Software Requirements Specifications

Version 1.0

4 September, 2025

Usaeed Rahman Tibro
24141123
usaeed.rahman.tibro@g.bracu.ac.bd

Contents

1. Introduction

CryptoFundHub is a decentralized crowdfunding platform built on blockchain technology. Unlike traditional platforms, it ensures transparency, trust, and security by leveraging Ethereum smart contracts. Campaign creators can launch funding campaigns directly on the blockchain, while contributors can fund projects using crypto assets (ETH on Sepolia).

The platform integrates **Thirdweb SDK** for Web3 tooling, **Next.js** for the frontend, and **Solidity smart contracts** for decentralized logic.

1.1 Key goals:

- Provide transparency in how funds are collected and distributed.
- Allow creators to launch campaigns trustlessly.
- Ensure that only the intended campaign creator can withdraw funds.
- Enable global participation without intermediaries.

1.2 Project Summary

CryptoFundHub is a decentralized crowdfunding platform built on the Ethereum blockchain using Solidity and Chainlink. The platform enables users to securely contribute funds in Ether (ETH) toward campaigns while enforcing a minimum USD-equivalent contribution through real-time price conversion. By leveraging smart contracts, CryptoFundHub ensures transparency, automation, and trust—eliminating the need for traditional intermediaries.

At its core, the platform allows individuals to fund campaigns directly from their crypto wallets. Each contributor's address and donation amount are securely recorded on-chain. All transactions are governed by verifiable, immutable smart contract logic. The platform uses Chainlink oracles to fetch accurate ETH/USD prices, ensuring that all contributions meet a minimum threshold in fiat value.

CryptoFundHub aims to provide a modern, secure, and decentralized alternative to traditional crowdfunding systems. It is designed to be modular, extensible, and transparent—laying the foundation for advanced features like multi-campaign support, donor dashboards, historical transaction tracking and automated campaign expiry in future iterations.

2. Functional and Non-Functional Requirements

2.1 Functional:

1. **ETH Funding Acceptance:**

   - This function is marked as `payable` and accepts contributions directly from the user's wallet.

   - The contract records each contributor's address.

2. **Minimum USD-equivalent Contribution Enforcement:**

   - Uses Chainlink Price Feeds to fetch the latest ETH/USD conversion rate.

   - Compares the incoming ETH value (`msg.value`) in USD to the threshold using `getConversionRate().`

   - If the value is below the minimum, the transaction is reverted.

3. **Track Funders and Contributions:**

   - An `address[]` array is used to store a list of funders.

   - A `mapping(address => uint256)` keeps track of how much each address has funded.

   - If the same user funds again, their total is updated cumulatively.

4. **Price Fetching via Chainlink:**

    - Uses the Chainlink AggregatorV3Interface at a specific address (`0x694A...`).

    - Data is fetched using `latestRoundData()` and normalized to 18 decimal places.

    - This provides a tamper-proof way to check the current ETH value in USD.

2.2 Non-Functional:

1. **Ethereum Testnet Compatibility:**

    - Ensures gas usage and behavior are realistic before deploying to mainnet.

    - Testnet oracles are used for price feeds.

2. **Security Best Practices:**

    - Use of `onlyOwner` to restrict access

    - Fallback-safe withdrawal using `call()` instead of `transfer()`

    - Avoiding integer overflows (Solidity 0.8+ does this automatically)

3. **Contract Verification on Etherscan:**

    - Ensures transparency and readability of the source code

    - Let's funders and external parties inspect the logic

4. **Gas efficiency:**

    - Reverts early if funding conditions are not met

- Clears arrays and mappings only when needed
- Uses efficient data structures and access patterns

5. **Modularity and Code Reusability:**

- `PriceConverter` is implemented as a separate library
- Allows code reuse in future contracts or upgrades
- Clean separation of logic and conversion calculations

3. Backend Development & Smart Contracts

3.1 Overview

CryptoFundHub uses a **serverless backend**: all business logic and data persistence live **on-chain** (Ethereum Sepolia), and the web app talks to contracts through the **thirdweb SDK**. There is **no custom REST/GraphQL server**. The "backend" is therefore:

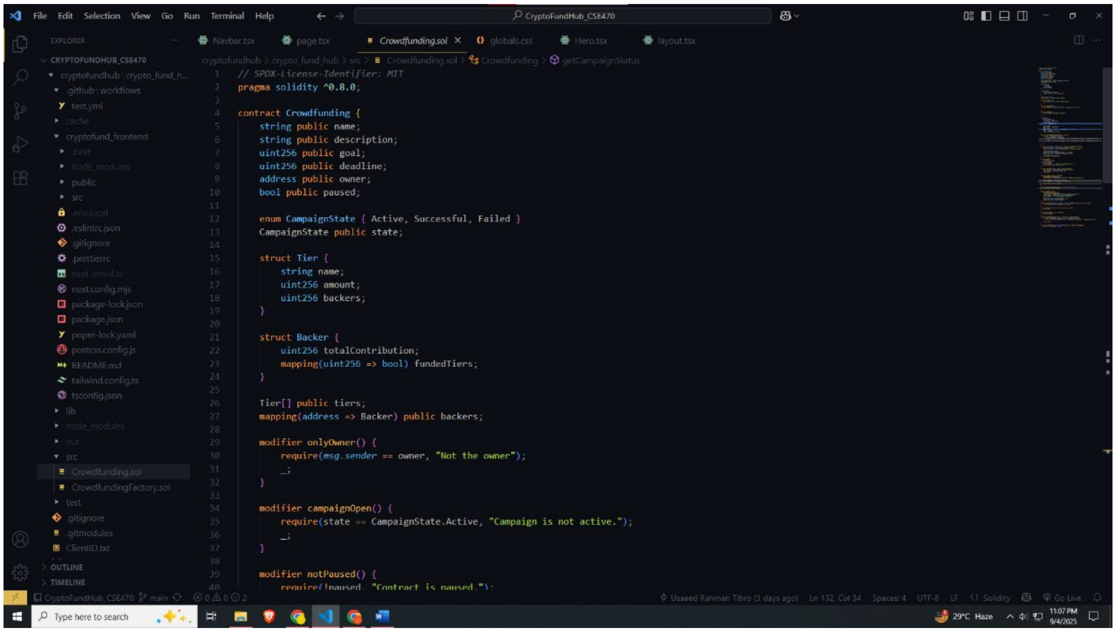
- the **Thirdweb client** (SDK configuration, wallet auth, RPC access), and
- the **contract integration layer** (read/write calls to CrowdfundingFactory and Crowdfunding).

This keeps the system trustless (no centralized DB), auditable, and easy to scale.

3.2 Smart Contracts

The Crowdfunding contract represents a **single fundraising campaign**.
It stores metadata (name, description, goal, duration), manages tiers of contributions, tracks backers, and enforces rules for funding, withdrawals, and refunds.

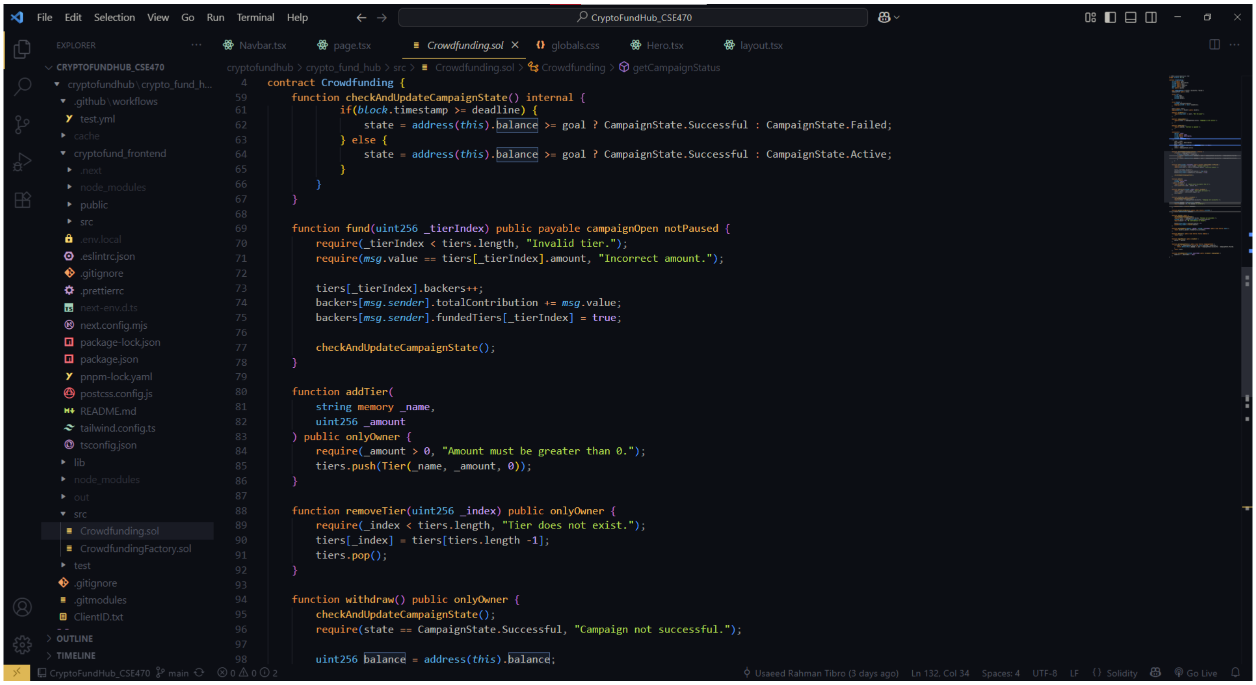
This contract is deployed by the CrowdfundingFactory, ensuring every campaign is **isolated** with its own funds and backers.



```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Crowdfunding {
    string public name;
    string public description;
    uint256 public goal;
    uint256 public deadline;
    address public owner;
    bool public paused;

    enum CampaignState { Active, Successful, Failed }
    CampaignState public state;

    struct Tier {
        string name;
        uint256 amount;
        uint256 backers;
    }

    struct Backer {
        uint256 totalContribution;
        mapping(uint256 => bool) fundedTiers;
    }

    Tier[] public tiers;
    mapping(address => Backer) public backers;

    modifier onlyOwner() {
        require(msg.sender == owner, "Not the owner");
        _;
    }

    modifier campaignOpen() {
        require(state == CampaignState.Active, "Campaign is not active.");
        _;
    }

    modifier notPaused() {
        require(!paused, "Contract is paused.");
```

The Crowdfunding.sol contract is the core component of the backend that manages an individual fundraising campaign. Each time a campaign is created through the factory, a new instance of this contract is deployed, isolating funds and contributors for that campaign. The contract stores campaign metadata, including its name, description, funding goal, deadline, and the address of the campaign creator. It also maintains a lifecycle state, which transitions between *Active*, *Successful*, and *Failed*, depending on whether the funding goal has been met within the deadline. To enhance control and safety, a pause mechanism is implemented, allowing the campaign owner to temporarily suspend operations in case of emergencies.

The contract is structured around tiers and backers. Tiers define contribution levels, where each tier has a name, a fixed contribution amount, and a counter to track the number of backers who supported it. Backers are represented by a mapping that records their total contributions and the specific tiers they have funded. This ensures accurate tracking of participant engagement across different funding levels. Contributions are processed through the fund function, which enforces that contributors must send the exact amount required by the tier they select. When funds are added, the system automatically checks whether the campaign goal has been reached, updating the campaign's state accordingly.

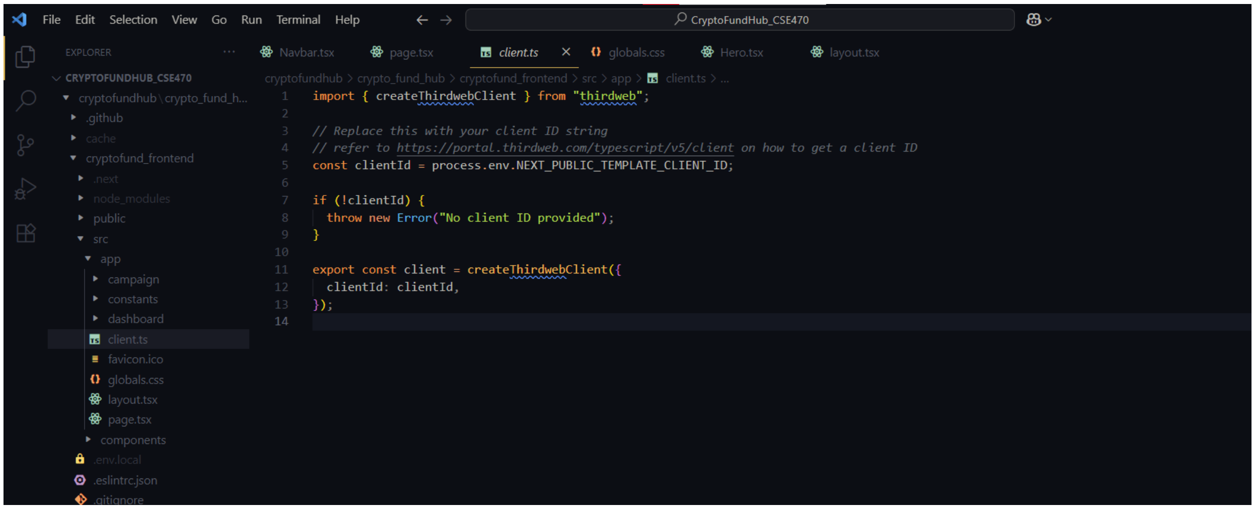


```
contract Crowdfunding {
    function checkAndUpdateCampaignState() internal {
            if(block.timestamp >= deadline) {
                state = address(this).balance >= goal ? CampaignState.Successful : CampaignState.Failed;
            } else {
                state = address(this).balance >= goal ? CampaignState.Successful : CampaignState.Active;
            }
        }
    }

    function fund(uint256 _tierIndex) public payable campaignOpen notPaused {
        require(_tierIndex < tiers.length, "Invalid tier.");
        require(msg.value == tiers[_tierIndex].amount, "Incorrect amount.");

        tiers[_tierIndex].backers++;
        backers[msg.sender].totalContribution += msg.value;
        backers[msg.sender].fundedTiers[_tierIndex] = true;

        checkAndUpdateCampaignState();
    }

    function addTier(
        string memory _name,
        uint256 _amount
    ) public onlyOwner {
        require(_amount > 0, "Amount must be greater than 0.");
        tiers.push(Tier(_name, _amount, 0));
    }

    function removeTier(uint256 _index) public onlyOwner {
        require(_index < tiers.length, "Tier does not exist.");
        tiers[_index] = tiers[tiers.length -1];
        tiers.pop();
    }

    function withdraw() public onlyOwner {
        checkAndUpdateCampaignState();
        require(state == CampaignState.Successful, "Campaign not successful.");

        uint256 balance = address(this).balance;
```

In addition to funding, the contract provides a set of administrative and utility functions for managing the campaign. The owner has the ability to add or remove tiers, withdraw funds once the campaign is marked successful, extend the deadline of an ongoing campaign, or pause it entirely. If the campaign fails to reach its goal, contributors can claim refunds, ensuring fairness and accountability. To prevent misuse, strict access control is enforced using modifiers such as onlyOwner, campaignOpen, and notPaused, which restrict functions to the campaign owner, ensure that the campaign is still active, and disable operations when paused, respectively.

3.3 **MataMask Integration**

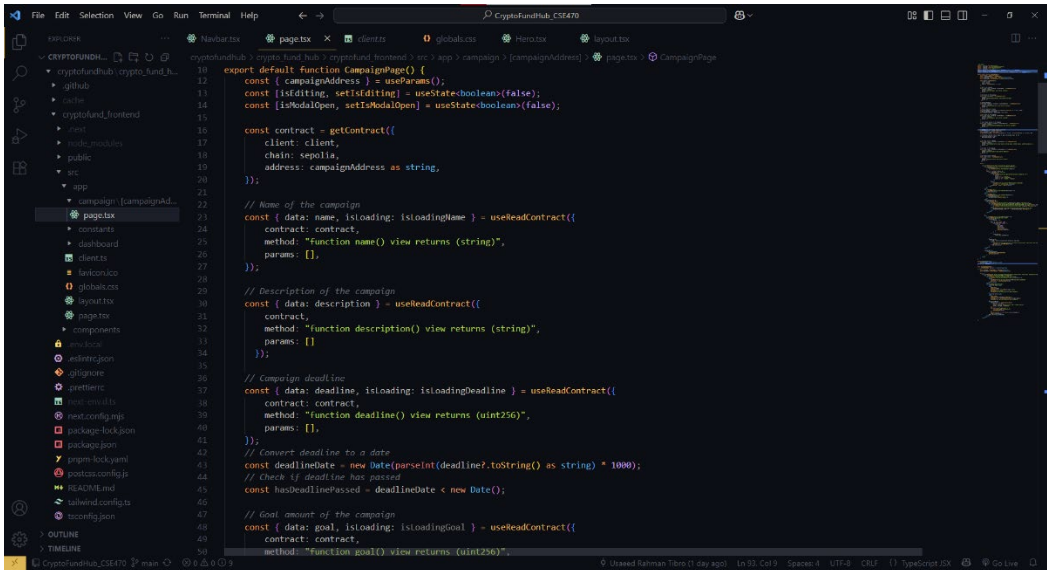
Allows users to connect their Ethereum wallet (e.g., MetaMask) to authenticate and interact with the dApp. The frontend uses Thirdweb's ConnectButton and useActiveAccount hook to establish a connection between the user's wallet and the dApp. Once connected, the wallet address is available globally and used to identify the campaign owner or contributor. The Navbar.tsx contains the ConnectButton from thirdweb/react while client.ts configures the Thirdweb client with the project's client ID.



```
import { createThirdwebClient } from "thirdweb";

// Replace this with your client ID string
// refer to https://portal.thirdweb.com/typescript/v5/client on how to get a client ID
const clientId = process.env.NEXT_PUBLIC_TEMPLATE_CLIENT_ID;

if (!clientId) {
  throw new Error("No client ID provided");
}

export const client = createThirdwebClient({
  clientId: clientId,
});
```

3.4 **Create Campaign**

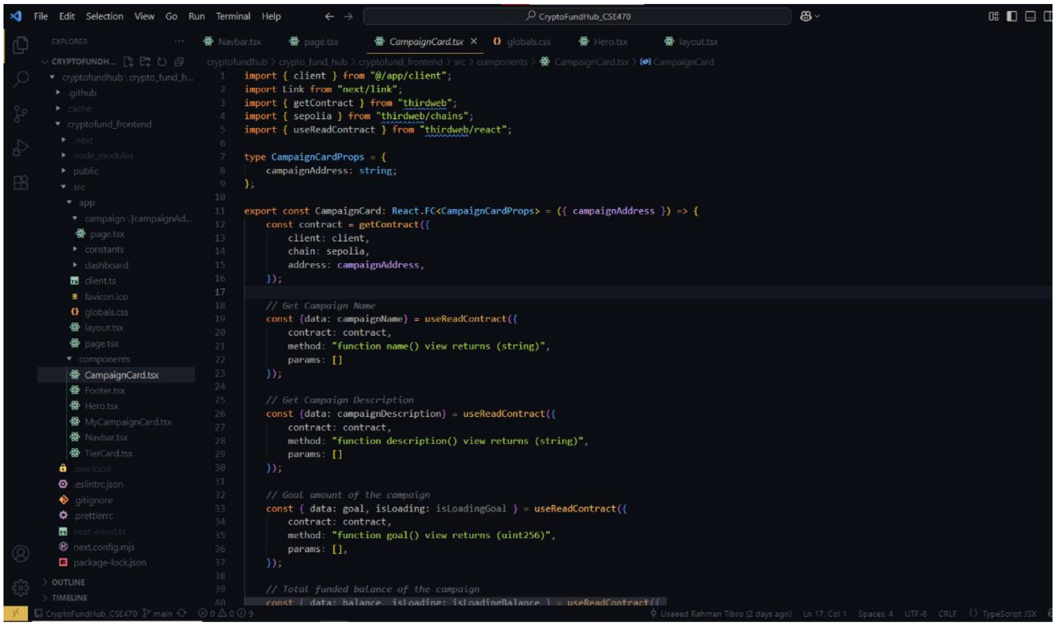
Enables users to launch their own crowdfunding campaigns. The user fills out a form in the **Create Campaign modal**. When submitted, the modal calls either the createCampaign function in the factory (CrowdfundingFactory.sol) or deploys a contract directly via Thirdweb SDK (deployPublishedContract). The campaign is then recorded on the blockchain and retrievable later by querying the factory. The smart contract CrowdfundingFactory.sol responsible for deploying new campaign instances. page.tsx (dashboard/[walletAddress]) contains the modal logic (CreateCampaignModal) where campaign creation is triggered.



```tsx
export default function CampaignPage() {
    const { campaignAddress } = useParams();
    const [isEditing, setIsEditing] = useState<boolean>(false);
    const [isModalOpen, setIsModalOpen] = useState<boolean>(false);

    const contract = getContract({
        client: client,
        chain: sepolia,
        address: campaignAddress as string,
    });

    // Name of the campaign
    const { data: name, isLoading: isLoadingName } = useReadContract({
        contract: contract,
        method: "function name() view returns (string)",
        params: [],
    });

    // Description of the campaign
    const { data: description } = useReadContract({
        contract,
        method: "function description() view returns (string)",
        params: []
      });

    // Campaign deadline
    const { data: deadline, isLoading: isLoadingDeadline } = useReadContract({
        contract: contract,
        method: "function deadline() view returns (uint256)",
        params: [],
    });
    // Convert deadline to a date
    const deadlineDate = new Date(parseInt(deadline?.toString() as string) * 1000);
    // Check if deadline has passed
    const hasDeadlinePassed = deadlineDate < new Date();

    // Goal amount of the campaign
    const { data: goal, isLoading: isLoadingGoal } = useReadContract({
        contract: contract,
        method: "function goal() view returns (uint256)",
```

3.5 Donating Funds

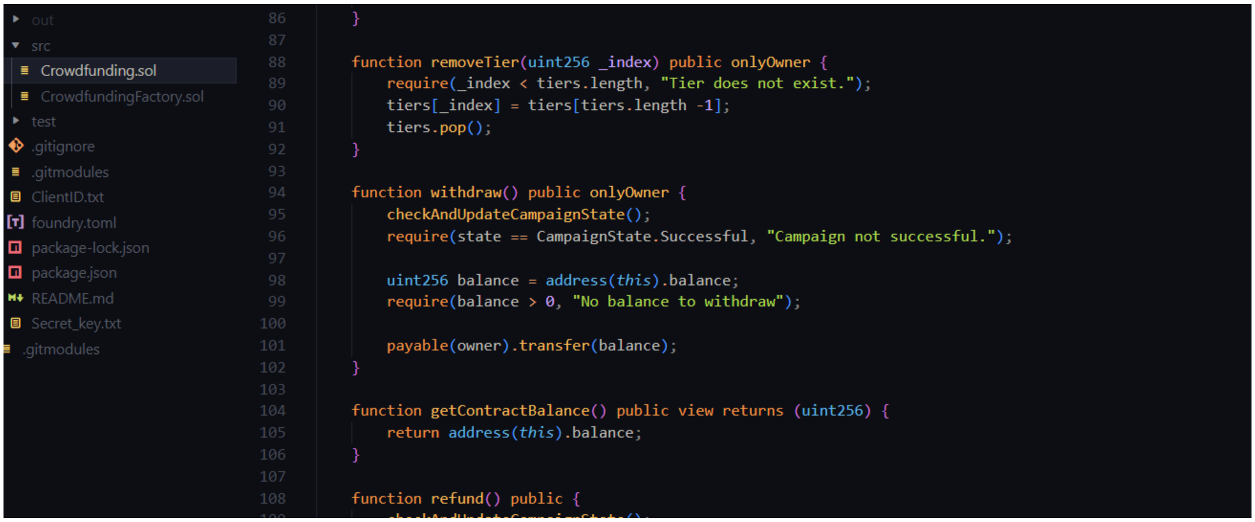
It Lets contributors donate ETH to campaigns through predefined tiers. The frontend calls the fund(uint256 _tierIndex) function in Crowdfunding.sol. The contributor must send the exact ETH amount specified in the tier. Once confirmed, the backer's contribution is recorded, and the campaign's state may update to Successful if the goal is reached. CampaignCard.tsx provides the UI and integrates the contract with the front end.



```tsx
import { client } from "@/app/client";
import Link from "next/link";
import { getContract } from "thirdweb";
import { sepolia } from "thirdweb/chains";
import { useReadContract } from "thirdweb/react";

type CampaignCardProps = {
    campaignAddress: string;
};

export const CampaignCard: React.FC<CampaignCardProps> = ({ campaignAddress }) => {
    const contract = getContract({
        client: client,
        chain: sepolia,
        address: campaignAddress,
    });

    // Get Campaign Name
    const {data: campaignName} = useReadContract({
        contract: contract,
        method: "function name() view returns (string)",
        params: []
    });

    // Get Campaign Description
    const {data: campaignDescription} = useReadContract({
        contract: contract,
        method: "function description() view returns (string)",
        params: []
    });

    // Goal amount of the campaign
    const { data: goal, isLoading: isLoadingGoal } = useReadContract({
        contract: contract,
        method: "function goal() view returns (uint256)",
        params: [],
    });

    // Total funded balance of the campaign
    const { data: balance, isLoading: isLoadingBalance } = useReadContract({
```

3.6 **Owner-only Withdrawals**

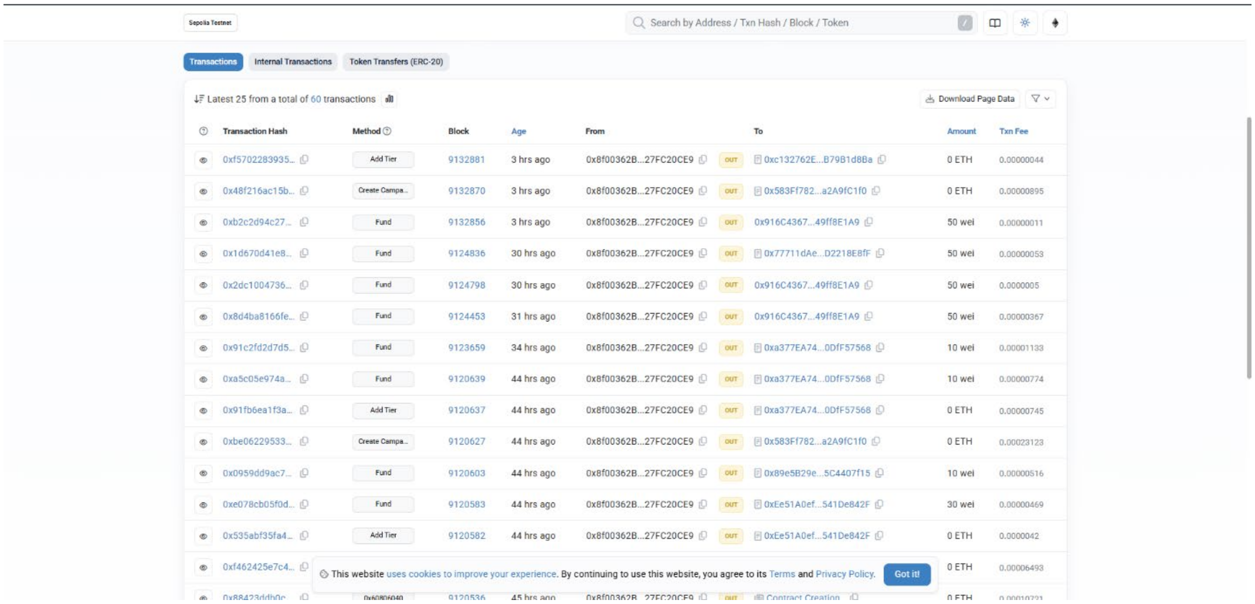
Ensures only campaign creators can withdraw funds from successful campaigns. The withdraw() function in Crowdfunding.sol is protected with the onlyOwner modifier. Once the campaign reaches its funding goal and is marked successful, the owner can call this function to transfer the balance to their wallet. page.tsx (dashboard/[walletAddress]) provides the owner UI for triggering withdrawals.



```
}

function removeTier(uint256 _index) public onlyOwner {
    require(_index < tiers.length, "Tier does not exist.");
    tiers[_index] = tiers[tiers.length -1];
    tiers.pop();
}

function withdraw() public onlyOwner {
    checkAndUpdateCampaignState();
    require(state == CampaignState.Successful, "Campaign not successful.");

    uint256 balance = address(this).balance;
    require(balance > 0, "No balance to withdraw");

    payable(owner).transfer(balance);
}

function getContractBalance() public view returns (uint256) {
    return address(this).balance;
}

function refund() public {
    checkAndUpdateCampaignState();
```

3.7 **Transaction History**

Tracks all blockchain transactions such as contributions, refunds, and withdrawals. Since every action (fund, withdraw, refund) is recorded on-chain, the frontend fetches these logs through contract reads or via Etherscan APIs. The mapping of backers in Crowdfunding.sol also helps track contribution history.



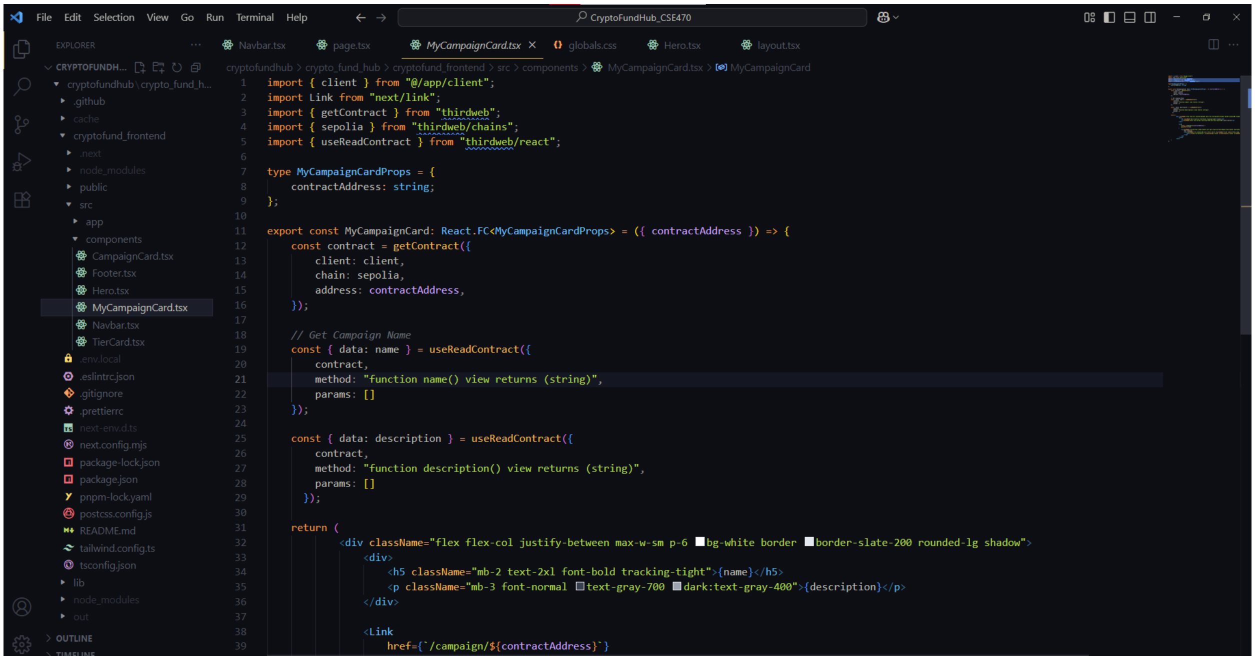
Latest 25 from a total of 60 transactions

Transaction Hash	Method	Block	Age	From		To	Amount	Txn Fee
0xf5702283935...	Add Tier	9132881	3 hrs ago	0x8f00362B...27FC20CE9	OUT	0xc132762E...B79B1d8Ba	0 ETH	0.00000044
0x48f216ac15b...	Create Campa...	9132870	3 hrs ago	0x8f00362B...27FC20CE9	OUT	0x583Ff782...a2A9fC1f0	0 ETH	0.00000895
0xb2c2d94c27...	Fund	9132856	3 hrs ago	0x8f00362B...27FC20CE9	OUT	0x916C4367...49ff8E1A9	50 wei	0.00000011
0x1d670d41e8...	Fund	9124836	30 hrs ago	0x8f00362B...27FC20CE9	OUT	0x77711dAe...D2218E8fF	50 wei	0.00000053
0x2dc1004736...	Fund	9124798	30 hrs ago	0x8f00362B...27FC20CE9	OUT	0x916C4367...49ff8E1A9	50 wei	0.0000005
0x8d4ba8166fe...	Fund	9124453	31 hrs ago	0x8f00362B...27FC20CE9	OUT	0x916C4367...49ff8E1A9	50 wei	0.00000367
0x91c2fd2d7d5...	Fund	9123659	34 hrs ago	0x8f00362B...27FC20CE9	OUT	0xa377EA74...0DfF57568	10 wei	0.00001133
0xa5c05e974a...	Fund	9120639	44 hrs ago	0x8f00362B...27FC20CE9	OUT	0xa377EA74...0DfF57568	10 wei	0.00000774
0x91fb6ea1f3a...	Add Tier	9120637	44 hrs ago	0x8f00362B...27FC20CE9	OUT	0xa377EA74...0DfF57568	0 ETH	0.00000745
0xbe06229533...	Create Campa...	9120627	44 hrs ago	0x8f00362B...27FC20CE9	OUT	0x583Ff782...a2A9fC1f0	0 ETH	0.00023123
0x0959dd9ac7...	Fund	9120603	44 hrs ago	0x8f00362B...27FC20CE9	OUT	0x89e5B29e...5C4407f15	10 wei	0.00000516
0xe078cb05f0d...	Fund	9120583	44 hrs ago	0x8f00362B...27FC20CE9	OUT	0xEe51A0ef...541De842F	30 wei	0.00000469
0x535abf35fa4...	Add Tier	9120582	44 hrs ago	0x8f00362B...27FC20CE9	OUT	0xEe51A0ef...541De842F	0 ETH	0.0000042
0xf462425e7c4...							0 ETH	0.00006493
0x88423ddb0c...	0x60806040	9120536	45 hrs ago	0x8f00362B...27FC20CE9	OUT	Contract Creation	0 ETH	0.00010721

This website uses cookies to improve your experience. By continuing to use this website, you agree to its Terms and Privacy Policy. Got it!

3.8 **Minimum Contribution Enforcement**

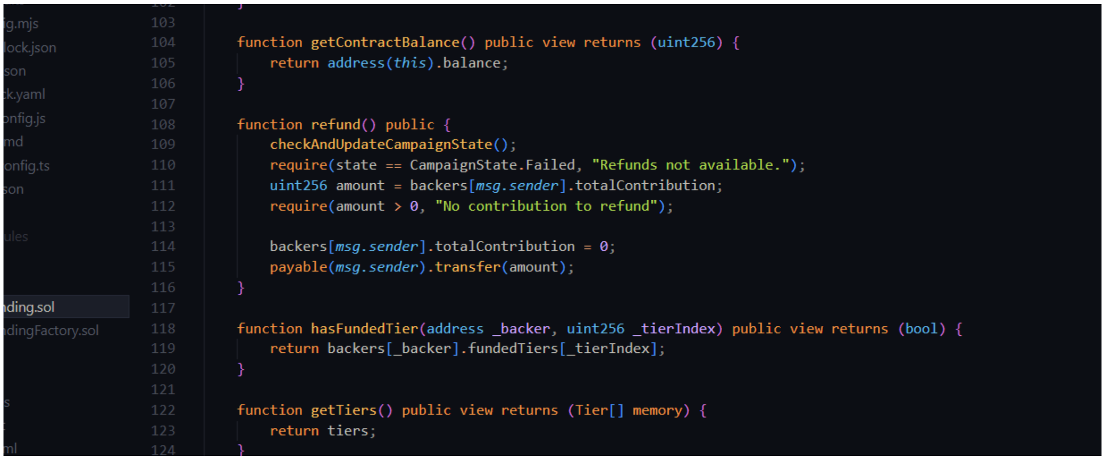
Prevents users from donating less than the tier amount, maintaining fairness. The fund() function in Crowdfunding.sol checks the condition require(msg.value == tiers[_tierIndex].amount). If the contribution does not match exactly, the transaction reverts. This guarantees standardization of tier contributions. Frontend component MyCampaignCard.tsx ensure users input the correct tier value before submitting.



```tsx
import { client } from "@/app/client";
import Link from "next/link";
import { getContract } from "thirdweb";
import { sepolia } from "thirdweb/chains";
import { useReadContract } from "thirdweb/react";

type MyCampaignCardProps = {
    contractAddress: string;
};

export const MyCampaignCard: React.FC<MyCampaignCardProps> = ({ contractAddress }) => {
    const contract = getContract({
        client: client,
        chain: sepolia,
        address: contractAddress,
    });

    // Get Campaign Name
    const { data: name } = useReadContract({
        contract,
        method: "function name() view returns (string)",
        params: []
    });

    const { data: description } = useReadContract({
        contract,
        method: "function description() view returns (string)",
        params: []
      });

    return (
            <div className="flex flex-col justify-between max-w-sm p-6 bg-white border border-slate-200 rounded-lg shadow">
                <div>
                    <h5 className="mb-2 text-2xl font-bold tracking-tight">{name}</h5>
                    <p className="mb-3 font-normal text-gray-700 dark:text-gray-400">{description}</p>
                </div>

                <Link
                    href={`/campaign/${contractAddress}`}
```

3.9 **Campaign Expiry System**

Automatically closes campaigns once deadlines pass, marking them as Successful or Failed. Each campaign is initialized with a deadline in Crowdfunding.sol. The function checkAndUpdateCampaignState() is called during key operations (funding, withdrawing,

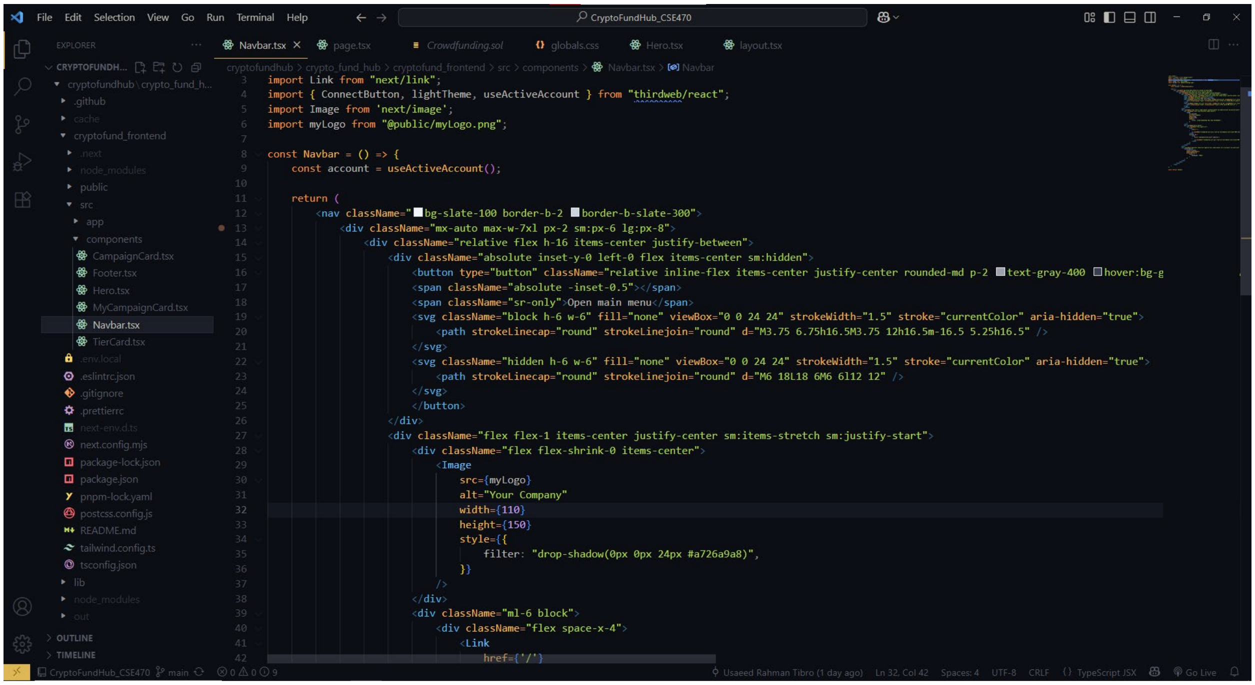


```solidity
    function getContractBalance() public view returns (uint256) {
        return address(this).balance;
    }

    function refund() public {
        checkAndUpdateCampaignState();
        require(state == CampaignState.Failed, "Refunds not available.");
        uint256 amount = backers[msg.sender].totalContribution;
        require(amount > 0, "No contribution to refund");

        backers[msg.sender].totalContribution = 0;
        payable(msg.sender).transfer(amount);
    }

    function hasFundedTier(address _backer, uint256 _tierIndex) public view returns (bool) {
        return backers[_backer].fundedTiers[_tierIndex];
    }

    function getTiers() public view returns (Tier[] memory) {
        return tiers;
```

refunding). If the deadline has passed, the contract decides whether the campaign is successful (goal reached) or failed (goal not reached). This ensures lifecycle automation.

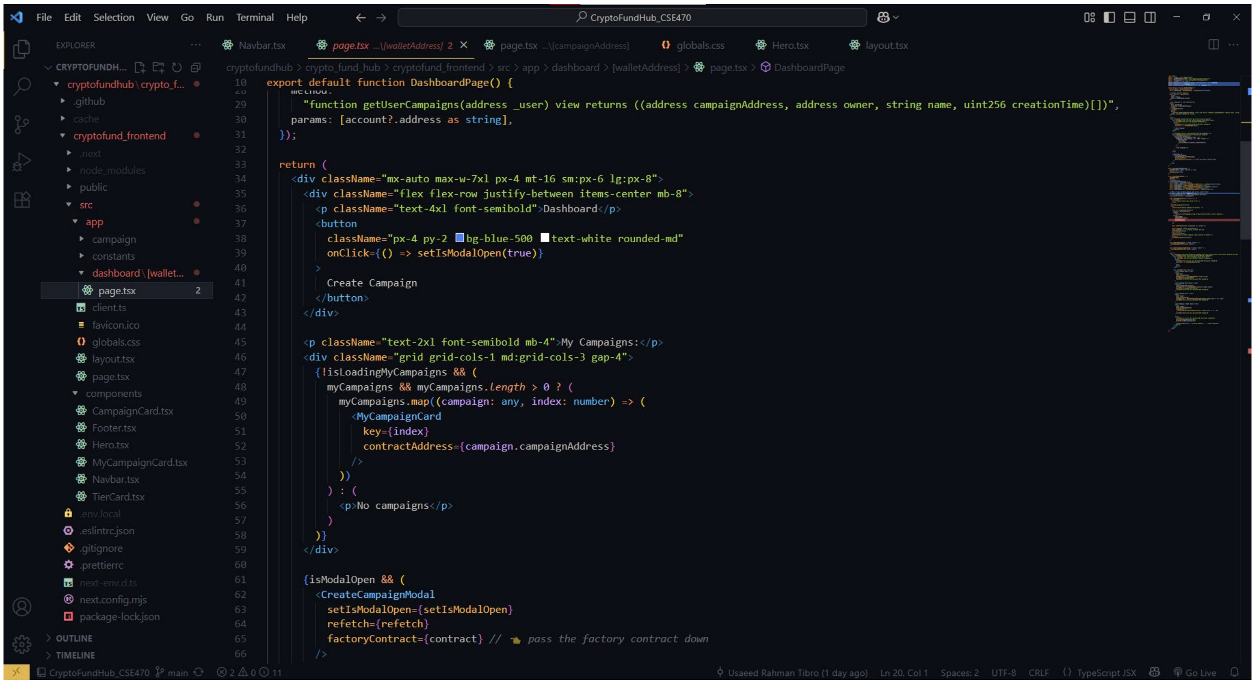
4. Frontend Development

The frontend of the application serves as the primary interface for user interaction, bridging blockchain logic with an intuitive and responsive design. Built using **Next.js** with **React** components and styled with **Tailwind CSS**, the frontend focuses on simplicity and accessibility, ensuring that both campaign owners and contributors can navigate the platform without requiring prior blockchain knowledge. The layout is structured around reusable components such as the **Navbar**, **Dashboard**, and **Campaign Cards**, which dynamically fetch and display data from the blockchain through the smart contract integration layer.



```tsx
import Link from "next/link";
import { ConnectButton, lightTheme, useActiveAccount } from "thirdweb/react";
import Image from 'next/image';
import myLogo from "@public/myLogo.png";

const Navbar = () => {
    const account = useActiveAccount();

    return (
        <nav className="bg-slate-100 border-b-2 border-b-slate-300">
            <div className="mx-auto max-w-7xl px-2 sm:px-6 lg:px-8">
                <div className="relative flex h-16 items-center justify-between">
                    <div className="absolute inset-y-0 left-0 flex items-center sm:hidden">
                        <button type="button" className="relative inline-flex items-center justify-center rounded-md p-2 text-gray-400 hover:bg-g
                        <span className="absolute -inset-0.5"></span>
                        <span className="sr-only">Open main menu</span>
                        <svg className="block h-6 w-6" fill="none" viewBox="0 0 24 24" strokeWidth="1.5" stroke="currentColor" aria-hidden="true">
                            <path strokeLinecap="round" strokeLinejoin="round" d="M3.75 6.75h16.5M3.75 12h16.5m-16.5 5.25h16.5" />
                        </svg>
                        <svg className="hidden h-6 w-6" fill="none" viewBox="0 0 24 24" strokeWidth="1.5" stroke="currentColor" aria-hidden="true">
                            <path strokeLinecap="round" strokeLinejoin="round" d="M6 18L18 6M6 6l12 12" />
                        </svg>
                        </button>
                    </div>
                    <div className="flex flex-1 items-center justify-center sm:items-stretch sm:justify-start">
                        <div className="flex flex-shrink-0 items-center">
                            <Image
                                src={myLogo}
                                alt="Your Company"
                                width={110}
                                height={150}
                                style={{
                                    filter: "drop-shadow(0px 0px 24px #a726a9a8)",
                                }}
                            />
                        </div>
                        <div className="ml-6 block">
                            <div className="flex space-x-4">
                                <Link
                                    href={'/'}
```

Users can seamlessly connect their wallets using the integrated Thirdweb SDK, after which the application adjusts its interface based on their account status — for example, showing the **Dashboard** button only when a wallet is connected. Campaigns are displayed in grid layouts with key details such as funding goals, raised amounts, and deadlines, all updated in real time by directly reading from deployed contracts. The frontend also provides modal-based forms for creating campaigns, making the process straightforward while abstracting the underlying contract deployment complexity.



```tsx
export default function DashboardPage() {
    method:
      "function getUserCampaigns(address _user) view returns ((address campaignAddress, address owner, string name, uint256 creationTime)[])",
    params: [account?.address as string],
  });

  return (
    <div className="mx-auto max-w-7xl px-4 mt-16 sm:px-6 lg:px-8">
      <div className="flex flex-row justify-between items-center mb-8">
        <p className="text-4xl font-semibold">Dashboard</p>
        <button
          className="px-4 py-2 bg-blue-500 text-white rounded-md"
          onClick={() => setIsModalOpen(true)}
        >
          Create Campaign
        </button>
      </div>

      <p className="text-2xl font-semibold mb-4">My Campaigns:</p>
      <div className="grid grid-cols-1 md:grid-cols-3 gap-4">
        {!isLoadingMyCampaigns && (
          myCampaigns && myCampaigns.length > 0 ? (
            myCampaigns.map((campaign: any, index: number) => (
              <MyCampaignCard
                key={index}
                contractAddress={campaign.campaignAddress}
              />
            ))
          ) : (
            <p>No campaigns</p>
          )
        )}
      </div>

      {isModalOpen && (
        <CreateCampaignModal
          setIsModalOpen={setIsModalOpen}
          refetch={refetch}
          factoryContract={contract} // 👈 pass the factory contract down
        />
```

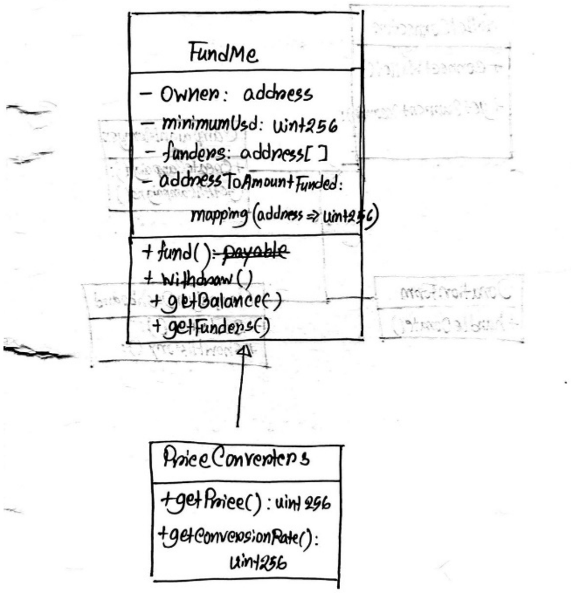
Overall, the frontend ensures a smooth user experience by hiding low-level blockchain operations behind simple UI components, presenting contract data in a familiar web format, and maintaining responsiveness across devices. It complements the backend contracts by giving users a clear, interactive way to create, manage, and fund campaigns in a decentralized manner.

5. Technology (Framework, Languages, Tools)

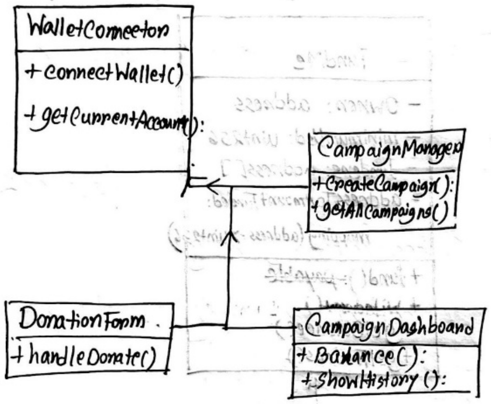
- **Frontend:** Next.js, React, Tailwind CSS, DaisyUI
- **Blockchain:** Solidity, Ethereum Sepolia Testnet
- **Web3 Tools:** Thirdweb SDK (React, Deploys, Contract Calls)
- **Development Tools:** Foundry (Forge), Hardhat (optional), GitHub

6. Class Diagram

Class Diagram for smart Contracts:



Class Diagram for Frontend (React + MetaMask interaction):


WalletConnector
+ connectWallet()
+ getCurrentAccount():
CampaignManager
+createCampaign():
+getAllCampaigns()
DonationForm
+ handleDonate()
CampaignDashboard
+ Balance():
+ showHistory():