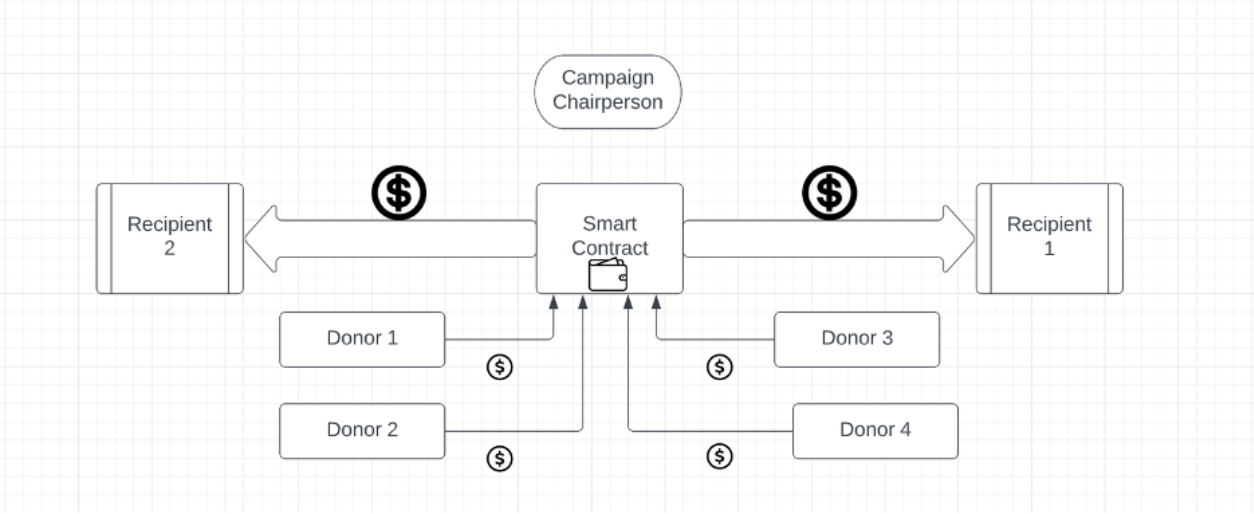
Trust Your Fund

Mohammed Junaid Shaik – mshaik6
Naman Agrawal - namanagr

Abstract:

The main goal of this project is to bring transparency to the incoming and outgoing funds of a Fund (eg: relief/cause funds). How we plan on implementing this is by allowing every person who donates money to see how and where the money that has been donated is flowing. This can be achieved by introducing a shared ledger for the fund that shows the balances, expenditure and the income. The tech-stack used for creating this dApp is Solidity, Truffle, React, Typescript and Web3.js.

Design Overview:

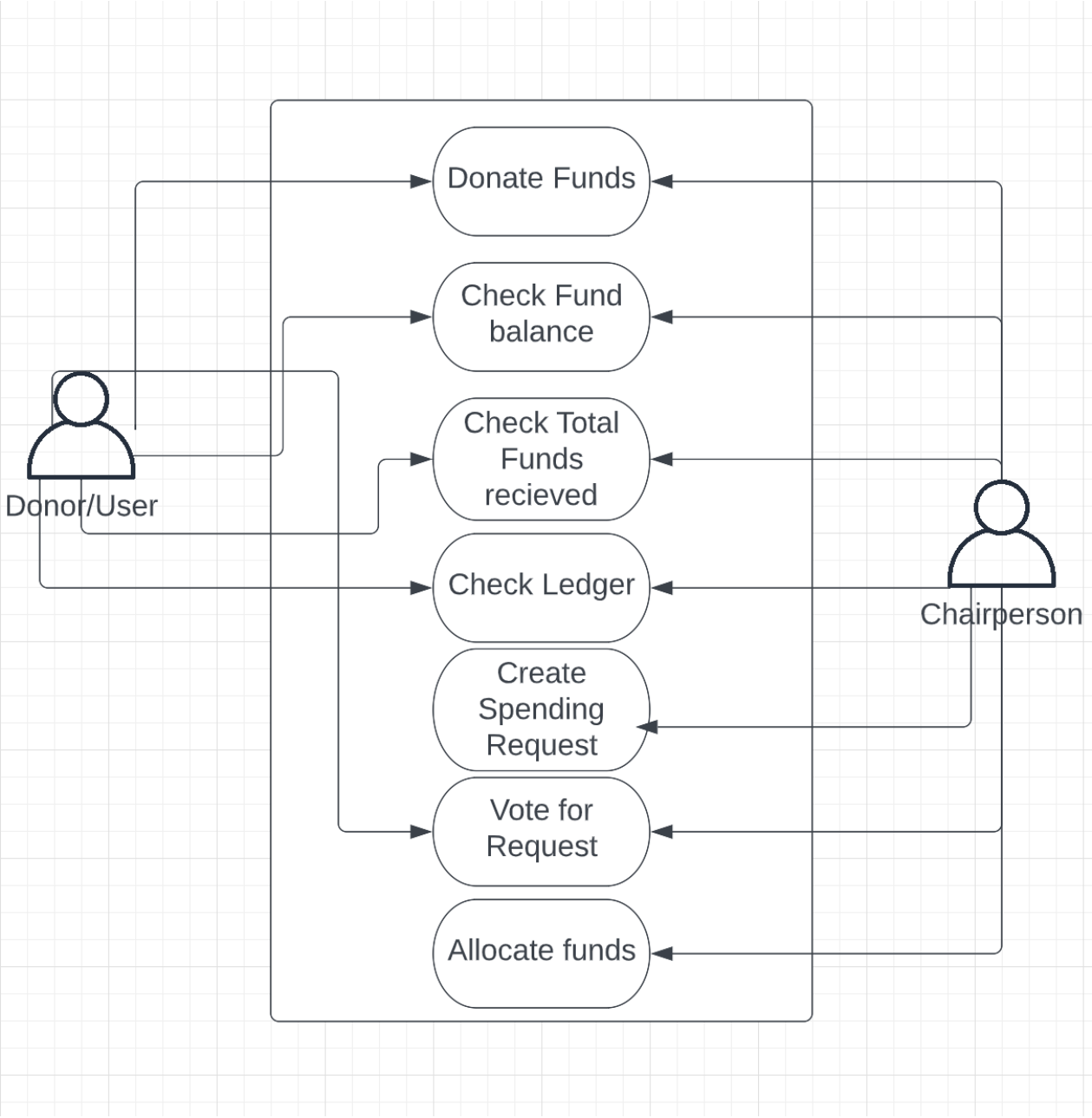


As we can see from the diagram above, we have three different types of users who will be accessing our system. They are:

- **Campaign Chairperson**: This is owner of the smart contract. It is also this entity that decides where the money that is collected goes. This can be a single person or a company who takes a collective decision.
- **Donors**: These are the people who donate in sympathy with the cause. These people can at any point view the current balance of the campaigns, donate to the and see where all the money donated to the campaign has been used.
- **Recipients**: Chairperson creates payment requests to said recipients from the fund which are approved by the voters(contributors).

Apart from this, there is one more actor in our application, the **smart contract**. The smart contract governs the receival and the spending of money and holds the money in a wallet.

Use case UML:



The above UML diagram shows the use case diagram which shows all the three users who are using the application. The Donor can donate funds, check the fund balance, know how much was donated in total and see a list of all the transactions. A donor also has the right to vote for any spending request. The Chairperson is the entity that creates the spending request and allocates the funds to the recipient once approved by the donors.

Contract Diagram:

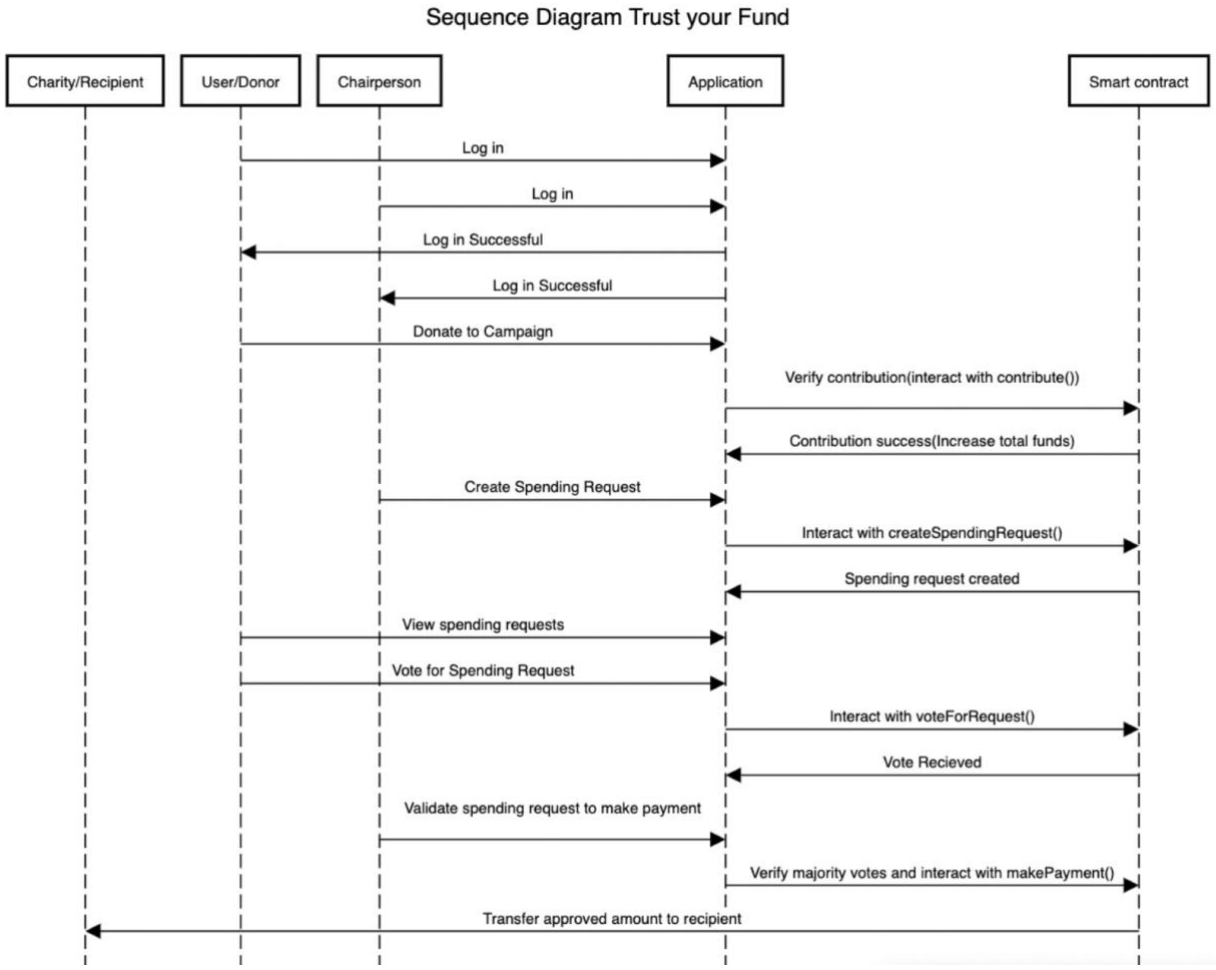
The first section of the contract diagram contains the contract name. The second section contains the data definitions or digital assets that have been declared. The third section contains a list of validations (modifiers). The fourth section contains a list of all the functions in the smart contract. Following is the contract diagram:

TYF

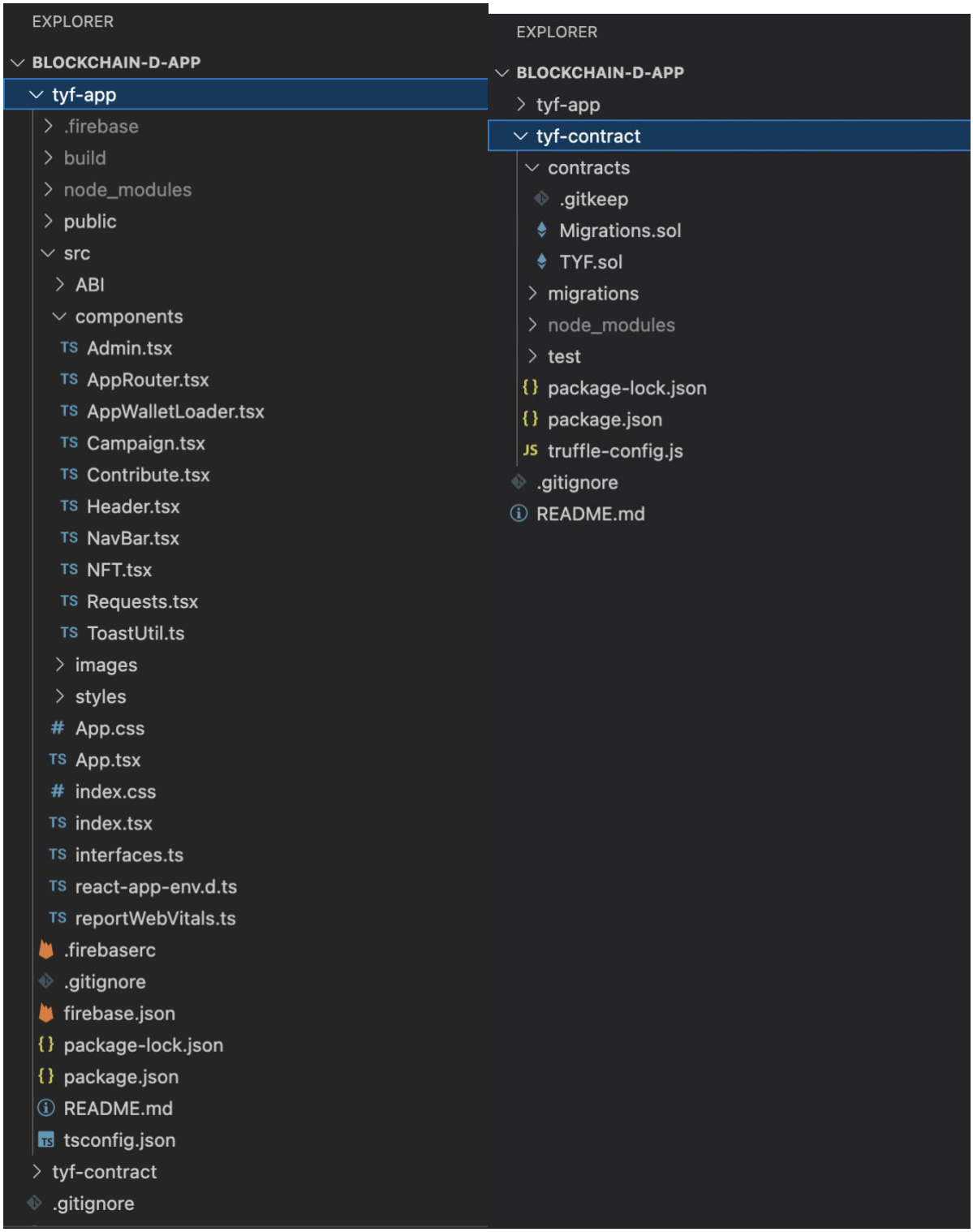
mapping(address => uint256) public contributions;
uint256 totalContributors;
uint256 minimumContribution;
uint256 raisedAmount;
address public admin;
struct Request {
string description;
uint256 value;
address recipient;
bool completed;
uint256 numberOfVoters;
mapping (address => bool) voters;
}
Request[] requests;

onlyAdmin () contributor ()
contribute () getFundBalance () getUniqueContributors () getLengthRequests () createSpendingRequest (string memory _description, address _recipient, uint256 _value) voteForRequest (uint256 index) makePayment (uint256 index)

Sequence Diagram:

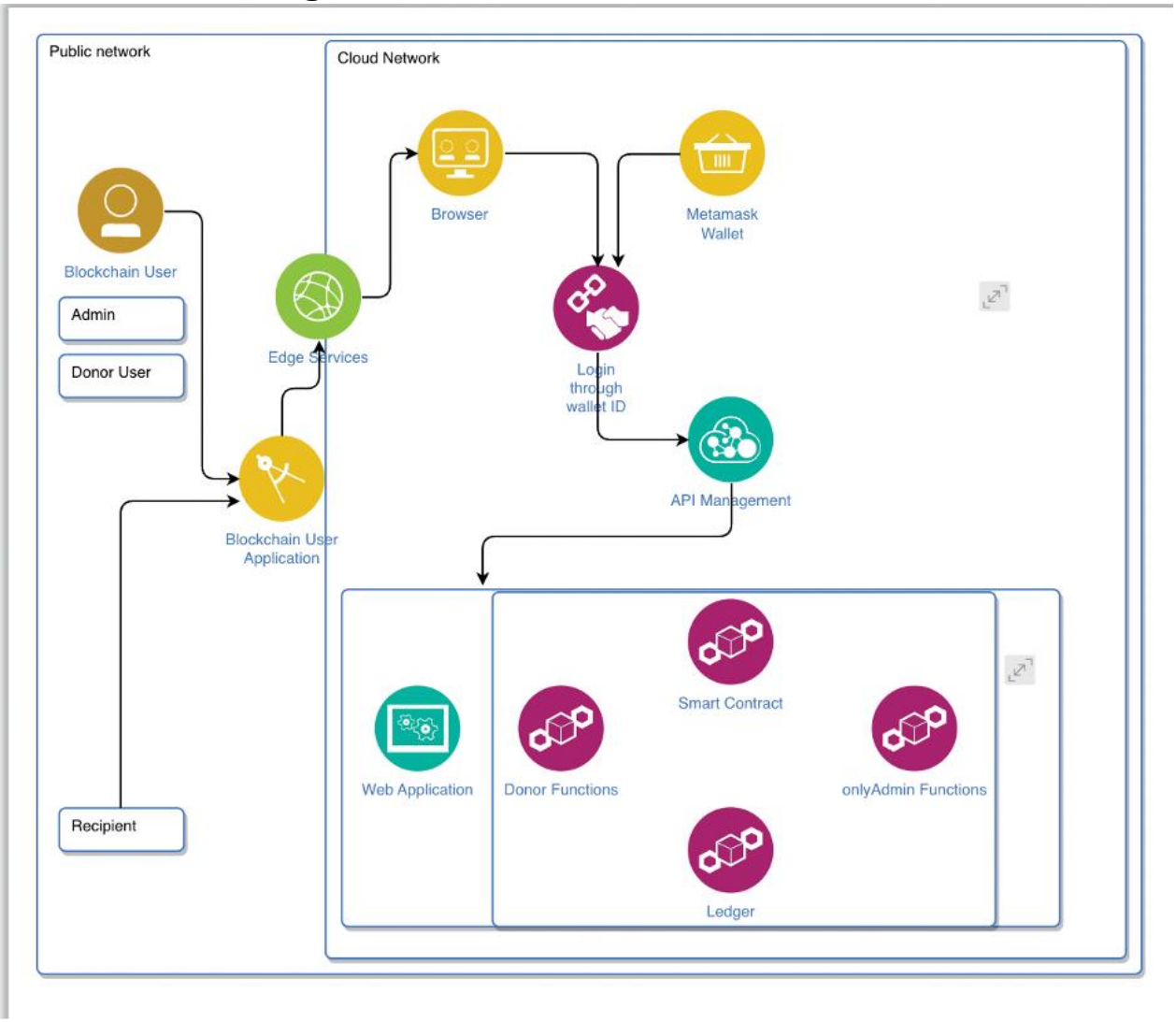


File Structure:



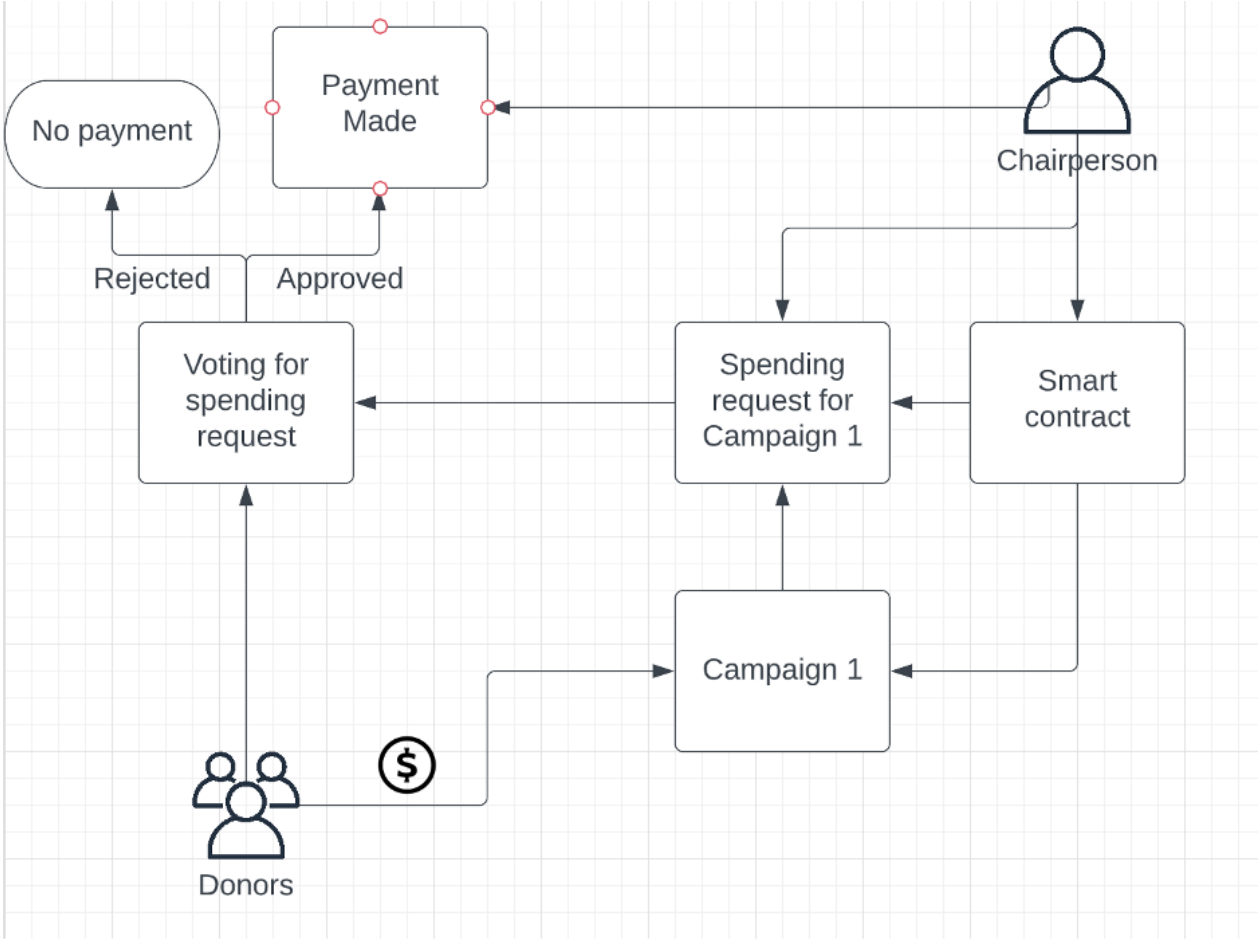
We have two main folders: tyf-app and tyf-contract. The former hosts the code for the web3 application, which includes all the web pages and some application layer logic. This has all the user facing code. The latter folder has the smart contract and other self-generated files from truffle.

Architecture Diagram:



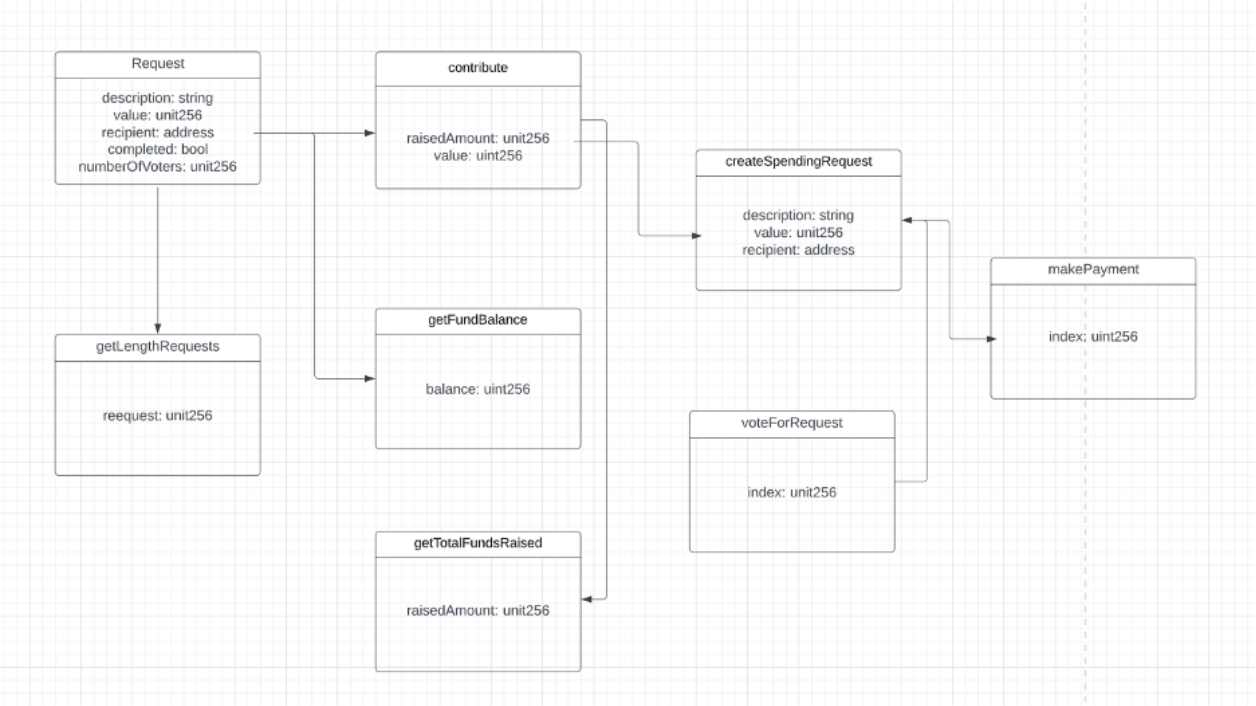
We have all the users of the application, namely the admin, donors and recipients in any public network. They interact with the application which through a browser and their metamask wallet through which they will be donating and receiving money. With their metamask wallet, once the authentication is done, they can log in to the application and access all the functionalities.
The smart contract and the ledger are part of a website which can be hosted on any cloud server, which can be authenticated by our web APIs.

Decision Flow:



The above UML diagram shows the flow of our application.
The contract governs the creation of campaigns in the first step. Once a campaign is created, it will be available to users for them to choose and donate to. The total fund balance is derived as a sum of all the money contributed to the campaigns wholly. The users will get the amount of money raised for a particular cause. Once a campaign receives enough money, the chairperson can create a spending request which can be validated by users of the blockchain by using the Voting option, and once that is approved, the payment is made. Till the number of votes are not approved by the users, the spending request remains stagnant.

Smart contract UML:



The various functions and modifiers used in our smart contract are as follows:

- Contribute: This is the main payable function which takes care of accepting donations by checking whether the amount being contributed is above the minimum contribution level set by the admin (more checks can be introduced) and if it satisfies the criteria, it will take in the funds and add it to the wallet.
- getFundBalance: Returns the amount of funds which are available to be spent i.e the remaining funds.
- getTotalFundsRaised: This tells us the total amount of money that has been raised till now by the smart contract (our application as a whole) in the entirety of the campaign
- getUniqueContributors: Gets the unique number of people who have donated to the fund.
- createSpendingRequest: This is created whenever we want to allocate collected funds towards a particular vendor/recipient. This is an onlyAdmin function, in terms that any public user cannot use this function.
- getLengthRequests: It is a public function which returns the number of requests which are available for donation. (We use this in conjunction with the public requests hashmap to display all the requests to the user in the application)
- voteForRequest: This is a public function available to all the users who are using the application. Using this the users can vote for and authorize the usage of funds to a particular recipient.
- makePayment: This function is also an onlyAdmin function which is used by the chairperson to finally allocate any funds which have been voted and approved by the users of the application.

Interact with the D-Application/Smart-Contract:

We recommend you interact with our app at https://tyf-dapp.web.app/, and not remix as we have tried to provide a simple intuitive user experience.

However, if you prefer to test the entirety of the methods of the smart contract, remix is recommended.

Screenshots of the TYF D-App and steps do deploy the app as your own are also provided later in this document.

Instructions to test the Smart Contract on Remix:

- Copy contents of the file "/tyf-contract/contracts/TYF.sol"
- Go to https://remix.ethereum.org and create a new file called "TYF.sol"
- Paste the contents in the file created
- Find and replace "../node_modules/" with ""
- Navigate to Solidity Compiler and Compile the contract. Proceed to next step once successful.
- Navigate to Deploy & Run Transactions, select "Injected Provider – Metamask" from the dropdown for "Environment". (Ensure you connected to Goerli net on Metamask).
- Now, there are two options, you can either test with smart contract associated with our deployed TYF d-app or you can choose to deploy the smart contract on your own and test it.
    - Option 1: Copy and paste "0x8d2a760cEDBe688d788bD9650Db5bC31688862fB" into the "At Address" section and click. You will see a deployed contact available below.
    - Option 2: Click on "Deploy". You will see a deployed contact available below.
- Select the deployed contract and should see all the available variables and methods for interacting with the smart contract.

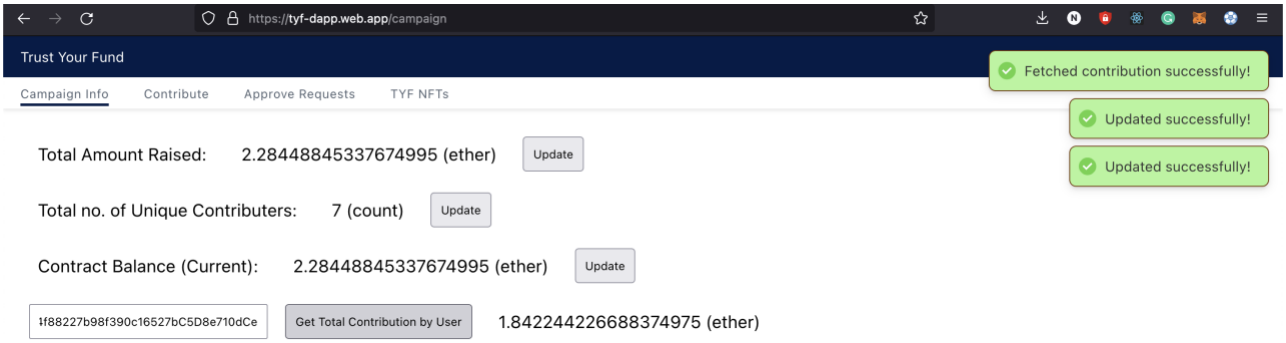
Key Methods to Interact with:

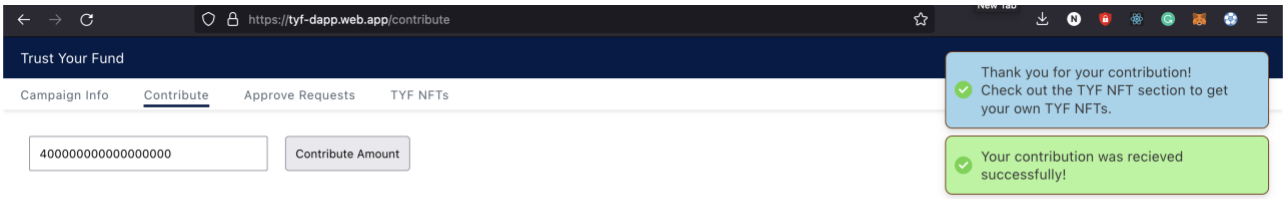
- contribute ()
- voteForRequest (uint256 index)
- makePayment (uint256 index) onlyAdmin
- createSpendingRequest (string memory _description, address _recipient, uint256 _value) onlyAdmin
- safeMint (address to, string memory uri) contributor
- transferNFT (address from, address to, uint256 tokenId) contributor

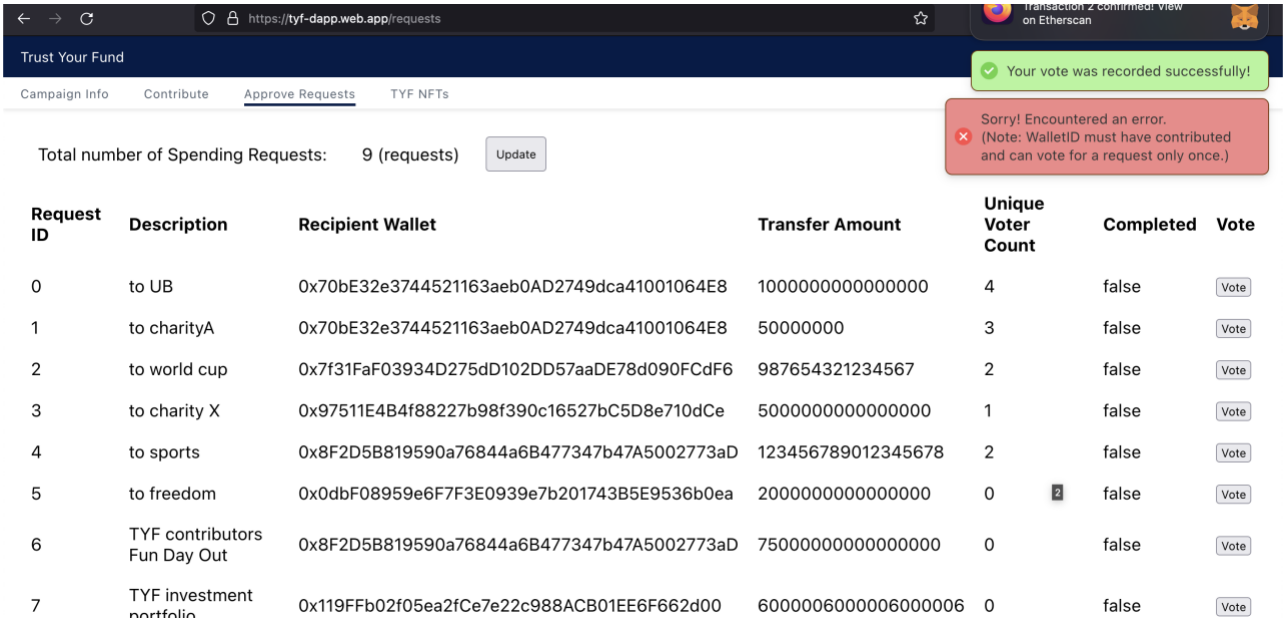
End-to-End Instructions to Compile, Build and Deploy the TYF D-Application as your own D-Application:

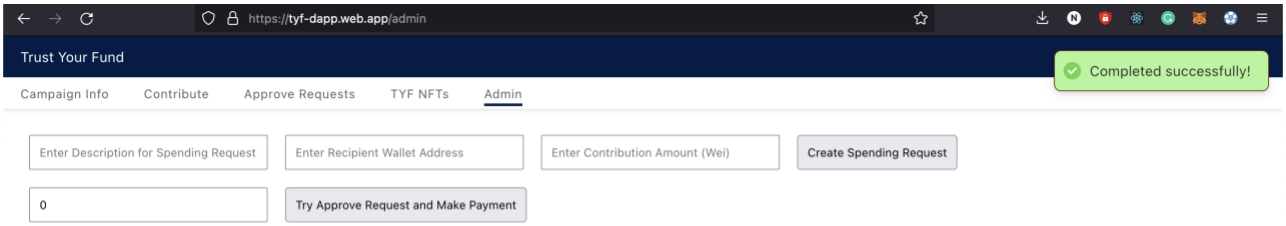
- Pre Requisites to be installed:
    - Browser with Metamask Plugin (recommended: https://www.mozilla.org/en-US/firefox/new/ && https://addons.mozilla.org/en-US/firefox/addon/ether-metamask/)
    - Node Package Manager (https://www.npmjs.com/package/npm/v/8.19.2)
    - Truffle Suite (https://trufflesuite.com/docs/truffle/how-to/install/)
    - Firebase CLI (https://firebase.google.com/docs/cli)
- Update the variables __MEMONIC__ and __INFURA_KEY__ with your own keys in the file "/tyf-contract/truffle-config.js"
- Start a terminal session in the root directory
- $ cd tyf-contract/
- $ npm install
- $ truffle compile
- $ truffle migrate --reset --network goerli
- $ cd ../tyf-app/
- $ npm install
- To run the app as a local server do the following, else skip to next step directly:
    - $ npm start
    - The app session should start in your browser automatically. Else visit localhost:3000/
    - Your app is ready to interact, follow the below to steps for production build and hosting.
- $ npm run build
- $ npm install -g firebase-tools
- Log in to the Firebase Console (https://firebase.google.com/) and create a new project.
- $ firebase login
- You will be redirected to a Google Firebase sign-in page. Proceed to next step on success.
- $ firebase init
- When you run this command, you will have to answer a few questions like the following:
    - Which Firebase CLI features do you want to set up? -> Select "Configure files for Firebase Hosting"
    - What do you want to use as your public directory? -> build
    - Configure as a single-page app (rewrite all urls to /index.html)? -> y
    - File build/index.html already exists. Overwrite? -> N
- $ firebase deploy
- Once you run this command, Firebase will then give you a unique hosting URL where your deployed app is located. (For example: https://tyf-dapp.web.app/)
- Visit the URL and interact with the Application!

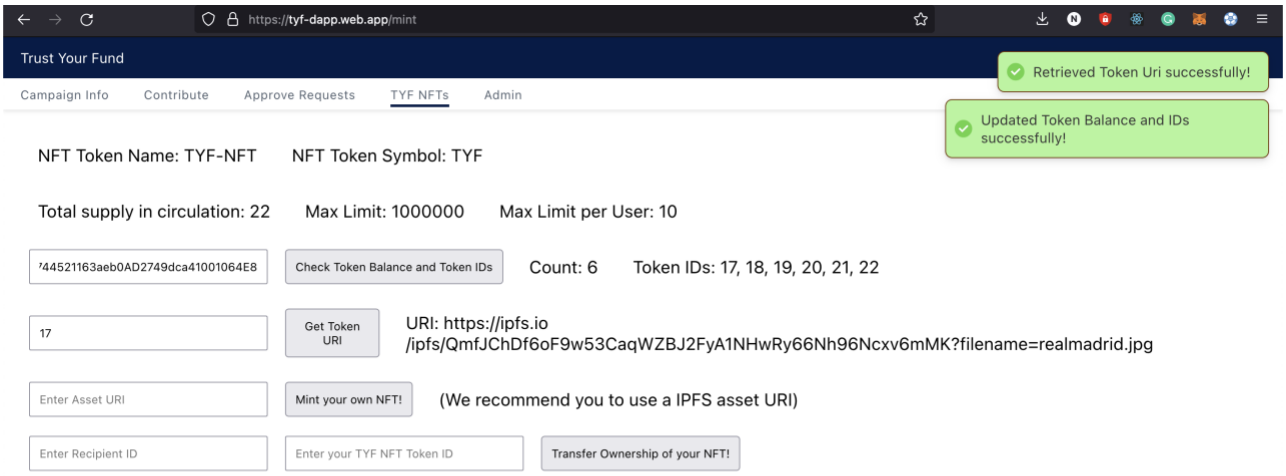
Screenshots:

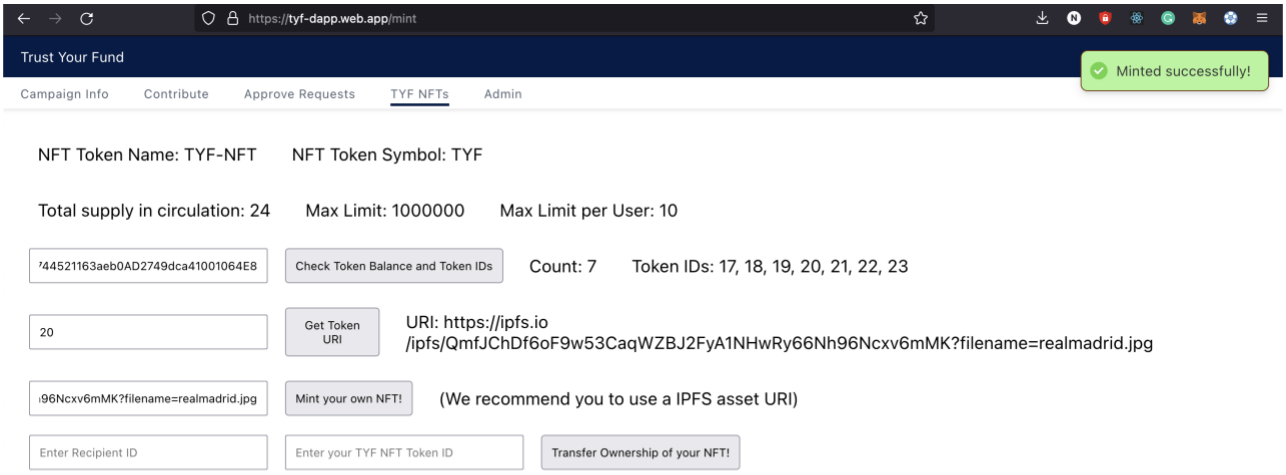







https://tyf-dapp.web.app/admin
Trust Your Fund
Completed successfully!
Campaign Info
Contribute
Approve Requests
TYF NFTs
Admin
Enter Description for Spending Request
Enter Recipient Wallet Address
Enter Contribution Amount (Wei)
Create Spending Request
0
Try Approve Request and Make Payment


https://tyf-dapp.web.app/mint
Trust Your Fund
Retrieved Token Uri successfully!
Updated Token Balance and IDs successfully!
Campaign Info
Contribute
Approve Requests
TYF NFTs
Admin
NFT Token Name: TYF-NFT
NFT Token Symbol: TYF
Total supply in circulation: 22
Max Limit: 1000000
Max Limit per User: 10
744521163aeb0AD2749dca41001064E8
Check Token Balance and Token IDs
Count: 6
Token IDs: 17, 18, 19, 20, 21, 22
17
Get Token URI
URI: https://ipfs.io /ipfs/QmfJChDf6oF9w53CaqWZBJ2FyA1NHwRy66Nh96Ncxv6mMK?filename=realmadrid.jpg
Enter Asset URI
Mint your own NFT!
(We recommend you to use a IPFS asset URI)
Enter Recipient ID
Enter your TYF NFT Token ID
Transfer Ownership of your NFT!


https://tyf-dapp.web.app/mint
Trust Your Fund
Minted successfully!
Campaign Info
Contribute
Approve Requests
TYF NFTs
Admin
NFT Token Name: TYF-NFT
NFT Token Symbol: TYF
Total supply in circulation: 24
Max Limit: 1000000
Max Limit per User: 10
744521163aeb0AD2749dca41001064E8
Check Token Balance and Token IDs
Count: 7
Token IDs: 17, 18, 19, 20, 21, 22, 23
20
Get Token URI
URI: https://ipfs.io /ipfs/QmfJChDf6oF9w53CaqWZBJ2FyA1NHwRy66Nh96Ncxv6mMK?filename=realmadrid.jpg
96Ncxv6mMK?filename=realmadrid.jpg
Mint your own NFT!
(We recommend you to use a IPFS asset URI)
Enter Recipient ID
Enter your TYF NFT Token ID
Transfer Ownership of your NFT!