IBC Final Report

Bhavik 2018385
Hardik 2018391
Ritvik 2018407

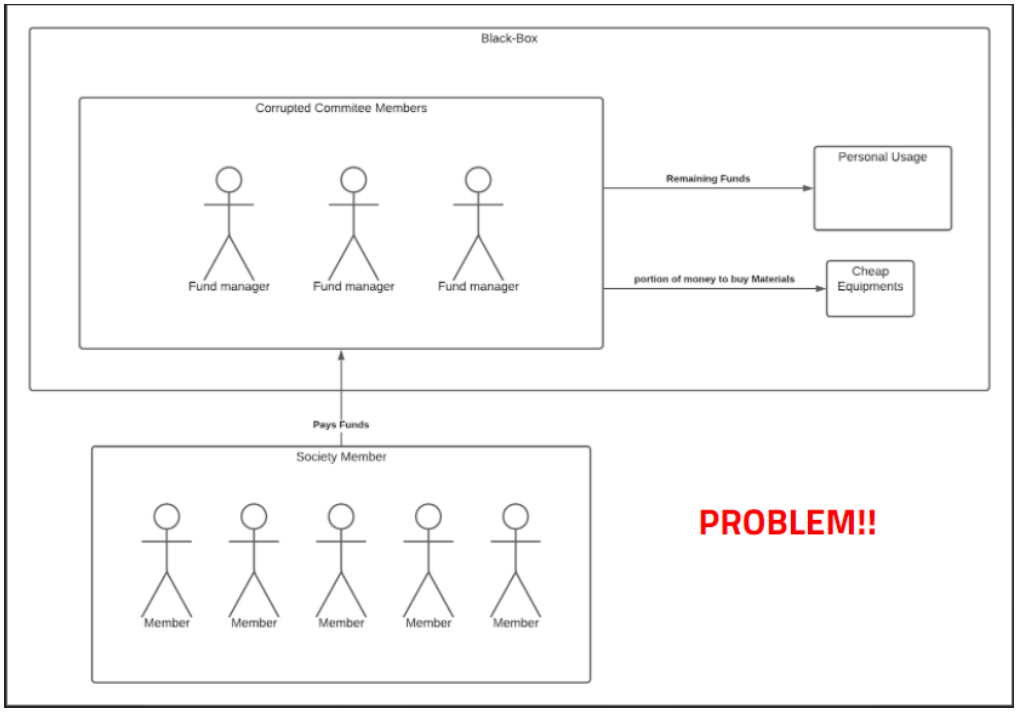
Initial Problem Statement and Solution

Modern Societies collect **Cooperative funds** from members, which are then later used to maintain and organize events in societies, So the organization looks something like in images below:

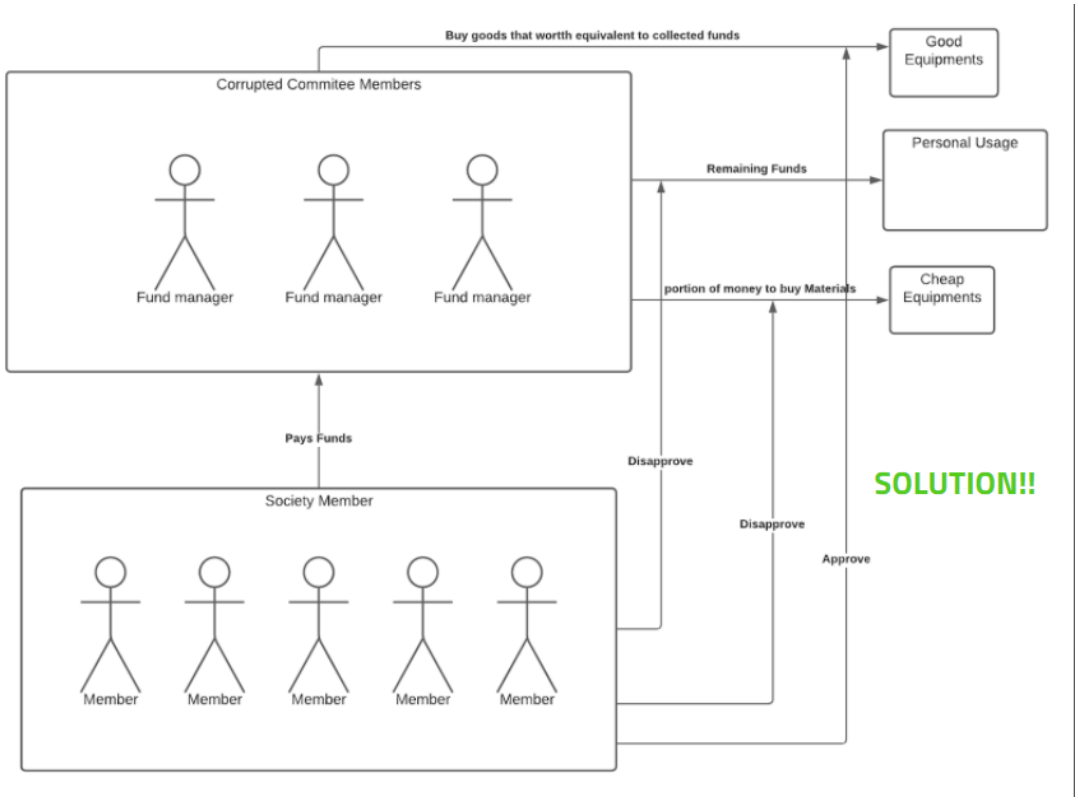
For example, Most of the people in urban cities live in societies. In a housing society, a small group of people who are generally the members of the society, are elected as the welfare committee, who have the following responsibilities:

1) Collect monthly maintenance from all members of the society and storing it as society funds.
2) Making decisions on where this fund would be used i.e. using the society funds for expenses related to regular maintenance-related works and giving salaries to all the employees like electricians, plumbers, cleaners, etc.
3) Managing various monthly/festival/cultural events, and all the expenses done by the committee would be funded by the contributions of the society members and a new fund would be created by the committee for every new event.

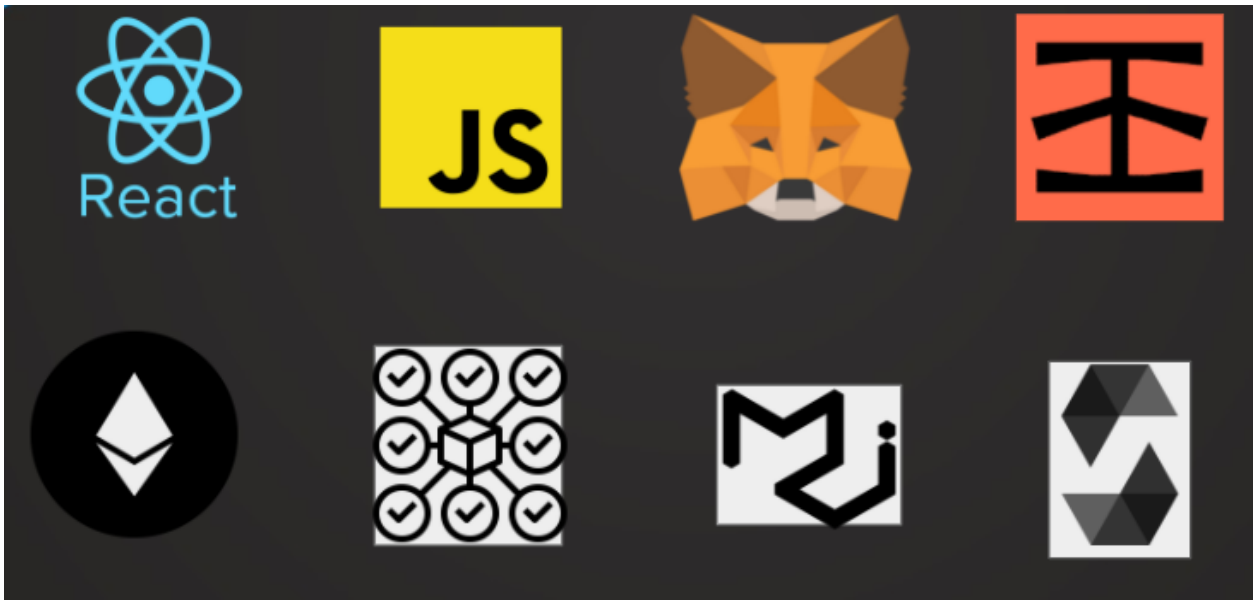
Now every member of society would like to have an assurance that all the money they are contributing is being spent in a fair manner, free of corruption. Hence there is a need for a robust and transparent system where all the expenditure done by the committee is visible to all the contributors of the fund.



We have implemented a DAPP application for the above problem in which the funds could only be approved after it has been verified by the majority of the society members. Basically, each and every time transaction would only take place, after the majority of the members give permission. So in this way, every society member would be aware where their money is being used and frauds could be avoided.



Tech Stack



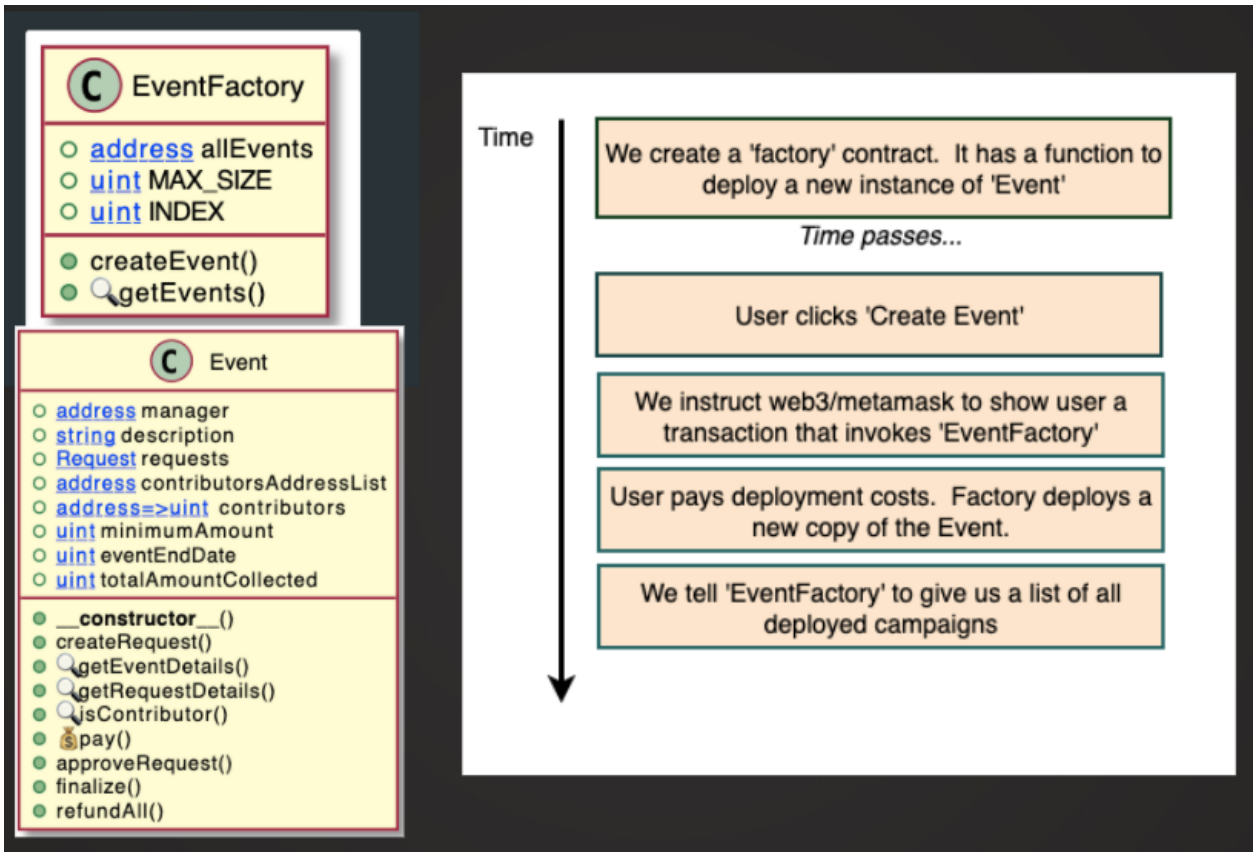
Deployment of contract on Rinkeby Test Network - Rinkeby is an Ethereum test network that allows for blockchain development testing before deployment on Mainnet, the main Ethereum network. As we dont want to spend “real” etherium, that's why we are using Rinkeby.

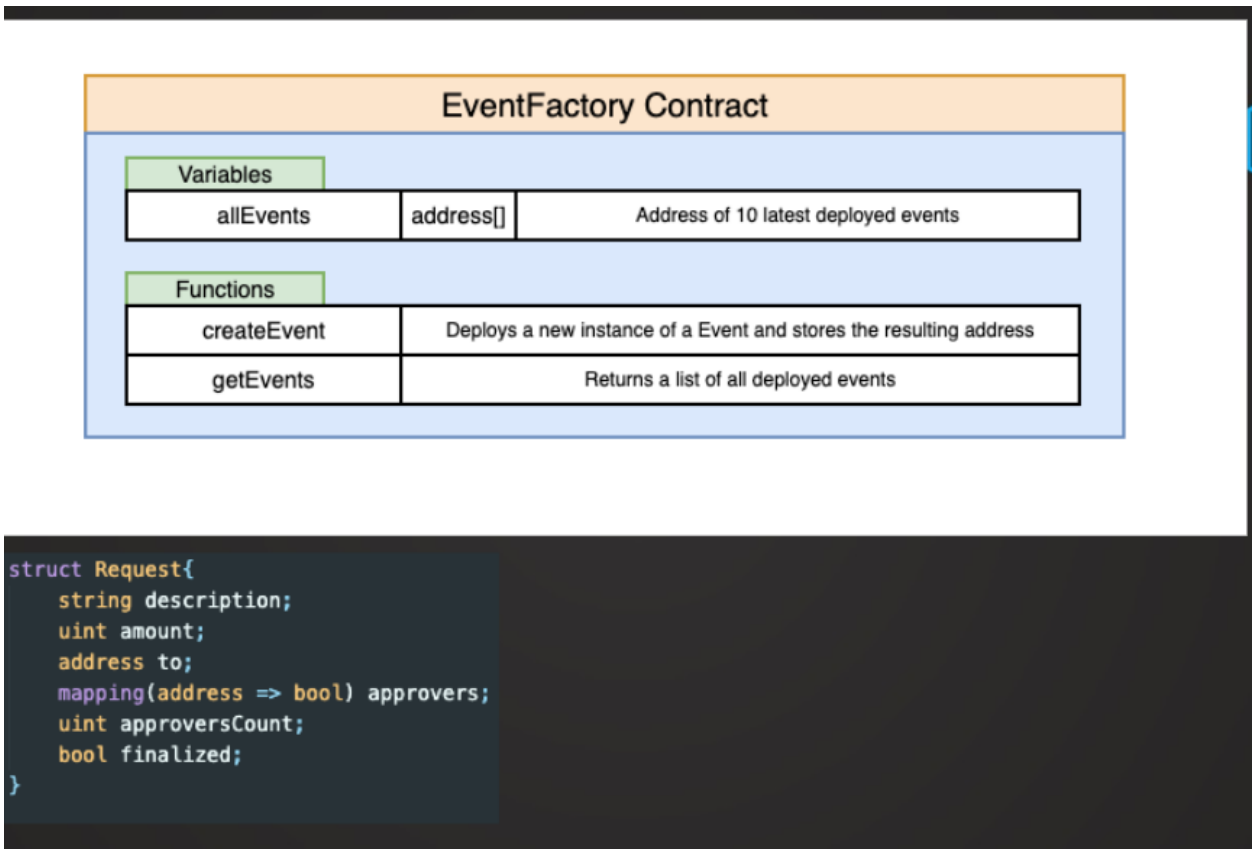
React - Frontend - React. js is an open-source JavaScript library that is used for building user interfaces specifically for single-page applications.

Web3 - web3 applications either run on blockchains, decentralized networks of many peer to peer nodes (servers), or a combination of the two that forms a crypto economic protocol. These apps are often referred to as dapps (decentralized apps), and you will see that term used often in the web3 space.

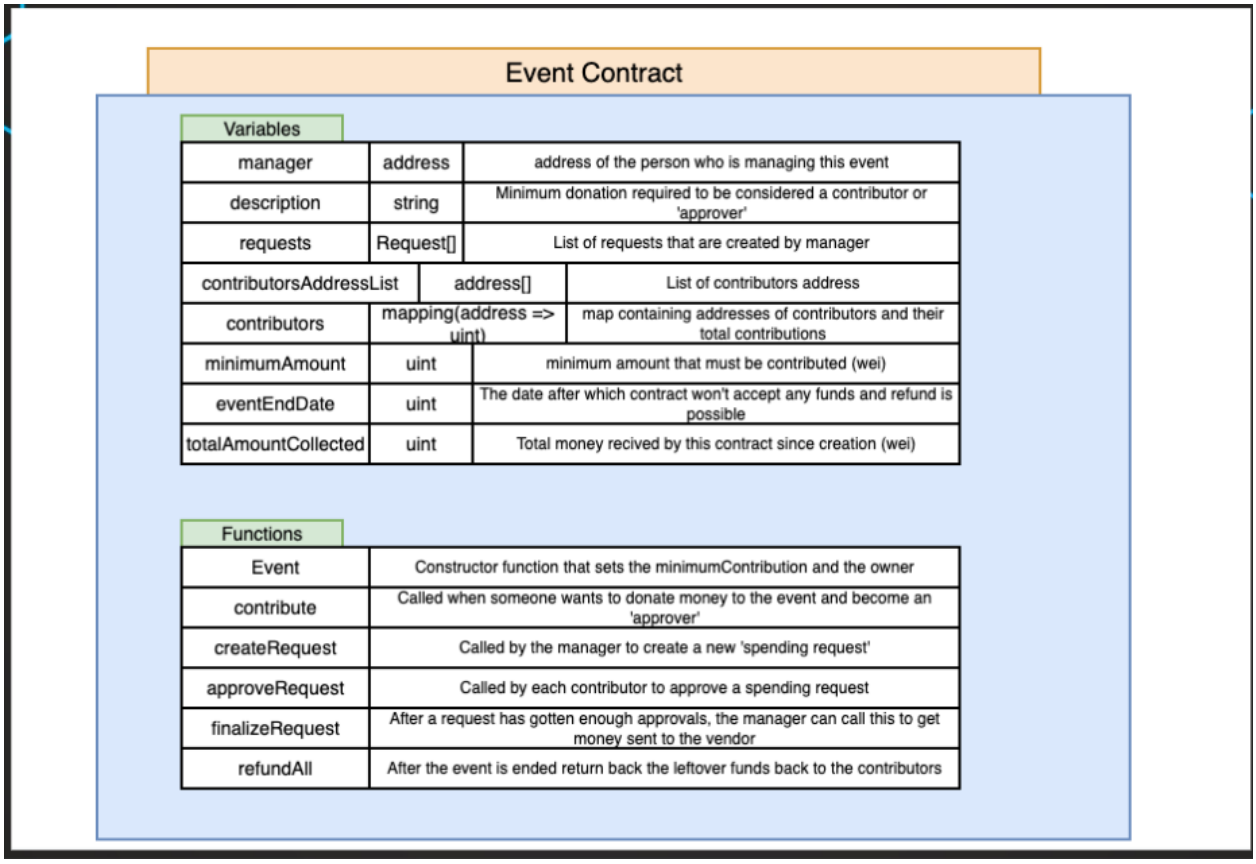
Solidity (Programming language) for contract - Solidity is an object-oriented programming language for writing smart contracts. It is used for implementing smart contracts on various blockchain platforms, most notably, Ethereum.

Smart Contract






```
struct Request{
    string description;
    uint amount;
    address to;
    mapping(address => bool) approvers;
    uint approversCount;
    bool finalized;
}
```



A "smart contract" is simply a program that runs on the Ethereum blockchain. It's a collection of code (its functions) and data (its state) that resides at a specific address on the Ethereum blockchain.

Smart contracts are a type of Ethereum account. This means they have a balance and they can send transactions over the network. However they're not controlled by a user, instead they are deployed to the network and run as programmed. User accounts can then interact with a smart contract by submitting transactions that execute a function defined on the smart contract. Smart contracts can define rules, like a regular contract, and automatically enforce them via the code. Smart contracts can not be deleted by default, and interactions with them are irreversible.

Solidity (Programming language) for contract - Solidity is an object-oriented programming language for writing smart contracts. It is used for implementing smart contracts on various blockchain platforms, most notably, Ethereum.

This is our smart contract above how it looks.

Functional Requirement

This section covers the formal Functional requirements that will be provided by our software. All these requirements were kept in mind while designing and coding the smart contract.

Event Creation

- Any valid user is able to create an Event (Create an instance of the contract)
- The person who creates instance will be referred to as the *manager* of this *Event*
- Manager is the person who is in power/incharge of the event
- Manager is supposedly responsible for handling the transfer of funds
- Manager while creating the instance/Event can set
    - A description for the event
    - Minimum amount required to contribute to become a contributor
    - Ending date for this event

Contributing to contract

- A user is able to contribute to the event by sending some money to the contract
- The user must send x units of money which must be greater than the minimum money that is set by the manager while creating the instance
- Once the user has paid minimum amount of money, then he/she will be referred to as contributor
- The money can be contributed only before the event end date, no contribution will be accepted after event has ended

Creating fund expenditure request

- The only way to spend money that has been collected by the contract is by creating a fund expenditure request
- The contract manager can create a request to spend money that is collected by the event
- This money should ideally be used to pay for event management (tent, food etc)
- While creating the request the manager should specify
    - Description about what is the purpose of this request (e.g. money for purchasing food)
    - Amount that will be spent to fulfil this request
    - Address of the person who is supposed to receive this money

Approving an request

- Once a fund expenditure request has been posted by the event manager it will be visible to all of the contributor
- The contributor can approve the fund expenditure request
- Each contributor approval for request will be counted at most once, irrespective of the amount he/she have contributed

- A request can only be approved by an contributor

Finalizing Request

- A fund expenditure request is said to be ready to be finalized once it has been approved by at least 50% of the total contributors count
- Once more than 50% contribution is done, the manager has to click on finalize request which will initiate and the person who was supposed to receive this money (the one who was set at the time of creating this request) will get that money
- A request can only be finalized at most once and that also is possible only when 50% approval is received

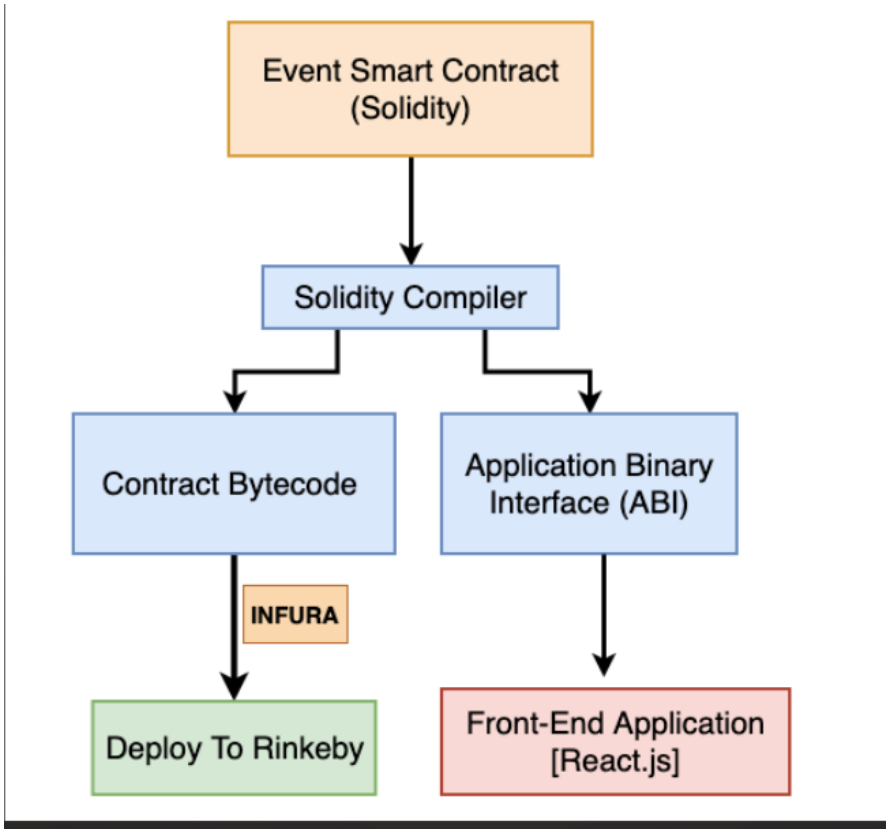
Event Expiry

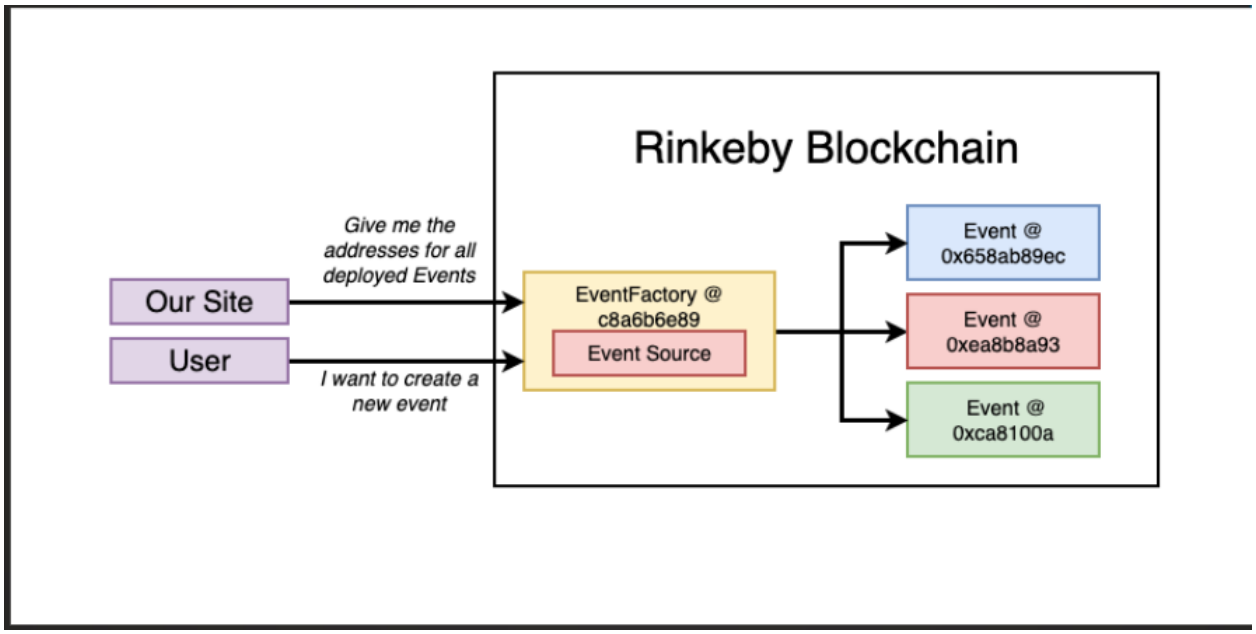
- The manager when created a contract specified an event end date
- The contract won't accept any money once the event end date has passed, no more contributions can be made.
- Upon event expiry, no more funds can be transferred
- Upon event ending no more request can be approved
- Refund of money is only possible when the event end date has already passed away i.e. event has expired

Refund all

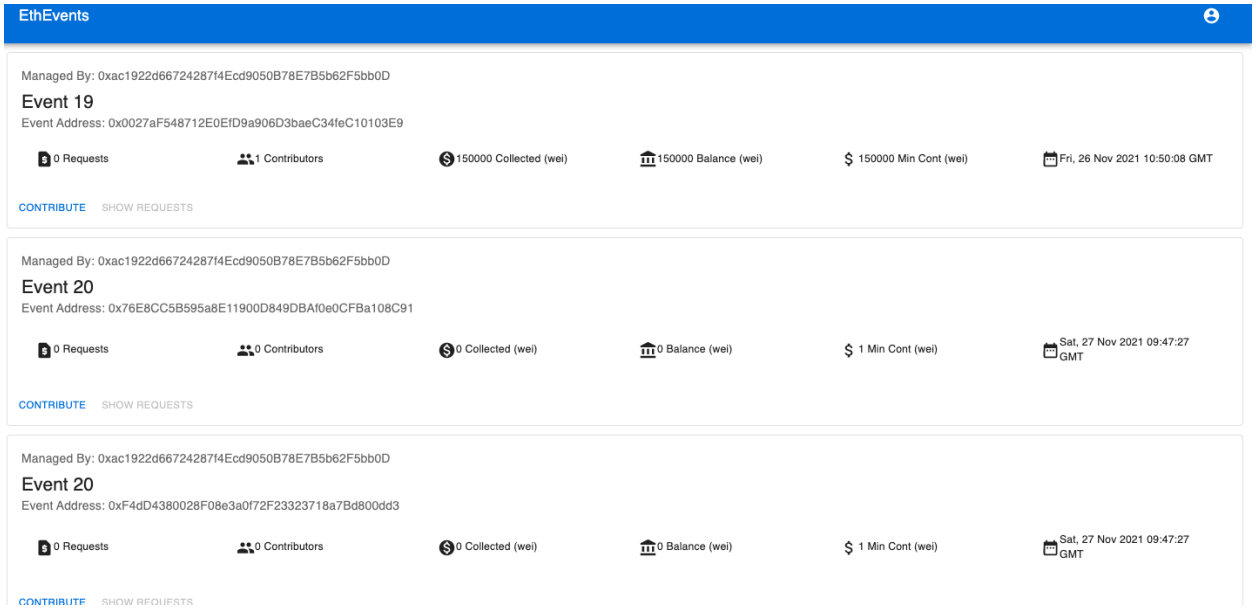
- Once the event has expired any of one of the contributor can initiate a refund all request
- Once this request is initiated all the contributors will receive the money that is proportionate to the amount they have contributed to the event i.e. on the basis of percentage
- Example
    - Suppose you pay 10 Rs to an Event A & it has total contribution of 100Rs, so in principle you have made 10% contribution to the total amount
    - Now suppose after the event has expired the contract has only 10Rs remaining balance (rest all is used for managing event)
    - So now upon refund, you will receive 10% of the remaining balance which is 1Rs, this 1Rs will be refunded to you account

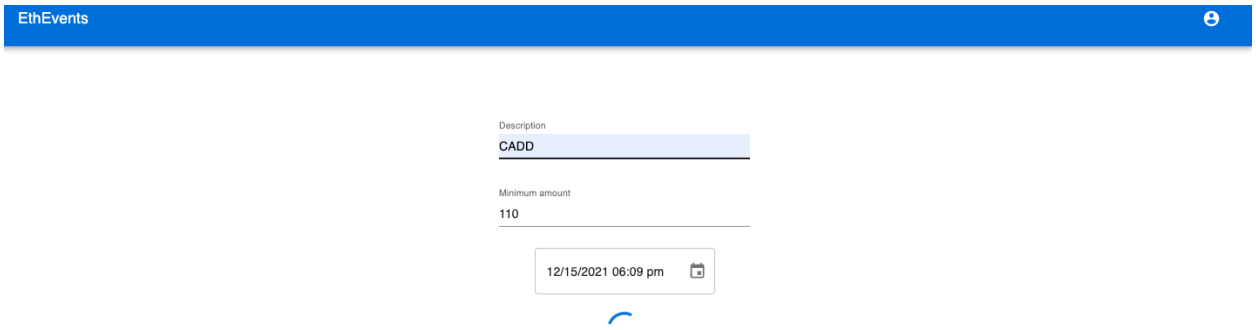
Compilation and Deployment



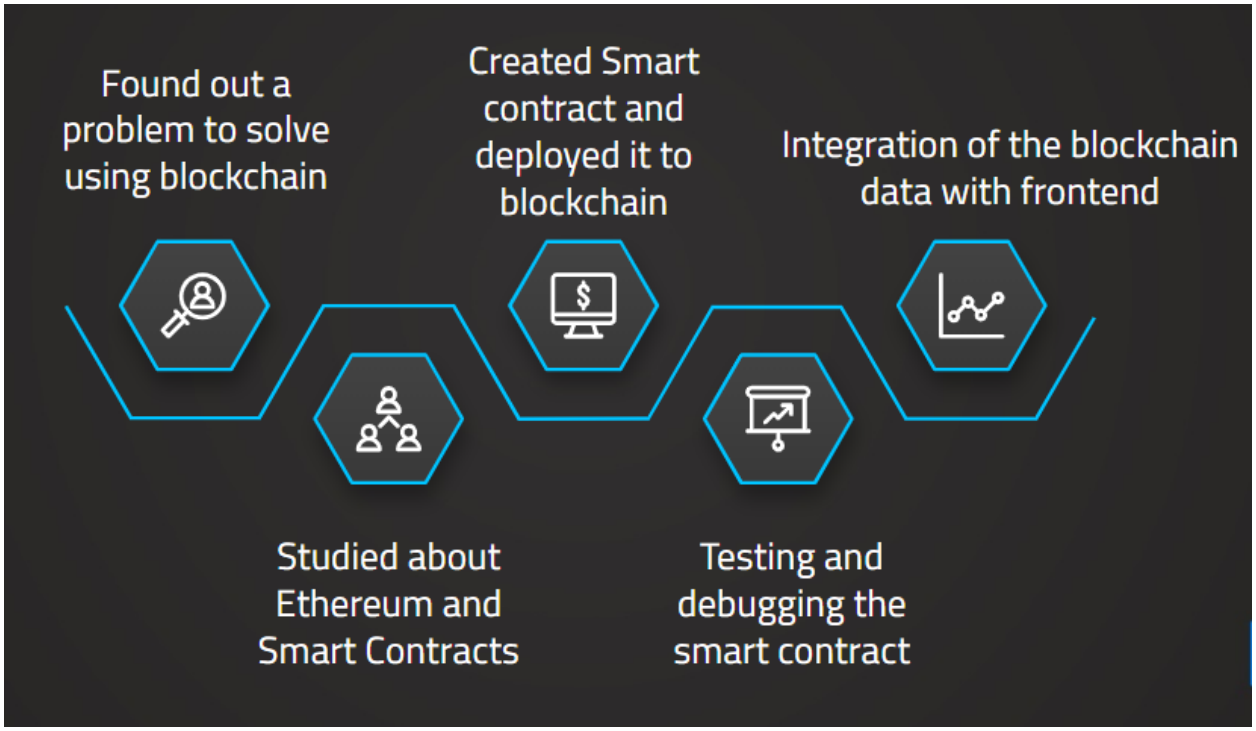


Some Screenshots from our project:





Workflow



Challenges faced

- Outdated Documentation
    - Ethereum and blockchain are relatively newer technologies
        - Ethereum was launched in 2015
    - Things and frameworks like versions of Web3, APIs endpoints and framework such as truffle are changing very rapidly.
    - So most of the problems that we searched on google (programmatic errors) had outdated solution so they didn't work

- Example writing code for **deploy.js** and **compile.js** took extensive research

Team Work

Work/Responsibilities Distribution:

#	Responsibilities	Person
1	Understanding Basics of Blockchain, React Frontend, Software requirement specification, Blockchain Development, Testing Blockchain Code	Hardik
2	Understanding Basics of Ethereum, Front-End event Handling, deciding Application policies/Rules, Smart Contract development, Testing Smart Contract Code	Bhavik
3	Understanding Basics of smart-contracts, Connection with Backend, Designing, Smart Contract & Blockchain Integration with Application, Testing Whole Application	Ritvik

Link to our LucidChart Project design:
https://lucid.app/lucidchart/942725e3-42d3-4b7b-a930-0a19a7997e33/edit?view_items=Dpfdbog_xAH-&invitationId=inv_96a46517-b5f0-4593-a0b8-b1dbb3db5efd

GITHUB Link to our project:
https://github.com/Guardianofgotham/IBC_2018391_2018407_2018385