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

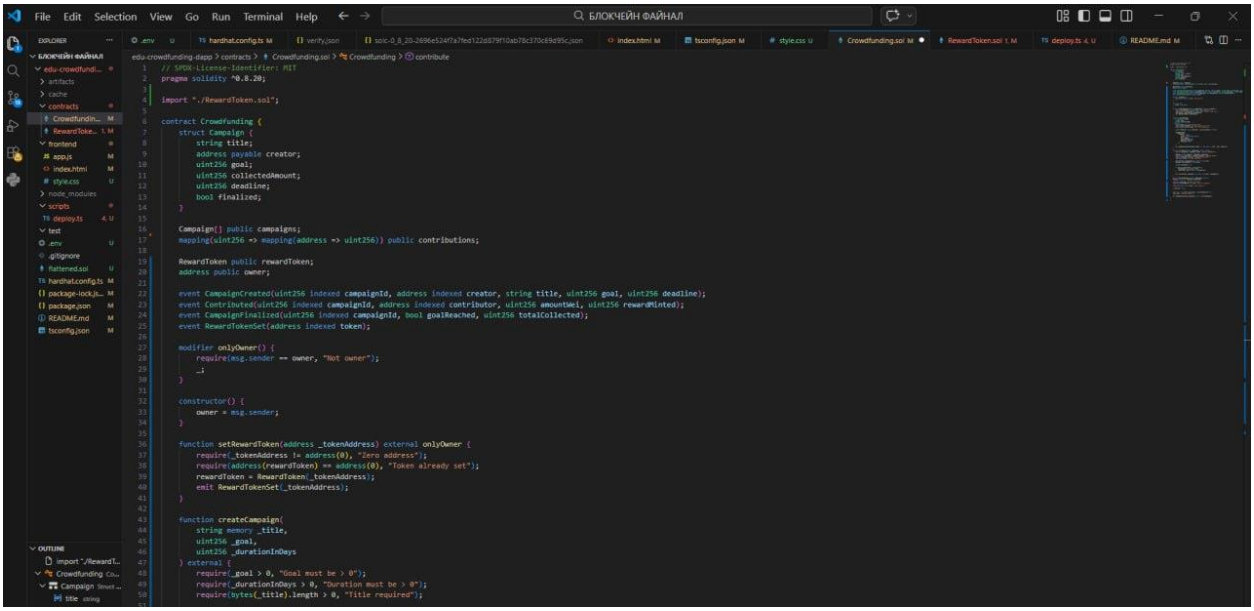
All components are fully connected and working together.

Smart Contracts Description

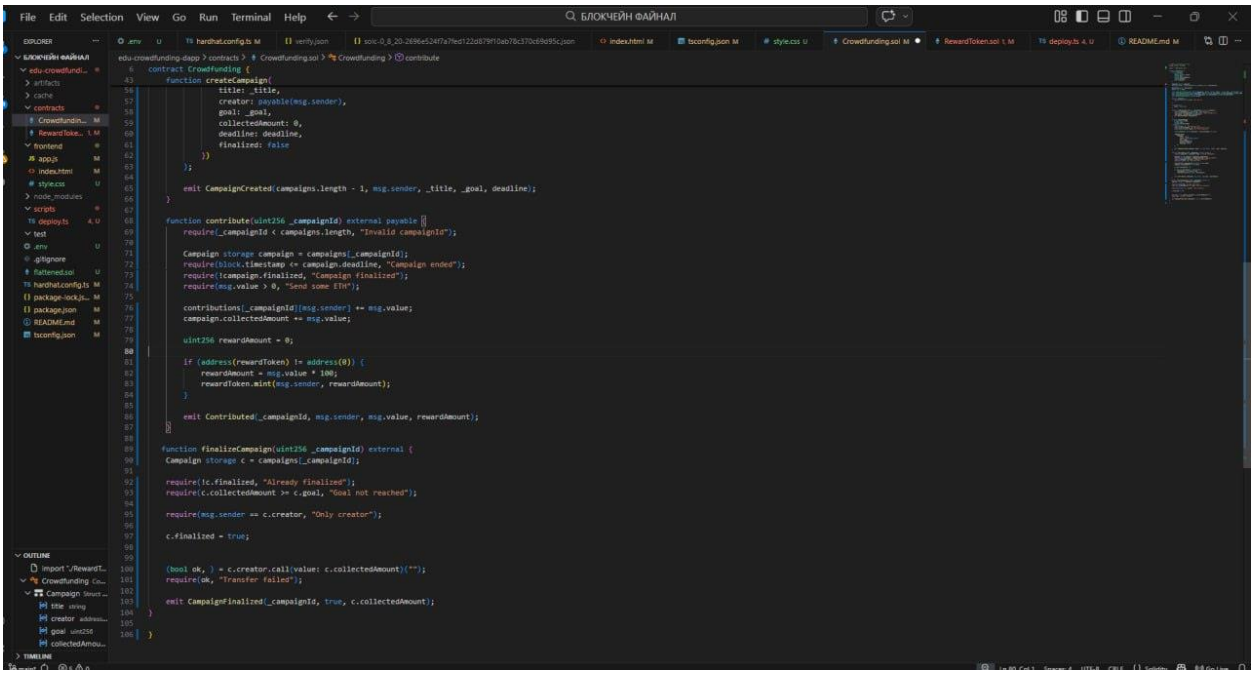
The project includes two smart contracts:

Crowdfunding.sol
RewardToken.sol

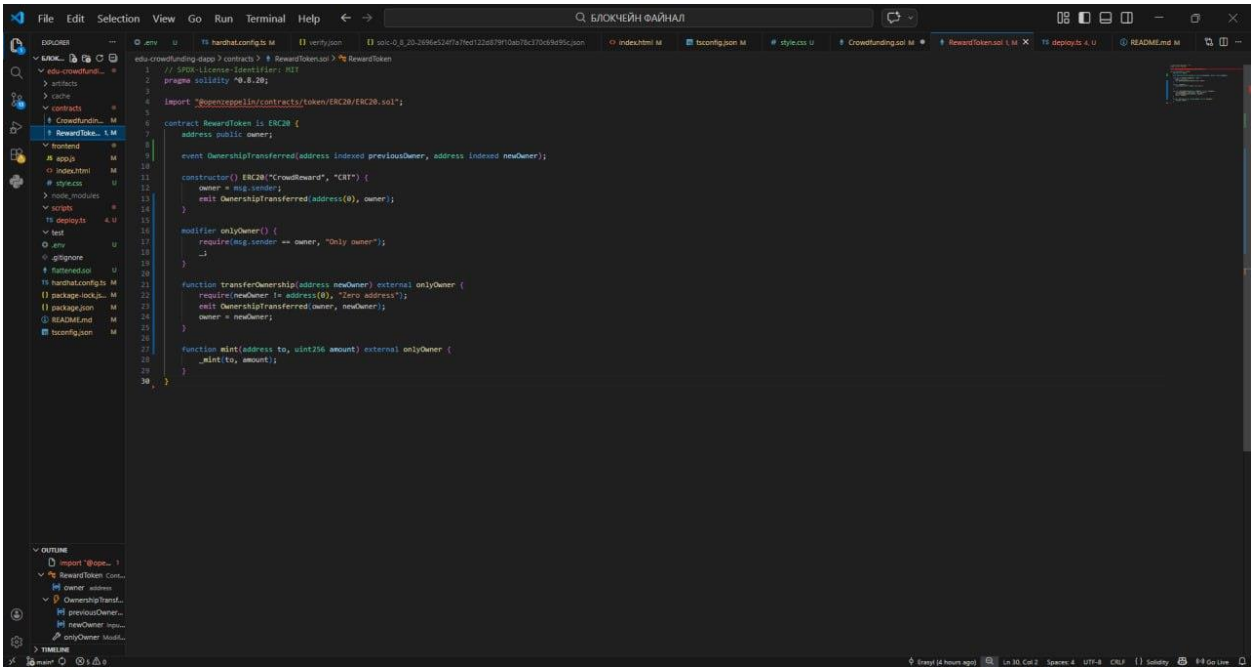
The Crowdfunding contract manages campaigns, contributions, and campaign finalization.
The RewardToken contract is an ERC-20 token used to reward contributors.



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

import "./RewardToken.sol";

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

    Campaign[] public campaigns;
    mapping(uint256 => mapping(address => uint256)) public contributions;

    RewardToken public rewardToken;
    address public owner;

    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 rewardMinted);
    event CampaignFinalized(uint256 indexed campaignId, bool goalReached, uint256 totalCollected);
    event RewardTokenSet(address indexed token);

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

    constructor() {
        owner = msg.sender;
    }

    function setRewardToken(address _tokenAddress) external onlyOwner {
        require(_tokenAddress != address(0), "Zero address");
        require(address(rewardToken) == address(0), "Token already set");
        rewardToken = RewardToken(_tokenAddress);
        emit RewardTokenSet(_tokenAddress);
    }

    function createCampaign(
        string memory _title,
        uint256 _goal,
        uint256 _durationInDays
    ) external {
        require(_goal > 0, "Goal must be > 0");
        require(_durationInDays > 0, "Duration must be > 0");
        require(bytes(_title).length > 0, "Title required");
```



```
contract Crowdfunding {
    function createCampaign(
                title: _title,
                creator: payable(msg.sender),
                goal: _goal,
                collectedAmount: 0,
                deadline: deadline,
                finalized: false
            })
        );

        emit CampaignCreated(campaigns.length - 1, msg.sender, _title, _goal, deadline);
    }

    function contribute(uint256 _campaignId) external payable {
        require(_campaignId < campaigns.length, "Invalid campaignId");

        Campaign storage campaign = campaigns[_campaignId];
        require(block.timestamp <= campaign.deadline, "Campaign ended");
        require(!campaign.finalized, "Campaign finalized");
        require(msg.value > 0, "Send some ETH");

        contributions[_campaignId][msg.sender] += msg.value;
        campaign.collectedAmount += msg.value;

        uint256 rewardAmount = 0;

        if (address(rewardToken) != address(0)) {
            rewardAmount = msg.value * 100;
            rewardToken.mint(msg.sender, rewardAmount);
        }

        emit Contributed(_campaignId, msg.sender, msg.value, rewardAmount);

    function finalizeCampaign(uint256 _campaignId) external {
        Campaign storage c = campaigns[_campaignId];

        require(!c.finalized, "Already finalized");
        require(c.collectedAmount >= c.goal, "Goal not reached");

        require(msg.sender == c.creator, "Only creator");

        c.finalized = true;


        (bool ok, ) = c.creator.call{value: c.collectedAmount}("");
        require(ok, "Transfer failed");

        emit CampaignFinalized(_campaignId, true, c.collectedAmount);
    }
}
```

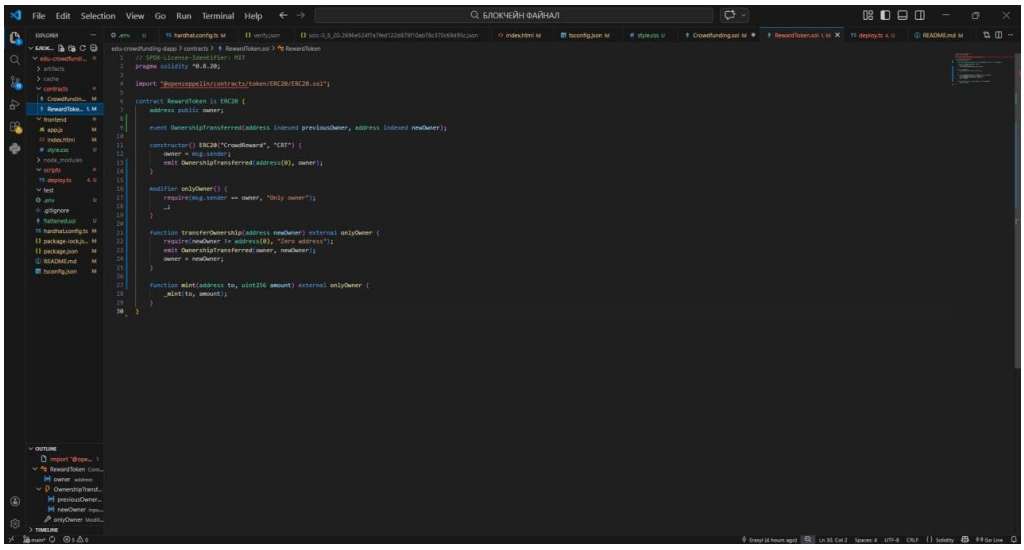


RewardToken Contract

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.



Crowdfunding Contract

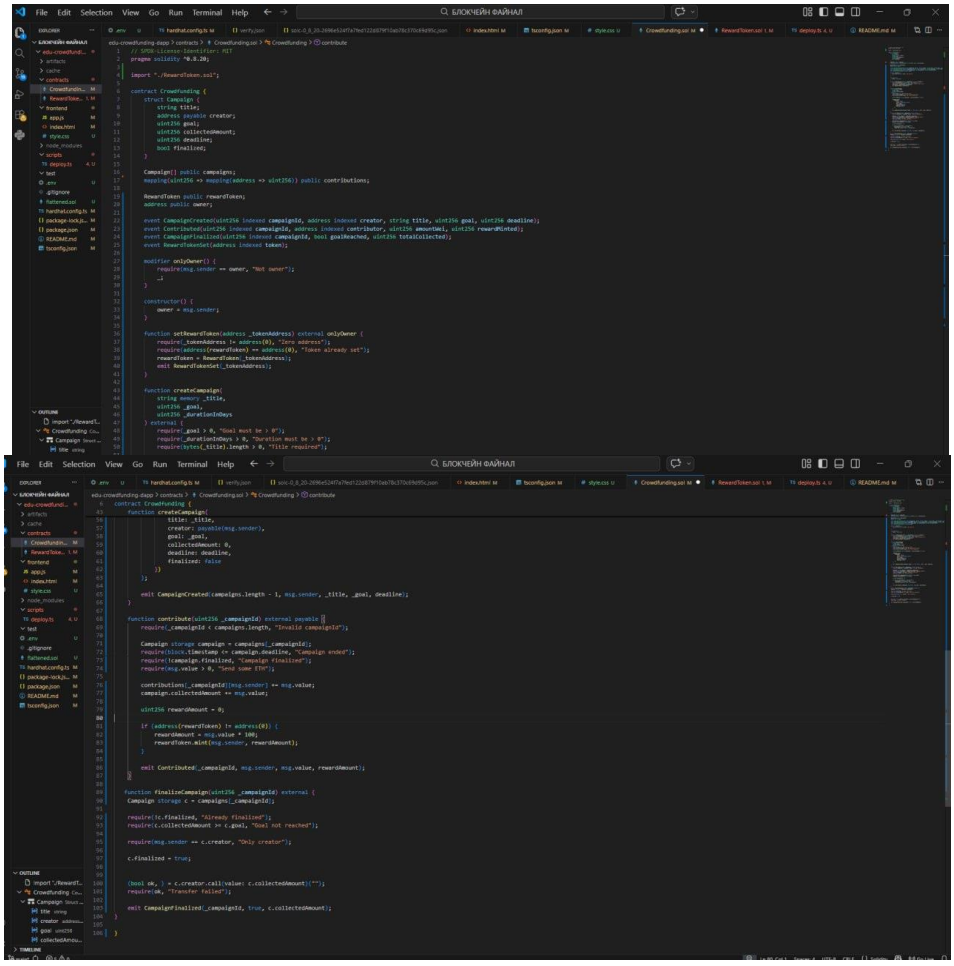
The Crowdfunding contract allows users to:

- 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

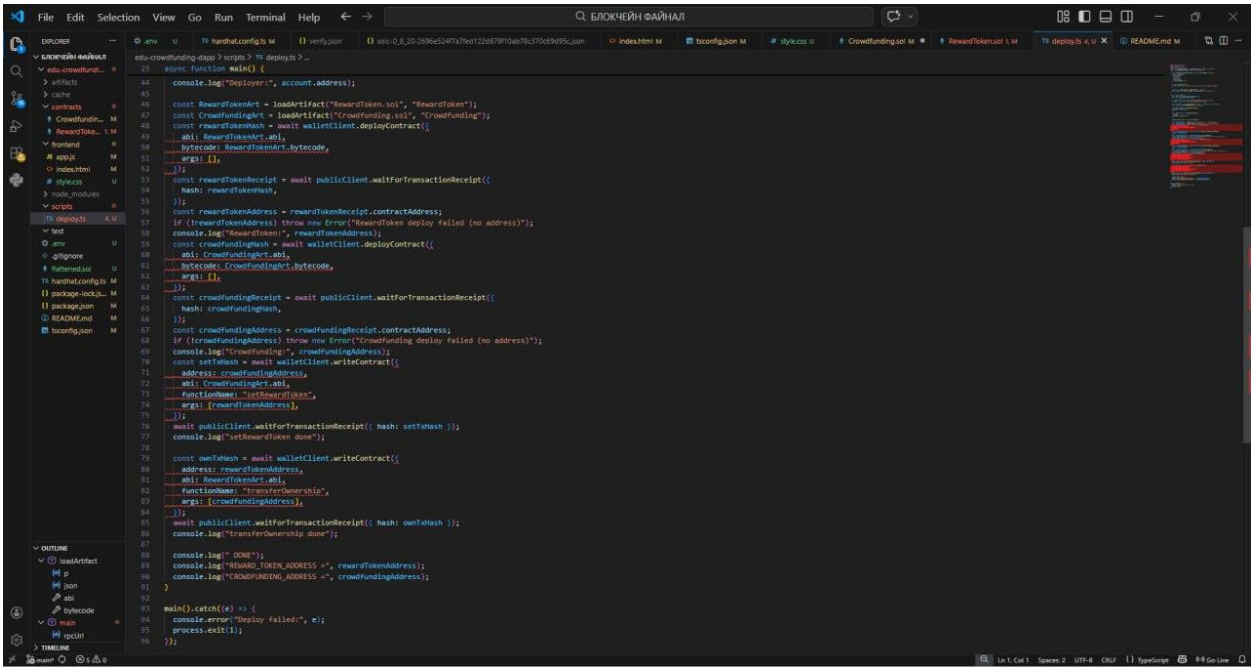
Campaigns are stored on-chain and can be accessed publicly, ensuring transparency.

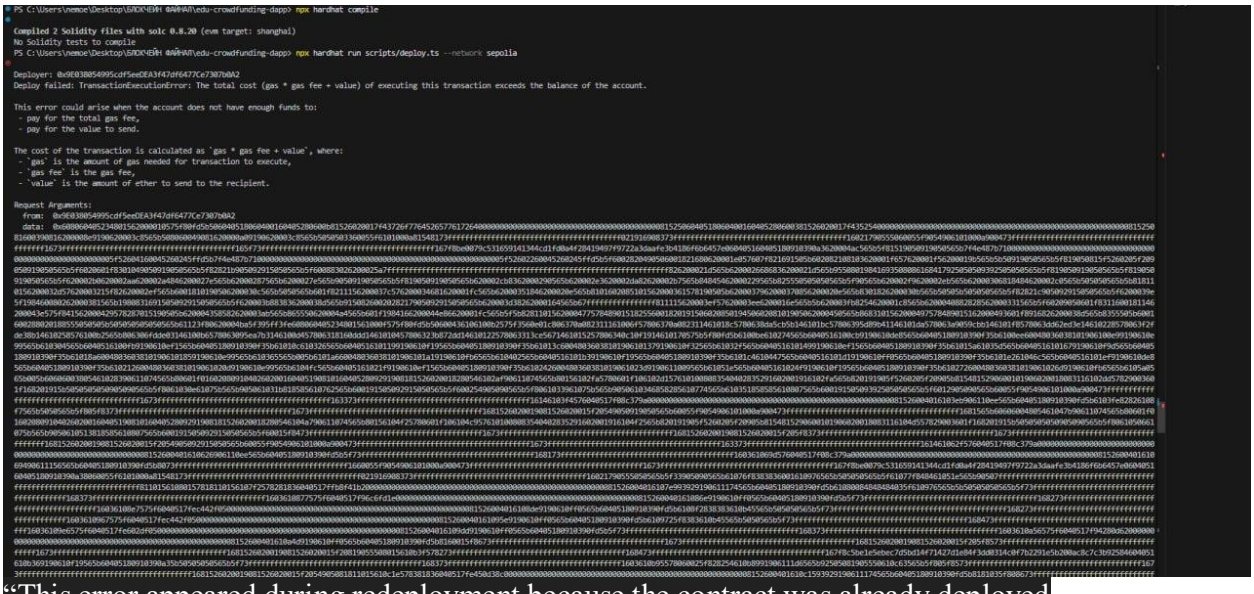


Deployment to Sepolia Network

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.



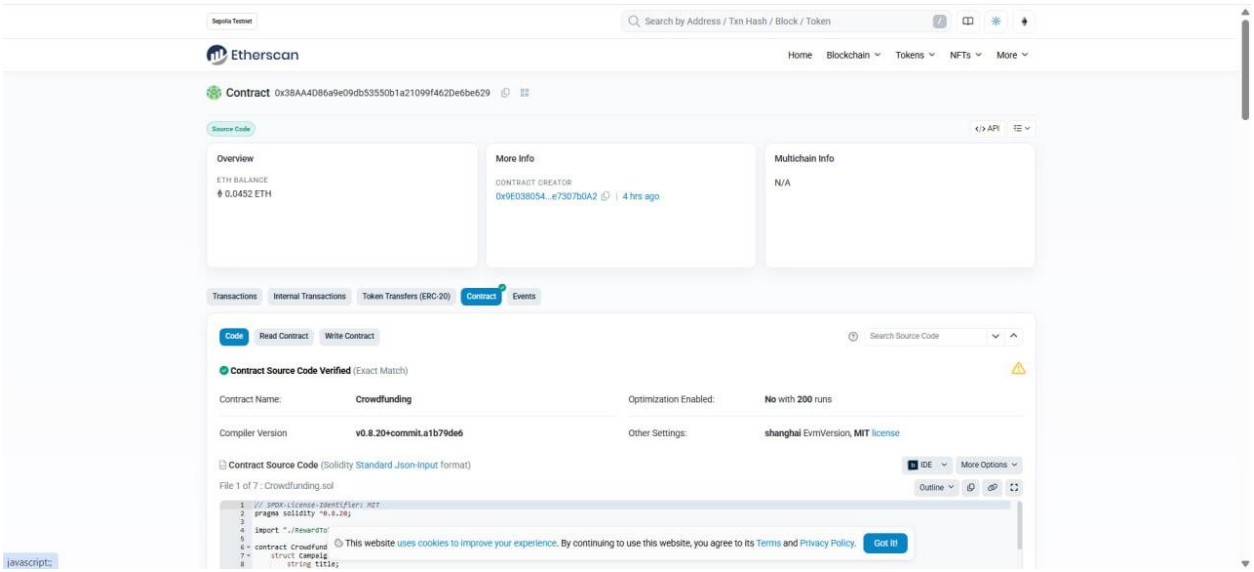


“This error appeared during redeployment because the contract was already deployed and the gas estimation reverted.
It is not a logical error in the smart contract,
but a deployment/runtime condition related to testnet execution.”

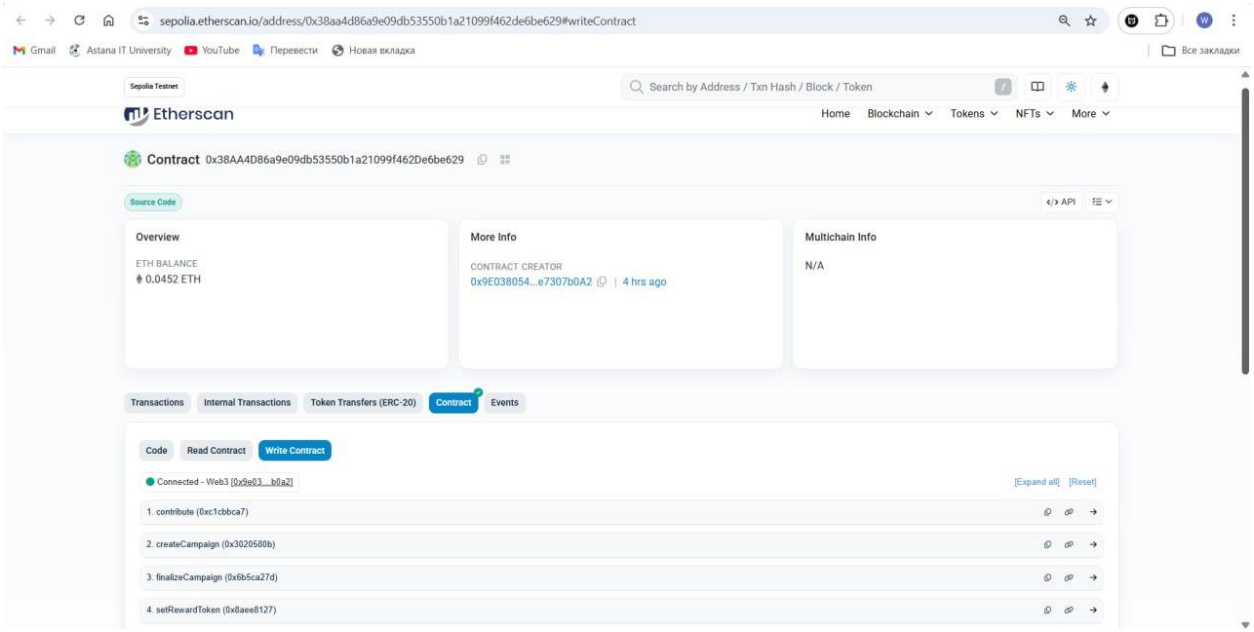
Contract Verification on Etherscan

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.

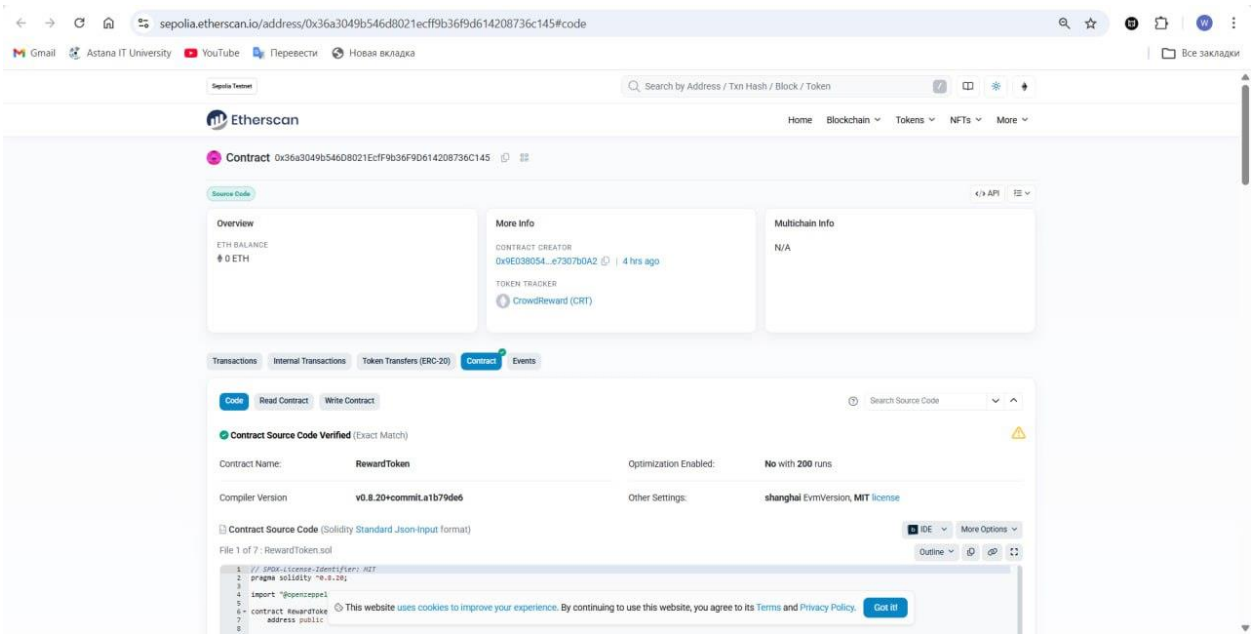
Successful verification confirms that the deployed bytecode matches the source code.



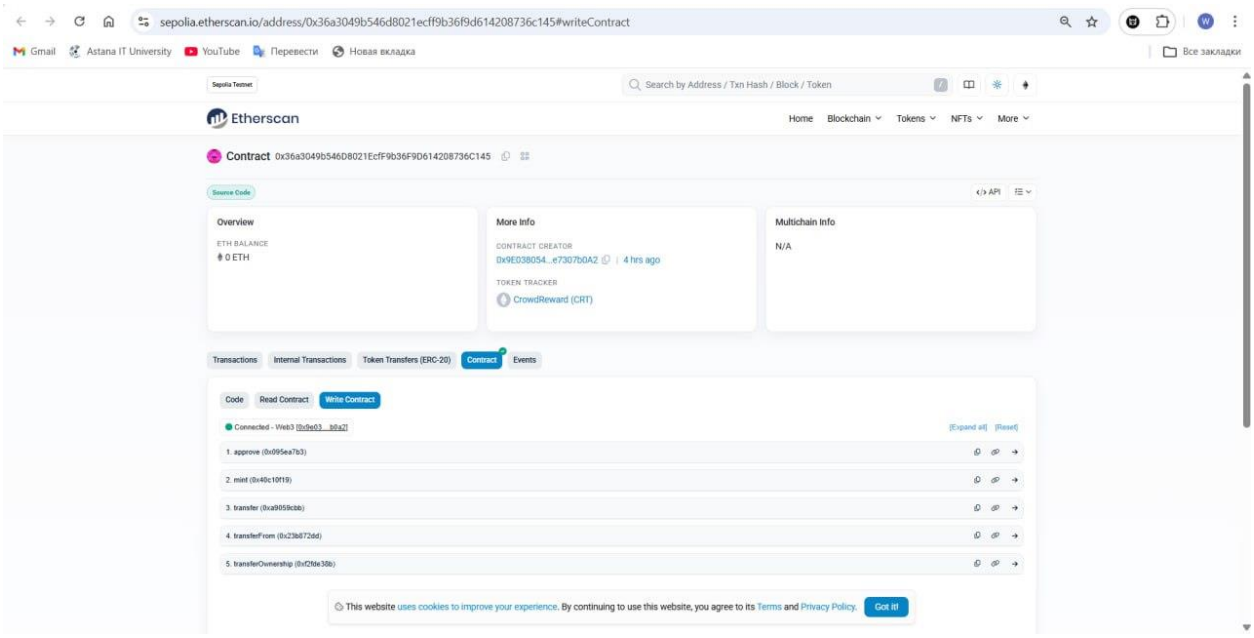
Crowdfunding



ALL FUNCTIONS OF CONTRACT CROWDFINDING IN ETHERSCAN AND CONNECT WITH METAMASK



RewardToken



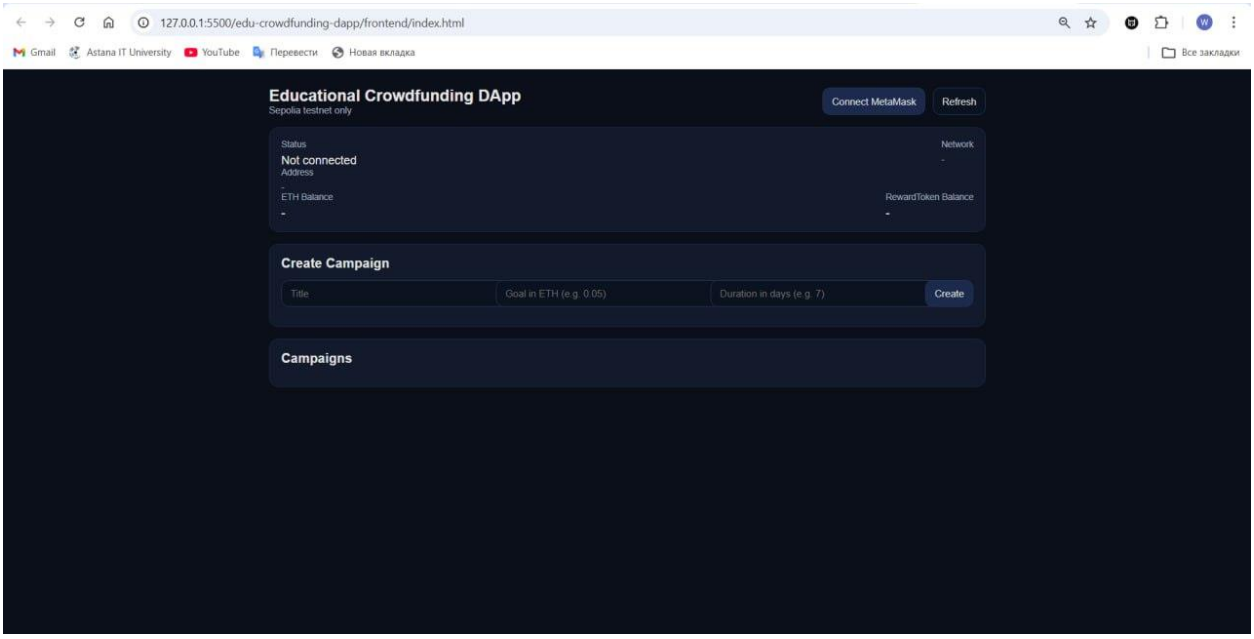
ALL FUNCTIONS OF CONTRACT RTB IN ETHERSCAN AND CONNECT WITH METAMASK

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:

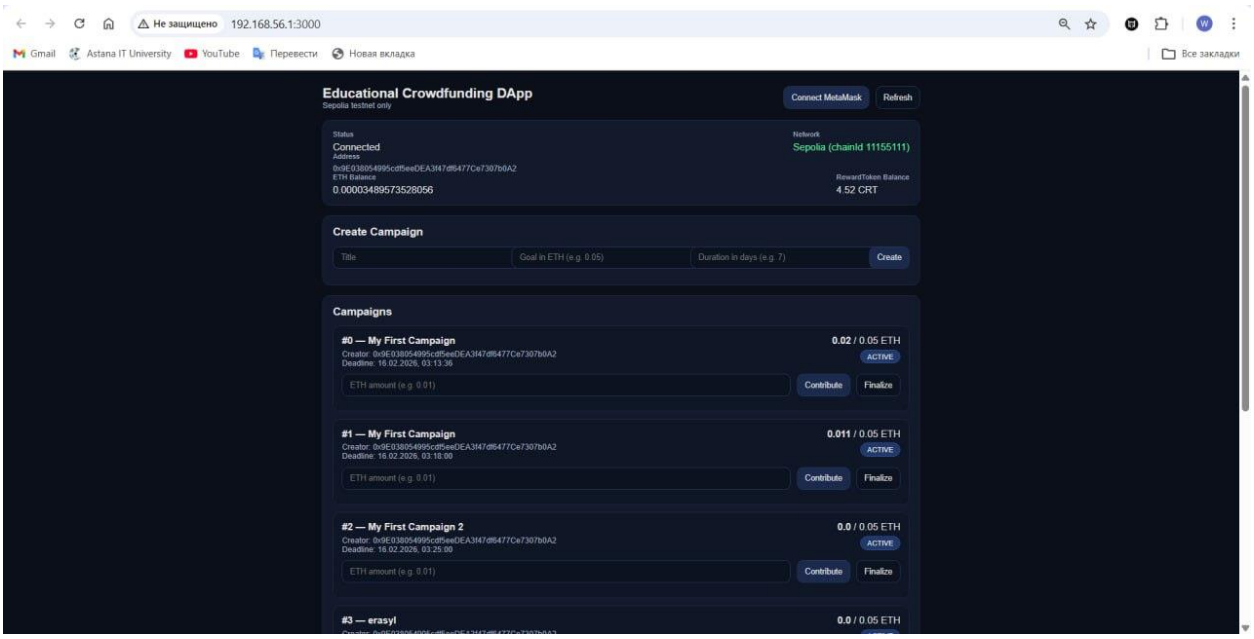
- MetaMask wallet connection
- Network validation (Sepolia only)
- Campaign creation
- ETH contributions
- Campaign finalization
- Display of reward token balance



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.

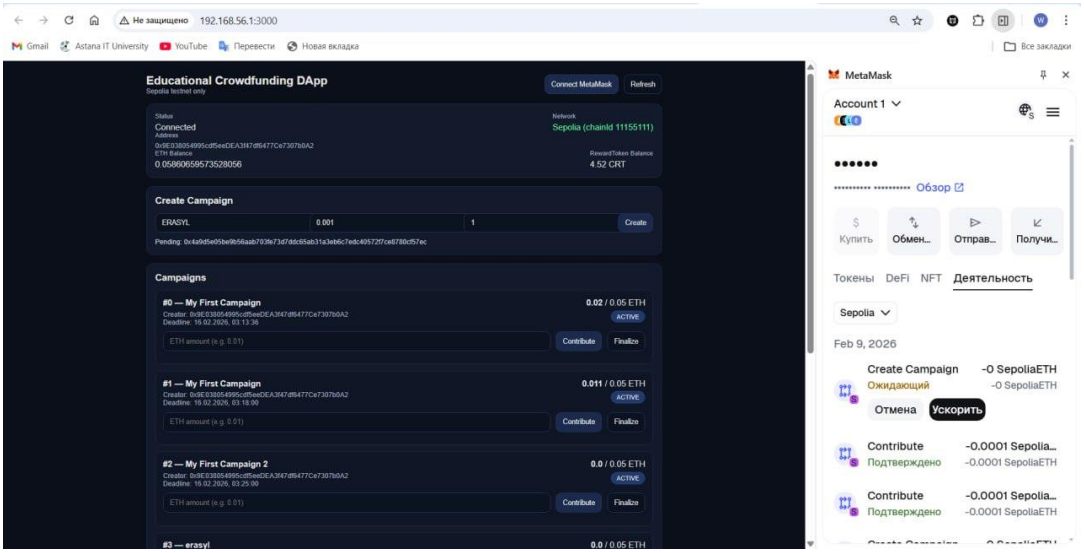


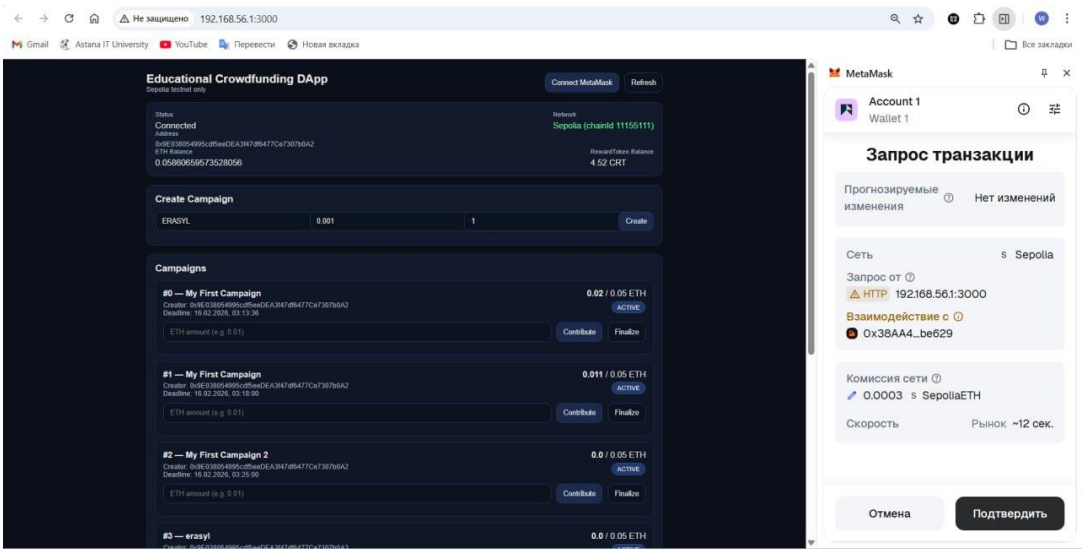
Creating a Campaign

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.



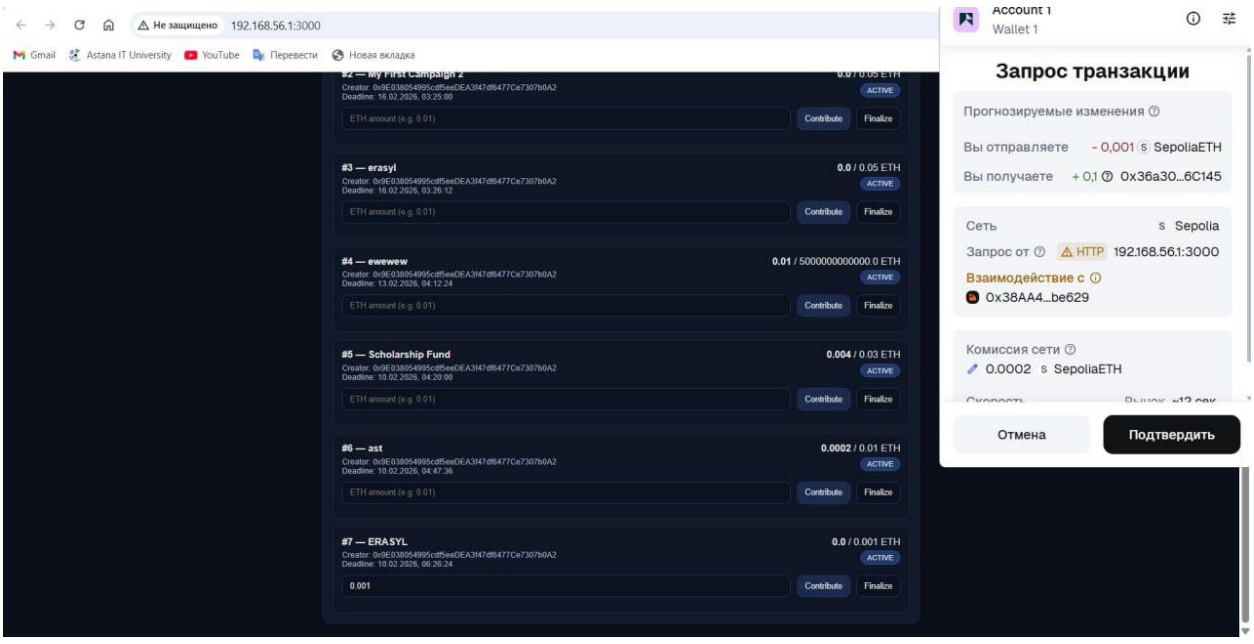


Contributing to a Campaign

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.



Finalizing a Campaign

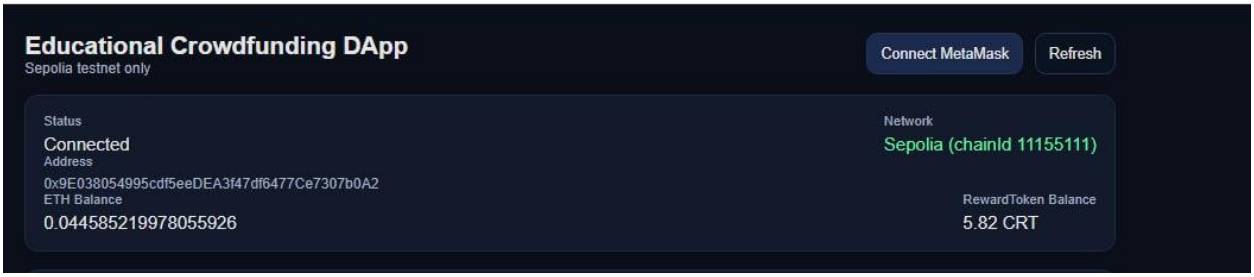
A campaign can be finalized when the funding goal is reached. Finalization marks the campaign as completed and prevents further contributions.

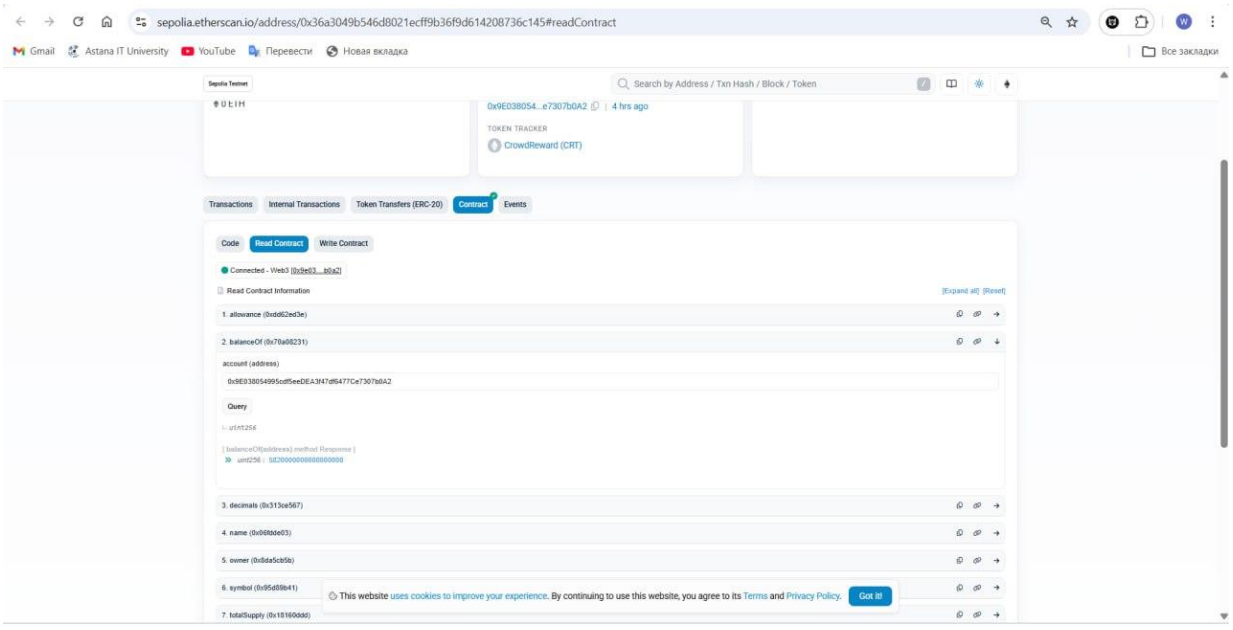
The finalized status is stored on-chain and visible to all users. This ensures that campaign results are transparent and cannot be modified.

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.





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.