**Project name:** Decentralized Crowdfunding DApp
**Course:** Blockchain Technologies 1
**Group:** SE-2433
**Team members:** Aibek Nurtay, Valeriy Fedorenko, Aizada Nazarbek
**Stack:** Solidity, JS, MetaMask, Ethereum Testnet

## 1) Project Overview:

This project impleplements a decentralized crowdfunding application that operates exclusively on the Ethereum testnet and uses only free test ETH and test tokens. The application supports creating campaigns, allowing users to contribute, and minting internal reward tokens to contributors. The use of Ethereum mainnet and real cryptocurrency is strictly prohibited and not used in this project

## 2) Application architecture:

The system consists of four main layers:

1. Frontend (HTML/JS):
   Provides UI for connecting MetaMask, creating campaigns, contributing, finalizing campaigns, and viewing balances
2. MetaMask Wallet:
   Handles account access permission requests, network selection, and user signing of transactions.
3. Ethereum Test Network:
   The blockchain environment where contracts are deployed and transactions are executed
4. Smart contracts:
   Crowdfunding.sol - manages campaigns, contributions, and campaign finalization
   RewardToken.sol - mints internal reward tokens during participating

**Data storage approach:**
All campaign state (title, goal, deadline, raised amount, finalized status) and contribution tracking are stored on-chain inside Crowdfunding.sol Token balances are stored in the ERC-20 contract as standard balances.

## 3) Design and Implementation Decisions

The application is deesigned to run only on test network Sepolia and uses only free test ETH and tokens. This directly satisfies the constraint that mainnet and real cryptocurrency are not allowed.

A custom ERC-20 token is minted automatically when a user contributes, purely for educational demonstration and without monetary value

The contract provides the required stages:

- Create a campaign with parameters (title, goal, deadline)
- Accept contributions in test ETH
- Track iindividual contributions
- Finalize after the deadline

# 4) Smart Contract Logic

## 4.1) Campaign creation

The smart contract layer is responsible for all critical state and validation.

**Function:** createCampaign(string title, uint256 goalWei, uint256 durationSeconds)
Creates a new campaign with:

- title (string)
- goalWei (funding goal in wei)
- deadline = bllock.timestamp + durationSeconds

**Validations** (typical)**:**

- Title is not empty
- Goal is greater than 0
- Duration is greater than 0

**State changes:**

- Increments campaignCount
- Stores a Campaign struct in campaigns[campaignId]
- Emits CampaignCreated(campaignId, creator, title, goalWei, deadline)

## 4.2) Contributing to a campaign

**Function:** contribute(uint256 campaignId) (payable)

Allows a user to send test ETh to an active campaign

**Validations** (typical)**:**

campaignId exists

- msg.value > 0
- Current time is before deadline

- Campaign is not finalized

**State changes:**

- Updates contributions[campaignId][msg.sender] += msg.value
- Updates campaigns[campaignId].raisedWei += msg.value

**Events:**

Emits Contributed(campaignId, contributor, amountWei)

### 4.3) Finalizing a campaign

**Function**: finalize(uint256 campaignId)
Finalizes a campaign after the deadline

**Validations (**typical**):**

- campaaignId exists
- block.timestamp >= deadline
- Not already finalized

**State changes:**

- Sets campaign.finalized = true
- Emits Finalized(campaignId)

## 5)Frontend-to-Blockchain Interaction

### 5.1) MetaMask connection and permissions

The client-side application interacts with the blockchain via MetaMask and a JavaaScript Ethereum library (ethers.js)

The UI requests wallet access using MetaMask and reads the connected address.

The flow includes:

- Requesting permission to access accounts
- Reading the selected account address and displaying it

### 5.2) Network validation

The frontend verifies that the currently selected network is a supported test network (Sepolia) before sending transactions

### 5.3) Transactions and monitoring outcomes

The UI supports:

- Creating campaigns
- Contributing to campaigns
- Finalizing campaigns
- Monitoring transaction outcomes (e.g., tx hash, confirmation)

All transactions are executed through MetaMask (user signs and broadcasts)

### 5.4) Balance display

The frontend displays:

- User test ETH balance (from provider getBalance)
- Reward token balance (via ERC-20 balanceOf)

## 6) Deployment and Execution Instructions

These steps describe how to compile, deploy, and run the application

### 6.1) Prequisites:

- Node.js + npm installed
- MetaMask installed in browser FINAL EXAMINATION PROJECT (1)
- A supported Ethereum test network selected in MetaMask (Sepolia/Holesky)

### 6.2) Instrall dependencies

From the project root (where package.json is located):

npm install --legacy-peer-deps

### 6.3) Compile contracts

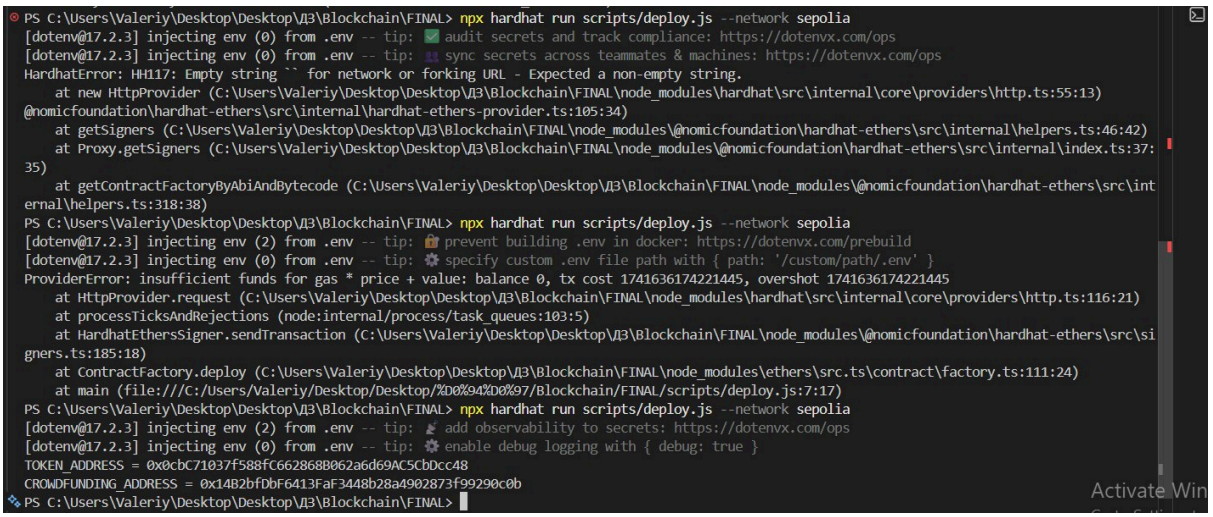
npm install --legacy-peer-deps

### 6.4) Configure environment for testnet deployment

Create a .env file (do not commit it) with:
SEPOLIA_RPC_URL=YOUR_RPC_URL

PRIVATE_KEY=YOUR_PRIVATE_KEY

### 6.5) Deploy to testnet

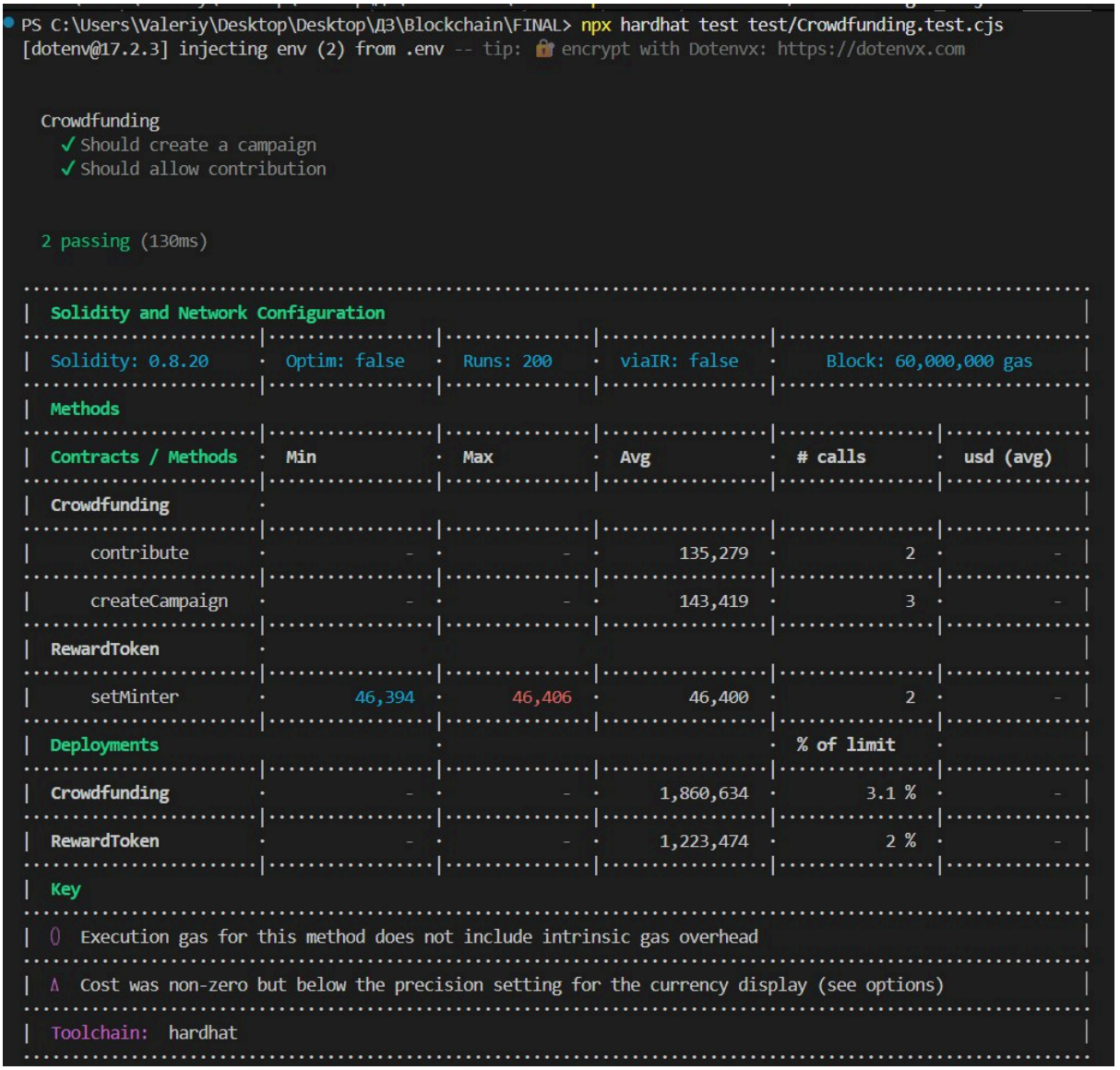



npx hardhat run scripts/deploy.js --network sepolia

After deployment, record:

TOKEN_ADDRESS = 0x…

CROWDFUNDING_ADDRESS = 0x…

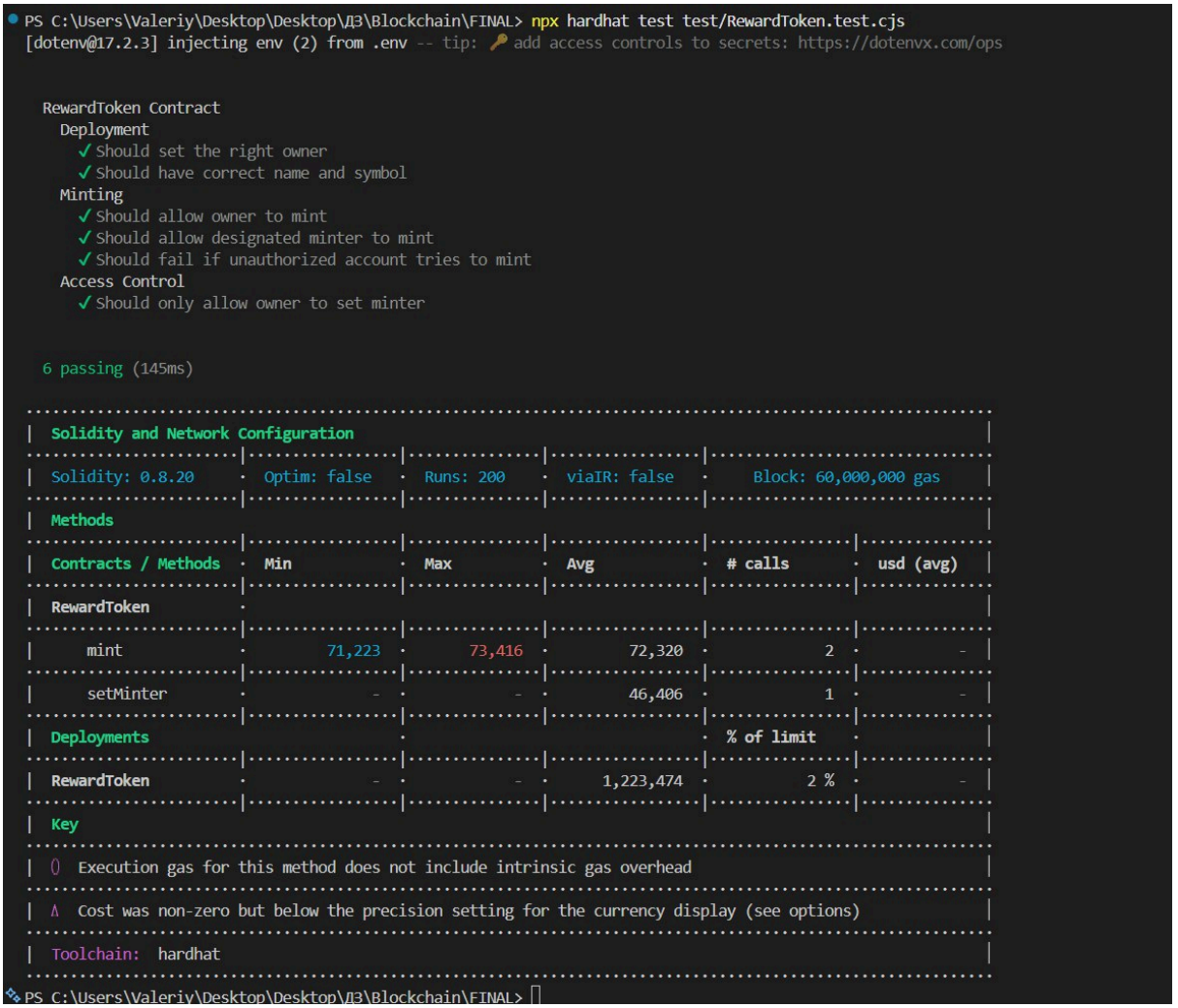


```
PS C:\Users\Valeriy\Desktop\Desktop\Д3\Blockchain\FINAL> npx hardhat test test/Crowdfunding.test.cjs
[dotenv@17.2.3] injecting env (2) from .env -- tip: 🔐 encrypt with Dotenvx: https://dotenvx.com


  Crowdfunding
    ✓ Should create a campaign
    ✓ Should allow contribution


  2 passing (130ms)

·-----------------------------------------------------------------------------------------------------------·
|  Solidity and Network Configuration                                                                       |
·----------------------|----------------|--------------|----------------|-------------------------------·
|  Solidity: 0.8.20    ·  Optim: false  ·  Runs: 200   ·  viaIR: false  ·     Block: 60,000,000 gas     |
·----------------------|----------------|--------------|----------------|-------------------------------·
|  Methods                                                                                                  |
·----------------------|----------------|--------------|----------------|---------------|---------------·
|  Contracts / Methods ·  Min           ·  Max         ·  Avg           ·  # calls      ·  usd (avg)    |
·----------------------|----------------|--------------|----------------|---------------|---------------·
|  Crowdfunding        ·                                                                                    |
·----------------------|----------------|--------------|----------------|---------------|---------------·
|      contribute      ·              -  ·           -  ·       135,279  ·            2  ·           -  |
·----------------------|----------------|--------------|----------------|---------------|---------------·
|      createCampaign  ·              -  ·           -  ·       143,419  ·            3  ·           -  |
·----------------------|----------------|--------------|----------------|---------------|---------------·
|  RewardToken         ·                                                                                    |
·----------------------|----------------|--------------|----------------|---------------|---------------·
|      setMinter       ·         46,394  ·      46,406  ·        46,400  ·            2  ·           -  |
·----------------------|----------------|--------------|----------------|---------------|---------------·
|  Deployments                           ·                               ·  % of limit   ·              |
·----------------------|----------------|--------------|----------------|---------------|---------------·
|  Crowdfunding        ·              -  ·           -  ·     1,860,634  ·       3.1 %  ·           -  |
·----------------------|----------------|--------------|----------------|---------------|---------------·
|  RewardToken         ·              -  ·           -  ·     1,223,474  ·          2 %  ·           -  |
·----------------------|----------------|--------------|----------------|---------------|---------------·
|  Key                                                                                                      |
·-----------------------------------------------------------------------------------------------------------·
|  ()  Execution gas for this method does not include intrinsic gas overhead                                |
·-----------------------------------------------------------------------------------------------------------·
|  Δ  Cost was non-zero but below the precision setting for the currency display (see options)              |
·-----------------------------------------------------------------------------------------------------------·
|  Toolchain:  hardhat                                                                                      |
·-----------------------------------------------------------------------------------------------------------·
```




## 6.6) Configure frontend contract addresses

Update the frontend config (example: frontend/js/config.js) with the deployed addresses and expected chainId: Sepolia chainId - 11155111

## 6.7) Run the frontend

Open the frontend using a static server or VS Code Live Server. Then:

1. Connect MetaMask
2. Confirm network validation
3. Create a campaign
4. Contribute test ETH
5. Finalize after deadline
6. Check ETH and token balances

# 7) Obtaining Test ETH

The application requires free test ETH to pay gas and contribute to campaigns

**Process used:**

1. Select the correct test network in MetaMask (Sepolia)
2. Copy your wallet address.
3. Use a faucet for that network to request test ETH.
4. Wait for the faucet transaction to complete and verify balance in MetaMask.

link for getting free ETH (faucet): Ethereum Sepolia Faucet