EDUFUND: DECENTRALIZED CROWDFUNDING DAPP

Blockchain 1 - Final Project

PRESENTED BY:

Kendebayev Nurassyl
Shalmanov Assylzhan
Demesh Makhmet

Agenda

1	Build a decentralized crowdfunding application
2	Practice smart contract development with Solidity
3	Demonstrate real blockchain interaction
4	Use ERC-20 tokens for reward logic
5	Integrate MetaMask with frontend

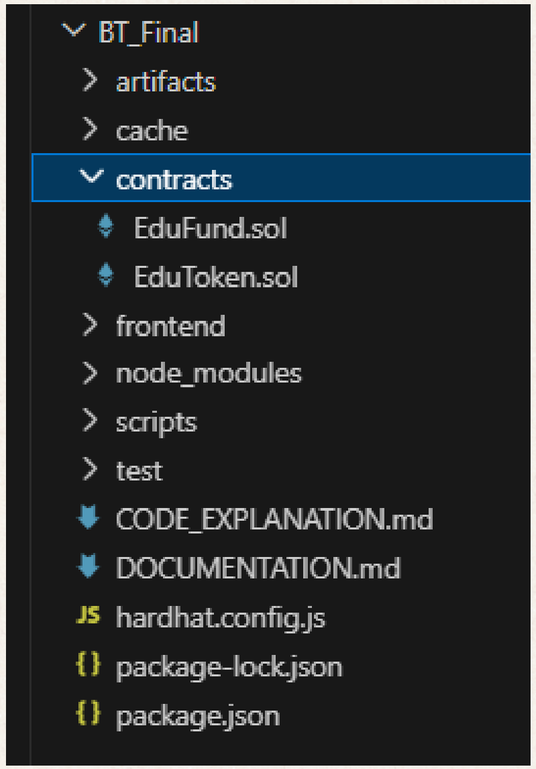
Smart Contracts Overview

On-chain Logic

EduFund.sol - crowdfunding logic

EduToken.sol – ERC-20 reward token

Contracts interact with each other
No real money, test ETH only



EduToken.sol

Purpose

ERC-20 token used as a reward
Minted automatically on donation
Has no real monetary value

Key Security Feature
Only EduFund contract can mint tokens
Uses onlyOwner modifier

EduToken.sol

BT_Final > contracts > EduToken.sol

```
1   // SPDX-License-Identifier: MIT
2   pragma solidity ^0.8.20;
3
4   import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
5   import "@openzeppelin/contracts/access/Ownable.sol";
6
7   /**
8    * @title EduToken
9    * @dev ERC-20 Reward Token for the EduFund platform.
10   * Minted only by the EduFund contract when users donate.
11   */
12  contract EduToken is ERC20, Ownable {
13
14      constructor(address initialOwner)
15          ERC20("EduReward", "EDU")
16          Ownable(initialOwner)
17      {
18          // No initial supply - tokens are minted on demand
19      }
20
21      function mint(address to, uint256 amount) external onlyOwner {
22          _mint(to, amount);
23      }
24  }
25
```

EduFund.sol

Core Responsibilities

- Create campaigns
- Accept ETH donations
- Track collected funds
- Finalize campaigns
- Mint reward tokens

```
contract EduFund {

    EduToken public eduToken;
    uint256 public campaignCount;
    uint256 public constant REWARD_RATE = 100;

    struct Campaign {
        address owner;
        string title;
        string description;
        uint256 goal;
        uint256 deadline;
        uint256 amountCollected;
        bool finalized;
    }
```

```
    function createCampaign(
        string calldata _title,
        string calldata _description,
        uint256 _goal,
        uint256 _duration
    ) external returns (uint256) {
        if (_goal == 0) revert InvalidGoal();
        if (_duration == 0) revert InvalidDuration();

        uint256 campaignId = campaignCount;
        campaignCount++;
```

Donation Flow

Donate Function

1st

Receives ETH (payable)

2nd

Updates campaign balance

3rd

Calculates reward

4th

Calls EduToken to mint tokens

```
function donate(uint256 _id) external payable {
    if (_id >= campaignCount) revert CampaignDoesNotExist();
    if (msg.value == 0) revert ZeroDonation();

    Campaign storage campaign = campaigns[_id];
    if (campaign.finalized) revert CampaignAlreadyFinalized();

    // Update campaign amount
    campaign.amountCollected += msg.value;

    // Track individual donation
    donations[_id][msg.sender] += msg.value;

    // 1 ETH = 100 EDU
    uint256 tokenReward = msg.value * REWARD_RATE;
    eduToken.mint(msg.sender, tokenReward);

    emit DonationReceived(_id, msg.sender, msg.value, tokenReward);

    emit FundSent(msg.sender, _id, msg.value, block.timestamp);
}
```

Withdraw and Security

Withdraw Logic

- Only campaign owner
- Goal must be reached
- Deadline must be passed
- Reentrancy protection (finalized = true first)

```
function withdraw(uint256 _id) external {
    if (_id >= campaignCount) revert CampaignDoesNotExist();

    Campaign storage campaign = campaigns[_id];

    if (msg.sender != campaign.owner) revert NotCampaignOwner();
    if (block.timestamp < campaign.deadline) revert DeadlineNotReached();
    if (campaign.amountCollected < campaign.goal) revert GoalNotReached();
    if (campaign.finalized) revert CampaignAlreadyFinalized();

    // Mark as finalized before transfer (reentrancy protection)
    campaign.finalized = true;

    uint256 amount = campaign.amountCollected;

    // Transfer funds to campaign owner
    (bool success, ) = payable(campaign.owner).call{value: amount}("");
    if (!success) revert TransferFailed();

    emit FundsWithdrawn(_id, campaign.owner, amount);
}
```

```
function refund(uint256 _id) external {
    Campaign storage campaign = campaigns[_id];

    if (block.timestamp < campaign.deadline) revert CampaignStillActive();
    if (campaign.amountCollected >= campaign.goal) revert GoalNotReached(); //

    uint256 donatedAmount = donations[_id][msg.sender];
    if (donatedAmount == 0) revert ZeroDonation();

    // Reset donation to prevent re-entrancy
    donations[_id][msg.sender] = 0;

    (bool success, ) = payable(msg.sender).call{value: donatedAmount}("");
    if (!success) revert TransferFailed();
}
```

Deployment Script

scripts/deploy.js

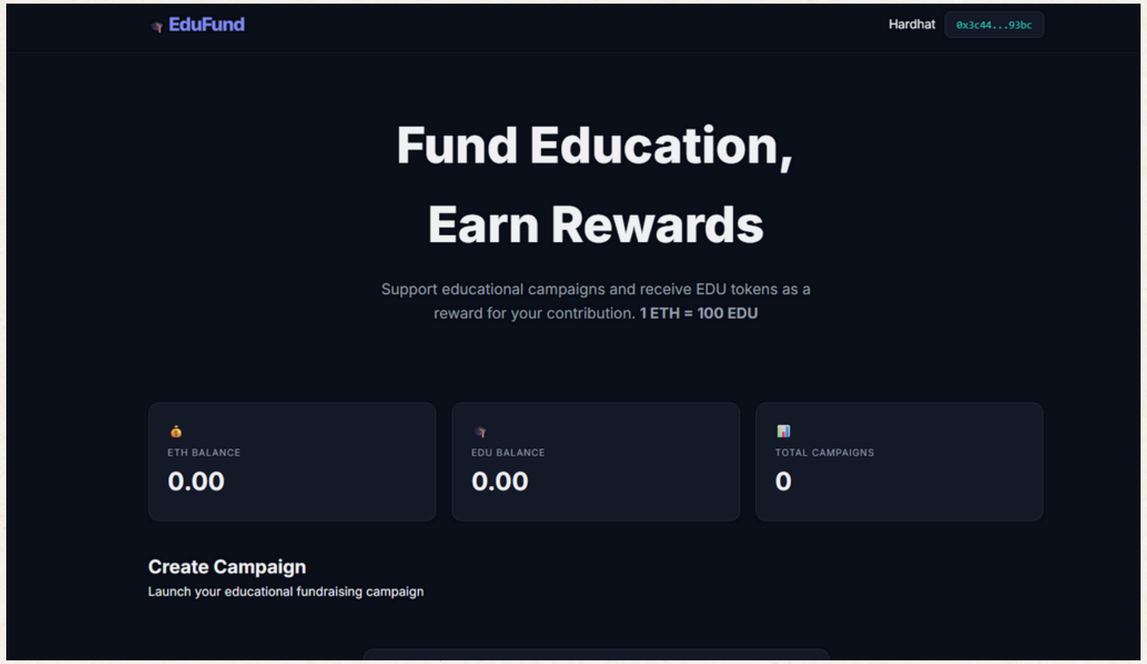
- Deploys EduToken
- Deploys EduFund
- Transfers token ownership
- Auto-writes addresses to frontend config

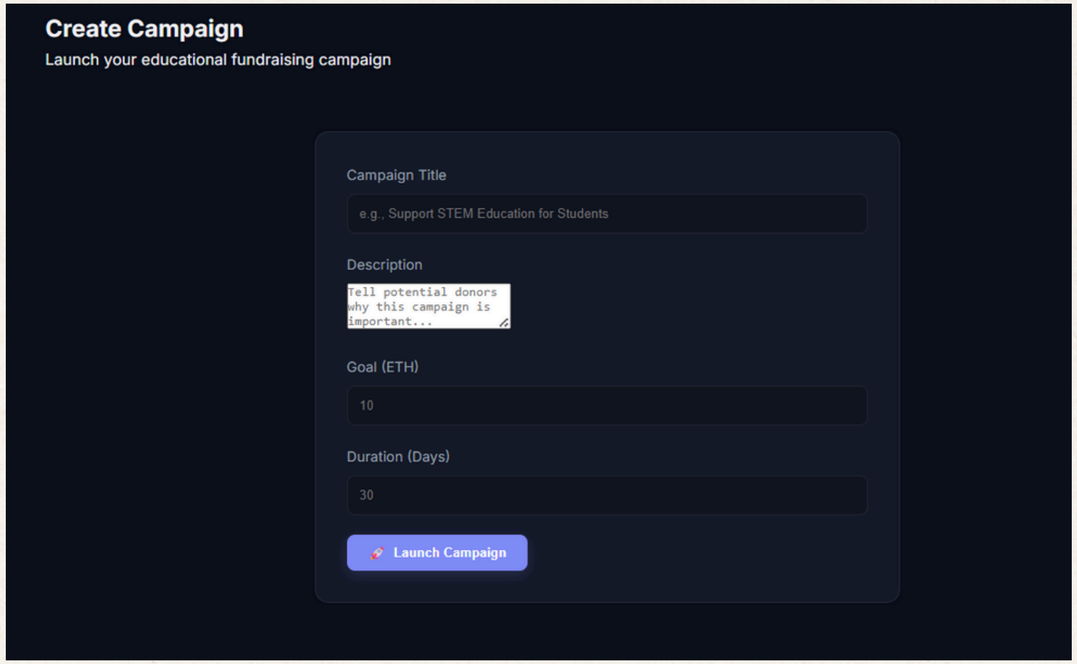
```
// Transfer Ownership
await eduToken.transferOwnership(await eduFund.getAddress());
console.log("Ownership transferred");

if (newOwner.toLowerCase() === eduFundAddress.toLowerCase()) {
    console.log("   ✓ Verified: EduFund can now mint EDU tokens");
} else {
    console.error("   X ERROR: Ownership transfer failed!");
    process.exit(1);
}
```

Frontend Interface

USER INTERFACE





MetaMask and Frontend Logic

HOW FRONTEND WORKS

- REQUESTS METAMASK CONNECTION
- CREATES PROVIDER AND SIGNER
- SENDS TRANSACTIONS
- WAITS FOR CONFIRMATION

```
// Wallet Connection
async function connectWallet() {
  if (!window.ethereum) {
    showToast("Please install MetaMask!", "error");
    return;
  }

  try {
    showLoading("Connecting to MetaMask...");
```

```
    // Initialize contracts
    await initializeContracts();

    // Update UI
    state.isConnected = true;
    updateWalletUI();
    await refreshData();

    hideLoading();
    showToast("Wallet connected successfully!", "success");

  } catch (error) {
    hideLoading();
    console.error("Connection error:", error);
    showToast("Failed to connect wallet", "error");
  }
```

End-to-End Data Flow

1	User clicks “Donate”
2	MetaMask asks for confirmation
3	ETH sent to EduFund
4	EduFund mints EDU tokens
5	UI updates balances