AITU

STUDENT FUNDRAISING *PROJECT*

by
Moldir, Kaisar, Arlan

AITU

Problem Statement

Crowdfunding is usually centralized, so contributors must trust a third party to handle money and rules fairly. In practice, transparency is limited: users can't always verify fund handling in real time, reward distribution depends on manual actions, and refunds may be delayed or unclear.

- Centralized custody of funds requires trust
- Limited transparency of campaign state and transactions
- Rewards are manual or inconsistent ("thank you" model)
- Refund process can be slow or unclear

Project Idea

Our solution is a tokenized crowdfunding DApp where contributions are made directly to a smart contract, and contributors automatically receive ERC-20 reward tokens. The reward is minted on-chain at the moment of contribution, so it does not depend on an admin or platform decisions.

- User contributes test ETH on Sepolia
- Smart contract mints ERC-20 tokens as rewards
- Reward is proportional to the contribution amount
- Campaign lifecycle: create → contribute → finalize → refund (if failed)

AITU

AITU

Goals and Functional Requirements

- Create campaigns (title, goal, duration)
- Accept ETH contributions and track per user
- Automatically mint reward tokens on contribution
- Finalize campaigns after deadline (success/fail)

The project implements the complete crowdfunding workflow on-chain. Creators can launch campaigns with a goal and a deadline, users can fund them, and the system decides the outcome deterministically. If the goal is reached, the creator receives funds; if not, contributors can claim refunds.

The application consists of a web frontend connected to MetaMask and two Solidity contracts deployed on Sepolia. The frontend is only an interface; all critical logic and state are stored on-chain, which increases transparency and reduces reliance on centralized control.

- MetaMask: wallet connection + transaction signing
- Frontend: HTML/CSS/JS + ethers.js
- Smart contracts: RewardToken + Crowdfunding logic
- Sepolia testnet: safe environment (test ETH only)

Architecture

AITU

AITU

Smart Contracts Overview

We separated responsibilities into two contracts to keep the system clean and secure. The token contract follows the ERC-20 standard, while the crowdfunding contract contains the business logic. Token minting is restricted, and ownership is assigned to the crowdfunding contract to enable automatic rewards.

RewardToken (ERC-20): balances, symbol, decimals, mint

TokenizedCrowdfunding: campaigns, contributions, finalize, refunds

Mint is restricted (onlyOwner)

Ownership transferred to crowdfunding contract

Automatic reward distribution on-chain

RewardToken Contract

RewardToken is an ERC-20 token used as a contribution reward. To prevent abuse, minting is restricted to the token owner. After deployment, the ownership is transferred to the crowdfunding contract so that it can mint tokens automatically during contributions.

- **ERC-20 standard implementation**
- **mint(to, amount) protected by onlyOwner**
- **Enables automatic token rewards**
- **Prevents arbitrary minting by random users**
- **Owner becomes the crowdfunding contract**

AITU

Crowdfunding Data Model

- Campaign fields: creator, title, goal, deadline, raised
- Status flags: finalized, successful
- Per-user contributions: contributions[id][user]
- Transparent state visible on-chain
- Deterministic outcome based on goal/deadline

AITU

Contribution and Token Reward Logic

The core feature is “contribute and get tokens automatically”. When a user calls contribute(campaignId) and sends ETH, the contract updates campaign totals and mints reward tokens immediately according to a fixed reward rate.

- contribute() is payable (ETH is sent to the contract)
- Checks campaign is active (not finalized, before deadline)
- Updates raised amount and user contribution
- Mints reward tokens based on reward rate
- Reward formula: tokens = contributed ETH × rate

Finalization Logic

Finalization locks the campaign outcome and prevents repeated actions. After the deadline, the contract determines success based on whether raised >= goal. If successful, funds go to the creator; if unsuccessful, refunds become available.

- Finalize allowed after campaign ends
- If raised >= goal → successful = true
- Successful → funds transferred to creator
- If raised < goal → successful = false
- Finalized flag prevents multiple finalizations

AITU

Refunds protect contributors when a campaign fails. A contributor can claim a refund only if the campaign is finalized and unsuccessful, and only if they actually contributed. After refunding, the contribution record is reset to prevent double refunds.

- Refund only if finalized = true and successful = false
- Refund only if user contribution > 0
- ETH returned to contributor from the contract
- Contribution set to 0 after refund
- Prevents double refund attacks

Refund Mechanism

AITU

AITU

Demo Results and Verification

The project was tested end-to-end on Sepolia: wallet connection, campaign creation, contributions, token rewards, finalization, and refund scenario. The demo proves that the full lifecycle works and that the reward/refund rules are enforced by smart contracts, not by a centralized platform.

- MetaMask connection and Sepolia network check
- Deploy contracts and configure frontend addresses
- Create campaign and contribute ETH
- Token balance increases after contribution
- Finalize success / finalize fail + claim refund

THANK *YOU*