Final Project Report: Decentralized Crowdfunding Platform

Course: Blockchain 1

Team: Yelzhan Zhandos, Ramazan Kozhabek, Issa Akhmet

GitHub Repository: https://github.com/Ramazan-droid/blockchain

1. Project Overview

1.1 Introduction

This project implements a fully functional Decentralized Crowdfunding Application (DApp) on the Ethereum blockchain. The platform allows users to create fundraising campaigns, contribute to campaigns using test Ether, and receive ERC-20 reward tokens as incentives. The application operates exclusively on the Ethereum Sepolia test network, ensuring no real cryptocurrency is involved while demonstrating real blockchain functionality.

1.2 Problem Statement

Traditional crowdfunding platforms face several challenges including:

- Centralized control and single points of failure
- Lack of transparency in fund management
- High intermediary fees (5-10% platform fees)
- Delayed fund disbursement and withdrawal restrictions
- Limited global accessibility

1.3 Our Solution

Our decentralized crowdfunding platform addresses these issues by leveraging blockchain technology to create:

- Transparent funding: All transactions are publicly verifiable on the blockchain
- Automated execution: Smart contracts automatically manage funds and rewards
- Global accessibility: Anyone with an Ethereum wallet can participate
- Reduced costs: Minimal gas fees replace high platform commissions
- Trustless environment: Code governs operations, not centralized entities

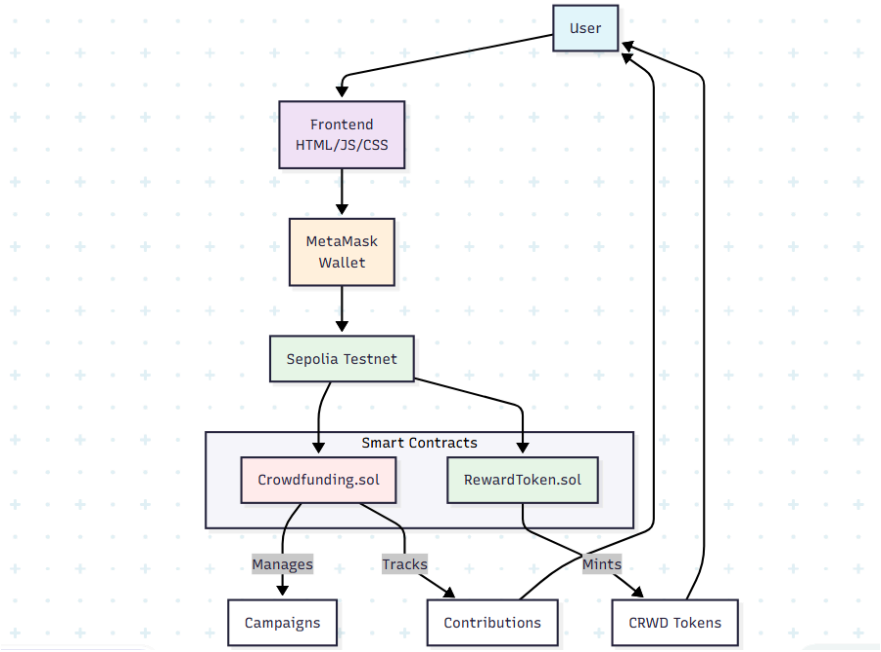
1.4 Educational Objectives

This project serves as a comprehensive demonstration of blockchain development skills including:

- Smart contract design and implementation in Solidity
- ERC-20 token standard application
- MetaMask wallet integration
- Frontend-blockchain communication
- Test network deployment and management

2. System Architecture

Our DApp follows a three-layer architecture that separates concerns while ensuring seamless integration between components:



Architectural Layers:

1. Presentation Layer: User interface built with HTML5, CSS3, and JavaScript
2. Application Layer: Smart contracts written in Solidity
3. Data Layer: Ethereum blockchain storage on Sepolia testnet

3. Smart Contract Implementation

3.1 Contract Structure

We implemented two main smart contracts:

Crowdfunding Contract (Crowdfunding.sol):

Core functionality includes:
- Campaign creation with customizable parameters
- Secure contribution mechanism with ETH tracking
- Automated reward token distribution
- Campaign status management and finalization
- Comprehensive event logging for frontend updates

RewardToken Contract (RewardToken.sol):
// Features:
- ERC-20 compliant token (OpenZeppelin implementation)
- Token name: "CrowdToken", Symbol: "CRWD"
- 18 decimal places for precision
- Restricted minting function (only callable by Crowdfunding contract)

- Pure educational purpose (no monetary value)

3.2 Key Contract Functions

A	B	C	D
Function	**Parameters**	**Returns**	**Description**
createCampaign()	title, goal, duration	campaignId	Creates new funding campaign
contribute()	campaignId, msg.value	-	Contributes ETH to campaign
finalizeCampaign()	campaignId	-	Ends campaign and distributes funds
getCampaign()	campaignId	Campaign struct	Returns campaign details
isActive()	campaignId	bool	Checks if campaign is active
mint()	recipient, amount	-	Mints reward tokens (internal)

3.3 Security Features

Our smart contracts incorporate multiple security measures:

1. Access Control: Critical functions restricted to authorized addresses
2. Input Validation: All parameters validated using require() statements
3. Reentrancy Protection: Follows checks-effects-interactions pattern
4. Deadline Enforcement: Automatic campaign expiration
5. Fund Safety: Contributions stored securely until campaign completion
6. Event Emission: Comprehensive logging for transparency

3.4 Contract Events

Event	Parameters	Purpose
CampaignCreated	id, title, goal, deadline	Notify frontend of new campaign
Contributed	campaignId, contributor, amount	Log contribution transactions
TokensMinted	recipient, amount	Track token distribution

4. Frontend Development

4.1 User Interface Components

The frontend provides an intuitive interface with the following modules:

Wallet Connection Module:

- MetaMask detection and connection handling

- Network validation (Sepolia only)
- Account address display
- Real-time balance updates (ETH and CRWD tokens)

Campaign Management Interface:

- Campaign creation form with validation
- Interactive campaign listing with progress visualization
- Contribution interface with amount selection
- Campaign status indicators (active/ended)

Dashboard Components:

- User wallet information panel
- Transaction status notifications
- Real-time campaign statistics
- Token balance display

4.2 MetaMask Integration

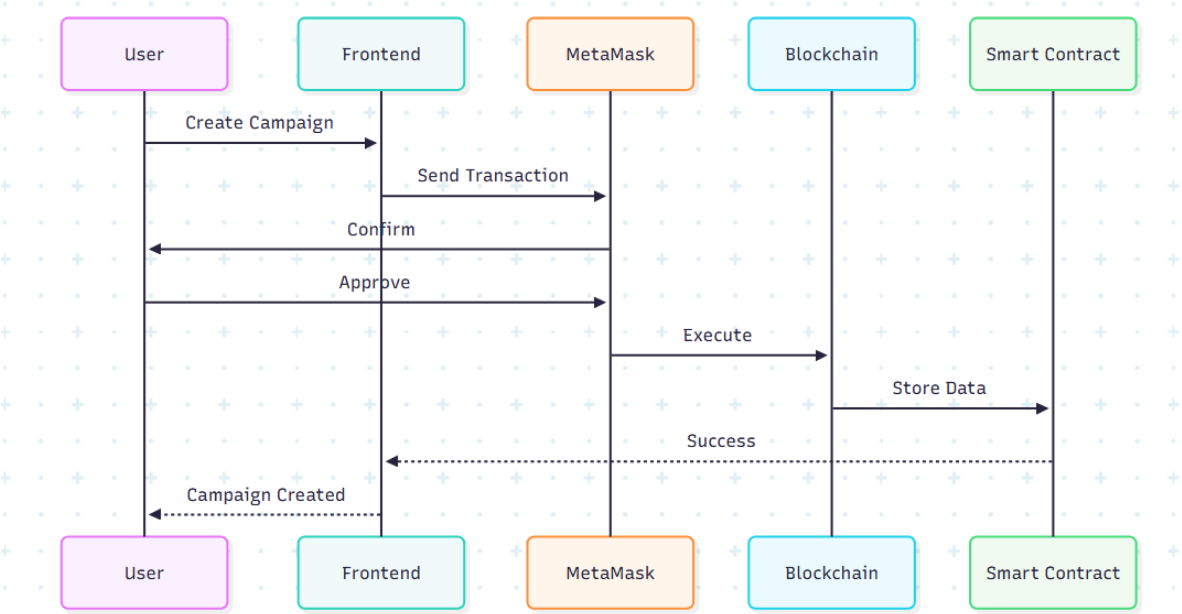
The application seamlessly integrates with MetaMask through:

// Key integration points:
1. Wallet Detection: Checking window.ethereum availability
2. Account Access: Using eth_requestAccounts API
3. Network Validation: Ensuring Sepolia network (chainId: 0xaa36a7)
4. Transaction Signing: MetaMask popup for user approval
5. Event Listening: Real-time account/network change detection

5. System Workflow

5.1 User Journey

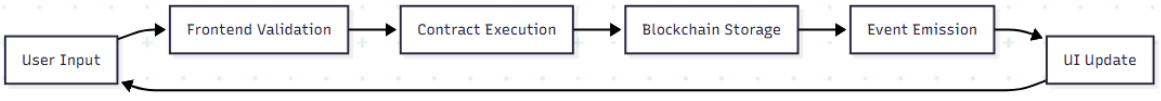
The platform supports the following primary user flows:



5.2 Transaction Flow

1. User initiates action through frontend interface
2. Frontend prepares transaction with correct parameters
3. MetaMask signs transaction after user approval
4. Transaction broadcast to Sepolia network
5. Smart contract executes the requested function
6. Events emitted for frontend consumption
7. UI updates with transaction results

5.3 Data Flow



6. Development Process

6.1 Development Environment Setup

1. Node.js environment with npm package management
2. Hardhat framework for smart contract development

3. TypeScript configuration for better development experience
4. Environment variables for secure key management
5. Git version control for collaborative development

6.2 Testing Strategy

We implemented comprehensive testing at multiple levels:

Unit Testing:

```
npx hardhat test
```

- Individual function testing
- Edge case validation
- Security vulnerability checks

Integration Testing:

- End-to-end transaction flows
- Contract interaction testing
- Frontend-blockchain communication

Network Testing:

- Sepolia testnet deployment
- Real transaction verification
- Gas optimization testing

6.3 Deployment Proces

```
# Step-by-step deployment:
1. npm install                # Install dependencies
2. npx hardhat compile        # Compile smart contracts
3. npx hardhat test           # Run test suite
4. Update .env file           # Configure environment variables
5. npx hardhat run scripts/deploy.js --network sepolia
6. Update frontend configuration # Set contract addresses
```

7. Technical Challenges & Solutions

7.1 Challenges Encountered

Challenge	Description	Solution Implemented
MetaMask Network Switching	Users on wrong network	Automatic network detection and switching prompts
Transaction Confirmation Delays	User uncertainty during waits	Clear loading indicators and status updates
Contract ABI Management	Large ABI files in frontend	Separate ABI files with only necessary functions
Test ETH Acquisition	Users needing test funds	Clear documentation on Sepolia faucet usage
Error Handling	Transaction failures	Comprehensive error messages and recovery options

7.2 Key Technical Decisions

1. Sepolia Testnet Selection: Modern, stable test network with good faucet availability
2. OpenZeppelin Implementation: Using battle-tested ERC-20 implementation for security
3. ethers.js over web3.js: Modern, TypeScript-friendly library with better documentation
4. Event-Driven Architecture: Efficient frontend updates without constant polling
5. Responsive Design: Mobile-first approach for wider accessibility

7.3 Performance Optimizations

- Gas Optimization: Efficient storage patterns and minimal on-chain operations
- Frontend Caching: Local storage for contract ABIs and user preferences
- Batch Operations: Combining related transactions where possible
- Lazy Loading: Loading campaign data only when needed

8. Educational Value

8.1 Learning Outcomes

This project provided comprehensive learning in:

Blockchain Fundamentals:

- Understanding of decentralized systems
- Smart contract development lifecycle
- Transaction lifecycle and gas mechanics
- Public/private key cryptography

Technical Skills:

- Solidity programming and best practices

- ERC-20 token standard implementation
- Frontend-blockchain integration
- Test network deployment and management

Development Practices:

- Version control with Git
- Environment variable management
- Comprehensive testing strategies
- Documentation and presentation skills

8.2 Course Requirement Alignment

Course Requirement	Our Implementation
Smart Contract Development	Full Solidity implementation
ERC-20 Token Usage	Custom CRWD token with minting
MetaMask Integration	Complete wallet connectivity
Test Network Usage	Sepolia deployment and testing
Frontend Development	Responsive HTML/JS interface
Documentation	Comprehensive technical documentation

9. Security Considerations

9.1 Security Measures Implemented

1. Test Network Only: No mainnet deployment, eliminating financial risk
2. Input Validation: All user inputs validated on-chain
3. Access Control: Restricted functions with modifiers
4. Event Logging: Complete transaction transparency
5. No Real Funds: Educational tokens only, no monetary value

9.2 Security Best Practices Followed

- Use of established libraries (OpenZeppelin)
- Comprehensive testing before deployment
- Environment variable management for private keys
- Regular dependency updates
- Code review and peer validation

9.3 Limitations & Assumptions

- Assumes MetaMask is installed and configured
- Relies on Sepolia network availability
- Requires test ETH for transactions
- Educational scope limits advanced features

10. Results & Evaluation

10.1 Functional Requirements Met

Campaign Creation: Users can create campaigns with custom parameters
Contribution System: Secure ETH contributions with tracking
Token Rewards: Automated ERC-20 token distribution
Wallet Integration: Full MetaMask connectivity
Network Operation: Sepolia testnet deployment
User Interface: Intuitive, responsive frontend

10.2 Technical Requirements Met

Solidity Smart Contracts: Two fully functional contracts
ERC-20 Token: Custom reward token implementation
Frontend Integration: Complete blockchain interaction
Transaction Handling: Secure MetaMask transaction flow
Event System: Real-time frontend updates
Error Handling: Comprehensive user feedback

11. Conclusion

11.1 Project Success

This project successfully demonstrates a complete decentralized crowdfunding application that meets all course requirements while providing practical blockchain development experience. The system is fully functional, secure, and educational.

11.2 Key Achievements

1. Complete DApp Development: End-to-end blockchain application from contracts to UI
2. Educational Focus: Clear learning outcomes aligned with course objectives
3. Technical Excellence: Modern development practices and security considerations
4. User-Centric Design: Intuitive interface with clear user journeys

11.3 Final Assessment

The Decentralized Crowdfunding Platform represents a successful implementation of blockchain technology for practical applications. It serves as both a functional system and an educational resource, demonstrating the potential of decentralized systems while providing hands-on learning in blockchain development.

The project successfully balances technical complexity with usability, making blockchain technology accessible while maintaining robust security and functionality. It stands as a comprehensive demonstration of the skills and knowledge gained throughout the Blockchain 1 course.

Appendices

A. Installation & Usage Instructions

Prerequisites:

- Node.js (v16 or higher)
- MetaMask browser extension
- Test ETH from Sepolia faucet

Usage Flow:

1. Connect MetaMask and ensure Sepolia network
2. Get test ETH from https://sepoliafaucet.com
3. Create campaigns or contribute to existing ones
4. Receive CRWD tokens for contributions
5. Monitor campaign progress through dashboard

B. Team Contributions

Team Member	Contribution Area	Key Responsibilities
Ramazan	Smart Contracts	Contract development, testing, deployment
Yelzhan	Frontend Development	UI/UX design, MetaMask integration
Issa	Documentation	Report writing, presentation preparation
All Members	Testing & Validation	Comprehensive system testing

C. References & Resources

1. OpenZeppelin Contracts: https://openzeppelin.com/contracts/
2. Hardhat Documentation: https://hardhat.org/docs
3. MetaMask Developer Docs: https://docs.metamask.io/
4. Ethereum Sepolia: https://sepolia.dev/
5. ethers.js Documentation: https://docs.ethers.org/v6/
6. Solidity Documentation: https://docs.soliditylang.org/