Decentralized Charity Platform (Crowdfunding)

Technical Documentation
University Final Project
Students: Tulegenov Alimzhan , Abdygalym Khamza , Sarsenbek Yerrkebulan

Supervisor: Course: Blockchain Systems Institution: AITU University Date: February 7, 2026

Table of Contents

1. Executive Summary

This project implements a decentralized charity crowdfunding platform on Ethereum. Users can create campaigns, donate test ETH, and automatically receive Supporter Tokens (ERC-20). The system is built with a simple 3-tier architecture: Frontend UI, Ethers.js integration, and Solidity smart contracts. The solution prioritizes transparency, traceability, and automatic reward distribution.

2. Project Purpose

Traditional charity systems depend on centralized intermediaries, which can reduce donor trust due to limited transparency and delayed reporting. By using blockchain:

Every donation is recorded on a public ledger and can be audited by anyone.
Fund flows can be verified independently without relying on a central party.
Reward distribution is executed automatically through smart contracts.

3. System Architecture

The platform follows a clear 3-tier architecture:

Presentation Layer: Static frontend for users to create and fund campaigns.
Integration Layer: Ethers.js for wallet connection and transactions.
Blockchain Layer: Solidity smart contracts for business logic and token minting.

3.1 Frontend

The frontend is a static web app (HTML/CSS/JS) served from `frontend/` . It:

- Displays campaigns and their status.
- Shows wallet balances (ETH and Supporter Tokens).
- Provides actions to create campaigns, contribute, and finalize.

3.2 Web3 Integration (Ethers.js)

Ethers.js (v6) is used in `frontend/app.js` to :

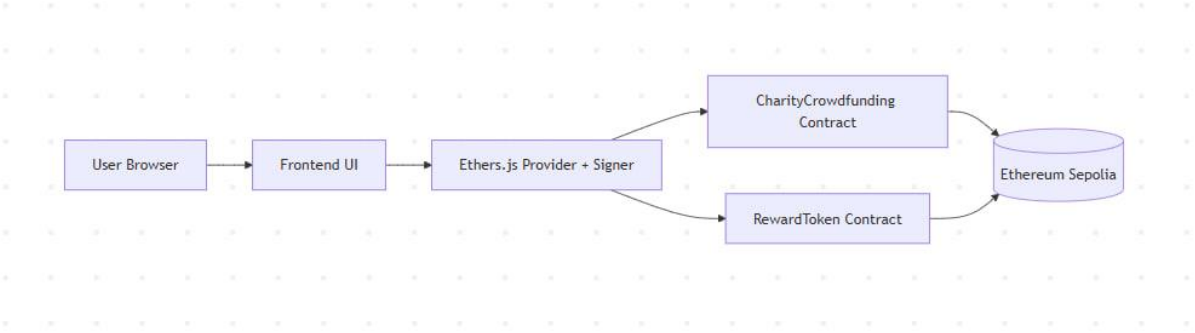
- Connect to MetaMask through `window.ethereum` .
- Read contract state (view functions).
- Sign and submit transactions (state changes).

3.3 Smart Contracts (Solidity)

The blockchain layer includes two contracts:

- `CharityCrowdfunding.sola` : Manages campaigns, donations, and finalization.
- `RewardToken.sol` : ERC-20 token used to reward donors.

Architecture Diagram



4. Design and Implementation Decisions

In developing the Decentralized Charity Platform, several architectural and technical choices were made to ensure security, transparency, and a seamless user experience.

Choice of ERC-20 Standard for Rewards: We implemented the "Supporter Token" (CRWD) using the ERC-20 standard to provide donors with a liquid, recognizable proof of contribution. Unlike simple receipts, ERC-20 tokens can be easily integrated into other DeFi ecosystems or displayed in any standard wallet like MetaMask, giving the "Proof of Support" tangible digital value within the project scope.

Automatic Minting Logic: A key decision was to trigger the mint function directly within the contribute transaction. This atomic approach ensures that a donor never has to wait for a manual distribution or a second transaction. The moment the ETH is sent to the campaign, the reward tokens are generated and sent to the donor's address, eliminating trust issues between the platform and the supporters.

Contract Ownership and Security: To prevent unauthorized token inflation, we implemented a restricted minting policy. During the deployment phase, the ownership of the RewardToken contract is transferred to the CharityCrowdfunding contract. This ensures that no individual—not even the developers—can mint tokens manually; they can only be created by the verified crowdfunding logic through a successful donation.

Decoupled Frontend via Ethers.js: We chose Ethers.js (v6) over other libraries because of its lightweight nature and robust handling of providers and signers. By utilizing BrowserProvider, we allow the frontend to remain entirely static and decentralized, as it relies solely on the user's MetaMask extension to interact with the Ethereum Sepolia network.

5. Smart Contract Logic

5.1 CharityCrowdfunding Contract

Core responsibilities:

Store campaigns and contribution history.
Accept ETH donations.
Mint Supporter Tokens for donors.
Finalize campaigns after deadlines.

Key data structures:

- `Campaign`:
  - `title`: campaign name
  - `creator`: campaign owner address
  - `goalWei`: funding goal in wei
  - `deadline`: Unix timestamp
  - `raisedWei`: total raised so far
  - `finalized`: finalization flag
- `campaigns[campaignId]`: maps ID to campaign data.
- `contributions[campaignId][contributor]`: donation history per user.

Key functions:

- `createCampaign(title, goalWei, durationSeconds)`
  - Validates inputs.
  - Sets a deadline using `block.timestamp + durationSeconds`.
  - Stores campaign and emits `CampaignCreated`.
- `contribute(campaignId)` payable:
  - Requires campaign to be active and not finalized.
  - Adds funds and updates contribution mapping.
  - Mints Supporter Tokens to donor.
  - Emits `Contributed`.
- `finalize(campaignId)`:
  - Can be called after the deadline.
  - Marks the campaign as finalized.
  - If goal reached, transfers all funds to creator.
  - Emits `Finalized`.

Reward formula:

Reward rate is 100 tokens per 1 ETH.
The mint amount is calculated as:

```
reward = msg.value * 100
```

5.2 RewardToken (Supporter Token) Contract

`RewardToken` is a standard ERC-20 token:

Name: Charity Reward Token
Symbol: CRWD
Decimals: 18 (default for ERC-20)

Only the contract owner can mint new tokens. During deployment, ownership is transferred to the `CharityCrowdfunding` contract so rewards are minted only through donations.

5.3 Events and Off-Chain Indexing

Events enable the frontend (or an indexer) to track changes:

- `CampaignCreated`
- `Contributed`
- `Finalized`

6. MetaMask Integration

The frontend handles wallet integration using MetaMask:

- Detects MetaMask via `window.ethereum`.
- Requests account access with `eth_requestAccounts`.
- Creates an Ethers `BrowserProvider` and `Signer`.
- Validates network against `frontend/contracts.json`.
- Disables actions if the user is on the wrong chain.

Listens to:

- `accountsChanged` to refresh balances and contracts.

`chainChanged` to re-check network and reload data.

Deployment Guide (Sepolia Testnet)

6.1 Prerequisites

Node.js v18+

MetaMask browser extension
Sepolia testnet access

6.2 Environment Configuration

Create a `.env` file at the project root:

```
SEPOLIA_RPC_URL="https://your-provider.example/v2/<api-key>"
SEPOLIA_PRIVATE_KEY="0xYOUR_PRIVATE_KEY"
```

6.3 Obtain Sepolia Test ETH

1. Open MetaMask and switch to Sepolia.
2. Copy your wallet address.
3. Use a trusted Sepolia faucet from a major provider or wallet to request test ETH.
4. Wait for the faucet transaction to confirm.
5. Verify your balance in MetaMask or a block explorer.

6.4 Deploy Contracts

```
npm install
npx hardhat run scripts/deploy.ts --network sepolia
```

The script deploys:

- `RewardToken`
- `CharityCrowdfunding` (linked to RewardToken)
  Transfers token ownership to CharityCrowdfunding

6.5 Configure Frontend

Update: `frontend/contracts.json`

`network` : "sepolia"

- `chainId` : 11155111
- `rewardToken.address` : deployed address
  `crowdfunding.address` : deployed address

6.6 Run the Frontend

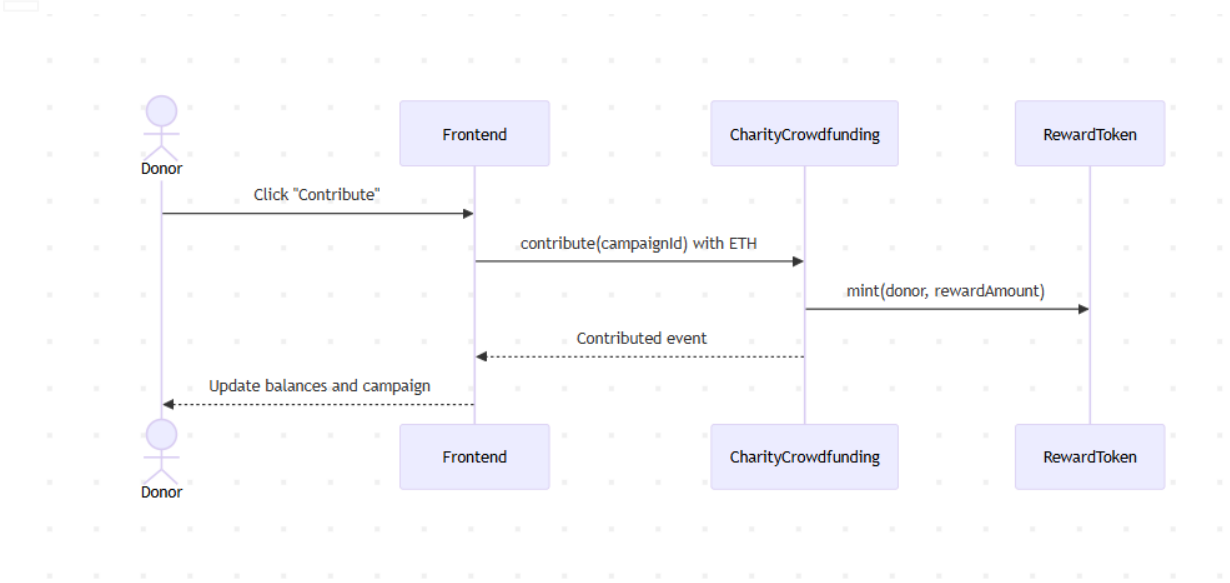
```
npx http-server frontend -p 8000
```

Open `http://localhost:8000` and connect MetaMask on Sepolia.

7. User Flow

1. User connects wallet in MetaMask.
2. User creates a campaign with a goal and duration.
3. Donors contribute test ETH.
4. Donors automatically receive Supporter Tokens (CRWD).
5. After deadline, anyone can finalize the campaign.
6. If goal reached, funds are transferred to the campaign creator.

Sequence Diagram



8. Security Considerations and Limitations

No refund mechanism if a campaign fails to reach its goal.
Finalization is time-based and must be called manually.
No pause or emergency withdrawal in the current design.
Contracts are not upgradeable; changes require redeployment.

9. Testing

Run the existing tests with:

```
npx hardhat test
```

Recommended additional tests:

Edge cases for zero values and expired campaigns.
Repeated finalization attempts.
Token minting only via the crowdfunding contract.

10. Future Improvements

Add refunds for unsuccessful campaigns.
Add campaign categories and metadata (images, descriptions).
Implement on-chain governance for charity verification.
Add indexer integration (e.g., The Graph) for fast querying.

11. Appendix: Key Files and Parameters

Key source files:

- `contracts/CharityCrowdfunding.sol`
- `contracts/RewardToken.sol`
- `scripts/deploy.ts`
- `frontend/app.js`
- `frontend/contracts.json`

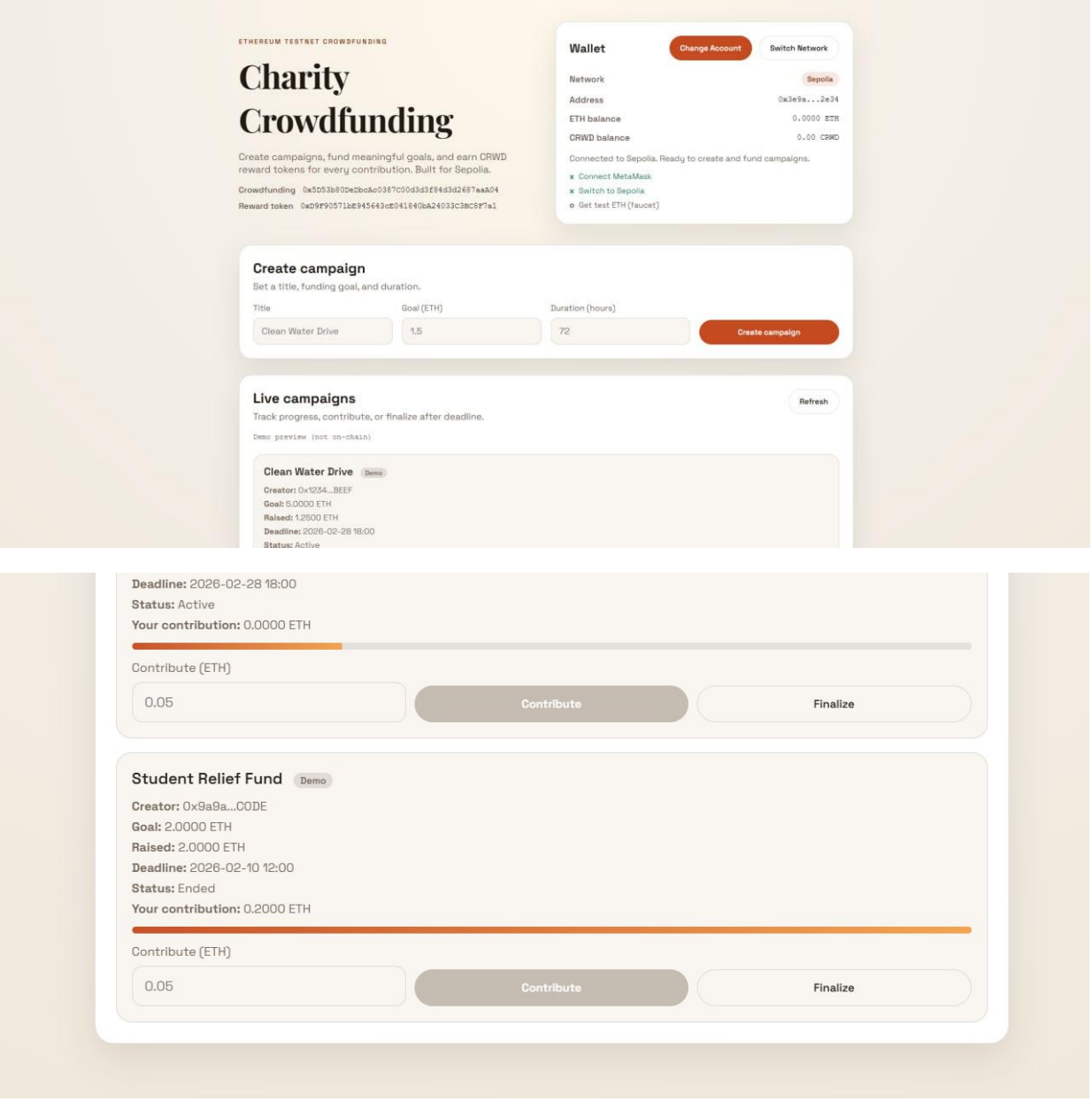
Key parameters:

Sepolia Chain ID: 11155111
Token symbol: CRWD
Reward rate: 100 tokens per 1 ETH

12. Appendix: Screenshots

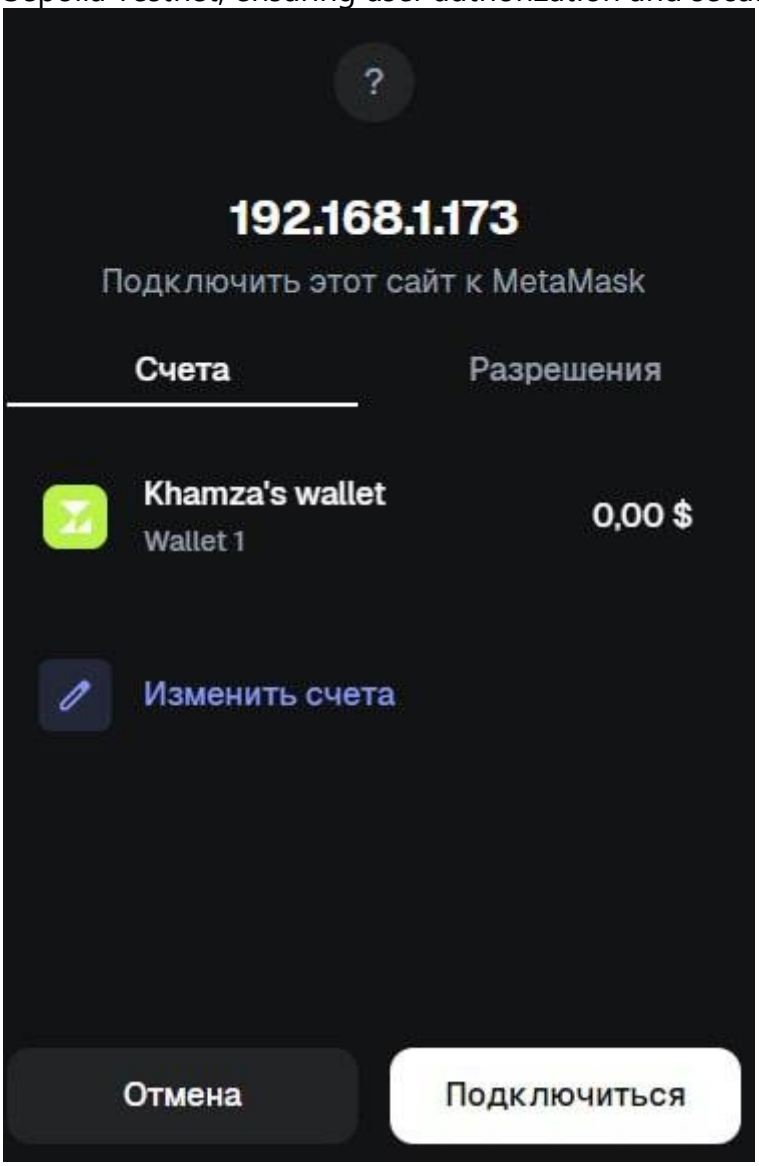
12.1 Operational DApp Interface

Caption: Figure 1: The main user interface displaying active charity campaigns and the current balance of ETH and Supporter Tokens (CRWD).



12.2 MetaMask Transaction Confirmation

Caption: Figure 2: MetaMask prompt for signing a donation transaction on the Sepolia Testnet, ensuring user authorization and secure interaction.



12.3 Smart Contract on Sepolia Etherscan

Caption: Figure 3: Verified smart contract on Etherscan Sepolia, demonstrating full transparency of the crowdfunding logic and transaction history.