**FINAL EXAMINATION PROJECT**

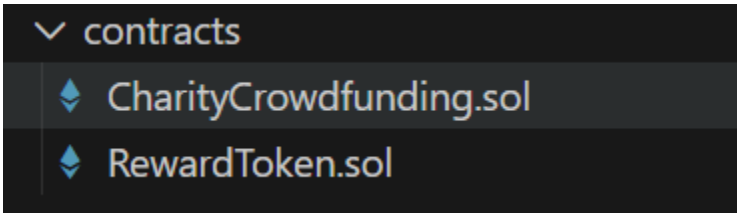
**Course: *Blockchain 1***
**Format: Group Project Technology**
**Stack: Solidity, JavaScript, MetaMask, Ethereum (Testnet)**

**Maratova Oryngul, Kazbek Aigerim, Bekzhan Danelya SE-2426**

## PURPOSE OF THE FINAL PROJECT

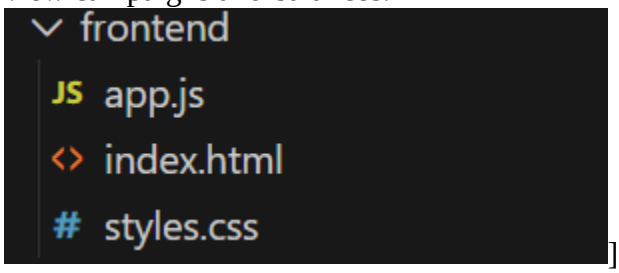
The project is a decentralized crowdfunding application built on the Ethereum test network (Sepolia).
The system consists of three main parts:

1. **Smart Contracts (Blockchain Layer)**
   Written in Solidity and deployed on Ethereum Sepolia.
   **CharityCrowdFunding.sol**
   **RewardToken.sol (ERC-20 standard)- for automatically rewarding with tokens.**
   They store campaign data, user contributions, and token rewards.
   All financial logic and security rules are handled inside smart contracts.




2. **Frontend (Client Layer)**
   Built using HTML, CSS, and JavaScript.
   The frontend provides the user interface where users can:

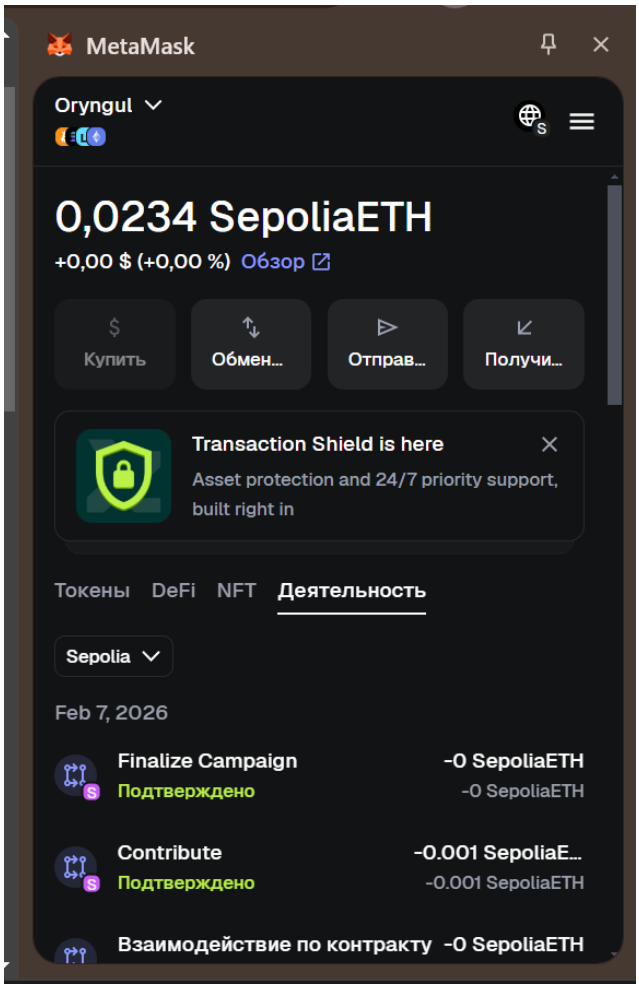
- connect their wallet,
- create campaigns,
- donate ETH,
- view campaigns and balances.



]

3. **MetaMask (Wallet Layer)**
   MetaMask acts as a bridge between the user and the blockchain.
   It signs transactions and gives permission to interact with smart contracts.

The architecture follows a typical DApp model:
**User → Frontend → MetaMask → Smart Contract → Blockchain.**




## Project Overview

It is a decentralized crowdfunding application operating exclusively on an Ethereum test network and utilizing free test tokens only.

The developed application enable:

- creation of crowdfunding campaigns;
- participation of users as contributors;
- issuance of internal reward tokens for contributions;
- secure interaction with the blockchain via MetaMask.

## Design and Implementation Decisions

Several design decisions were made to ensure simplicity and clarity:

- **Sepolia Test Network** was chosen to avoid real money usage.
- **OpenZeppelin Libraries** were used for ERC-20 token and ownership logic to ensure security and reliability.
- **Custom Errors** were added to reduce gas usage and improve debugging.
- **Reward Token System** was implemented to demonstrate tokenization concepts.
- **Duration Limits** were added to campaigns to prevent unrealistic deadline.

## Smart Contract Logic Description

**CharityCrowdfunding.sol**

This contract manages all crowdfunding operations:

- **createCampaign**
  Creates a new campaign with title, funding goal, duration, and beneficiary address.
  Includes validation checks (goal > 0, valid duration, non-empty title).
- **contribute**
  Allows users to send ETH to active campaigns.
  Updates total raised amount and mints reward tokens automatically.
- **finalizeCampaign**
  Can be executed only after the deadline.
  If the funding goal is reached, ETH is transferred to the beneficiary.
  Otherwise, the campaign is marked as failed.
- **refund**
  Allows contributors to get their ETH back if the campaign failed and was finalized.
- **getCampaign / getStatus**
  Helper functions for the frontend to read campaign data in one call.

**RewardToken.sol**

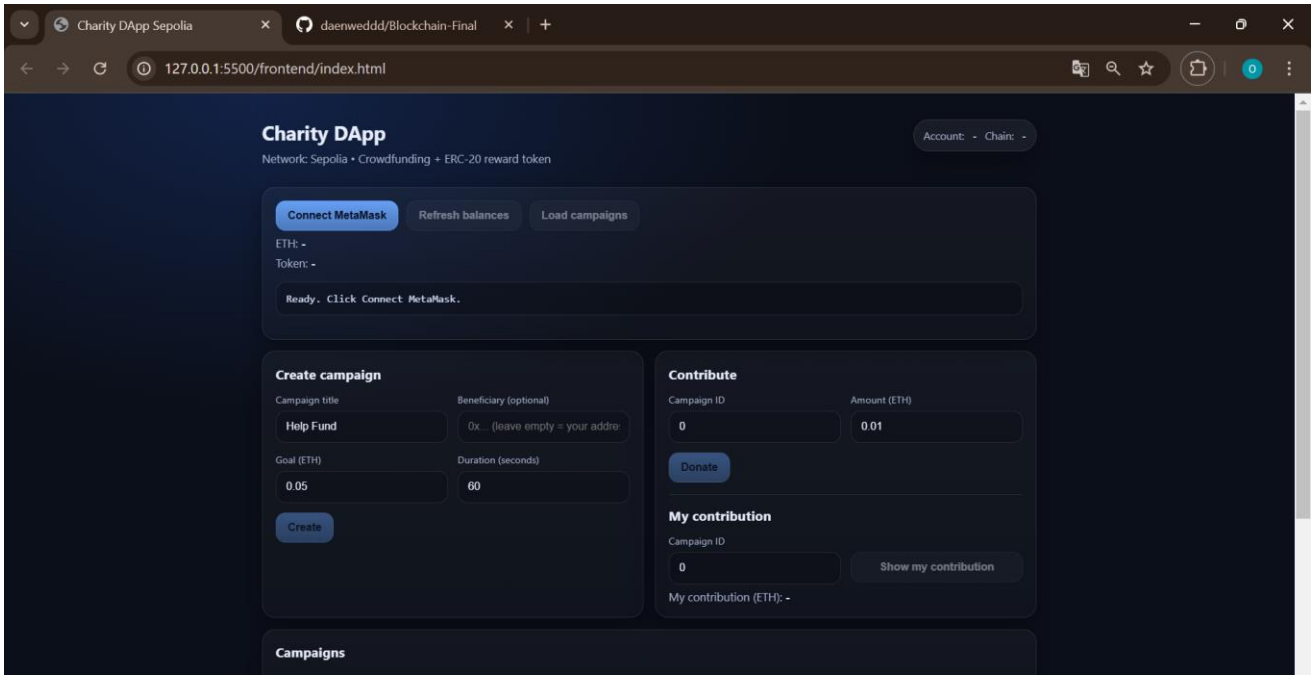
This is a custom ERC-20 token used only inside the application:

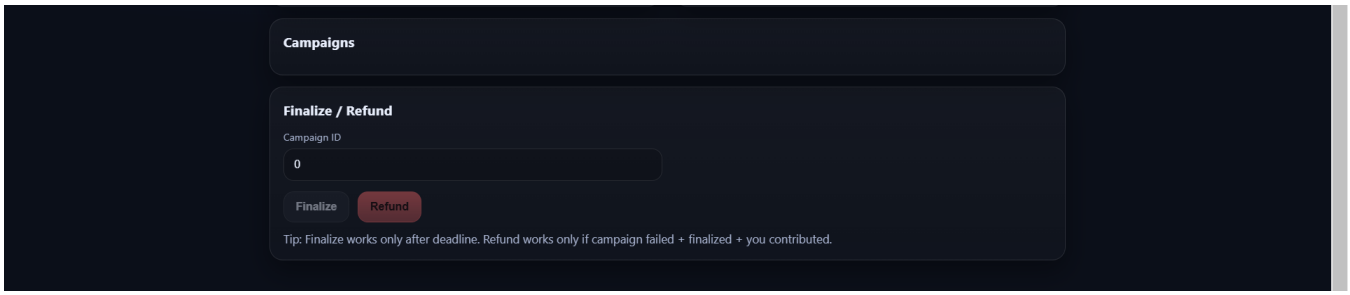
- Tokens are minted automatically when users donate.
- Only the crowdfunding contract is allowed to mint tokens.

## Frontend-to-Blockchain Interaction

The frontend communicates with the blockchain using **Ethers.js** library.
Interaction flow:

1. User clicks **Connect MetaMask**.
2. MetaMask requests wallet access.
3. The application checks if the network is Sepolia.
4. When a user performs an action (create, donate, finalize, refund),
   a transaction is sent through MetaMask.
5. MetaMask asks the user to confirm the transaction.
6. After confirmation, the smart contract executes the function.
7. The frontend updates UI with new balances and campaign data.






# Deployment and Execution Instructions

**Smart Contract Deployment**

1. Install dependencies using npm install.
2. Compile contracts using npx hardhat compile
3. Take RPC URL from alchemy
4. Configure .env with RPC URL and private key.
5. Deploy contracts using:
6. npx hardhat run scripts/deploy.js --network sepolia
7. Copy deployed contract addresses into app.js frontend configuration.

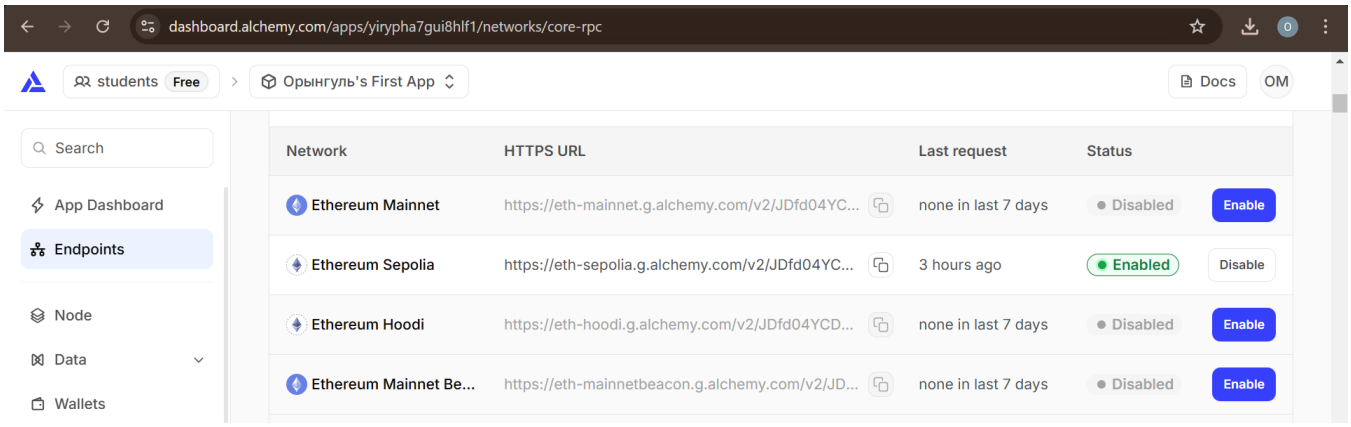
**Running Frontend**

- Open index.html using Live Server or a local server.
- Connect MetaMask.
- Ensure Sepolia network is selected.
- Start creating campaigns and interacting with the DApp.

## Obtaining Test ETH

Test ETH is required for transactions on Sepolia.
It can be obtained from free public faucets such as:

- Alchemy Sepolia Faucet
- Other public Ethereum test faucets
  We take it from Google cloud web3 Ethereum Sepola Faucets




Users simply paste their wallet address into the faucet website and receive small amounts of test ETH.
This ETH has **no real financial value** and is used only for testing transactions.

**Conclusion:**
This project demonstrates the full lifecycle of a decentralized application including smart contract development, token integration, MetaMask interaction, and frontend-blockchain communication in a secure test environment.