# GROUP 1

EZRA ANONDA - 20/03563

DIAR STEPHEN - 22/02321

DENIS GITAU - 22/01876

KAREN NDEGWA - 21/00351

Github Link: https://github.com/king-luvaha/crowd-funding

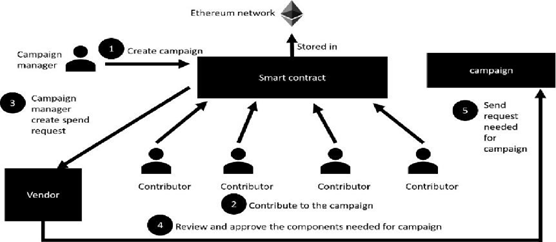
Github Link:

Github Link:

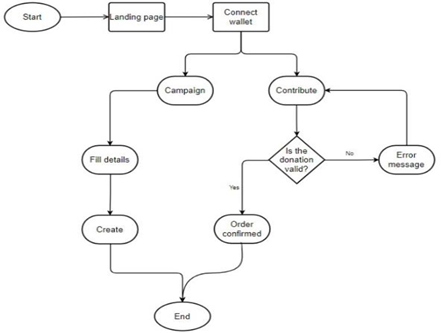
Github Link:

This is a smart contract for a crowdfunding platform app.

## **Project Design**



## **Data Flow Diagram**



## **The System Code**

// SPDX-License-Identifier: UNLICENSED

pragma solidity ^0.8.9;

contract CrowdFunding {

struct Campaign {

address owner;

string title;

string description;

uint256 target;

uint256 deadline;

uint256 amountCollected;

string image;

address[] donators;

uint256[] donations;

}

mapping(uint256 => Campaign) public campaigns;

uint256 public numberOfCampaigns = 0;

function createCampaign(address \_owner, string memory \_title, string memory \_description, uint256 \_target, uint256 \_deadline, string memory \_image) public returns (uint256) {

require(\_deadline > block.timestamp, "The deadline should be a date in the future.");

Campaign storage campaign = campaigns[numberOfCampaigns];

campaign.owner = \_owner;

campaign.title = \_title;

campaign.description = \_description;

campaign.target = \_target;

campaign.deadline = \_deadline;

campaign.amountCollected = 0;

campaign.image = \_image;

numberOfCampaigns++;

return numberOfCampaigns - 1;

}

function donateToCampaign(uint256 \_id) public payable {

uint256 amount = msg.value;

Campaign storage campaign = campaigns[\_id];

campaign.donators.push(msg.sender);

campaign.donations.push(amount);

// Update the state before transferring funds

campaign.amountCollected += amount;

(bool sent,) = payable(campaign.owner).call{value: amount}("");

require(sent, "Failed to send Ether");

}

function getDonators(uint256 \_id) view public returns (address[] memory, uint256[] memory) {

return (campaigns[\_id].donators, campaigns[\_id].donations);

}

function getCampaigns() public view returns (Campaign[] memory) {

Campaign[] memory allCampaigns = new Campaign[](numberOfCampaigns);

for(uint i = 0; i < numberOfCampaigns; i++) {

Campaign storage item = campaigns[i];

allCampaigns[i] = item;

}

return allCampaigns;

}

}

## **Overview**

The CrowdFunding smart contract is designed to facilitate crowdfunding campaigns on the Ethereum blockchain. It allows users to create campaigns, donate to them, and track the contributions made by various donors. The contract is written in Solidity and is compatible with version 0.8.9 and above.

## Contract structure

## **Campaign Struct**

The Campaign struct represents a crowdfunding campaign and contains the following fields:

* address owner: The address of the campaign creator.
* string title: The title of the campaign.
* string description: A brief description of the campaign.
* uint256 target: The funding goal for the campaign.
* uint256 deadline: The timestamp indicating when the campaign ends.
* uint256 amountCollected: The total amount of funds collected so far.
* string image: A URL or reference to an image representing the campaign.
* address[] donators: An array of addresses that have donated to the campaign.
* uint256[] donations: An array of donation amounts corresponding to each donor.

## **State Variables**

* mapping(uint256 => Campaign) public campaigns: A mapping that stores all campaigns indexed by a unique ID.
* uint256 public numberOfCampaigns: A counter that tracks the total number of campaigns created.

## **Functions**

### ‘createCampaign’

1. function createCampaign(address \_owner, string memory \_title, string memory \_description, uint256 \_target, uint256 \_deadline, string memory \_image) public returns (uint256) {
2. Campaign storage campaign = campaigns[numberOfCampaigns];
3. require(campaign.deadline < block.timestamp, "The deadline should be a date in the future.");
4. campaign.owner = \_owner;
5. campaign.title = \_title;
6. campaign.description = \_description;
7. campaign.target = \_target;
8. campaign.deadline = \_deadline;
9. campaign.amountCollected = 0;
10. campaign.image = \_image;
11. numberOfCampaigns++;
12. return numberOfCampaigns - 1;
13. }

**Parameters:**

* address \_owner: The address of the campaign creator.
* string memory \_title: The title of the campaign.
* string memory \_description: A description of the campaign.
* uint256 \_target: The funding target for the campaign.
* uint256 \_deadline: The deadline for the campaign (must be a future date).
* string memory \_image: An image URL for the campaign.

**Returns:** The ID of the newly created campaign.

**Description:** This function allows users to create a new crowdfunding campaign. It requires that the deadline is set in the future. Upon successful creation, it increments the numberOfCampaigns counter.

### ‘donateToCampaign’

function donateToCampaign(uint256 \_id) public payable {

uint256 amount = msg.value;

Campaign storage campaign = campaigns[\_id];

campaign.donators.push(msg.sender);

campaign.donations.push(amount);

(bool sent,) = payable(campaign.owner).call{value: amount}("");

if(sent) {

campaign.amountCollected = campaign.amountCollected + amount;

}

}

**Parameters:**

* `uint256 \_id`: The ID of the campaign to which the user wants to donate.

**Description:** This function allows users to donate Ether to a specific campaign. The donation amount is taken from the transaction value. The donor's address and the donation amount are recorded in the campaign's respective arrays. If the transfer to the campaign owner is successful, the amount collected for the campaign is updated.

### 

### ‘getDonators’

function getDonators(uint256 \_id) view public returns (address[] memory, uint256[] memory) {

return (campaigns[\_id].donators, campaigns[\_id].donations);

}

**Parameters:**

* uint256 \_id: The ID of the campaign.

**Returns:** An array of donor addresses and an array of corresponding donation amounts.

**Description:** This function retrieves the list of donors and their respective donation amounts for a specified campaign.

### ‘getCampaigns’

function getCampaigns() public view returns (Campaign[] memory) {

Campaign[] memory allCampaigns = new Campaign[](numberOfCampaigns);

for(uint i = 0; i < numberOfCampaigns; i++) {

Campaign storage item = campaigns[i];

allCampaigns[i] = item;

}

return allCampaigns;

}

**Returns:** An array containing all campaigns.

**Description:** This function returns an array of all campaigns created in the contract, allowing users to view the details of each campaign.

# Steps to Interact with the Contract

## Deploying the Contract

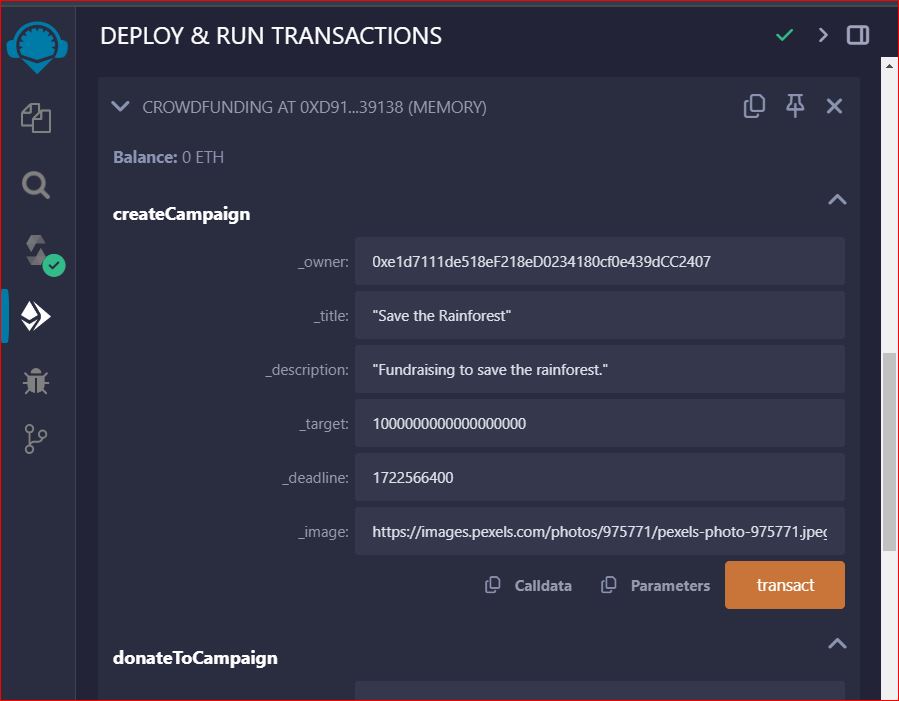
To deploy the ‘Crowdfunding’ contract:

* Open Remix IDE or your preferred environment.
* Create a new Solidity file and paste the contract code.
* Compile the contract using the Solidity compiler.
* Deploy the contract by selecting it from the dropdown menu and clicking the "Deploy" button. Make sure your wallet is connected and has sufficient Ether for gas fees.

## Creating a Campaign

Once the contract is deployed, you can create a campaign by calling the ‘createCampaign’ function:

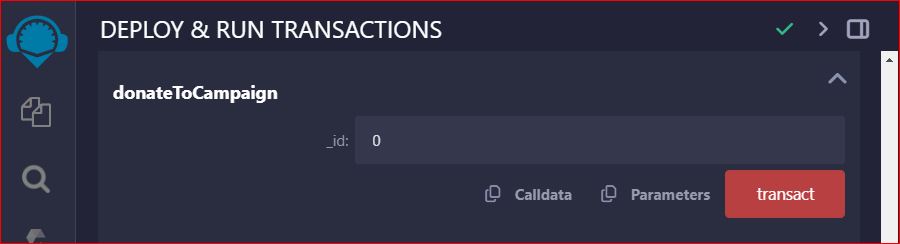
* **Function Call:** ‘createCampaign’
* **Parameters:**
  + \_owner: The address of the campaign owner (usually your wallet address).
  + \_title: A string representing the title of the campaign.
  + \_description: A string describing the campaign.
  + \_target: A uint256 representing the funding goal (in wei).
  + \_deadline: A uint256 representing the deadline (timestamp in seconds).
  + \_image: A string URL pointing to an image related to the campaign.



## Donating to a Campaign

To donate to an existing campaign, use the ‘donateToCampaign’ function:

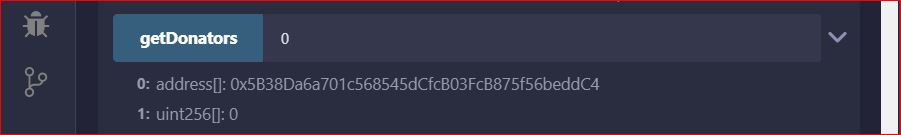
* **Function Call:** ‘donateToCampaign’.
* **Parameters:**
  + \_id: The ID of the campaign you want to donate to.



## Retrieving Donators and Donations

To view the list of donators and their respective donation amounts for a specific campaign, call the getDonators function:

* Function Call: getDonators
* Parameters:
  + \_id: The ID of the campaign.



## Getting All Campaigns

To retrieve all campaigns created in the contract, use the getCampaigns function:

* **Function Call:** getCampaigns

