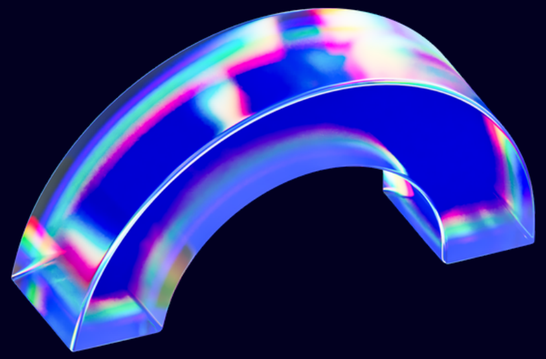
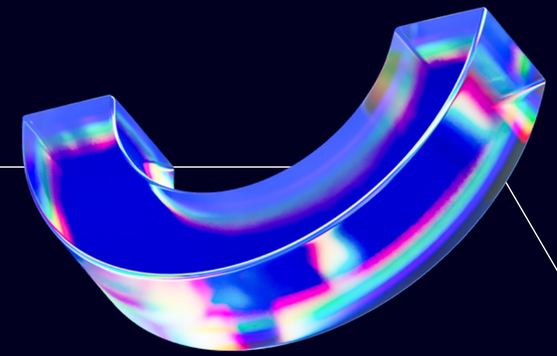


Decentralized Crowdfunding Application

BLOCKCHAIN 1 — FINAL PROJECT

Tamirlan Meiirzhan Zharkynai



Project Goals and Objectives

Main Goal

Comprehensive assessment of competencies in the field of smart contracts and DApp development, demonstrating the practical application of blockchain technologies.

Key Objectives

- Design and implement secure Solidity smart contracts
- Develop client-side with blockchain integration via JavaScript
- Connect MetaMask for secure wallet interaction
- Implement a tokenization system with internal reward tokens

Overview of the Application Architecture

- Decentralized Infrastructure: The application is built on a modular architecture where business logic is strictly separated from asset management.
- Contract Interaction: The Donation.sol contract acts as an "orchestrator," initiating cross-contract calls to the RewardToken.sol contract via the ERC-20 interface.
- On-Chain State Management: All campaign data is stored on the blockchain using mappings and arrays of structs, ensuring data immutability and transparency.
- External Integration: The frontend connects via the Ethers.js v6 provider, bridging the gap between the user's wallet (MetaMask) and the blockchain node.

Technology Stack

Smart Contracts

Solidity v0.8.20 for creating reliable and secure contracts

Frontend

HTML5, CSS3, Vanilla JavaScript for the user interface

Blockchain Library

Ethers.js v6 for interacting with Ethereum

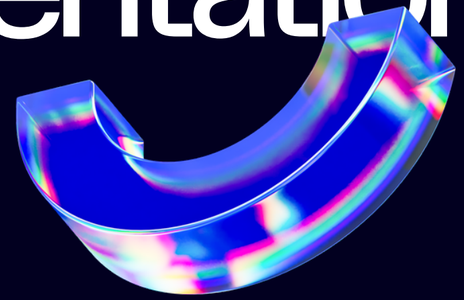
Development

Hardhat Localhost Testnet for development and testing

Wallet

MetaMask Integration for managing transactions

Design and Implementation Decisions



Security by Design

Before any ETH is moved, the require() function acts as a gatekeeper, checking if the campaign is active and the ID is valid.

```
unction donate(uint256 _campaignId) public payable {
    require(_campaignId < campaigns.length, "Campaign does not ex
    require(msg.value > 0, "Send some ETH");
    require(campaigns[_campaignId].active, "Campaign is not activ
```

Access Control

Integration of the MINTER_ROLE via OpenZeppelin restricts token issuance rights solely to the authorized Donation contract, preventing unauthorized minting.

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

Gas Optimization

Using struct Campaign allows for efficient data grouping, which minimizes gas costs during on-chain write operations.

```
function createCampaign(string memory _title, uint256 _goal) public {
    campaigns.push(Campaign({
        creator: msg.sender,
        title: _title,
        goal: _goal,
        raised: 0,
        active: true
    }));
    emit CampaignCreated(campaigns.length - 1, _title, _goal, msg.sender);
}
```

TypeScript Reliability

Description of Smart Contract Logic

```
function createCampaign(string memory _title, uint256 _goal) public {
    campaigns.push(Campaign({
        creator: msg.sender,
        title: _title,
        goal: _goal,
        raised: 0,
        active: true
    }));
                                                    msg.sender);
```

1) Entry Point

START: The user calls the donate(id) function and sends ETH (msg.value) contract.

2) Validation Phase (Checks)

Before any state changes occur, the contract runs security checks:

CHECK 1: Valid ID require(id < campaigns.length) Ensures the campaign exists within the array bounds.

CHECK 2: Campaign Status require(campaign.active == true) Verifies that the fundraising campaign is still open and has not been closed or paused.

3) State Updates and Logic (Actions)

If the checks pass, the contract executes the following logic:

ACTION 1: Update Balance campaign.raised += msg.value The total amount of ETH raised for this specific campaign is updated on the blockchain.

ACTION 2: Mint Reward Tokens rewardToken.mint(msg.sender, amount) The contract automatically triggers the minting process to send reward tokens to the donor's address.

4) Finalization

EVENT: Logging Data emit DonationReceived(...) A transaction event is broadcasted to the blockchain logs, allowing the frontend (website) to update the UI in real-time.

END: The transaction is successfully confirmed and added to the block.

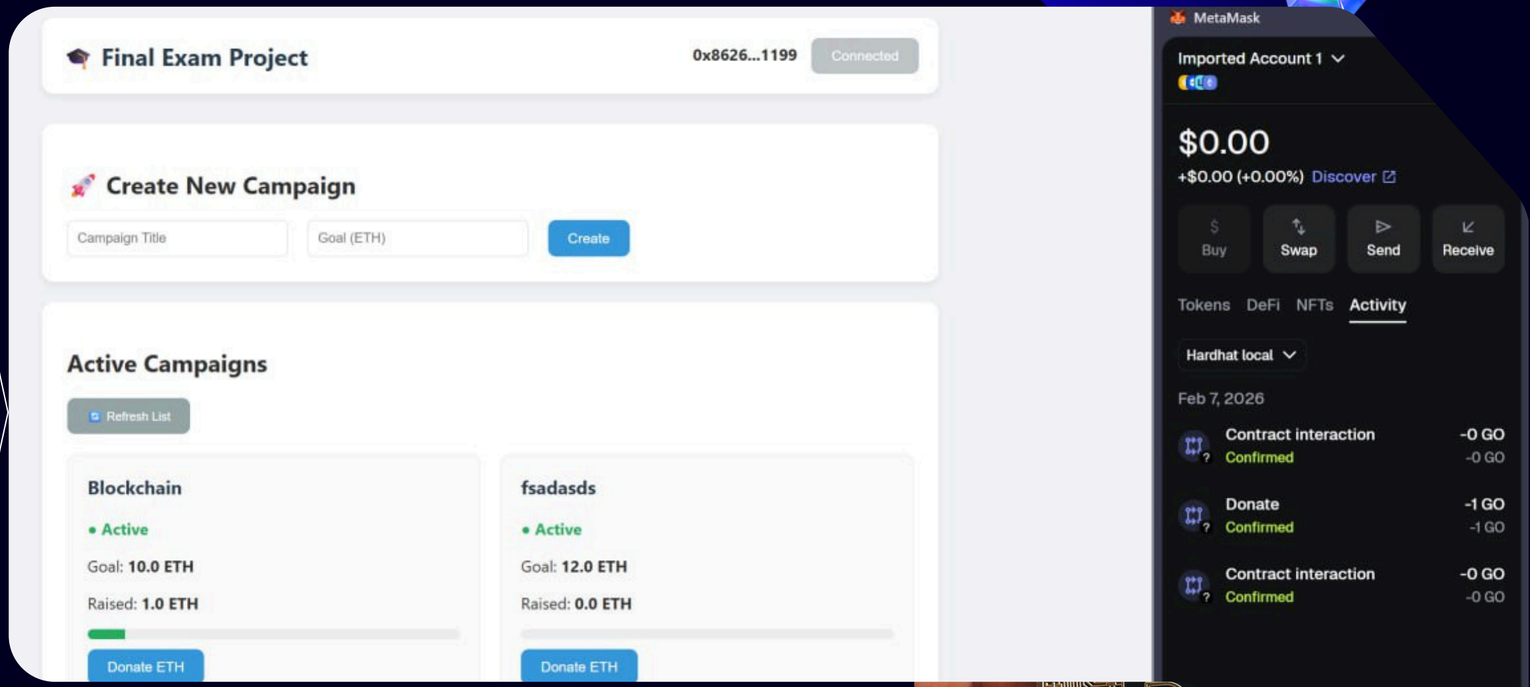
Frontend-to-Blockchain Interaction

Wallet Connectivity: Integration with the MetaMask extension via the window.ethereum object for user authorization and transaction signing.

Data Synchronization: Use of call methods for gasless reading of campaign progress and send transactions for altering the blockchain state.

Network Validation: Implementation of network ID checks to ensure the user is connected to the correct environment (Hardhat Localhost or Sepolia).

Transaction Lifecycle: The interface awaits transaction mining confirmation before updating the UI, ensuring data consistency for the user.



Deployment and Execution Instructions

1. Environment Prep

npm install

Hardhat 3.1.6

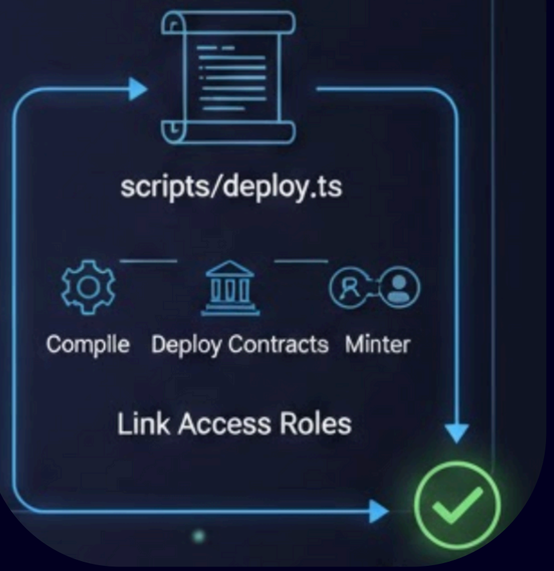
OpenZepelen

Sandbox Node

npx hardhat node

Local EVM with 10k ETH each

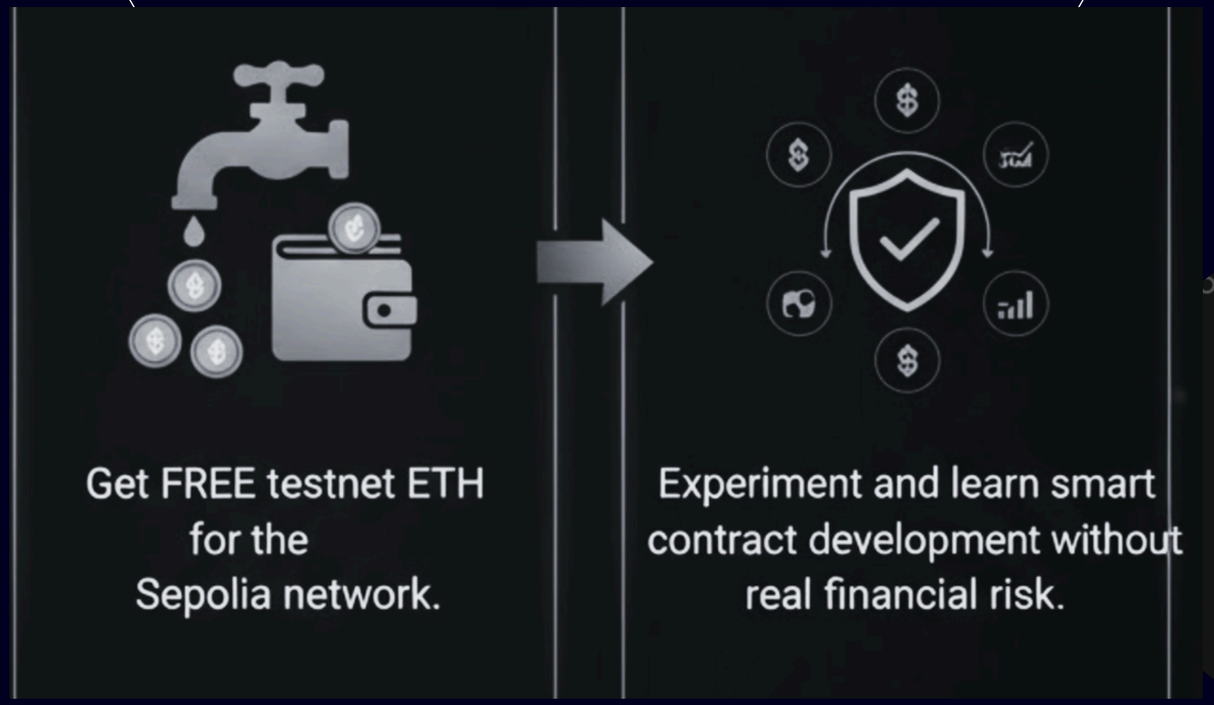
3. Automated Deployment




```
(base) merniebkz@MacBook-Air-Zharkynai donation-app % node scripts/deploy.ts
🚀 Starting manual deployment...
👤 Deployer address: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
⏳ Deploying Donation...

============================================================
🎉 DEPLOYMENT SUCCESSFUL!
📍 Donation App:  0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512
🪙 Reward Token: 0xCafac3dD18aC6c6e92c921884f9E4176737C052c
============================================================

(base) merniebkz@MacBook-Air-Zharkynai donation-app %
```




```
(base) merniebkz@MacBook-Air-Zharkynai donation-app % npx hardhat node
Started HTTP and WebSocket JSON-RPC server at http://127.0.0.1:8545/

Accounts
========

WARNING: Funds sent on live network to accounts with publicly known private keys

Account #0:  0xf39fd6e51aad88f6f4ce6ab8827279cfffb92266 (10000 ETH)
Private Key: 0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80

Account #1:  0x70997970c51812dc3a010c7d01b50e0d17dc79c8 (10000 ETH)
Private Key: 0x59c6995e998f97a5a0044966f0945389dc9e86dae88c7a8412f4603b6b78690d

                0x3c44cdddb6a900fa2b585dd299e03d12fa4293bc (10000 ETH)
                0x5de4111afa1a4b94908f83103eb1f1706367c2e68ca870fc3fb9a804cdab365a

                f79bf6eb2c4f870365e785982e1f101e93b906 (10000 ETH)
                2118294e51e653712a81e05800f419141751be58f605c371e15141b007a6
```

Process for Obtaining Test ETH

100%

Full-Stack DApp

Complete integration from smart contracts to user interface

100%

Blockchain Integration

Seamless frontend-Ethereum communication via Ethers.js

100%

Security Standards

Secure fund management with comprehensive validation

Conclusion

This project successfully demonstrates practical blockchain application development, meeting all academic requirements while showcasing comprehensive understanding of decentralized systems, smart contract security, and Web3 integration patterns.

The implementation represents production-quality code ready for evaluation, with clear documentation and extensible architecture for future enhancements.