAdminIndex, i)
      console.log(transaction)
      const purpose = transaction[0]
      const value = transaction[1].toNumber()
      const proofLink = transaction[2]
      const verified = transaction[3]


      // Create the html for the task

      const $newViewAllTransactionsTemplateDonor =
$viewAllTransactionsTemplateDonor.clone()

$newViewAllTransactionsTemplateDonor.find('.globalIndexTransactionDonor').h
tml(globalIndex)

$newViewAllTransactionsTemplateDonor.find('.adminTransactionDonor').html(pr
ojAdmin)

$newViewAllTransactionsTemplateDonor.find('.adminProjIndexTransactionDonor'
).html(projAdminIndex)

$newViewAllTransactionsTemplateDonor.find('.TransactionIndexDonor').html(i)

$newViewAllTransactionsTemplateDonor.find('.purposeTransactionDonor').html(
purpose)

$newViewAllTransactionsTemplateDonor.find('.valueTransactionDonor').html(va
lue)

$newViewAllTransactionsTemplateDonor.find('.linkTransactionDonor').html(pro
ofLink)

$newViewAllTransactionsTemplateDonor.find('.verifiedTransactionDonor').html
(verified)


$('#allTransactionsDonor').append($newViewAllTransactionsTemplateDonor)

      $newViewAllTransactionsTemplateDonor.show()
    }
  },
  //ROUTING FUNCS
  goToDonorHome: async () => {
    console.log('selected goToDonorHome')
    App.setLoading(305)
  },

  goToDonate: async () => {
```

```
  console.log('selected goToDonate')
  App.setLoading(302)
},

goToViewAllProjectsDonor: async () => {
  console.log('selected goToViewAllProjectsD')
  App.setLoading(301)
},

goToViewAllAdminsDonor: async () => {
  console.log('selected goToViewAllAdminsD')
  App.setLoading(304)
},

goToViewTransactionsDonor: async () => {
  console.log('selected goToViewTransactionsD')
  App.setLoading(303)
},
// END OF DONOR FUNCS
```

```
setLoading: (stateNum) => {
  console.log("MOVING TO STATE")
  console.log(stateNum)
  if (App.mainState === 0) {
    document.getElementById('MAIN-PAGE').hidden = false
    document.getElementById('PROJECT-CREATOR').hidden = true
    document.getElementById('MODERATOR').hidden = true
    document.getElementById('DONOR').hidden = true
  }
  else if (App.mainState === 1) {
    document.getElementById('MAIN-PAGE').hidden = true
    document.getElementById('PROJECT-CREATOR').hidden = false
    document.getElementById('MODERATOR').hidden = true
    document.getElementById('DONOR').hidden = true

    document.getElementById('loader').hidden = true
    document.getElementById('createProjectHTML').hidden = true
    document.getElementById('viewMyProjectsHTML').hidden = true
    document.getElementById('viewAllProjectsHTML').hidden = true
    document.getElementById('viewAllAdminsHTML').hidden = true
    document.getElementById('withdrawFromProjectHTML').hidden = true
    document.getElementById('viewAllTransactionsHTML').hidden = true
    document.getElementById('submitProofHTML').hidden = true
    document.getElementById('viewReputationHTML').hidden = true
    document.getElementById('homeHTML').hidden = true
```

```
      if (stateNum == 101) {
        document.getElementById('loader').hidden = false
        App.loading = true
      } else if (stateNum == 102) {
        document.getElementById('createProjectHTML').hidden = false
      } else if (stateNum == 103) {
        document.getElementById('viewMyProjectsHTML').hidden = false
      } else if (stateNum == 104) {
        document.getElementById('viewAllProjectsHTML').hidden = false
      } else if (stateNum == 105) {
        document.getElementById('viewAllAdminsHTML').hidden = false
      } else if (stateNum == 106) {
        document.getElementById('withdrawFromProjectHTML').hidden = false
      } else if (stateNum == 107) {
        document.getElementById('viewAllTransactionsHTML').hidden = false
      } else if (stateNum == 108) {
        document.getElementById('submitProofHTML').hidden = false
      } else if (stateNum == 109) {
        document.getElementById('viewReputationHTML').hidden = false
      } else if (stateNum == 110) {
        document.getElementById('homeHTML').hidden = false
      }
    }
    else if (App.mainState === 2) {
      document.getElementById('MAIN-PAGE').hidden = true
      document.getElementById('PROJECT-CREATOR').hidden = true
      document.getElementById('MODERATOR').hidden = false
      document.getElementById('DONOR').hidden = true

      document.getElementById('modHomeHTML').hidden = true
      document.getElementById('validateTransactionHTML').hidden = true
      document.getElementById('viewModTokenHTML').hidden = true

      if (stateNum === 201) {
        document.getElementById('validateTransactionHTML').hidden = false
      } else if (stateNum === 202) {
        document.getElementById('viewModTokenHTML').hidden = false
      } else {
        document.getElementById('modHomeHTML').hidden = false
      }
    }
    else if (App.mainState === 3) {
      document.getElementById('MAIN-PAGE').hidden = true
      document.getElementById('PROJECT-CREATOR').hidden = true
      document.getElementById('MODERATOR').hidden = true
      document.getElementById('DONOR').hidden = false

      document.getElementById('allProjectDonorHTML').hidden = true
      document.getElementById('donateHTML').hidden = true
      document.getElementById('viewAllTransactionsDonorHTML').hidden = true
      document.getElementById('viewAllAdminsDonorHTML').hidden = true
      document.getElementById('donorHomeHTML').hidden = true

      if (stateNum == 301) {
        document.getElementById('allProjectDonorHTML').hidden = false
      } else if (stateNum == 302) {
        document.getElementById('donateHTML').hidden = false
      } else if (stateNum == 303) {
        document.getElementById('viewAllTransactionsDonorHTML').hidden =
false
      } else if (stateNum == 304) {
```

```
        document.getElementById('viewAllAdminsDonorHTML').hidden = false
      } else {
        document.getElementById('donorHomeHTML').hidden = false
      }
    }

  },
}

$(() => {
  $(window).load(() => {
    App.load()
  })
})
```

Submissions Overview

Background Information [what is this?]

Batch file name: initial draft team 11.pdf

Report generated on: 15/04/2021, 01:12:39 PM

Checking Parameters [what is this?]

Matching scope(s): Within submission, Internet

Leniency: Detailed matching with threshold 70%

Minimum sentence length: Sentences with more than or equal to 3 meaningful words were checked

Similarity Statistics

Similarity Statistics [what is this?]

Total number of documents: 1

Number of documents which can be processed: 1

Number of documents which cannot be processed: 0

Show 10 entries Search:

Entry	Document	Status	Similarity	Action
1	initial_draft_team_11.pdf	processed	107/844=12.60%	View details

Showing 1 to 1 of 1 entries First Previous 1 Next Last