INSTITUTION
Astana IT University

DATE
12 February 2026

GreenPulse

Decentralized Crowdfunding Platform

Empowering environmental impact through transparent, token-incentivized fundraising on the Ethereum blockchain.

PROJECT TYPE
Blockchain Technology 1
Final Project Report

PRESENTED BY
Aibek Nazarbek
Sergey Nurtilek
Beisenbek Nazly

Project Overview

Core Purpose

To demonstrate the practical application of blockchain technology in Decentralized Finance (DeFi) by building a transparent, immutable crowdfunding ecosystem for environmental initiatives.

Platform Mechanics

- Users create campaigns for environmental projects
- Contributors fund campaigns using Ethereum (ETH)
- Automatic distribution of **"LEAF" ERC-20 tokens** as rewards

Value Proposition

- **Trust-Minimized:** Removing centralized intermediaries
- **Transparent:** All fund flows visible on-chain
- **Secure:** Smart contracts control withdrawals and refunds

Technical Scope

- Solidity Smart Contracts
- Hardhat Testing Framework
- Angular Frontend + Ethers.js
- MetaMask Integration

CURRENT LANDSCAPE

Limitations of Centralized Crowdfunding

Traditional platforms act as gatekeepers, introducing systemic risks and inefficiencies that undermine the core purpose of community funding.

Trust & Custody Risks

Reliance on intermediaries to hold funds creates custodial risk. Users must trust third parties not to mismanage or freeze assets.

Single Point of Failure

Centralized servers are vulnerable to hacks, DDoS attacks, and technical downtime, potentially halting fundraising campaigns.

Censorship Risks

Platforms can arbitrarily block campaigns, freeze accounts, or de-platform users due to political or corporate pressure.

Opaque Accounting

Financial records are kept in private databases. Contributors cannot verify if funds are actually being used for the stated project.

High Platform Fees

Intermediaries charge significant fees (5-15%) for service and payment processing, reducing the actual capital reaching the project.

Delayed Settlements

Creators often wait weeks for funds to clear banking systems after a campaign ends, delaying project kick-off.

Strategic Objectives

Defining the core pillars of success for a secure, scalable, and practical decentralized crowdfunding ecosystem.

01

Security

Implementing robust smart contracts with strict access control and **ReentrancyGuard** protection to prevent attacks.

02

Scalability

Utilizing a modular architecture with **CampaignFactory** to deploy independent per-campaign contracts.

03

Efficiency

Optimizing gas usage through **batched reads**, minimal storage writes, and efficient data structures.

04

Practicality

Demonstrating full-stack Web3 integration using **Angular, Ethers.js, and MetaMask** to bridge the gap between users and blockchain logic.

05

Rigor

Ensuring reliability through comprehensive **automated testing**, CI/CD pipelines, and professional software development methodologies.

System Architecture Layers

A robust three-tier stack combining secure Solidity smart contracts, rigorous testing frameworks, and a modern reactive frontend.

Smart Contract Layer

CORE LOGIC

- `Campaign.sol`
- `CampaignFactory.sol`
- `GreenToken.sol (LEAF)`

Development Layer

TESTING & OPTIMIZATION

- Hardhat Framework
- Mocha & Chai Testing
- Gas Reporting

Frontend Layer

USER INTERFACE

- Angular (TypeScript)
- Ethers.js Integration
- MetaMask Wallet
- Tailwind CSS

Component Interaction Flow

Frontend

User Interface Layer

COMPONENTS
- Angular Components
- Reactive Signals
- Tailwind UI

LIBRARY
- Ethers.js Provider

Wallet

Injected Provider

MetaMask
Transaction Signing

Blockchain

Ethereum Network

Ethereum Node
State Storage & EVM Execution

Smart Contracts

On-Chain Logic

CampaignFactory.sol

↓

Campaign.sol (Instance)

↓

GreenToken.sol (ERC20)

Presentation Layer | Web3 Injection | Network Layer | Application Logic

Campaign Lifecycle Sequence

From campaign creation to fund withdrawal, every step is automated and secured by smart contracts.

1. **CREATOR** — **Create Campaign**
   User calls Factory to initialize a new project with goal & deadline.

2. **FACTORY** — **Deploy Contract**
   Factory deploys a new independent Campaign contract instance.

3. **CONTRIBUTOR** — **Contribute ETH**
   Supporters send ETH directly to the specific Campaign address.

4. **CAMPAIGN** — **Trigger Reward**
   Contract verifies deposit and calls Factory to request token minting.

5. **FACTORY** — **Mint LEAF**
   Factory mints LEAF tokens to contributor (100 LEAF per 1 ETH).

6. **CREATOR** — **Withdraw Funds**
   Creator withdraws raised ETH only after funding goal is successfully met.

User Action | System/Factory Logic | Smart Contract Logic

Core Smart Contract Components

A modular architecture ensuring security, scalability, and automated incentives across the ecosystem.

Campaign.sol

CORE LOGIC INSTANCE

- **Contribution Management**
  Accepts ETH, tracks donor balances, and updates raised amount securely.
- **Withdrawal Logic**
  Restricted to creator via `onlyCreator`; allowed only if `raised >= goal`.
- **Refund Mechanism**
  Contributors can reclaim ETH if the funding goal is missed by the deadline.

CampaignFactory.sol

ORCHESTRATOR & REGISTRY

- **Campaign Deployment**
  Deploys new isolated contract instances for each project to ensure scalability.
- **Central Registry**
  Maintains an array of all deployed campaigns for frontend discovery.
- **Reward Authority**
  Acts as the trusted authority to mint LEAF rewards when valid contributions occur.

GreenToken.sol

ERC-20 INCENTIVE

- **Standard Implementation**
  Based on OpenZeppelin ERC-20 standard for compatibility with wallets.
- **Restricted Minting**
  Only the `CampaignFactory` address is authorized to mint new tokens.
- **Incentive Model**
  Rewards environmental impact: 100 LEAF tokens minted per 1 ETH contributed.

Platform Highlights

Secure and incentivized crowdfunding ecosystem.

Transparent Fundraising

Decentralized architecture ensures all contributions are immutable and publicly verifiable on-chain.

TRUSTLESS

Automatic Rewards

Smart contracts automatically mint and distribute 100 LEAF tokens for every 1 ETH contributed.

INCENTIVIZED

Secure Fund Management

Funds are held in smart contracts. Withdrawals are cryptographically guarded and only permitted upon goal success.

- OnlyCreator Access
- Goal Verification
- Reentrancy Protection

Auto-Refund Logic

If a campaign fails to reach its goal, contributors can claim a full refund via the smart contract.

SAFETY NET

Efficient Data Retrieval

Optimized functions return campaign stats in a single JSON-RPC call, ensuring a fast UI.

GAS OPTIMIZED

Optimization Techniques

Strategies to minimize transaction costs and enhance contract logic.

Constant Time Lookups `O(1) Gas`

Utilized `mapping` over arrays for contribution tracking.

External Visibility `Calldata`

Methods use `external` to read directly from calldata, saving memory.

Batched Data Retrieval `1 RPC Call`

Aggregated stats in `getSummary()` to reduce network overhead.

Storage Caching `Reduced SLOAD`

Cached state variables in stack memory during complex loops and logic.

Compiler Optimization `Solc 0.8.28`

Optimizer enabled with `runs: 200` for lean bytecode.

Impact

Directly reducing costs for end-users.

~20%
GAS REDUCTION

O(1)
SCALING

200
OPT. RUNS

Communication Bridge

Ethers.js connects Angular to the blockchain via `window.ethereum`.

`ethers.BrowserProvider` → `MetaMask`

Provider

READ-ONLY CONNECTION

- Reads blockchain state
- No user permission required
- Zero gas cost

```
const data = await contract.getSummary();
return data;
```

VS

Signer

WRITE & EXECUTE

- Sends state transactions
- Requires MetaMask popup
- Incurs Gas fees (ETH)

```
const tx = await contract.contribute({
  value: parseEther("1.0")
});
```

Reactive State Management (Angular Signals)

Signals trigger UI updates instantly when blockchain data changes. `tokenBalance.set(val)` updates the view without a refresh.

Fortified Smart Contract Architecture

Defense-in-depth strategy combining secure coding patterns, access controls, and rigorous testing.

Reentrancy Protection

Applied OpenZeppelin's `nonReentrant` modifier to prevent recursive call attacks during ETH transfers.

`ReentrancyGuard`

Checks-Effects-Interactions

Strict logical ordering: validate inputs, update state, and then perform external calls to minimize attack surfaces.

`Logic Pattern`

Role-Based Access

Utilized `Ownable`. Only the Factory can mint tokens; only Campaign Creators can withdraw funds.

`AccessControl`

Overflow Safety

Leveraged Solidity 0.8.x built-in overflow/underflow checks, eliminating legacy SafeMath library overhead.

`Solidity ^0.8.0`

Comprehensive Validation

Hardhat test suite covering edge cases and malicious flows.

100% COVERAGE	25+ UNIT TESTS	PASSED STATUS

Iterative Development Lifecycle

A systematic approach from smart contract architecture to production-ready deployment.

01 Smart Contract Design

Architecting modular contracts, defining data structures, and establishing access control rules.

`Solidity` `UML`

02 Unit Testing

Validating logic with comprehensive test suites covering success paths and edge cases.

`Hardhat` `Mocha` `Chai`

03 Gas Optimization

Analyzing gas reports and refining code to minimize execution costs for end users.

`Gas Reporter` `Assembly`

04 Frontend Integration

Connecting the Angular UI to blockchain state via Web3 provider injection.

`Angular` `Ethers.js` `Signals`

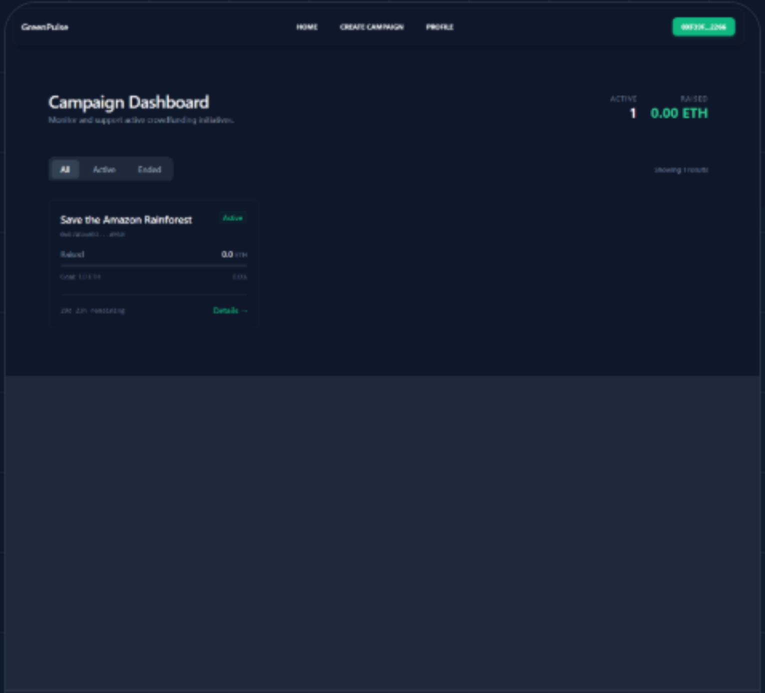
05 Deployment & CI/CD

Automated pipelines for testing, compilation, and network deployment.

`GitHub Actions` `Scripts`

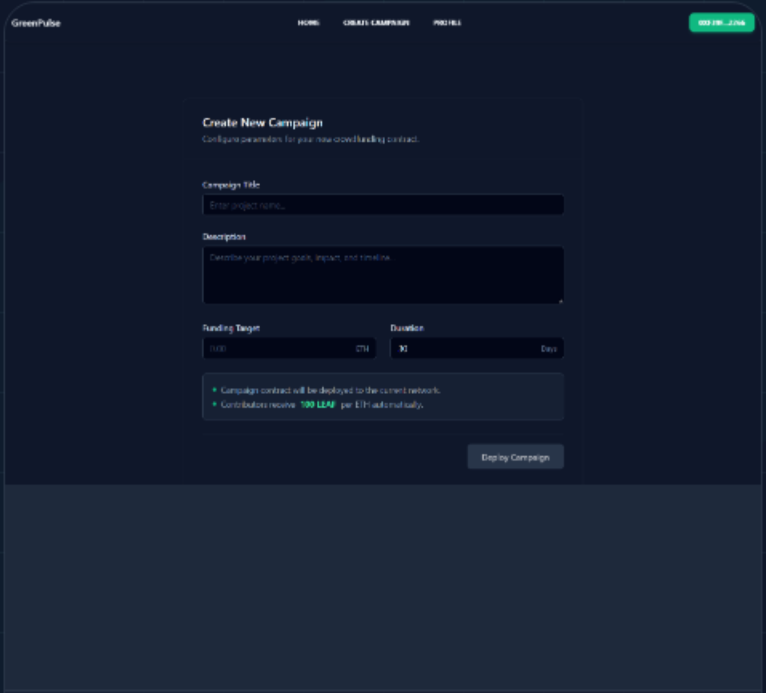
User Interface

A modern, responsive experience designed for transparency and ease of use.



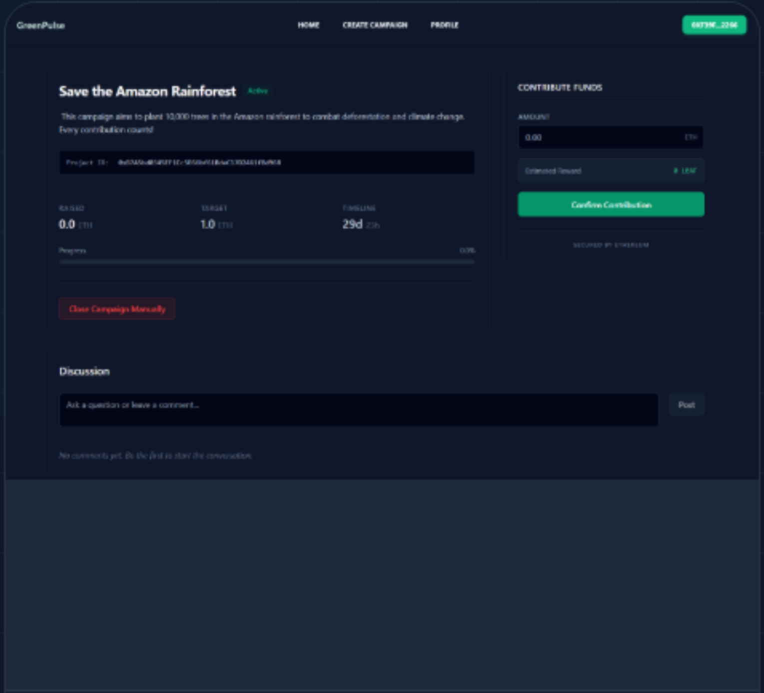
01 Campaign Discovery

Home dashboard allows users to browse active environmental campaigns, filtering by category or funding status.



02 Campaign Creation

Intuitive form for creators to deploy new smart contracts. Defines funding goals, deadlines, and project details.



03 Contribution View

Detailed tracking of funds raised vs goal. Features real-time ETH contribution input and progress visualization.

Current Challenges

Layer-1 Limitations

High Gas Fees

Network congestion causes transaction costs to spike, making small contributions ($10-$50) economically unviable for average users.

Throughput Limits

Ethereum Mainnet processes ~15-30 TPS. During high-demand campaigns, transaction confirmation times can degrade user experience.

Storage Costs

On-chain storage is expensive. Storing extensive campaign descriptions and media directly on Ethereum is cost-prohibitive.

Scaling Solutions

Layer-2 & Sidechains

Optimistic Rollups

Assumes transactions are valid by default. Significant gas savings while inheriting L1 security.

`Optimism / Arbitrum`

zk-Rollups

Uses zero-knowledge proofs for validity. Offers higher throughput and instant finality compared to optimistic approaches.

`zkSync / StarkNet`

Polygon Integration

Deploying to Polygon POS sidechain offers near-zero gas fees and fast block times (~2s), ideal for micro-funding campaigns.

`Production Ready`

Technical Evolution

From centralized testing to a fully decentralized autonomous ecosystem for green energy funding.

PHASE 2.0

The Graph Integration

Implementing Subgraphs to index on-chain events. This enables millisecond-speed queries for campaign filtering and contributor history without direct RPC calls.

GraphQL · AssemblyScript · Event Indexing

PHASE 2.5

IPFS / Filecoin Storage

Migrating campaign metadata and high-res impact media to decentralized storage. Removes reliance on AWS/Cloudinary for project documentation.

Content-Addressing · Pinata · CID Persistence

PHASE 3.0

Layer-2 Expansion

Deploying to Optimism and Arbitrum. Drastically reduces gas costs by 95%, making micro-contributions (under $5) economically viable for the first time.

Optimistic Rollups · L2 Bridges · Scalability

PHASE 4.0

Governance DAO

Transitioning to community control. Token holders will vote on-chain for platform fee adjustments, new project approvals, and treasury allocations.

Tally Integration · Snapshot · Quadratic Voting

Key Achievements

Successfully delivered a robust, decentralized fundraising ecosystem through rigorous engineering.

Full-Stack Web3 Integration

Seamlessly connected Solidity smart contracts with a reactive Angular frontend using Ethers.js, providing a smooth user experience.

Secure Contract Engineering

Implemented ReentrancyGuard, Checks-Effects-Interactions patterns, and rigorous access control to prevent common DeFi vulnerabilities.

Gas-Optimized Design

Reduced transaction costs through O(1) mapping lookups, batched data retrieval, and strategic memory variable usage.

Automated Testing & CI/CD

Established a reliable DevOps pipeline with GitHub Actions, ensuring every commit passes a comprehensive Mocha/Chai test suite.

Modular Architecture

Designed a scalable factory pattern that deploys isolated campaign contracts, ensuring system resilience and clean separation of concerns.

> GreenPulse demonstrates a practical DeFi application that successfully bridges the gap between decentralized technology and environmental impact.

Trust-Minimized Logic

By replacing centralized intermediaries with immutable smart contracts, we ensure transparent fund tracking and guaranteed refunds without third-party reliance.

Tokenized Incentives

The LEAF token model introduces a circular economy, transforming passive donations into active community engagement through tangible digital rewards.

Scalable Foundation

With a modular factory architecture and gas-optimized code, the system provides a robust blueprint for real-world Web3 crowdfunding platforms.