

Educational Crowdfunding Decentralized Application
Final Project Report
Students Name: U. Erasyl, A. Yerasyl, Y. Shynbergen
Group – SE-2439

This project is an Educational Crowdfunding Decentralized Application (DApp) built on the Ethereum blockchain using the Sepolia test network. The goal of the project is to demonstrate how blockchain technology can be used to create a transparent and decentralized crowdfunding system with smart contracts and a simple web interface.

The application allows users to create fundraising campaigns, contribute Ethereum (ETH) to campaigns, receive reward tokens, and finalize campaigns when the funding goal is reached.

System Overview

The system consists of three main parts:

1. Smart contracts written in Solidity
2. Deployment and verification on the Sepolia test network
3. A frontend web application that interacts with the smart contracts using MetaMask

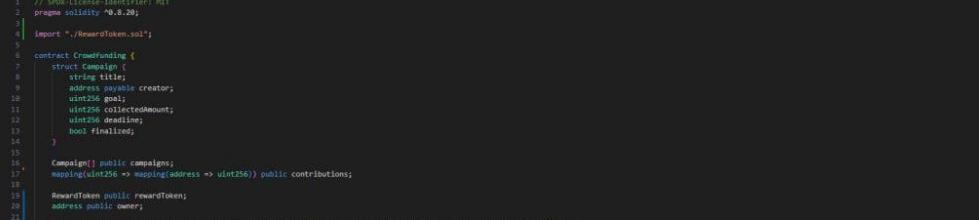
All components are fully connected and working together.

Smart Contracts Description

The project includes two smart contracts:

Crowdfunding.sol
RewardToken.sol

The Crowdfunding contract manages campaigns, contributions, and campaign finalization.
The RewardToken contract is an ERC-20 token used to reward contributors.



```
hardhat config is M verify.json is M solc-0.8-269e6524f7a7ed122d879fd9a79c70c695c.json is M index.html is M tsconfig.json is M style.css is M Crowdfund.sol is M RewardToken.sol is M deploy.js is M README.md is M
```

```
edu-crowdfunding-deep-3 contracts > Crowdfund.sol > Crowdfund > contribute
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.25;
4
5 import "./RewardToken.sol";
6
7 contract Crowdfund {
8     struct Campaign {
9         string title;
10        address payable creator;
11        uint256 goal;
12        uint256 amountCollected;
13        uint256 deadline;
14        bool finalized;
15    }
16
17    Campaign[] public campaigns;
18    mapping(uint256 => mapping(address => uint256)) public contributions;
19
20    RewardToken public rewardToken;
21    address public owner;
22
23    event CampaignCreated(uint256 indexed campaignId, address indexed creator, string title, uint256 goal, uint256 deadline);
24    event Contributed(uint256 indexed campaignId, address indexed contributor, uint256 amountCollected, uint256 rewardUnlocked);
25    event ReachedGoal(uint256 indexed campaignId, bool goalReached, uint256 totalCollected);
26    event RewardDistributed(address indexed token);
27
28    modifier onlyOwner() {
29        require(msg.sender == owner, "Not owner");
30        -
31    }
32
33    constructor() {
34        owner = msg.sender;
35    }
36
37    function setRewardToken(address _tokenAddress) external onlyOwner {
38        require(_tokenAddress != address(0), "Zero address");
39        rewardToken = RewardToken(_tokenAddress);
40        require(rewardToken != address(0), "Token already set");
41        rewardToken.setTokenAddress(_tokenAddress);
42    }
43
44    function createCampaign(
45        string memory _title,
46        uint256 _goal,
47        uint256 _durationInDays
48    ) external {
49        require(_goal > 0, "Goal must be > 0");
50        require(_durationInDays > 0, "Duration must be > 0");
51        require(bytes(_title).length > 0, "Title required");
52    }
53}
```

The screenshot shows the Truffle IDE interface with the RewardToken contract code open. The code is written in Solidity and follows the ERC-20 standard. It includes a constructor, ownership transfer logic, and a minting function restricted to the owner.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "openzeppelin/contracts/token/ERC20/ERC20.sol";

contract RewardToken is ERC20 {
    address public owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    constructor() ERC20("CrowdReward", "CR") {
        owner = msg.sender;
        emit OwnershipTransferred(address(0), owner);
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Only owner");
        _;
    }

    function transferOwnership(address newOwner) external onlyOwner {
        require(newOwner != address(0), "Zero address");
        emit OwnershipTransferred(owner, newOwner);
        owner = newOwner;
    }

    function mint(address to, uint256 amount) external onlyOwner {
        _mint(to, amount);
    }
}
```

RewardToken Contract

RewardToken is an ERC-20 token deployed on the Sepolia network. It follows the standard ERC-20 interface and includes minting functionality.

The mint function is restricted and can only be called by the Crowdfunding contract. This design ensures that users cannot mint tokens manually and can only receive tokens as a reward for contributing to campaigns.

This restriction increases security and prevents abuse of token creation.

The screenshot shows the Truffle IDE interface with the Crowdfunding contract code open. The code is written in Solidity and interacts with the RewardToken contract to manage campaign funding.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "openzeppelin/contracts/token/ERC20/ERC20.sol";
import "openzeppelin/contracts/access/Ownable.sol";

contract Crowdfunding is ERC20, Ownable {
    RewardToken rewardToken;

    struct Campaign {
        string name;
        uint256 goal;
        uint256 amount;
        uint256 deadline;
        mapping(address => uint256) contributions;
    }

    mapping(uint256 => Campaign) campaigns;
    uint256 campaignCount;

    event CampaignCreated(uint256 indexed id, string name, uint256 goal, uint256 deadline);
    event ContributionReceived(uint256 indexed id, address indexed contributor, uint256 amount);
    event CampaignFunded(uint256 indexed id, address indexed beneficiary, uint256 amount);
    event CampaignCompleted(uint256 indexed id, address indexed beneficiary, uint256 amount);
    event CampaignCancelled(uint256 indexed id, address indexed beneficiary, uint256 amount);

    constructor() ERC20("CrowdFund", "CFD") {
        rewardToken = RewardToken(0x...); // Replace with actual RewardToken address
    }

    function createCampaign(string memory _name, uint256 _goal, uint256 _deadline) external onlyOwner {
        uint256 id = campaignCount++;
        campaigns[id] = Campaign({name: _name, goal: _goal, amount: 0, deadline: _deadline});
        emit CampaignCreated(id, _name, _goal, _deadline);
    }

    function contribute(uint256 id, uint256 amount) external {
        Campaign storage campaign = campaigns[id];
        require(campaign.deadline > block.timestamp, "Campaign has ended");
        require(amount > 0, "Amount must be positive");
        campaign.contributions[msg.sender] += amount;
        emit ContributionReceived(id, msg.sender, amount);
    }

    function withdraw(uint256 id) external {
        Campaign storage campaign = campaigns[id];
        require(campaign.amount > 0, "Campaign has not funded yet");
        require(block.timestamp > campaign.deadline, "Campaign has not completed");
        require(campaign.contributions[msg.sender] > 0, "Contributor has not funded");
        uint256 amount = campaign.contributions[msg.sender];
        campaign.contributions[msg.sender] = 0;
        emit CampaignFunded(id, msg.sender, amount);
    }

    function cancelCampaign(uint256 id) external onlyOwner {
        Campaign storage campaign = campaigns[id];
        require(campaign.amount > 0, "Campaign has not funded yet");
        require(block.timestamp < campaign.deadline, "Campaign has not started");
        uint256 amount = campaign.amount;
        campaign.amount = 0;
        emit CampaignCancelled(id, msg.sender, amount);
    }

    function mint(address to, uint256 amount) external onlyOwner {
        rewardToken.mint(to, amount);
    }
}
```

Crowdfunding Contract

The Crowdfunding contract allows users to:

- Create a new crowdfunding campaign
- Contribute ETH to an existing campaign
- Automatically receive reward tokens for contributions
- Finalize a campaign when the funding goal is reached

Each campaign contains the following data:

- Title
- Creator address
- Funding goal (in wei)
- Collected amount
- Deadline
- Finalized status

Campaigns are stored on-chain and can be accessed publicly, ensuring transparency.

The image shows two side-by-side code editors, both displaying the same Solidity smart contract code for a crowdfunding platform. The code is organized into several files: `crowd.sol`, `crowdfund.sol`, `index.html`, `style.css`, `script.js`, and `README.md`. The `crowdfund.sol` file contains the main logic for creating campaigns, contributing, and managing rewards. The `crowd.sol` file contains the interface for the crowdfunding contract. The `index.html` and `style.css` files provide a simple web-based user interface, while `script.js` handles client-side interactions. The `README.md` file provides deployment instructions.

```

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

import "./Rewards.sol";

contract Crowdfunding {
    struct Campaign {
        string title;
        address creator;
        uint256 goal;
        uint256 deadline;
        uint256 collectedAmount;
        bool finalized;
    }

    mapping(uint256 => Campaign) public campaigns;
    mapping(address => mapping(address => uint256)) public contributions;
    Rewards rewards;
}

event CampaignCreated(uint256 indexed campaignId, address indexed creator, string title, uint256 goal, uint256 deadline);
event Contributed(uint256 indexed campaignId, address indexed contributor, uint256 amountWei, uint256 rewardWei);
event CampaignFinalized(uint256 indexed campaignId, bool goalReached, uint256 totalCollected);

modifier onlyOwner() {
    require(msg.sender == owner, "Not owner");
}
constructor() {
    owner = msg.sender;
}
function withdrawFunds(address _to) external onlyOwner {
    require(_to != address(0), "Zero address");
    require(rewards.rewards[_to] == address(0), "Token already set");
    uint256 amount = rewards.rewards[_to];
    emit RewardWithdrawn(_to, amount);
}
function createCampaign(string memory _title, uint256 _goal, uint256 _deadline) external {
    require(_goal > 0, "Goal must be > 0");
    require(_durationInDays > 0, "Duration must be > 0");
    require(_title.length > 0, "Title required");
    uint256 campaignId = campaigns.length + 1;
    msg.sender;
    uint256 rewardWei = 0;
    uint256 rewardAmount = 0;
    if (address(rewardWei) != address(0)) {
        rewardAmount = msg.value * 100;
        rewards.list[msg.sender, rewardAmount];
    }
    emit CampaignCreated(campaignId, msg.sender, _title, _goal, deadline);
}

function contribute(uint256 _campaignId) external payable {
    require(_campaignId < campaigns.length, "Invalid campaignId");
    Campaign storage c = campaigns[_campaignId];
    require(block.timestamp < c.deadline, "Campaign ended");
    require(c.collectedAmount < c.goal, "Goal not reached");
    require(c.collectedAmount + msg.value > c.goal, "Goal not reached");
    c.collectedAmount += msg.value;
    c.collectedWei += msg.value;
    rewards.list[msg.sender, rewardAmount];
}

function finalizeCampaign(uint256 _campaignId) external {
    Campaign storage c = campaigns[_campaignId];
    require(c.collectedAmount >= c.goal, "Goal not reached");
    require(c.collectedAmount >= c.goal, "Goal not reached");
    require(c.creator == msg.sender, "Only creator");
    c.finalized = true;
}

function transferReward(uint256 _campaignId) external {
    uint256 amountWei = c.creator.call.value(c.collectedAmount)("");
    require(amountWei != "0", "Transfer failed");
    require(amountWei == msg.value, "Amount mismatch");
    emit TransferReward(_campaignId, msg.sender, amountWei);
}

contract Rewards {
    mapping(address => uint256) rewards;
}
    
```

Deployment to Sepolia Network

Both smart contracts were compiled using Solidity version 0.8.20 and deployed to the Sepolia test network using Hardhat.

After deployment, the contract addresses were saved and used in the frontend application.

“This error appeared during redeployment because the contract was already deployed and the gas estimation reverted.

It is not a logical error in the smart contract, but a deployment/runtime condition related to testnet execution.”

Contract Verification on Etherscan

After deployment, both contracts were verified on Sepolia Etherscan using the Standard JSON Input method. This allows anyone to view the source code and interact with the contracts directly through Etherscan.

Successful verification confirms that the deployed bytecode matches the source code.

The screenshot shows the Etherscan interface for the Sepolia Testnet. A specific contract is selected, displaying its overview, more info, and multichain info. The 'Contract' tab is active, showing the contract name 'Crowdfunding', compiler version 'v0.8.20+commit.a1b79de6', and optimization status. Below this, the 'Source Code' tab is selected, showing the Solidity code for the contract:

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 import "./Reward.sol";
5
6 contract Crowdfunding {
7     struct Campaign {
8         string title;
```

Crowdfunding

The screenshot shows the Etherscan interface for the Sepolia Testnet, similar to the previous one but with a different URL in the browser address bar: `sepolia.etherscan.io/address/0x38aa4d86a9e09db53550b1a21099f462de6be629#writeContract`. The 'Contract' tab is active, showing the same details as before. The 'Write Contract' tab is selected, and the interface displays a list of functions that can be called:

- Connected - Web3 [0x8e03...b0a2]
- 1. contribute (0xc1bbca7)
- 2. createCampaign (0x3020580b)
- 3. finalizeCampaign (0xb5ca27d)
- 4. setRewardToken (0xdæee8127)

ALL FUNCTIONS OF CONTRACT CROWDFUNDING IN ETHERSCAN AND CONNECT WITH METAMASK

Contract Address: 0x36a3049b546d8021ecff9b36f9d614208736c145

Overview

ETH BALANCE: 0 ETH

More Info

CONTRACT CREATOR: 0x9E038054...e7307b0A2 | 4 hrs ago

TOKEN TRACKER: CrowdReward (CRT)

Multichain Info

N/A

Transactions **Internal Transactions** **Token Transfers (ERC-20)** **Contract** **Events**

Code **Read Contract** **Write Contract**

Contract Source Code Verified (Exact Match)

Contract Name: RewardToken Optimization Enabled: No with 200 runs

Compiler Version: v0.8.20+commit.a1b79de6 Other Settings: shanghai EvmVersion, MIT license

Contract Source Code (Solidity Standard Json Input format)

```

1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.20;
3 
4 import "@openzeppelin/...
5 
6 + contract RewardToken {
7     address public ...
8 }
```

This website uses cookies to improve your experience. By continuing to use this website, you agree to its Terms and Privacy Policy. [Got It!](#)

RewardToken

Contract Address: 0x36a3049b546d8021ecff9b36f9d614208736c145

Overview

ETH BALANCE: 0 ETH

More Info

CONTRACT CREATOR: 0x9E038054...e7307b0A2 | 4 hrs ago

TOKEN TRACKER: CrowdReward (CRT)

Multichain Info

N/A

Transactions **Internal Transactions** **Token Transfers (ERC-20)** **Contract** **Events**

Code **Read Contract** **Write Contract**

Connected - Web3 [0xde03...30a2]

- 1. approve (0xb95ea7b3)
- 2. mint (0x40c1019)
- 3. transfer (0xa959cbb)
- 4. transferFrom (0x23b772dd)
- 5. transferOwnership (0xf2fe35b)

This website uses cookies to improve your experience. By continuing to use this website, you agree to its Terms and Privacy Policy. [Got It!](#)

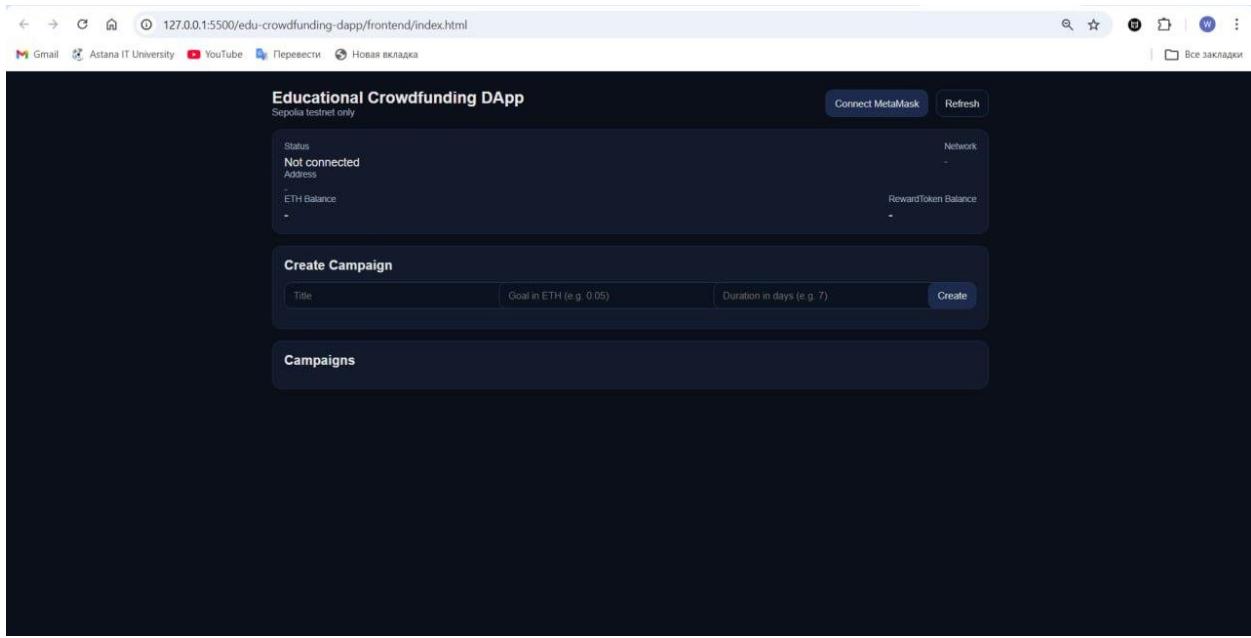
ALL FUNCTIONS OF CONTRACT RTB IN ETHERSCAN AND CONNECT WITH METAMASK

Frontend Application

The frontend is a simple web application built with HTML, CSS, and JavaScript. It uses the ethers.js library to interact with the blockchain.

The frontend provides the following features:

- MetaMask wallet connection
- Network validation (Sepolia only)
- Campaign creation
- ETH contributions
- Campaign finalization
- Display of reward token balance



MetaMask Integration

Users connect their MetaMask wallet to the application. The system checks whether the user is connected to the Sepolia test network. If the wrong network is selected, the application shows a warning message.

Once connected, the user address, ETH balance, and RewardToken balance are displayed.

Educational Crowdfunding DApp
Sepolia testnet only

Status: Connected
Address: 0x038054995cd5ee0EA347d8477Ce7307b0A2
ETH Balance: 0.00003489573528056 RewardToken Balance: 4.52 CRT

Create Campaign

Title: Goal in ETH (e.g. 0.05) Duration in days (e.g. 7) Create

Campaigns

- #0 — My First Campaign Creator: 0x038054995cd5ee0EA347d8477Ce7307b0A2 Deadline: 16.02.2026, 03:13:36

0.02 / 0.05 ETH ACTIVE

Contribute Finalize
- #1 — My First Campaign Creator: 0x038054995cd5ee0EA347d8477Ce7307b0A2 Deadline: 16.02.2026, 03:18:00

0.011 / 0.05 ETH ACTIVE

Contribute Finalize
- #2 — My First Campaign 2 Creator: 0x038054995cd5ee0EA347d8477Ce7307b0A2 Deadline: 16.02.2026, 03:25:00

0.0 / 0.05 ETH ACTIVE

Contribute Finalize
- #3 — erasyl Creator: 0x038054995cd5ee0EA347d8477Ce7307b0A2

0.0 / 0.05 ETH

Creating a Campaign

A connected user can create a new campaign by entering:

- Campaign title
- Funding goal in ETH
- Duration in days

After submitting the form, a transaction is sent to the blockchain, and the campaign is stored in the smart contract.

Educational Crowdfunding DApp
Sepolia testnet only

Status: Connected
Address: 0x038054995cd5ee0EA347d8477Ce7307b0A2
ETH Balance: 0 05890659573528056 RewardToken Balance: 4.52 CRT

Create Campaign

ERASYL 0.001 1 Create

Pending: 0x4a805e0be0b61aab703e73d7dd61ab11a3eb6c7edc40572fca8780cf57e

Campaigns

- #0 — My First Campaign Creator: 0x038054995cd5ee0EA347d8477Ce7307b0A2 Deadline: 16.02.2026, 03:13:36

0.02 / 0.05 ETH ACTIVE

Contribute Finalize
- #1 — My First Campaign Creator: 0x038054995cd5ee0EA347d8477Ce7307b0A2 Deadline: 16.02.2026, 03:18:00

0.011 / 0.05 ETH ACTIVE

Contribute Finalize
- #2 — My First Campaign 2 Creator: 0x038054995cd5ee0EA347d8477Ce7307b0A2 Deadline: 16.02.2026, 03:25:00

0.0 / 0.05 ETH ACTIVE

Contribute Finalize
- #3 — erasyl Creator: 0x038054995cd5ee0EA347d8477Ce7307b0A2

0.0 / 0.05 ETH

MetaMask

Account 1

Обзор Купить Обмен... Отправ... Получ...

Токены DeFi NFT Деятельность

Sepolia

Feb 9, 2026

Create Campaign -0 SepoliaETH
Ожидаящий -0 SepoliaETH
Отмена Ускорить

Contribute -0.0001 Sepolia...
Подтверждено -0.0001 SepoliaETH

Contribute -0.0001 Sepolia...
Подтверждено -0.0001 SepoliaETH

The screenshot shows the Educational Crowdfunding DApp interface. On the left, there's a 'Create Campaign' section where a user has entered 'ERASYL' and a value of '0.001'. On the right, a list of existing campaigns is shown, including '#0 — My First Campaign', '#2 — My First Campaign', and '#3 — erasyl'. Each campaign entry includes details like creator address, deadline, current progress (e.g., '0.02 / 0.05 ETH'), and status ('ACTIVE'). To the right of the campaigns is a MetaMask wallet overlay titled 'Запрос транзакции' (Transaction Request). It displays the transaction details: 'Sepolia' network, 'Account 1' (Wallet 1), and a message: 'Прогнозируемые изменения' (Predicted changes) with 'Нет изменений' (No changes). It also shows the transaction parameters: 'Сеть' (Network) set to Sepolia, 'Запрос от' (From) set to '192.168.56.1:3000', and 'Взаимодействие с' (Interaction with) set to '0x38AA4_be629'. Below these are fields for 'Комиссия сети' (Network fee) set to '0.0003 s SepollaETH' and 'Скорость' (Speed) set to 'Рынок ~12 сек.' (Market ~12 sec.). At the bottom are 'Отмена' (Cancel) and 'Подтвердить' (Confirm) buttons.

Contributing to a Campaign

Any user can contribute ETH to an active campaign. When a contribution is made:

- ETH is transferred to the campaign
- The collected amount is updated
- Reward tokens are minted and sent to the contributor

The frontend updates campaign progress in real time after the transaction is confirmed.

This screenshot is similar to the one above but shows the transaction confirmation process. The MetaMask overlay now displays the transaction details: 'Вы отправляете - 0,001 s SepollaETH' (You are sending - 0,001 s SepollaETH) and 'Вы получаете + 0,1 0x38AA4_be629' (You receive + 0,1 0x38AA4_be629). The transaction status is shown as 'Предоставлено' (Submitted). The rest of the interface and the campaign list remain the same as in the previous screenshot.

#7 — ERASYL

Creator: 0x9E038054995cdf5eeDEA3f47df6477Ce7307b0A2

Deadline: 10.02.2026, 06.26.24

0.002 / 0.001 ETH

ACTIVE

Contribute Finalize

Pending: 0x909d4df4fb191e2624a185d4471592a4bca2661712aa21ab82d062b53a18c797

Finalizing a Campaign

A campaign can be finalized when the funding goal is reached. Finalization marks the campaign as completed and prevents further contributions.

The finalized status is stored on-chain and visible to all users. This ensures that campaign results are transparent and cannot be modified.

Reward Token Distribution

After contributing to a campaign, users receive RewardToken (CRT). The token balance is displayed in the frontend and can also be verified on Etherscan.

This demonstrates how smart contracts can be used to implement incentive mechanisms in decentralized applications.

Educational Crowdfunding DApp

Sepolia testnet only

Status: Connected

Address: 0x9E038054995cdf5eeDEA3f47df6477Ce7307b0A2

ETH Balance: 0.044585219978055926

Network: Sepolia (chainId 11155111)

RewardToken Balance: 5.82 CRT

Connect MetaMask Refresh

sepolia.etherscan.io/address/0x36a3049b546d8021ecff9b36f9d614208736c145#readContract

Transactions Internal Transactions Token Transfers (ERC-20) Contract Events

Code Read Contract Write Contract

Connected - Web3 [Index 3, 0xa]

Rest Contract Information

1. allowance (0xb082ed3e)

2. balanceOf (0x7a0a8231)

account (address)

0x9E038054995cdf5eeDEA3f47df6477Ce7307b0A2

Query

0x909d4df4fb191e2624a185d4471592a4bca2661712aa21ab82d062b53a18c797

3. decimals (0x313e567)

4. name (0x080ded03)

5. owner (0xdada5cb0a)

6. symbol (0x95d0bb41)

7. totalSupply (0x18160dd5)

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

Security and Transparency

The system ensures security and transparency through:

- Restricted token minting
- On-chain campaign data
- Public contract verification
- Immutable transaction history

All important actions require blockchain transactions and user confirmation via MetaMask.

Conclusion

This project successfully demonstrates a working decentralized crowdfunding platform using Ethereum smart contracts and a web-based frontend.

The application allows users to create campaigns, contribute ETH, receive reward tokens, and finalize campaigns in a transparent and secure way. The use of blockchain technology ensures trust, immutability, and decentralization.

All project requirements were fully implemented, deployed, tested, and demonstrated on the Sepolia test network.