ETHCrowdfund DApp

Final Project Report

Solidity,ERC-20,Metamask,Hardhat

School of Computer Engineering

Course: Blockchain Technologies

Uni: Astana IT University (AITU)

Group: SE-2434

Date: February 2026

Students: Ayan A., Askartau, Ruslan D.

Github link: https://github.com/ayanvrsn/Final-project

Introduction

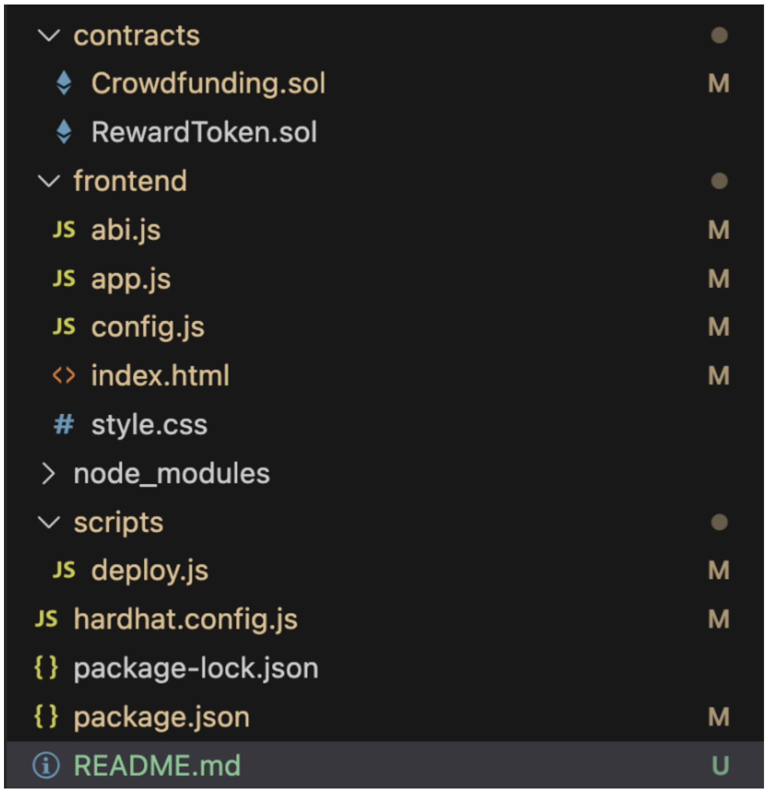
This report documents the development, deployment, and validation of a decentralized Crowdfund rewards application based on Hard hat, Solidity, and ethers.js.

The system solves the problem of transparent fundraising for campaigns with a reward program: participants contribute to ETH, and upon successful completion of the campaign they receive ERC-20 tokens.

In addition to demonstrating functionality, special emphasis is placed on the analysis of architectural solutions, risks and limitations of the current implementation.

Screenshots of each step and features with description:

Design structure and separation of layers



```
contracts
  Crowdfunding.sol        M
  RewardToken.sol
frontend
  abi.js                  M
  app.js                  M
  config.js               M
  index.html              M
  style.css
node_modules
scripts
  deploy.js               M
hardhat.config.js         M
package-lock.json
package.json              M
README.md                 U
```

Install dependencies

npm install

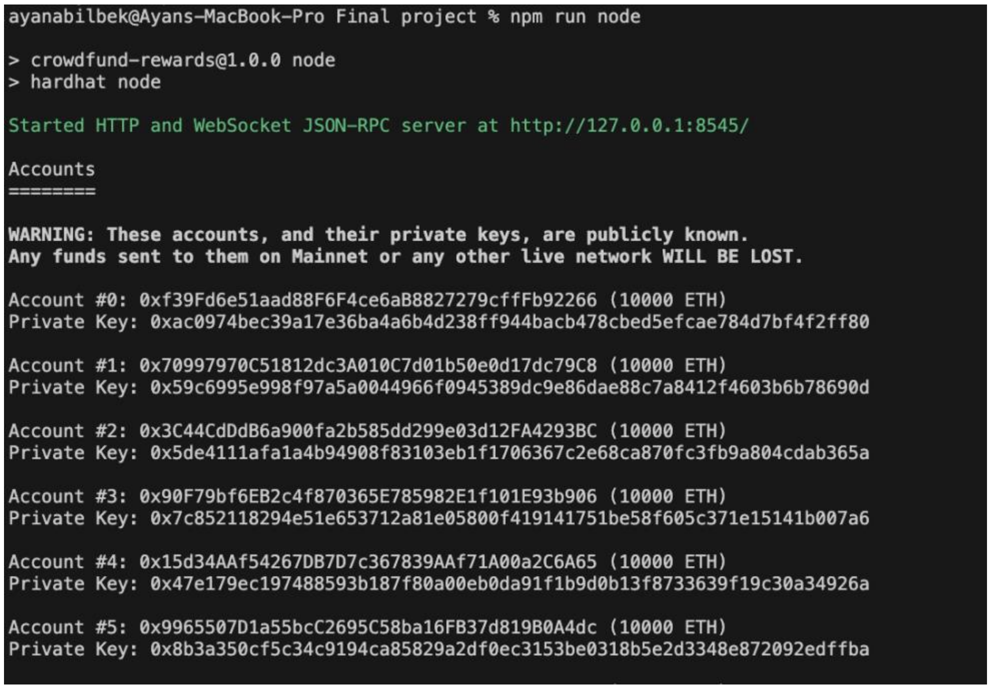
Compile contracts

npm run compile

Start local blockchain

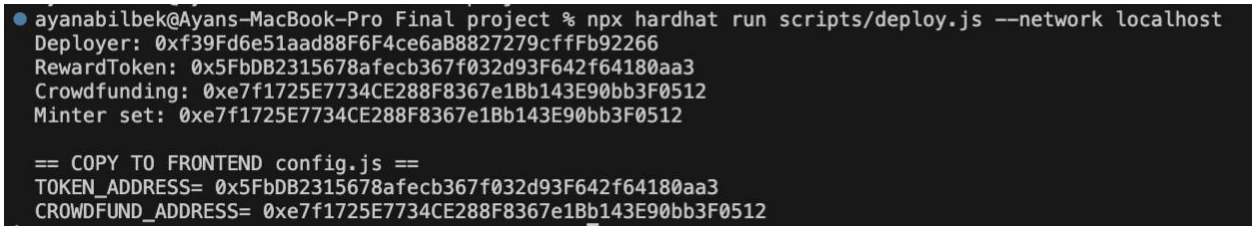
npm run node

Terminal: npm run node



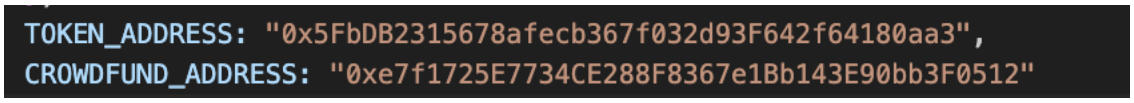
```
ayanabilbek@Ayans-MacBook-Pro Final project % npm run node

> crowdfund-rewards@1.0.0 node
> hardhat node

Started HTTP and WebSocket JSON-RPC server at http://127.0.0.1:8545/

Accounts
========

WARNING: These accounts, and their private keys, are publicly known.
Any funds sent to them on Mainnet or any other live network WILL BE LOST.

Account #0: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266 (10000 ETH)
Private Key: 0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80

Account #1: 0x70997970C51812dc3A010C7d01b50e0d17dc79C8 (10000 ETH)
Private Key: 0x59c6995e998f97a5a0044966f0945389dc9e86dae88c7a8412f4603b6b78690d

Account #2: 0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC (10000 ETH)
Private Key: 0x5de4111afa1a4b94908f83103eb1f1706367c2e68ca870fc3fb9a804cdab365a

Account #3: 0x90F79bf6EB2c4f870365E785982E1f101E93b906 (10000 ETH)
Private Key: 0x7c852118294e51e653712a81e05800f419141751be58f605c371e15141b007a6

Account #4: 0x15d34AAf54267DB7D7c367839AAf71A00a2C6A65 (10000 ETH)
Private Key: 0x47e179ec197488593b187f80a00eb0da91f1b9d0b13f8733639f19c30a34926a

Account #5: 0x9965507D1a55bcC2695C58ba16FB37d819B0A4dc (10000 ETH)
Private Key: 0x8b3a350cf5c34c9194ca85829a2df0ec3153be0318b5e2d3348e872092edffba
```

Terminal: npx hardhat run scripts/deploy.js --network localhost



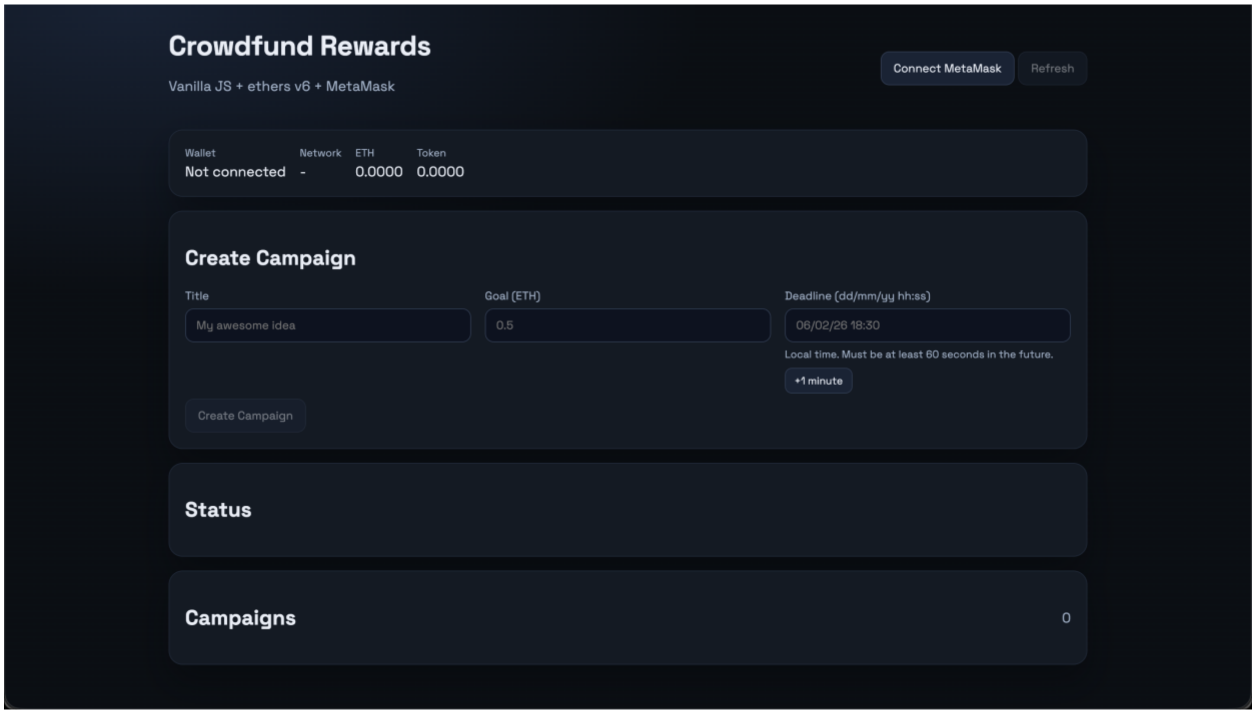
```
● ayanabilbek@Ayans-MacBook-Pro Final project % npx hardhat run scripts/deploy.js --network localhost
  Deployer: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
  RewardToken: 0x5FbDB2315678afecb367f032d93F642f64180aa3
  Crowdfunding: 0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512
  Minter set: 0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512

  == COPY TO FRONTEND config.js ==
  TOKEN_ADDRESS= 0x5FbDB2315678afecb367f032d93F642f64180aa3
  CROWDFUND_ADDRESS= 0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512
```

Update token and contract addresses:



```
TOKEN_ADDRESS: "0x5FbDB2315678afecb367f032d93F642f64180aa3",
CROWDFUND_ADDRESS: "0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512"
```

Frontend UI:



Crowdfund Rewards

Vanilla JS + ethers v6 + MetaMask

Wallet	Network	ETH	Token
Not connected	-	0.0000	0.0000

Create Campaign

Title: My awesome idea

Goal (ETH): 0.5

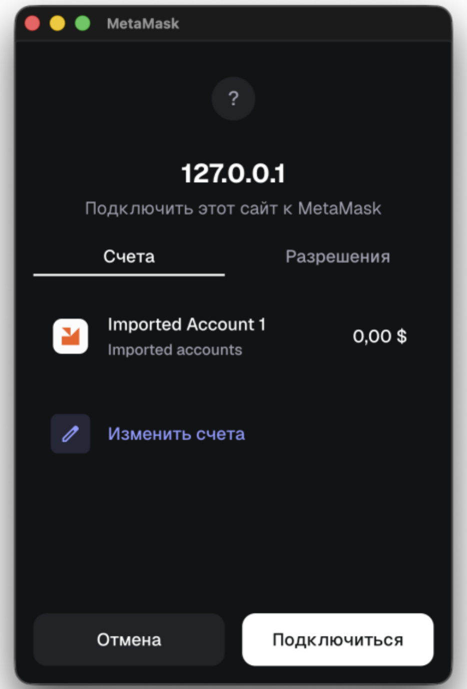
Deadline (dd/mm/yy hh:ss): 06/02/26 18:30

Local time. Must be at least 60 seconds in the future.

Status

Campaigns 0

Connect Metamask wallet



127.0.0.1

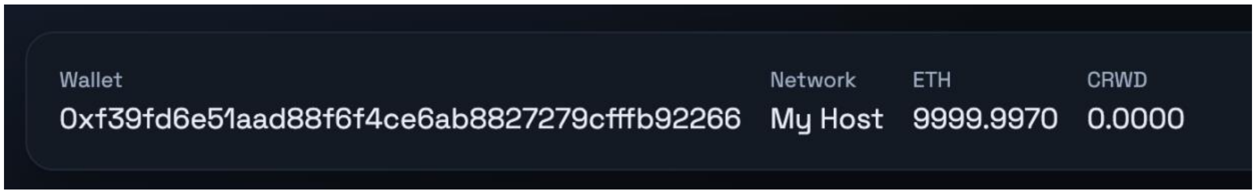
Подключить этот сайт к MetaMask

Imported Account 1 — 0,00 $

Imported accounts

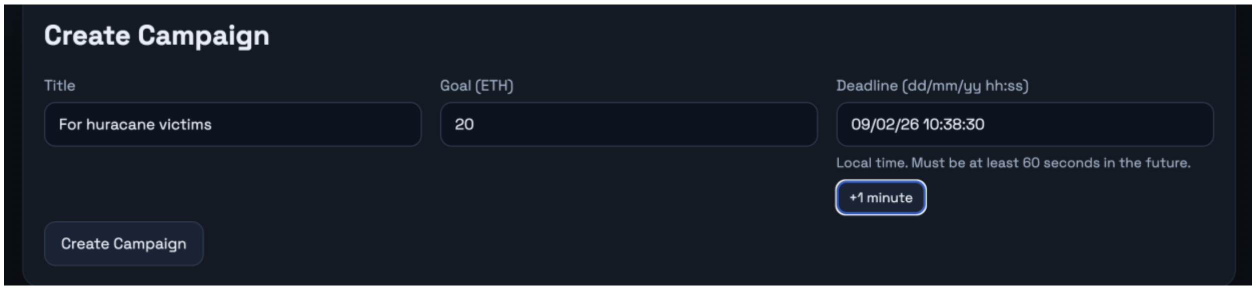
User approval for connecting

After connection



Wallet	Network	ETH	CRWD
0xf39fd6e51aad88f6f4ce6ab8827279cfffb92266	My Host	9999.9970	0.0000

Creation of campaign



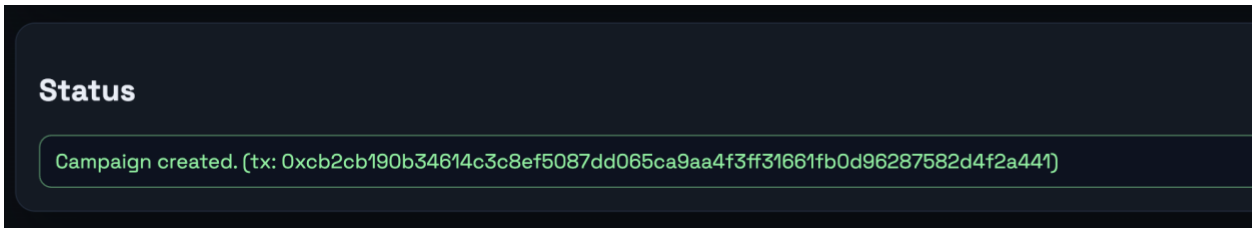
Create Campaign

Title: For huracane victims

Goal (ETH): 20

Deadline (dd/mm/yy hh:ss): 09/02/26 10:38:30

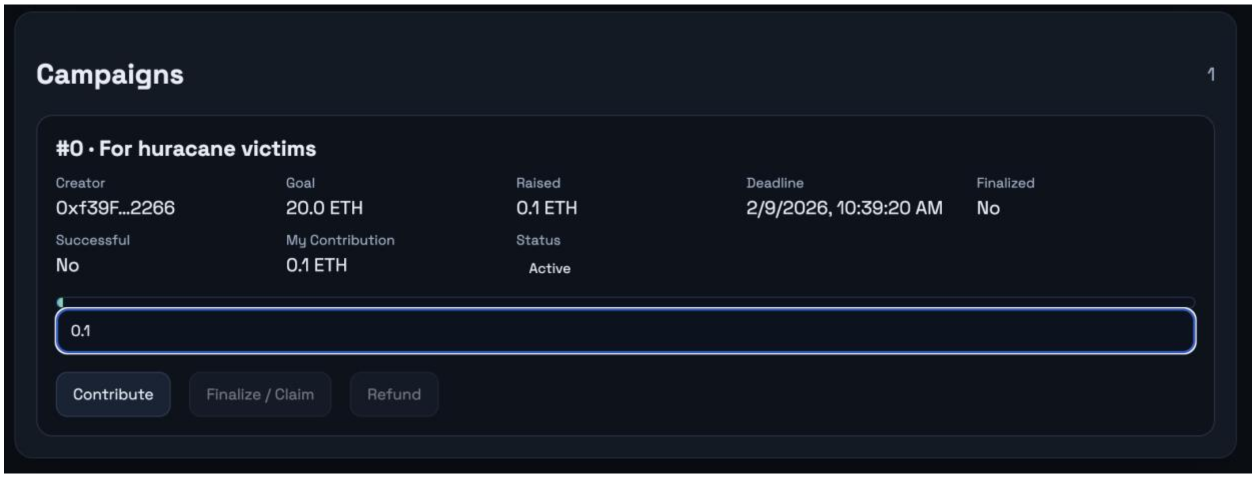
Local time. Must be at least 60 seconds in the future.



Status

Campaign created. (tx: 0xcb2cb190b34614c3c8ef5087dd065ca9aa4f3ff31661fb0d96287582d4f2a441)

Contribute to campaign



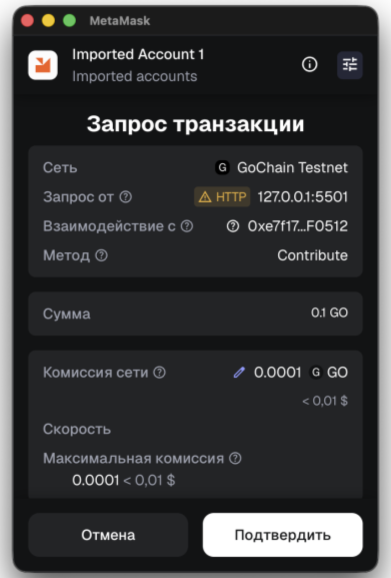
Campaigns 1

#0 · For huracane victims

Creator	Goal	Raised	Deadline	Finalized
0xf39F...2266	20.0 ETH	0.1 ETH	2/9/2026, 10:39:20 AM	No

Successful	My Contribution	Status
No	0.1 ETH	Active

0.1

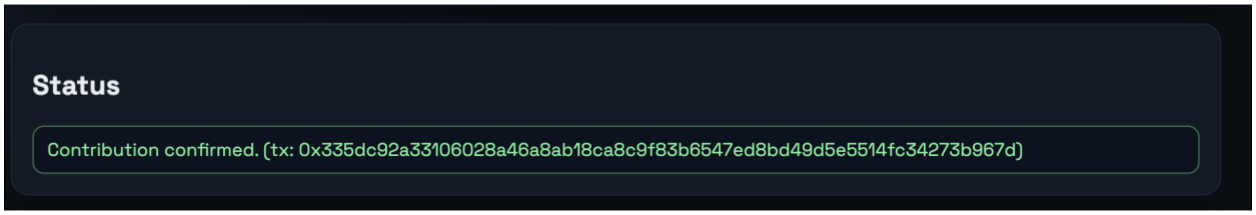


MetaMask

Imported Account 1
Imported accounts

Запрос транзакции

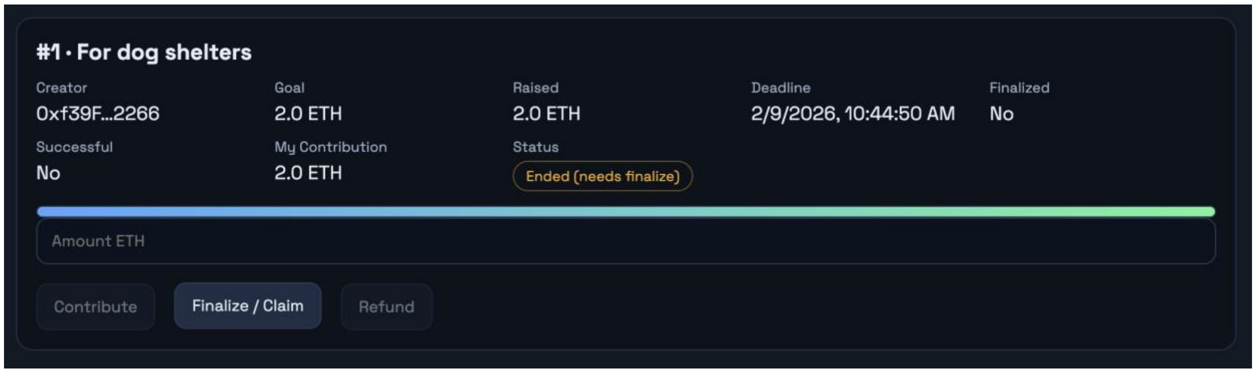
Сеть	GoChain Testnet
Запрос от	HTTP 127.0.0.1:5501
Взаимодействие с	0xe7f17...F0512
Метод	Contribute
Сумма	0.1 GO
Комиссия сети	0.0001 GO < 0,01 $
Скорость	
Максимальная комиссия	0.0001 < 0,01 $



Status

Contribution confirmed. (tx: 0x335dc92a33106028a46a8ab18ca8c9f83b6547ed8bd49d5e5514fc34273b967d)

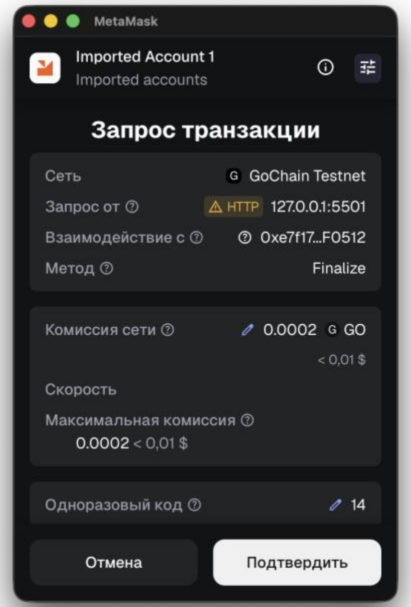
Reward after finalization when goal achieved:



#1 • For dog shelters

Creator	Goal	Raised	Deadline	Finalized
0xf39F...2266	2.0 ETH	2.0 ETH	2/9/2026, 10:44:50 AM	No

Successful	My Contribution	Status
No	2.0 ETH	Ended (needs finalize)


MetaMask

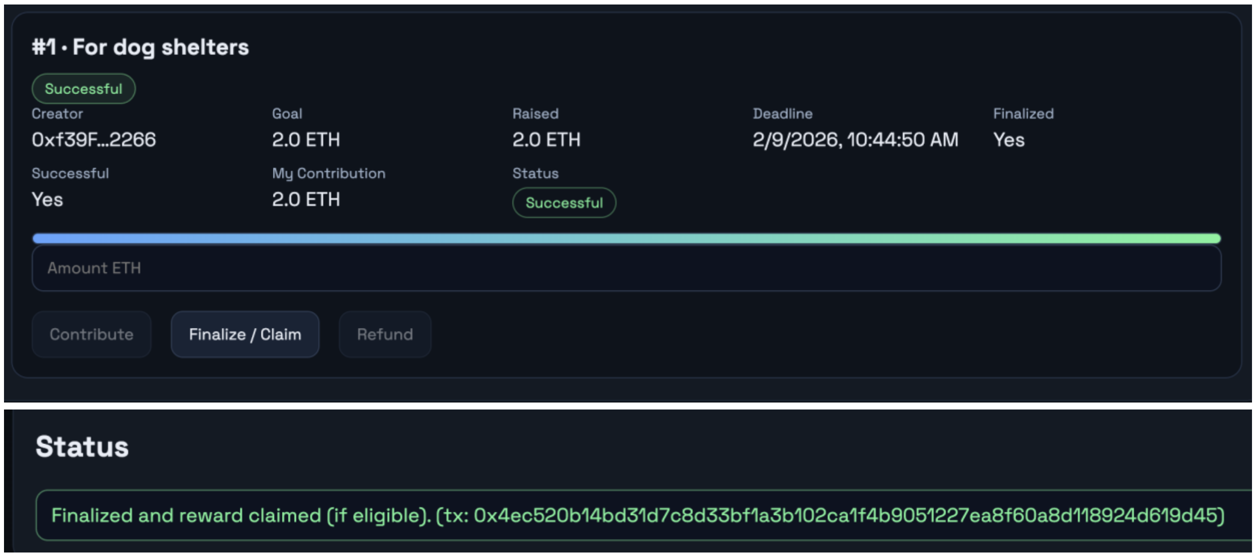
Imported Account 1

Imported accounts

Запрос транзакции

Сеть	GoChain Testnet
Запрос от	HTTP 127.0.0.1:5501
Взаимодействие с	0xe7f17...F0512
Метод	Finalize

Комиссия сети	0.0002 GO
	< 0,01 $
Скорость	
Максимальная комиссия	0.0002 < 0,01 $
Одноразовый код	14


#1 · For dog shelters

Successful

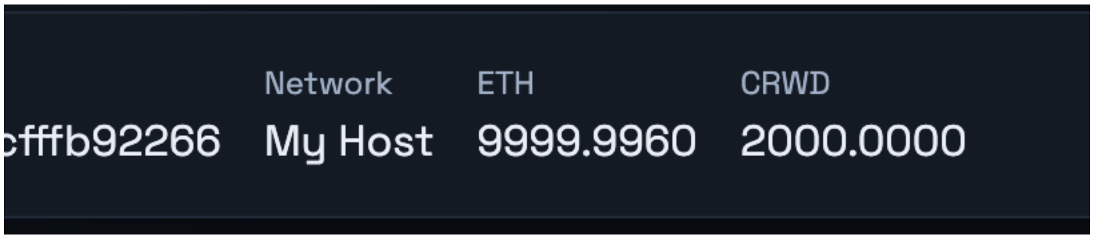
Creator	Goal	Raised	Deadline	Finalized
0xf39F...2266	2.0 ETH	2.0 ETH	2/9/2026, 10:44:50 AM	Yes

Successful	My Contribution	Status
Yes	2.0 ETH	Successful

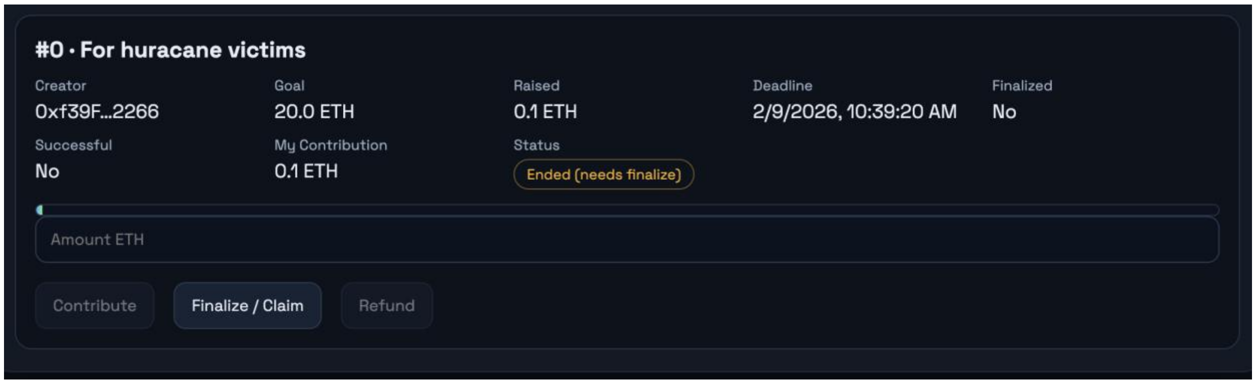
Status

Finalized and reward claimed (if eligible). (tx: 0x4ec520b14bd31d7c8d33bf1a3b102ca1f4b9051227ea8f60a8d118924d619d45)

And as reward CRWD tokens are added to contributors (Formula: **Reward = ETH contributed * 1000**)

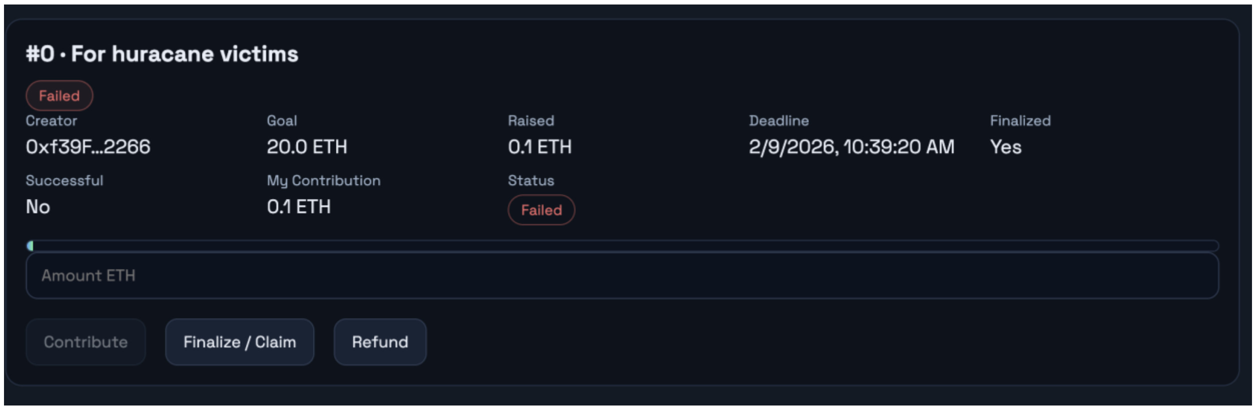

	Network	ETH	CRWD
cfffb92266	My Host	9999.9960	2000.0000

Refund if goal not achieved


#0 • For huracane victims

Creator	Goal	Raised	Deadline	Finalized
0xf39F...2266	20.0 ETH	0.1 ETH	2/9/2026, 10:39:20 AM	No

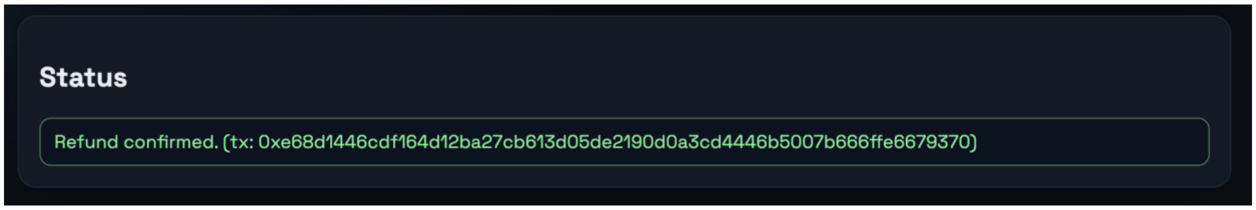
Successful	My Contribution	Status
No	0.1 ETH	Ended (needs finalize)


#0 • For huracane victims

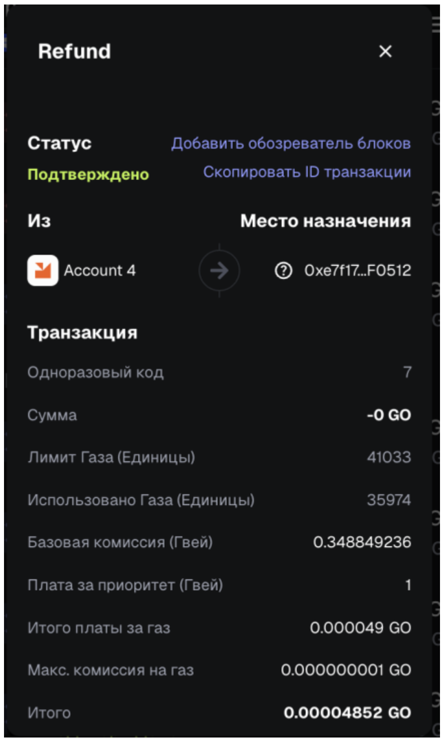
Failed

Creator	Goal	Raised	Deadline	Finalized
0xf39F...2266	20.0 ETH	0.1 ETH	2/9/2026, 10:39:20 AM	Yes

Successful	My Contribution	Status
No	0.1 ETH	Failed


Status

Refund confirmed. (tx: 0xe68d1446cdf164d12ba27cb613d05de2190d0a3cd4446b5007b666ffe6679370)



Refund

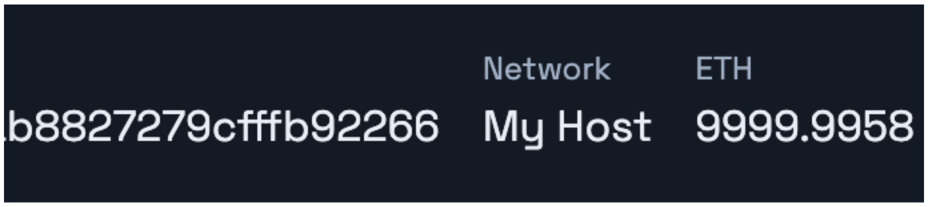
Статус	Добавить обозреватель блоков
Подтверждено	Скопировать ID транзакции
Из	Место назначения
Account 4	0xe7f17...F0512

Транзакция

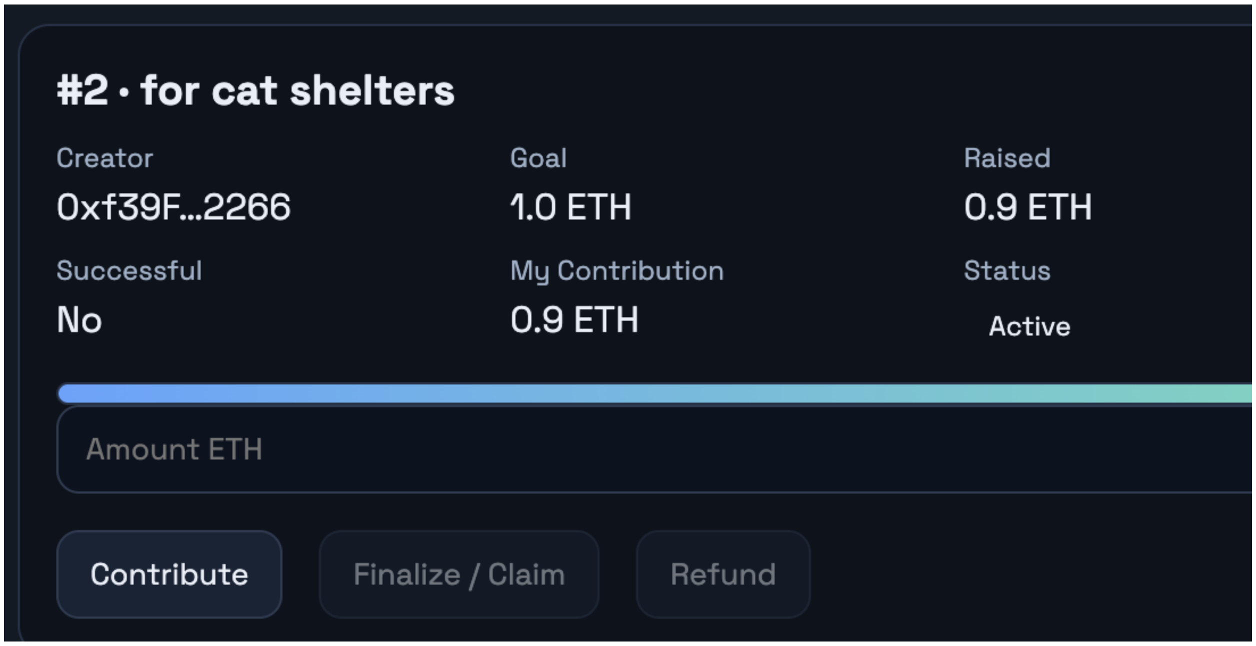
Одноразовый код	7
Сумма	-0 GO
Лимит Газа (Единицы)	41033
Использовано Газа (Единицы)	35974
Базовая комиссия (Гвей)	0.348849236
Плата за приоритет (Гвей)	1
Итого платы за газ	0.000049 GO
Макс. комиссия на газ	0.000000001 GO
Итого	0.00004852 GO

One more example for refund

Before contribution



	Network	ETH
b8827279cfffb92266	My Host	9999.9958

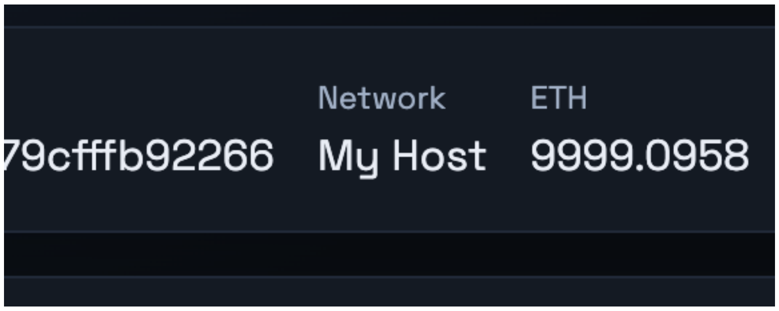


#2 • for cat shelters

Creator	Goal	Raised
0xf39F...2266	1.0 ETH	0.9 ETH

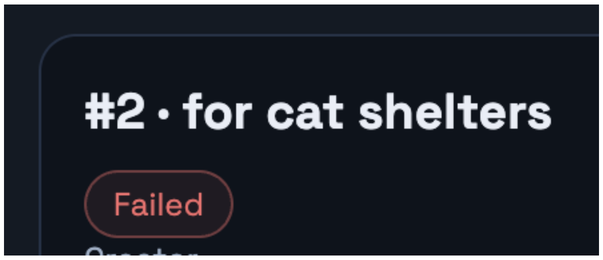
Successful	My Contribution	Status
No	0.9 ETH	Active

After contribution of 9 ETH

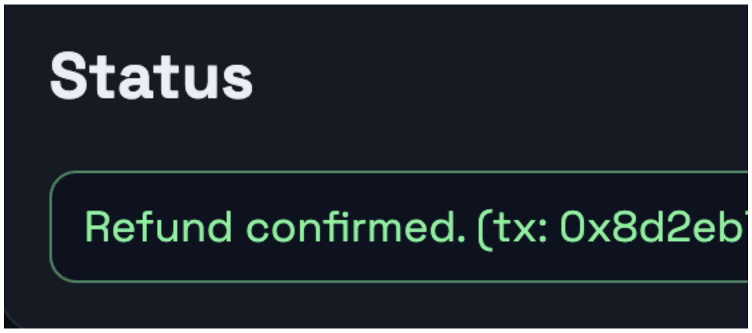


	Network	ETH
79cfffb92266	My Host	9999.0958

Campaign failed and we refund



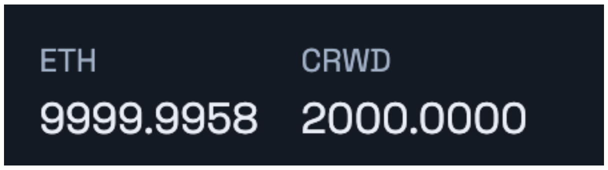
#2 • for cat shelters



Status

Refund confirmed. (tx: 0x8d2eb

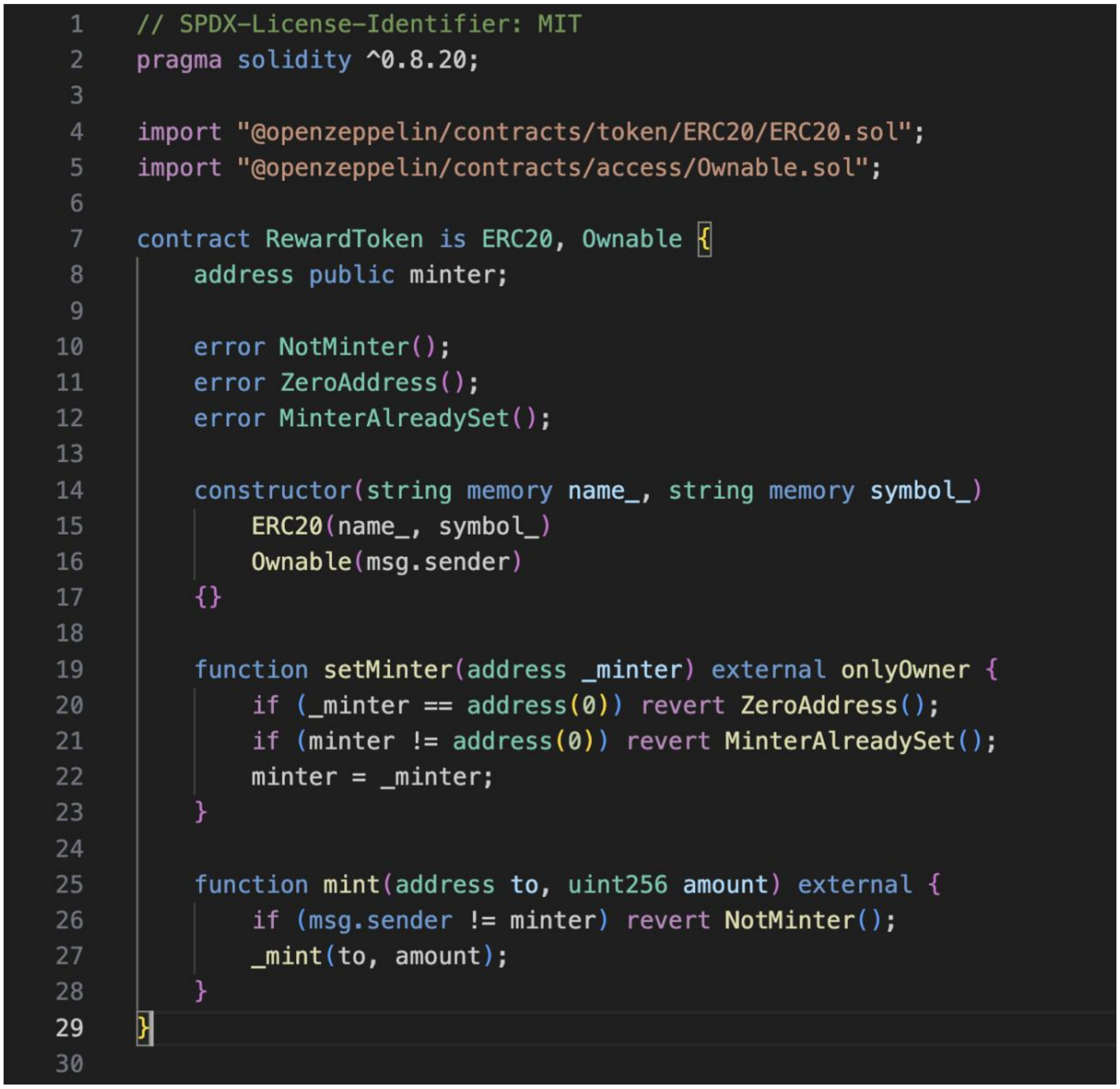
After we finalized campaign and refunded it



ETH	CRWD
9999.9958	2000.0000

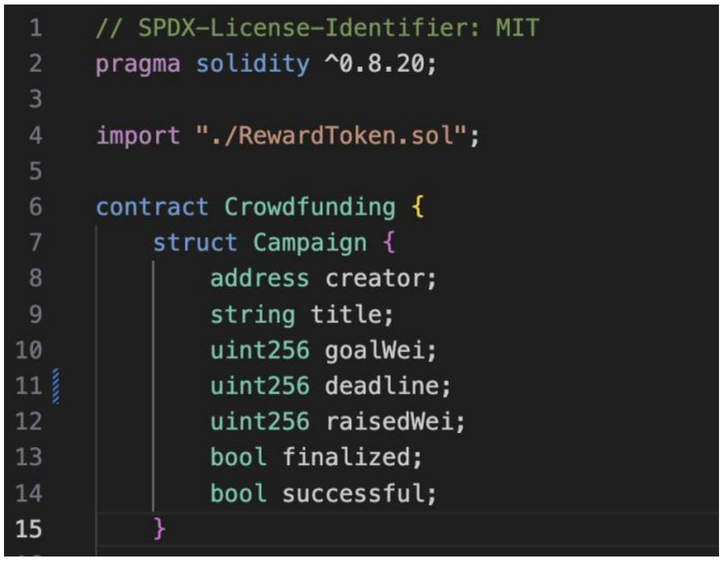
RewardToken.sol

Minting control of reward tokens

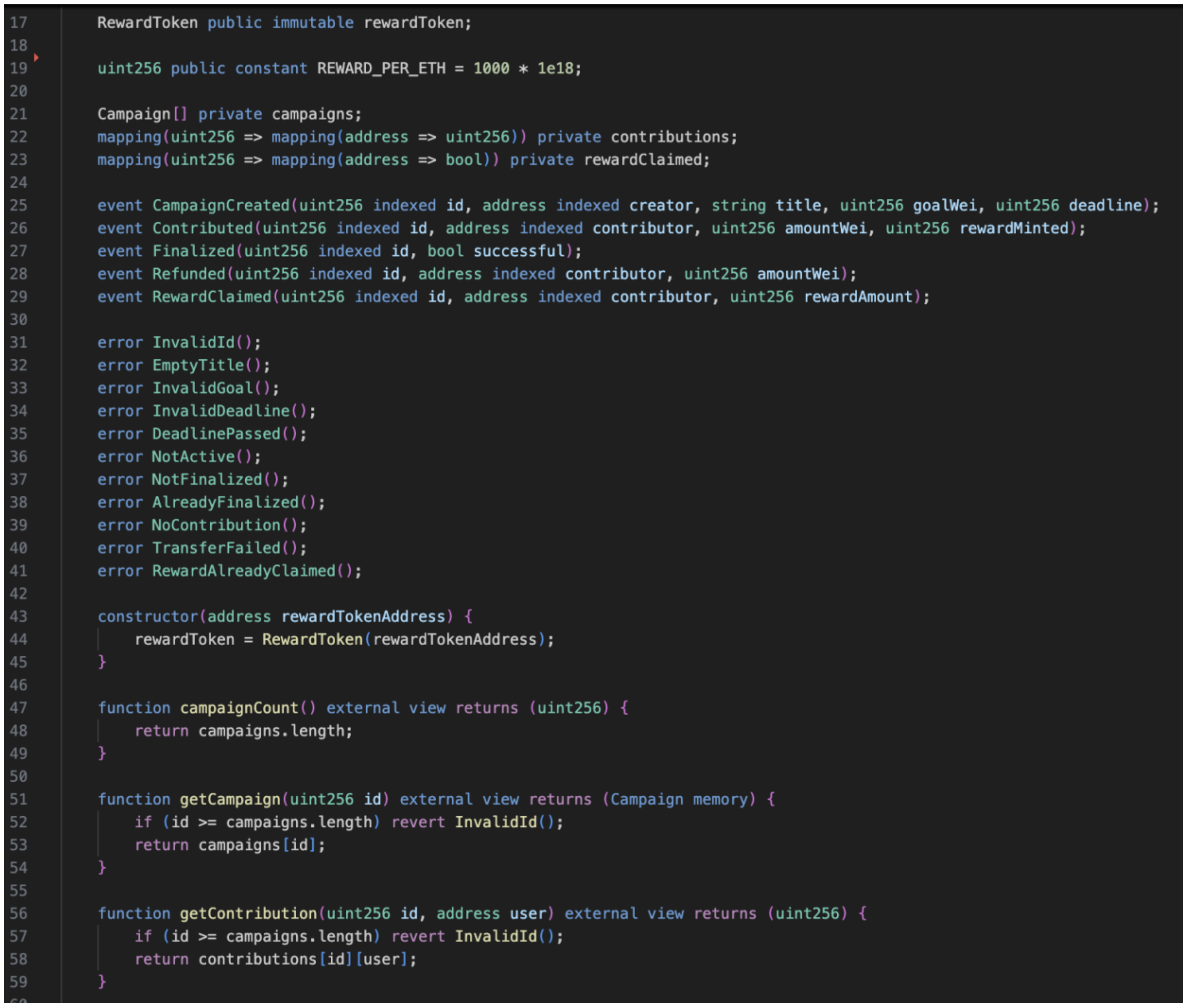


```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

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

contract RewardToken is ERC20, Ownable {
    address public minter;

    error NotMinter();
    error ZeroAddress();
    error MinterAlreadySet();

    constructor(string memory name_, string memory symbol_)
        ERC20(name_, symbol_)
        Ownable(msg.sender)
    {}

    function setMinter(address _minter) external onlyOwner {
        if (_minter == address(0)) revert ZeroAddress();
        if (minter != address(0)) revert MinterAlreadySet();
        minter = _minter;
    }

    function mint(address to, uint256 amount) external {
        if (msg.sender != minter) revert NotMinter();
        _mint(to, amount);
    }
}

```

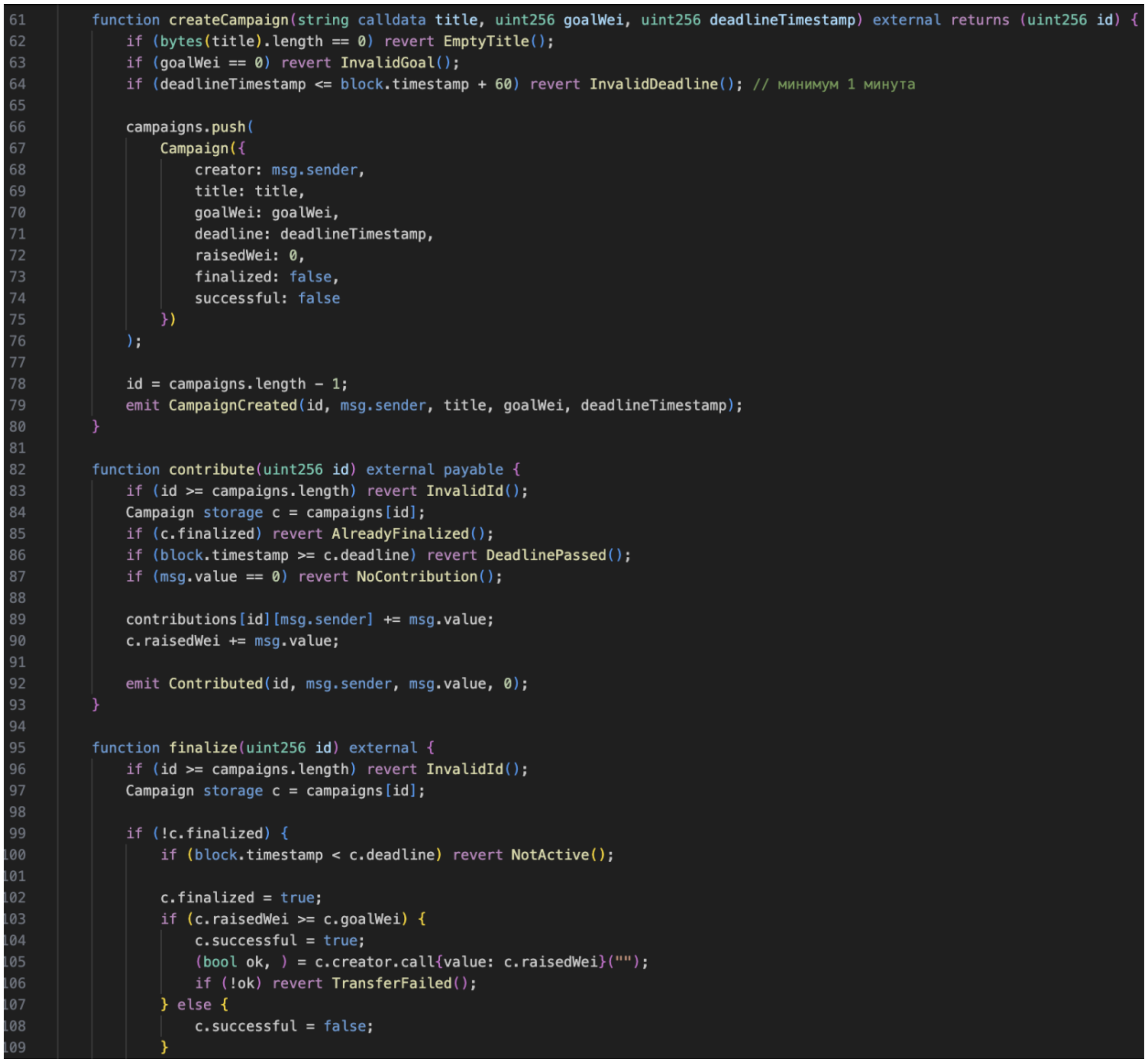
Crowdfunding.sol
Shows that the system takes into account the contribution of each participant and prevents double claims.



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

import "./RewardToken.sol";

contract Crowdfunding {
    struct Campaign {
        address creator;
        string title;
        uint256 goalWei;
        uint256 deadline;
        uint256 raisedWei;
        bool finalized;
        bool successful;
    }
```

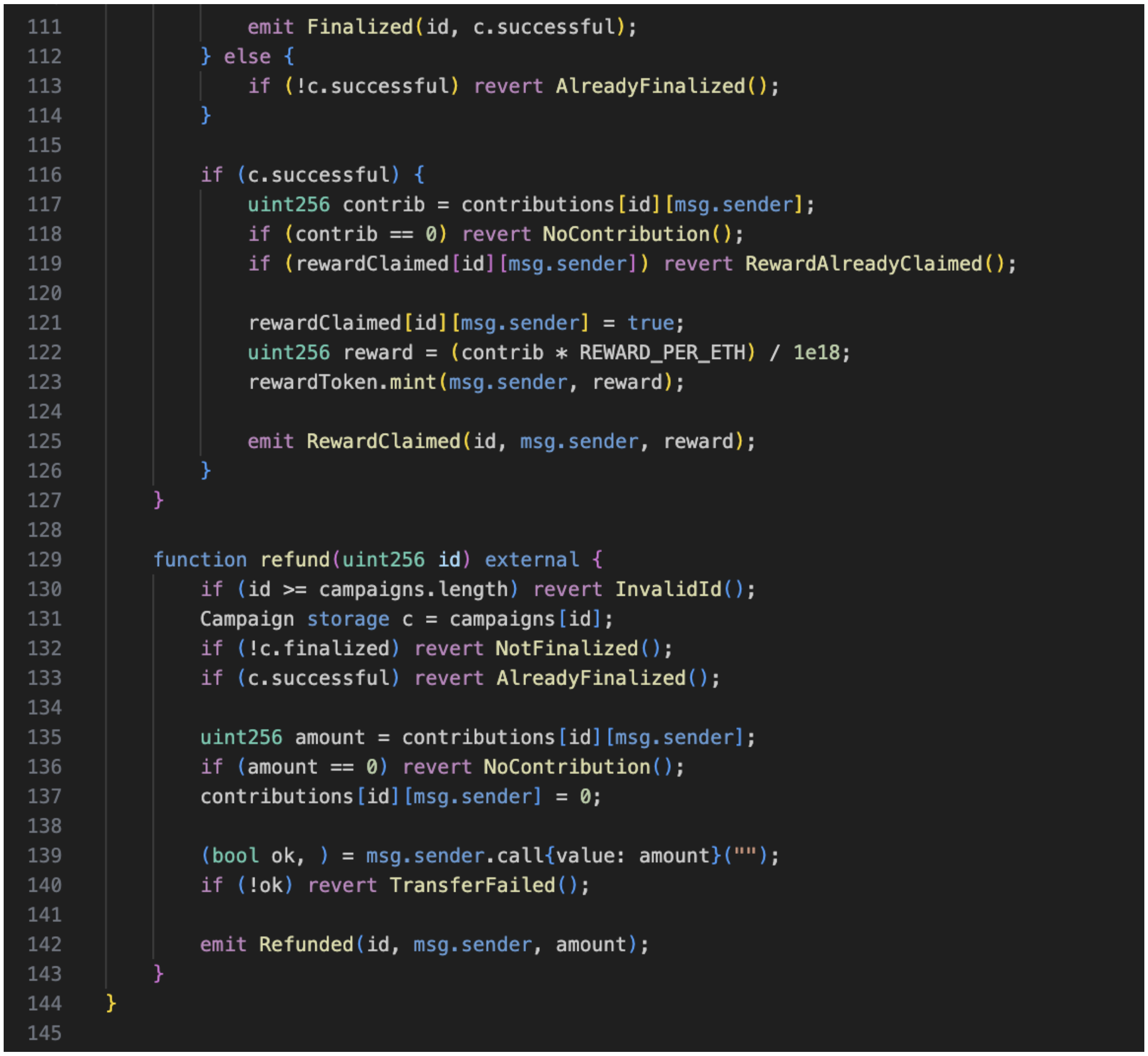


```solidity
    RewardToken public immutable rewardToken;

    uint256 public constant REWARD_PER_ETH = 1000 * 1e18;

    Campaign[] private campaigns;
    mapping(uint256 => mapping(address => uint256)) private contributions;
    mapping(uint256 => mapping(address => bool)) private rewardClaimed;

    event CampaignCreated(uint256 indexed id, address indexed creator, string title, uint256 goalWei, uint256 deadline);
    event Contributed(uint256 indexed id, address indexed contributor, uint256 amountWei, uint256 rewardMinted);
    event Finalized(uint256 indexed id, bool successful);
    event Refunded(uint256 indexed id, address indexed contributor, uint256 amountWei);
    event RewardClaimed(uint256 indexed id, address indexed contributor, uint256 rewardAmount);

    error InvalidId();
    error EmptyTitle();
    error InvalidGoal();
    error InvalidDeadline();
    error DeadlinePassed();
    error NotActive();
    error NotFinalized();
    error AlreadyFinalized();
    error NoContribution();
    error TransferFailed();
    error RewardAlreadyClaimed();

    constructor(address rewardTokenAddress) {
        rewardToken = RewardToken(rewardTokenAddress);
    }

    function campaignCount() external view returns (uint256) {
        return campaigns.length;
    }

    function getCampaign(uint256 id) external view returns (Campaign memory) {
        if (id >= campaigns.length) revert InvalidId();
        return campaigns[id];
    }

    function getContribution(uint256 id, address user) external view returns (uint256) {
        if (id >= campaigns.length) revert InvalidId();
        return contributions[id][user];
    }
```



```solidity
    function createCampaign(string calldata title, uint256 goalWei, uint256 deadlineTimestamp) external returns (uint256 id) {
        if (bytes(title).length == 0) revert EmptyTitle();
        if (goalWei == 0) revert InvalidGoal();
        if (deadlineTimestamp <= block.timestamp + 60) revert InvalidDeadline(); // минимум 1 минута

        campaigns.push(
            Campaign({
                creator: msg.sender,
                title: title,
                goalWei: goalWei,
                deadline: deadlineTimestamp,
                raisedWei: 0,
                finalized: false,
                successful: false
            })
        );

        id = campaigns.length - 1;
        emit CampaignCreated(id, msg.sender, title, goalWei, deadlineTimestamp);
    }

    function contribute(uint256 id) external payable {
        if (id >= campaigns.length) revert InvalidId();
        Campaign storage c = campaigns[id];
        if (c.finalized) revert AlreadyFinalized();
        if (block.timestamp >= c.deadline) revert DeadlinePassed();
        if (msg.value == 0) revert NoContribution();

        contributions[id][msg.sender] += msg.value;
        c.raisedWei += msg.value;

        emit Contributed(id, msg.sender, msg.value, 0);
    }

    function finalize(uint256 id) external {
        if (id >= campaigns.length) revert InvalidId();
        Campaign storage c = campaigns[id];

        if (!c.finalized) {
            if (block.timestamp < c.deadline) revert NotActive();

            c.finalized = true;
            if (c.raisedWei >= c.goalWei) {
                c.successful = true;
                (bool ok, ) = c.creator.call{value: c.raisedWei}("");
                if (!ok) revert TransferFailed();
            } else {
                c.successful = false;
            }
```

The full coverage of the business process: from creation to payment/refund.



```solidity
            emit Finalized(id, c.successful);
        } else {
            if (!c.successful) revert AlreadyFinalized();
        }

        if (c.successful) {
            uint256 contrib = contributions[id][msg.sender];
            if (contrib == 0) revert NoContribution();
            if (rewardClaimed[id][msg.sender]) revert RewardAlreadyClaimed();

            rewardClaimed[id][msg.sender] = true;
            uint256 reward = (contrib * REWARD_PER_ETH) / 1e18;
            rewardToken.mint(msg.sender, reward);

            emit RewardClaimed(id, msg.sender, reward);
        }
    }

    function refund(uint256 id) external {
        if (id >= campaigns.length) revert InvalidId();
        Campaign storage c = campaigns[id];
        if (!c.finalized) revert NotFinalized();
        if (c.successful) revert AlreadyFinalized();

        uint256 amount = contributions[id][msg.sender];
        if (amount == 0) revert NoContribution();
        contributions[id][msg.sender] = 0;

        (bool ok, ) = msg.sender.call{value: amount}("");
        if (!ok) revert TransferFailed();

        emit Refunded(id, msg.sender, amount);
    }
}
```

Analysis

1) Architectural analysis

The project architecture is chosen rationally: the reward token is placed in a separate contract, and the crowdfunding logic is isolated in Crowdfunding. This reduces coupling and simplifies auditing. The deployment script centralizes dependency linking (transferring the token address to the crowdfunding constructor and assigning minter), which reduces the likelihood of a manual configuration error.

2) Business logic analysis

The contract covers the full cycle of the campaign: creation -> fundraising -> finalizing -> distribution of outcomes (payment to the author upon success / refund to participants upon failure). The selected contributions[id][user] storage model allows you to make accurate per-user accounting and correctly calculate the reward. The rewardClaimed mechanism prevents the reward from being received again.

3) Security analysis

Strengths:

Minting restriction via a separate minter.
Prohibition of repeated claim reward.
Refund is based on the pull model, which is safer than mass automatic payments.
In refund, the deposit balance is reset to zero, then an external call is made (correct order versus reentrancy).

Areas of attention:

Finalize combines two responsibilities: campaign finalizing and claim rewards. This works functionally, but complicates readability and auditing.

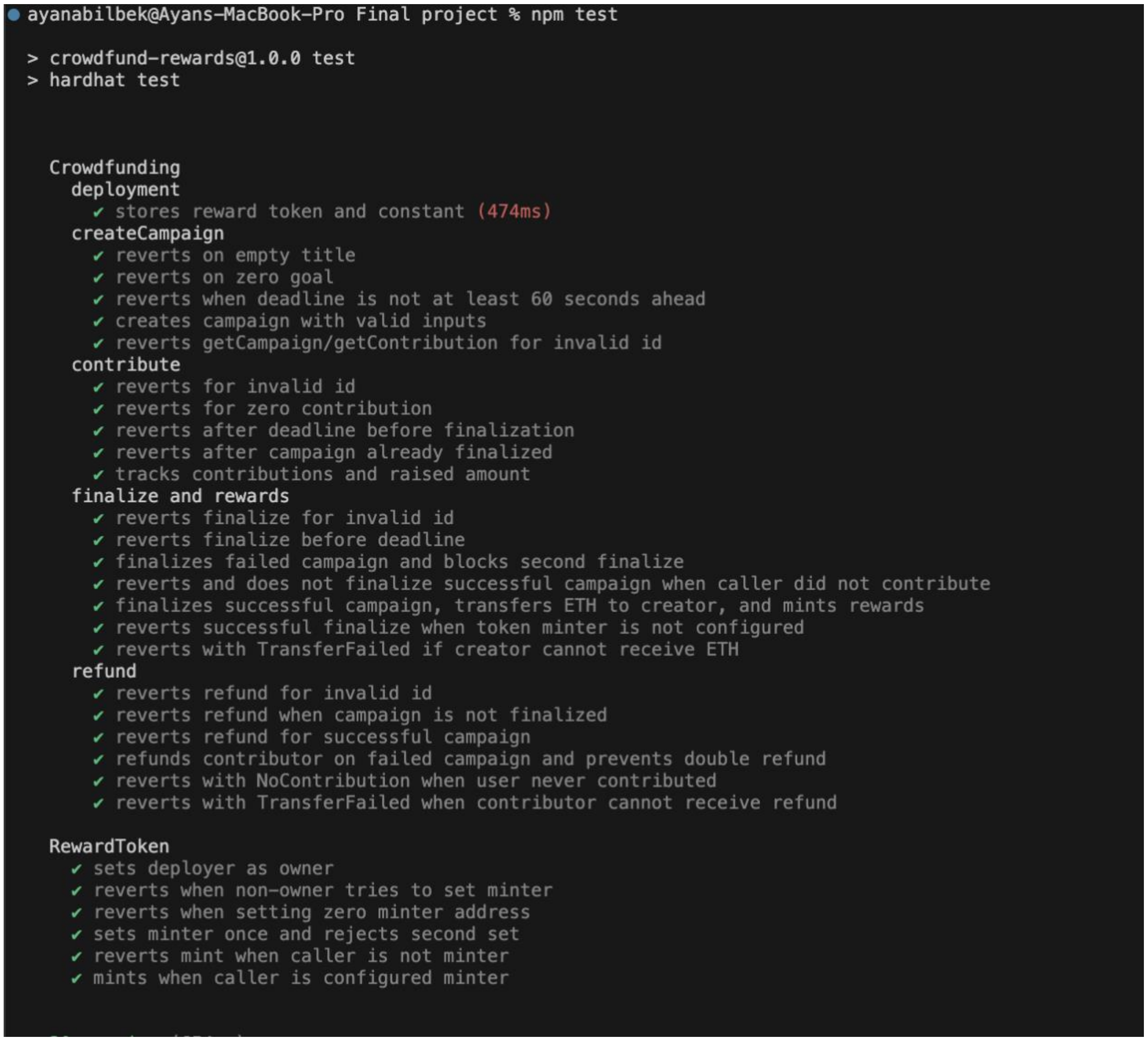
UNIT tests

Which we didn't include in Github
It was not in requirements

Cases covered:

1. **RewardToken:** OwnableUnauthorizedAccount, ZeroAddress, MinterAlreadySet, NotMinter, successful mint.
2. **Crowdfunding.createCampaign:** empty title, goal=0, deadline <=(now+60), correct creation.
3. **Crowdfunding.contribute:** InvalidId, NoContribution, after deadline, after finalize, accumulation of deposits.
4. **Crowdfunding.finalize:** Invalid, notable, failed/successful scenarios, reward branding, re-branding, case without minter (NotMinter), TransferFailed upon payment to the creator.

Crowdfunding.refund: Invalid, NotFinalized, refund in a successful campaign, double refund, refund without deposit, TransferFailed when returning to the contract without receive.



```
ayanabilbek@Ayans-MacBook-Pro Final project % npm test

  > crowdfund-rewards@1.0.0 test
  > hardhat test



    Crowdfunding
      deployment
        ✔ stores reward token and constant (474ms)
      createCampaign
        ✔ reverts on empty title
        ✔ reverts on zero goal
        ✔ reverts when deadline is not at least 60 seconds ahead
        ✔ creates campaign with valid inputs
        ✔ reverts getCampaign/getContribution for invalid id
      contribute
        ✔ reverts for invalid id
        ✔ reverts for zero contribution
        ✔ reverts after deadline before finalization
        ✔ reverts after campaign already finalized
        ✔ tracks contributions and raised amount
      finalize and rewards
        ✔ reverts finalize for invalid id
        ✔ reverts finalize before deadline
        ✔ finalizes failed campaign and blocks second finalize
        ✔ reverts and does not finalize successful campaign when caller did not contribute
        ✔ finalizes successful campaign, transfers ETH to creator, and mints rewards
        ✔ reverts successful finalize when token minter is not configured
        ✔ reverts with TransferFailed if creator cannot receive ETH
      refund
        ✔ reverts refund for invalid id
        ✔ reverts refund when campaign is not finalized
        ✔ reverts refund for successful campaign
        ✔ refunds contributor on failed campaign and prevents double refund
        ✔ reverts with NoContribution when user never contributed
        ✔ reverts with TransferFailed when contributor cannot receive refund

    RewardToken
      ✔ sets deployer as owner
      ✔ reverts when non-owner tries to set minter
      ✔ reverts when setting zero minter address
      ✔ sets minter once and rejects second set
      ✔ reverts mint when caller is not minter
      ✔ mints when caller is configured minter
```

Frontend UX and integration analysis

Frontend correctly implements wallet-connect, chainId validation, transaction status display, and on-chain data update. Plus, there is deadline validation and button locking in an invalid state.

Conclusion

As a result, a working crowdfunding DApp system with reward mechanics has been implemented, where the on-chain part is responsible for the consistency of financial logic, and the frontend provides an accessible user interaction scenario through MetaMask. Architecturally, the solution scales and is understandable to maintain due to the separation of the token contract, the business contract and the UI layer.

The analysis showed that the key functions were performed correctly: creating campaigns, accounting for deposits, finalizing, accruing tokens and refunds in case of failure.