Ilnur, Elkham, Alisher. SE-2439

Final Project

1. Application Overview

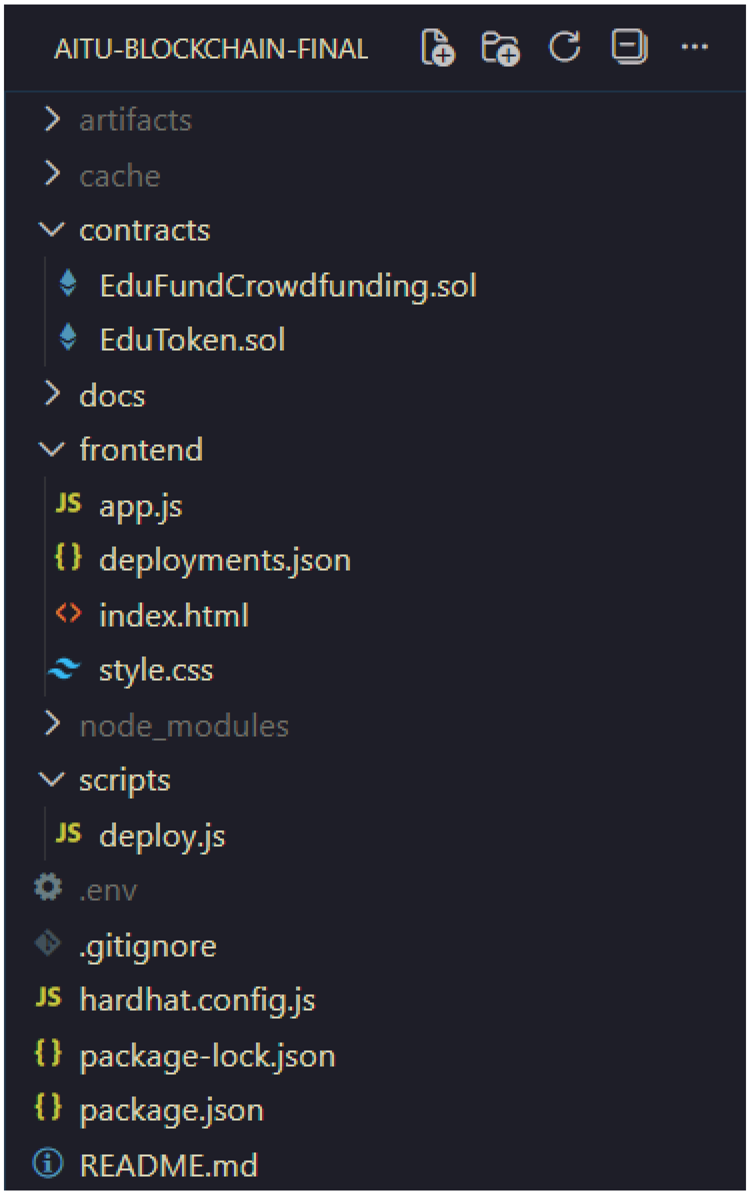
This project is a decentralized crowdfunding application developed as part of the Blockchain 1 final examination project. The application allows users to create crowdfunding campaigns, contribute test ETH, and receive internal ERC-20 reward tokens. The system operates exclusively on an Ethereum test network and uses MetaMask for secure wallet interaction. The project is open-source and can be accessed through this github link.

2. Application Architecture

The application follows a standard decentralized application architecture.

Components:

- Smart contracts
  - Crowdfunding logic
  - ERC-20 reward token
- Frontend
  - User interface
  - Blockchain interaction
- Wallet integration
  - MetaMask
- Development Environment
  - Hardhat
  - Ethereum test network (sepolia)



```
AITU-BLOCKCHAIN-FINAL
> artifacts
> cache
v contracts
    EduFundCrowdfunding.sol
    EduToken.sol
> docs
v frontend
    app.js
    deployments.json
    index.html
    style.css
> node_modules
v scripts
    deploy.js
  .env
  .gitignore
  hardhat.config.js
  package-lock.json
  package.json
  README.md
```

Architecture Flow:

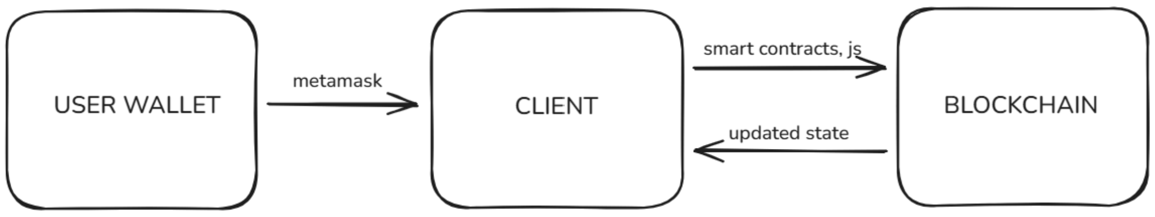
User connects MetaMask wallet

Frontend interacts with smart contracts via JavaScript

Transactions are signed and executed through MetaMask

Smart contracts update blockchain state

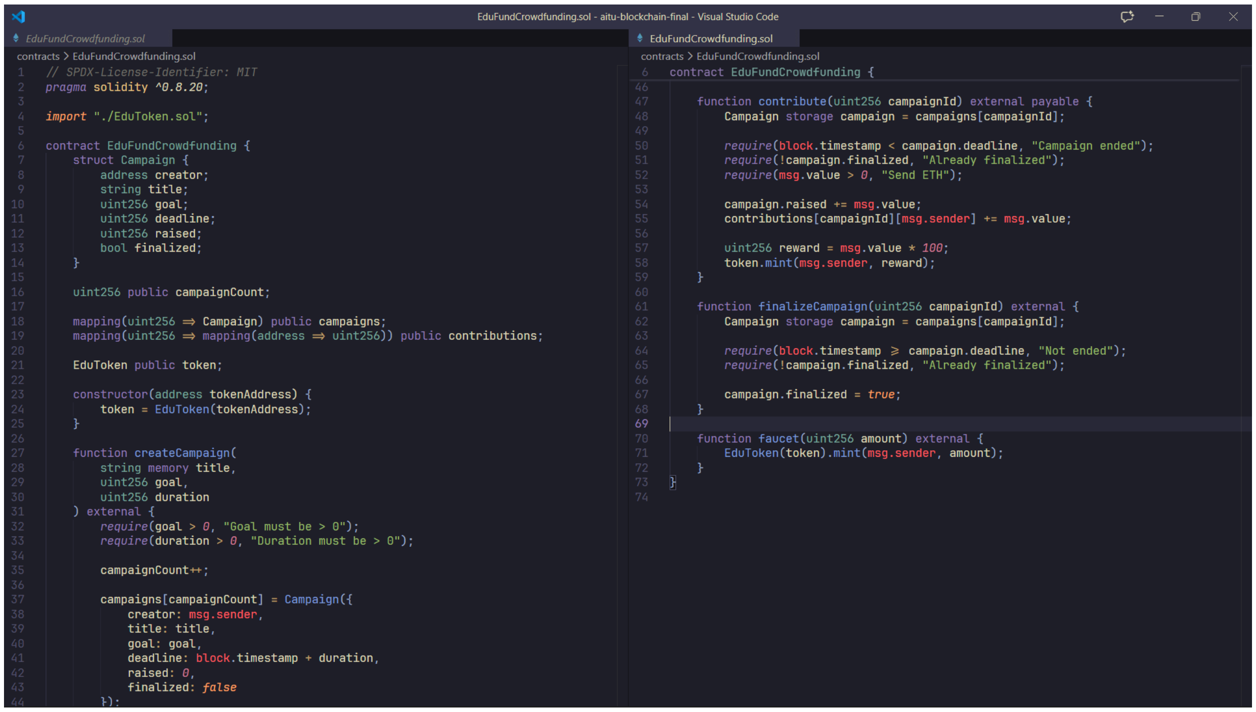
Frontend displays updated data (balances, campaigns)



3. Smart Contract Logic

3.1 Crowdfunding Contract

The crowdfunding smart contract is responsible for creating campaigns (with title, funding goal, and deadline), accepting ETH contributions, tracking individual user contributions, finalizing campaigns after deadline, and issuing reward tokens based on contribution amount. Each campaign is stored on-chain and is publicly accessible.

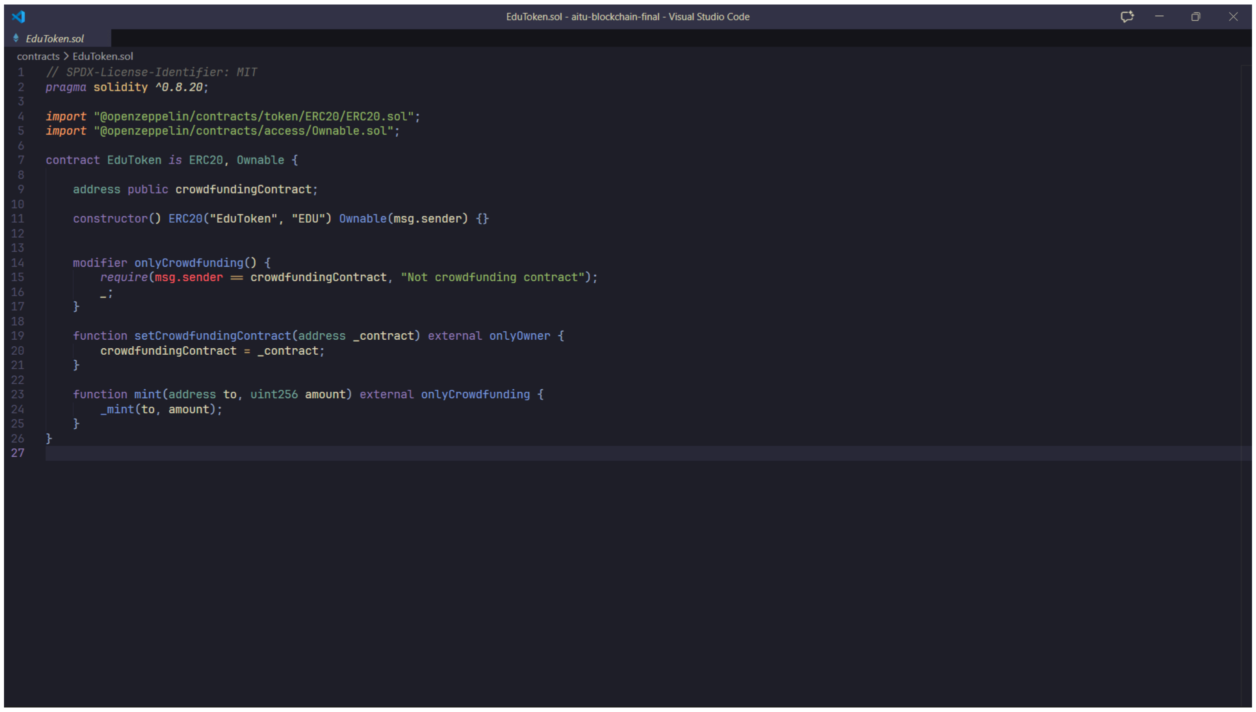

EduFundCrowdfunding.sol - aitu-blockchain-final - Visual Studio Code
EduFundCrowdfunding.sol
contracts > EduFundCrowdfunding.sol
EduFundCrowdfunding.sol
contracts > EduFundCrowdfunding.sol

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "./EduToken.sol";

contract EduFundCrowdfunding {
    struct Campaign {
        address creator;
        string title;
        uint256 goal;
        uint256 deadline;
        uint256 raised;
        bool finalized;
    }

    uint256 public campaignCount;

    mapping(uint256 => Campaign) public campaigns;
    mapping(uint256 => mapping(address => uint256)) public contributions;

    EduToken public token;

    constructor(address tokenAddress) {
        token = EduToken(tokenAddress);
    }

    function createCampaign(
        string memory title,
        uint256 goal,
        uint256 duration
    ) external {
        require(goal > 0, "Goal must be > 0");
        require(duration > 0, "Duration must be > 0");

        campaignCount++;

        campaigns[campaignCount] = Campaign({
            creator: msg.sender,
            title: title,
            goal: goal,
            deadline: block.timestamp + duration,
            raised: 0,
            finalized: false
        });
```

```solidity
contract EduFundCrowdfunding {

    function contribute(uint256 campaignId) external payable {
        Campaign storage campaign = campaigns[campaignId];

        require(block.timestamp < campaign.deadline, "Campaign ended");
        require(!campaign.finalized, "Already finalized");
        require(msg.value > 0, "Send ETH");

        campaign.raised += msg.value;
        contributions[campaignId][msg.sender] += msg.value;

        uint256 reward = msg.value * 100;
        token.mint(msg.sender, reward);
    }

    function finalizeCampaign(uint256 campaignId) external {
        Campaign storage campaign = campaigns[campaignId];

        require(block.timestamp >= campaign.deadline, "Not ended");
        require(!campaign.finalized, "Already finalized");

        campaign.finalized = true;
    }

    function faucet(uint256 amount) external {
        EduToken(token).mint(msg.sender, amount);
    }
}
```

3.2 ERC-20 Reward Token

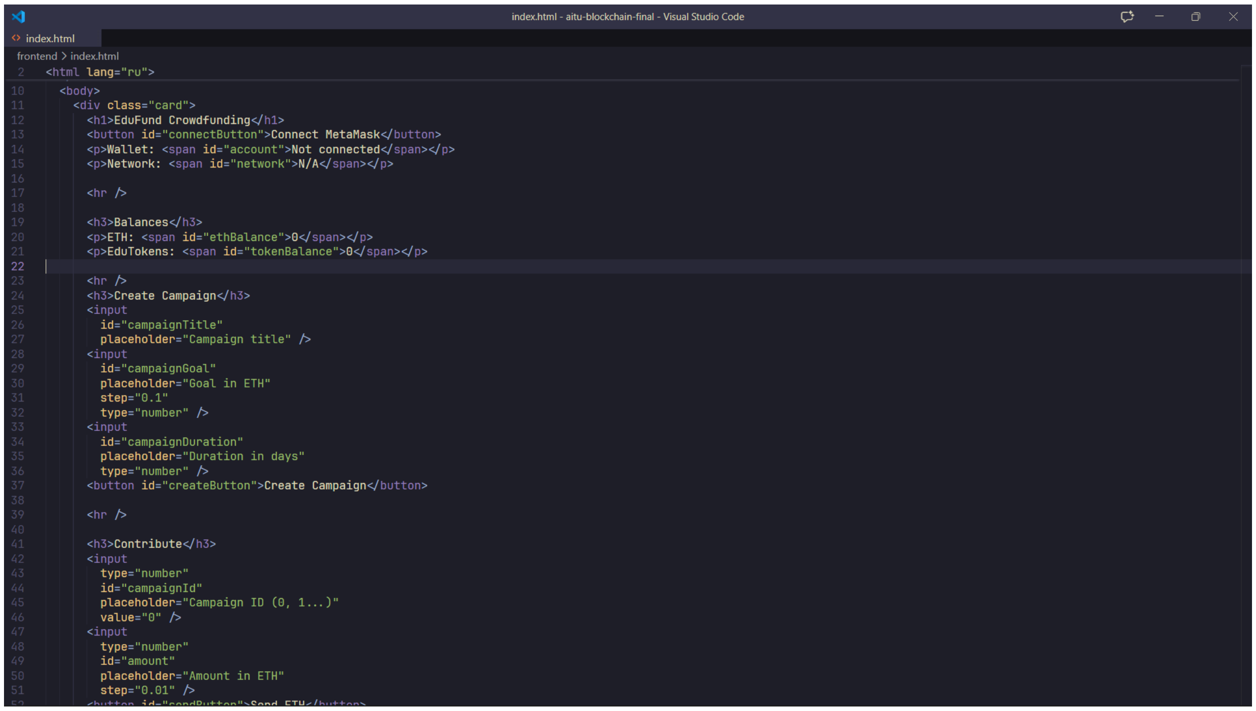
The project includes a custom ERC-20 token used as a reward mechanism. Token characteristics are based on OpenZeppelin ERC-20 standard. Tokens are minted automatically during participation. Importantly, they have no real monetary value and are used only for educational purposes. Tokens are distributed proportionally to the amount of ETH contributed.



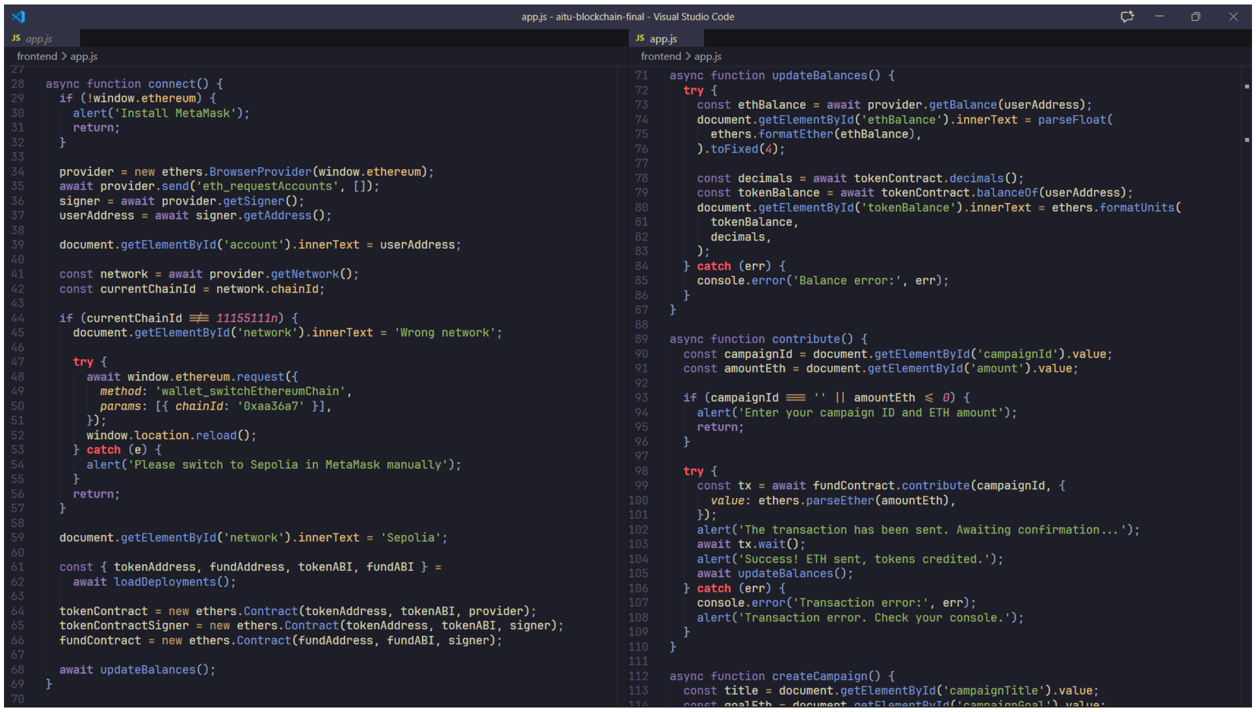
```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract EduToken is ERC20, Ownable {

    address public crowdfundingContract;

    constructor() ERC20("EduToken", "EDU") Ownable(msg.sender) {}


    modifier onlyCrowdfunding() {
        require(msg.sender == crowdfundingContract, "Not crowdfunding contract");
        _;
    }

    function setCrowdfundingContract(address _contract) external onlyOwner {
        crowdfundingContract = _contract;
    }

    function mint(address to, uint256 amount) external onlyCrowdfunding {
        _mint(to, amount);
    }
}

```

4. Frontend and Blockchain Interaction

The frontend is implemented using HTML, CSS, and JavaScript. Features include MetaMask wallet connection, wallet address display, network verification, campaign creation form, campaign contribution functionality, ETH balance display, ERC-20 token balance display, transaction status monitoring, and more. JavaScript uses the Ethereum provider injected by MetaMask to interact with deployed smart contracts.

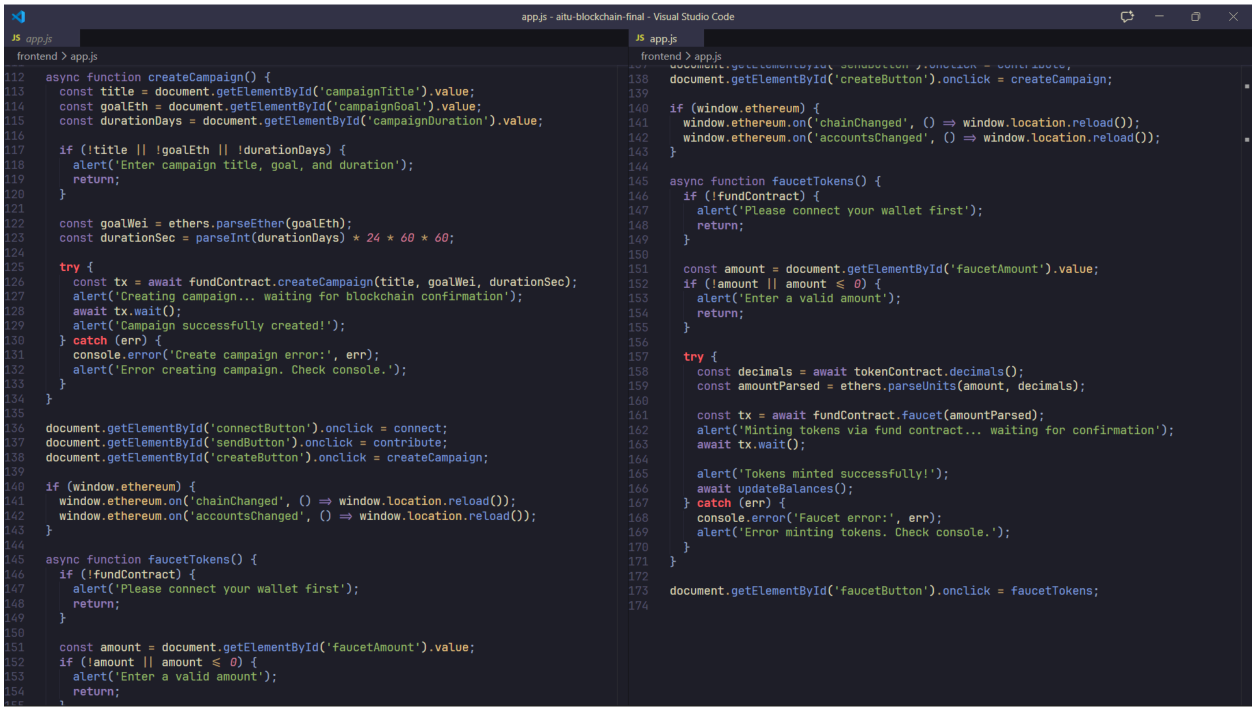


```html
<html lang="ru">
  <body>
    <div class="card">
      <h1>EduFund Crowdfunding</h1>
      <button id="connectButton">Connect MetaMask</button>
      <p>Wallet: <span id="account">Not connected</span></p>
      <p>Network: <span id="network">N/A</span></p>

      <hr />

      <h3>Balances</h3>
      <p>ETH: <span id="ethBalance">0</span></p>
      <p>EduTokens: <span id="tokenBalance">0</span></p>

      <hr />
      <h3>Create Campaign</h3>
      <input
        id="campaignTitle"
        placeholder="Campaign title" />
      <input
        id="campaignGoal"
        placeholder="Goal in ETH"
        step="0.1"
        type="number" />
      <input
        id="campaignDuration"
        placeholder="Duration in days"
        type="number" />
      <button id="createButton">Create Campaign</button>

      <hr />

      <h3>Contribute</h3>
      <input
        type="number"
        id="campaignId"
        placeholder="Campaign ID (0, 1...)"
        value="0" />
      <input
        type="number"
        id="amount"
        placeholder="Amount in ETH"
        step="0.01" />
      <button id="sendButton">Send ETH</button>
```



```javascript
async function connect() {
  if (!window.ethereum) {
    alert('Install MetaMask');
    return;
  }

  provider = new ethers.BrowserProvider(window.ethereum);
  await provider.send('eth_requestAccounts', []);
  signer = await provider.getSigner();
  userAddress = await signer.getAddress();

  document.getElementById('account').innerText = userAddress;

  const network = await provider.getNetwork();
  const currentChainId = network.chainId;

  if (currentChainId !== 11155111n) {
    document.getElementById('network').innerText = 'Wrong network';

    try {
      await window.ethereum.request({
        method: 'wallet_switchEthereumChain',
        params: [{ chainId: '0xaa36a7' }],
      });
      window.location.reload();
    } catch (e) {
      alert('Please switch to Sepolia in MetaMask manually');
    }
    return;
  }

  document.getElementById('network').innerText = 'Sepolia';

  const { tokenAddress, fundAddress, tokenABI, fundABI } =
    await loadDeployments();

  tokenContract = new ethers.Contract(tokenAddress, tokenABI, provider);
  tokenContractSigner = new ethers.Contract(tokenAddress, tokenABI, signer);
  fundContract = new ethers.Contract(fundAddress, fundABI, signer);

  await updateBalances();
}

```

```javascript
async function updateBalances() {
  try {
    const ethBalance = await provider.getBalance(userAddress);
    document.getElementById('ethBalance').innerText = parseFloat(
      ethers.formatEther(ethBalance),
    ).toFixed(4);

    const decimals = await tokenContract.decimals();
    const tokenBalance = await tokenContract.balanceOf(userAddress);
    document.getElementById('tokenBalance').innerText = ethers.formatUnits(
      tokenBalance,
      decimals,
    );
  } catch (err) {
    console.error('Balance error:', err);
  }
}

async function contribute() {
  const campaignId = document.getElementById('campaignId').value;
  const amountEth = document.getElementById('amount').value;

  if (campaignId === '' || amountEth <= 0) {
    alert('Enter your campaign ID and ETH amount');
    return;
  }

  try {
    const tx = await fundContract.contribute(campaignId, {
      value: ethers.parseEther(amountEth),
    });
    alert('The transaction has been sent. Awaiting confirmation...');
    await tx.wait();
    alert('Success! ETH sent, tokens credited.');
    await updateBalances();
  } catch (err) {
    console.error('Transaction error:', err);
    alert('Transaction error. Check your console.');
  }
}

async function createCampaign() {
  const title = document.getElementById('campaignTitle').value;
  const goalEth = document.getElementById('campaignGoal').value;
```



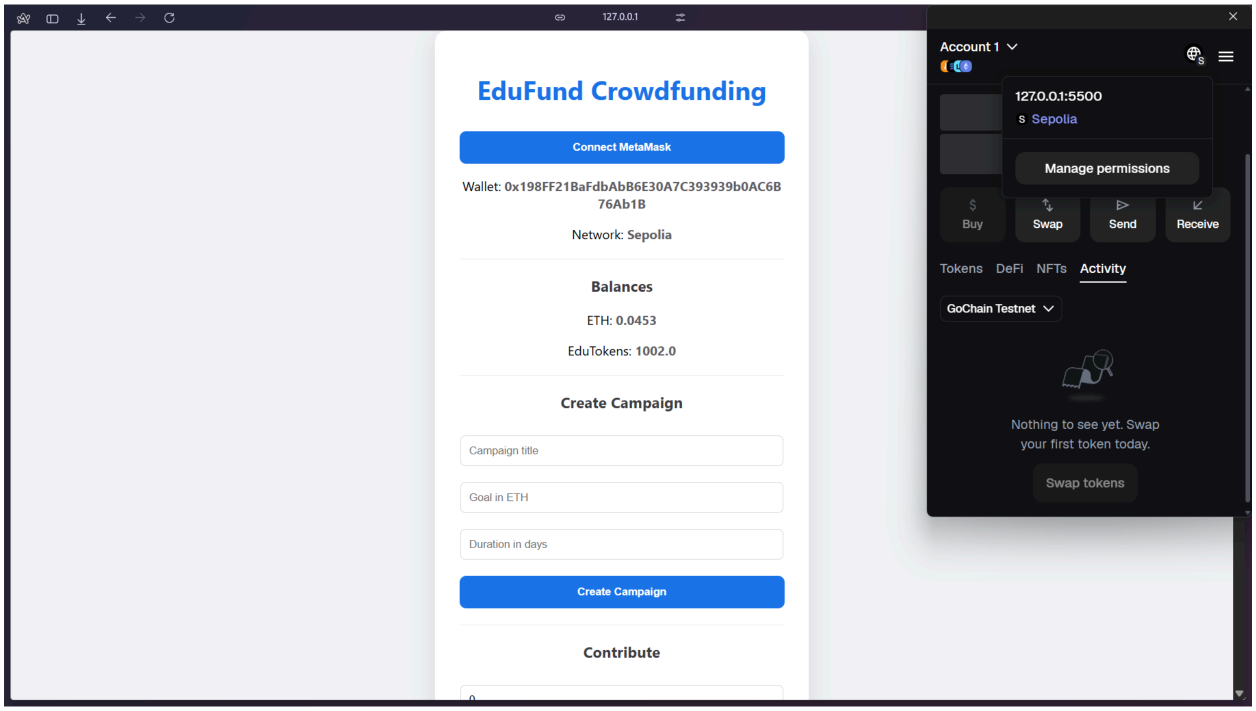
```
async function createCampaign() {
  const title = document.getElementById('campaignTitle').value;
  const goalEth = document.getElementById('campaignGoal').value;
  const durationDays = document.getElementById('campaignDuration').value;

  if (!title || !goalEth || !durationDays) {
    alert('Enter campaign title, goal, and duration');
    return;
  }

  const goalWei = ethers.parseEther(goalEth);
  const durationSec = parseInt(durationDays) * 24 * 60 * 60;

  try {
    const tx = await fundContract.createCampaign(title, goalWei, durationSec);
    alert('Creating campaign... waiting for blockchain confirmation');
    await tx.wait();
    alert('Campaign successfully created!');
  } catch (err) {
    console.error('Create campaign error:', err);
    alert('Error creating campaign. Check console.');
  }
}

document.getElementById('connectButton').onclick = connect;
document.getElementById('sendButton').onclick = contribute;
document.getElementById('createButton').onclick = createCampaign;

if (window.ethereum) {
  window.ethereum.on('chainChanged', () ⇒ window.location.reload());
  window.ethereum.on('accountsChanged', () ⇒ window.location.reload());
}

async function faucetTokens() {
  if (!fundContract) {
    alert('Please connect your wallet first');
    return;
  }

  const amount = document.getElementById('faucetAmount').value;
  if (!amount || amount ⩽ 0) {
    alert('Enter a valid amount');
    return;
```

```
document.getElementById('createButton').onclick = createCampaign;

if (window.ethereum) {
  window.ethereum.on('chainChanged', () ⇒ window.location.reload());
  window.ethereum.on('accountsChanged', () ⇒ window.location.reload());
}

async function faucetTokens() {
  if (!fundContract) {
    alert('Please connect your wallet first');
    return;
  }

  const amount = document.getElementById('faucetAmount').value;
  if (!amount || amount ⩽ 0) {
    alert('Enter a valid amount');
    return;
```

```
  }

  try {
    const decimals = await tokenContract.decimals();
    const amountParsed = ethers.parseUnits(amount, decimals);

    const tx = await fundContract.faucet(amountParsed);
    alert('Minting tokens via fund contract... waiting for confirmation');
    await tx.wait();

    alert('Tokens minted successfully!');
    await updateBalances();
  } catch (err) {
    console.error('Faucet error:', err);
    alert('Error minting tokens. Check console.');
  }
}

document.getElementById('faucetButton').onclick = faucetTokens;
```

5. MetaMask Integration

MetaMask integration is mandatory and fully implemented. The application requests permission to access user accounts, verifies the selected Ethereum test network, sends transactions through MetaMask, and requires user confirmation for each blockchain action. This ensures secure and transparent interaction with the blockchain.


127.0.0.1

EduFund Crowdfunding

Connect MetaMask

Wallet: 0x198FF21BaFdbAbB6E30A7C393939b0AC6B76Ab1B

Network: Sepolia

Balances

ETH: 0.0453

EduTokens: 1002.0

Create Campaign

Campaign title

Goal in ETH

Duration in days

Create Campaign

Contribute

Account 1

127.0.0.1:5500

Sepolia

Manage permissions

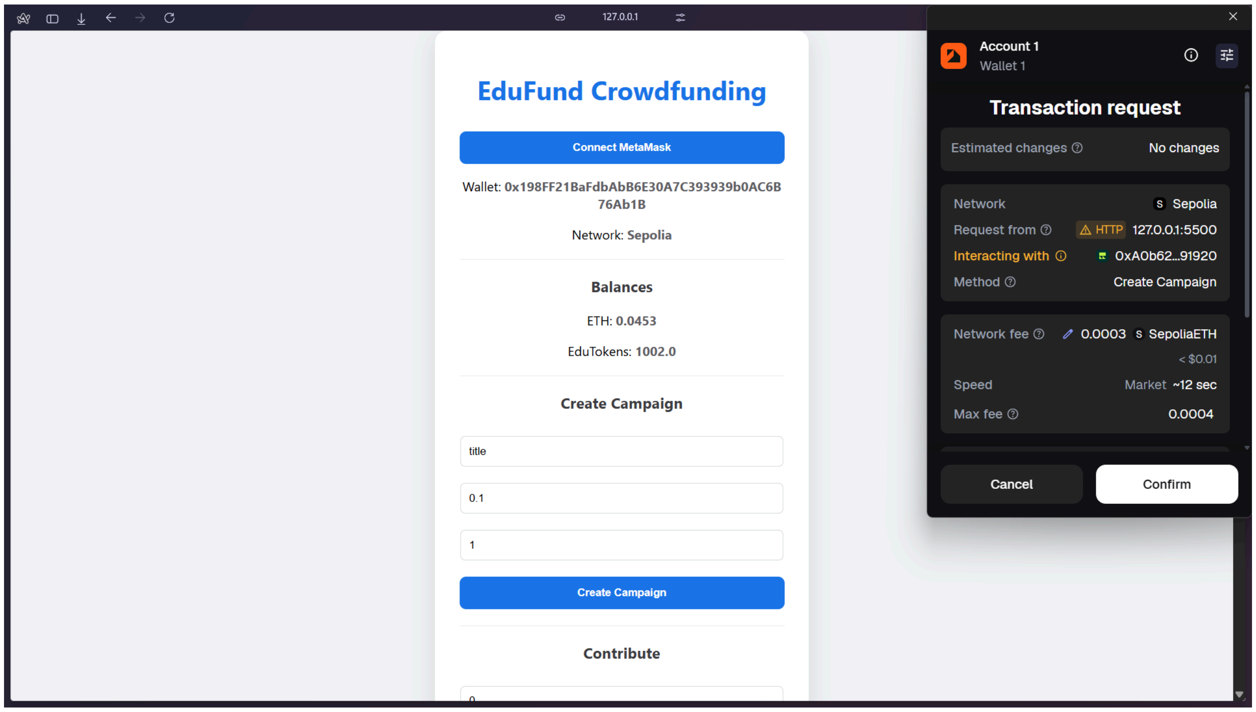
Buy Swap Send Receive

Tokens DeFi NFTs Activity

GoChain Testnet

Nothing to see yet. Swap your first token today.

Swap tokens


127.0.0.1

EduFund Crowdfunding

Connect MetaMask

Wallet: 0x198FF21BaFdbAbB6E30A7C393939b0AC6B76Ab1B

Network: Sepolia

Balances

ETH: 0.0453

EduTokens: 1002.0

Create Campaign

title

0.1

1

Create Campaign

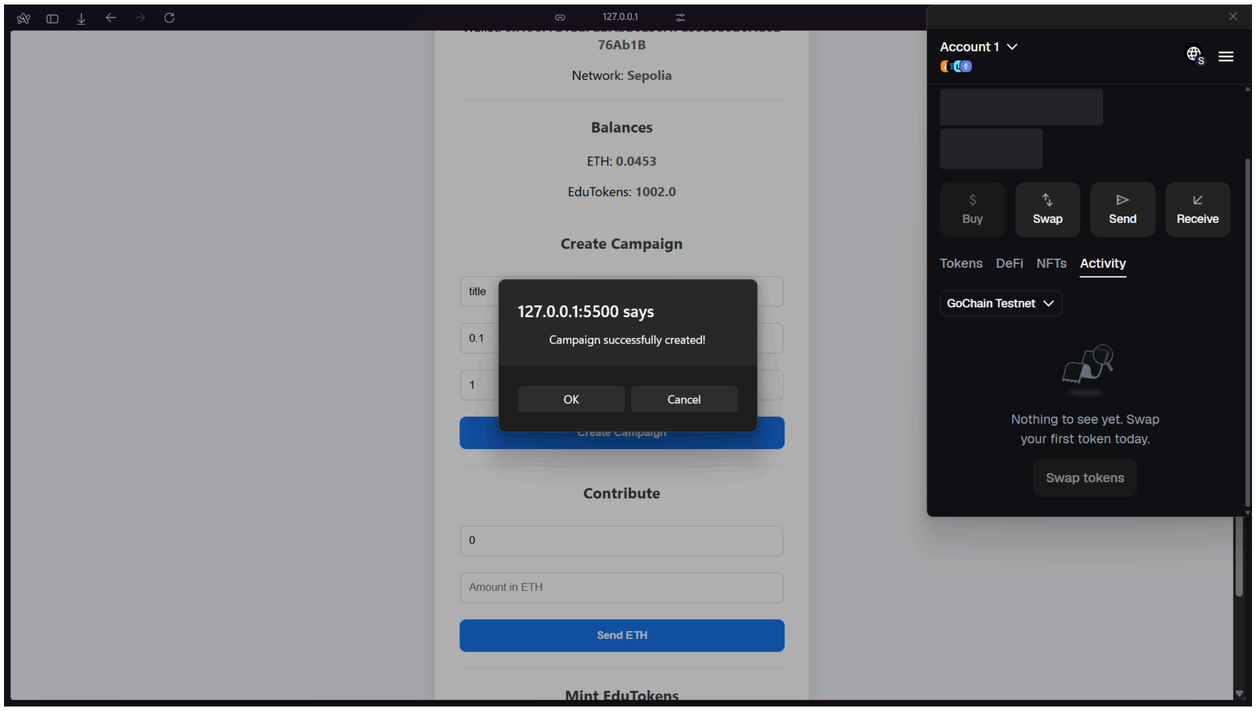
Contribute

Account 1
Wallet 1

Transaction request

Estimated changes	No changes
Network	Sepolia
Request from	HTTP 127.0.0.1:5500
Interacting with	0xA0b62...91920
Method	Create Campaign
Network fee	0.0003 SepoliaETH < $0.01
Speed	Market ~12 sec
Max fee	0.0004

Cancel Confirm



76Ab1B

Network: Sepolia

Balances

ETH: 0.0453

EduTokens: 1002.0

Create Campaign

title

0.1

1

127.0.0.1:5500 says

Campaign successfully created!

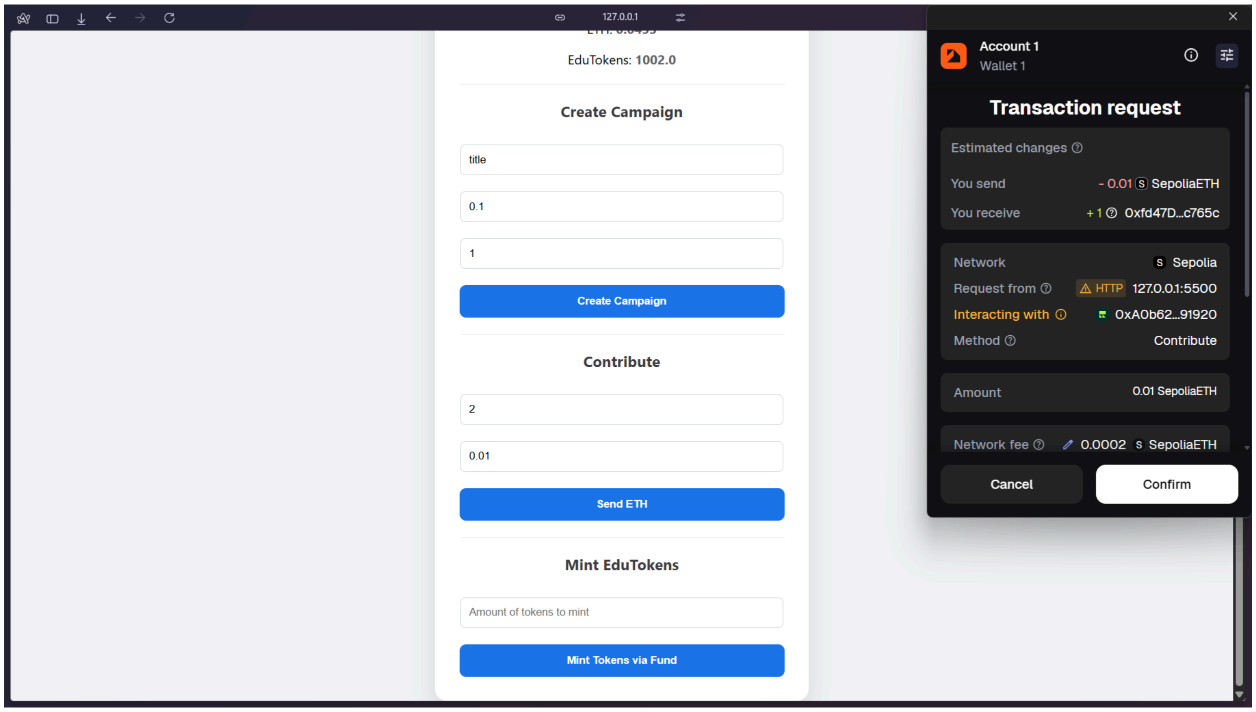
Contribute

0

Amount in ETH

Mint EduTokens

Nothing to see yet. Swap your first token today.



EduTokens: 1002.0

Create Campaign

title

0.1

1

Contribute

2

0.01

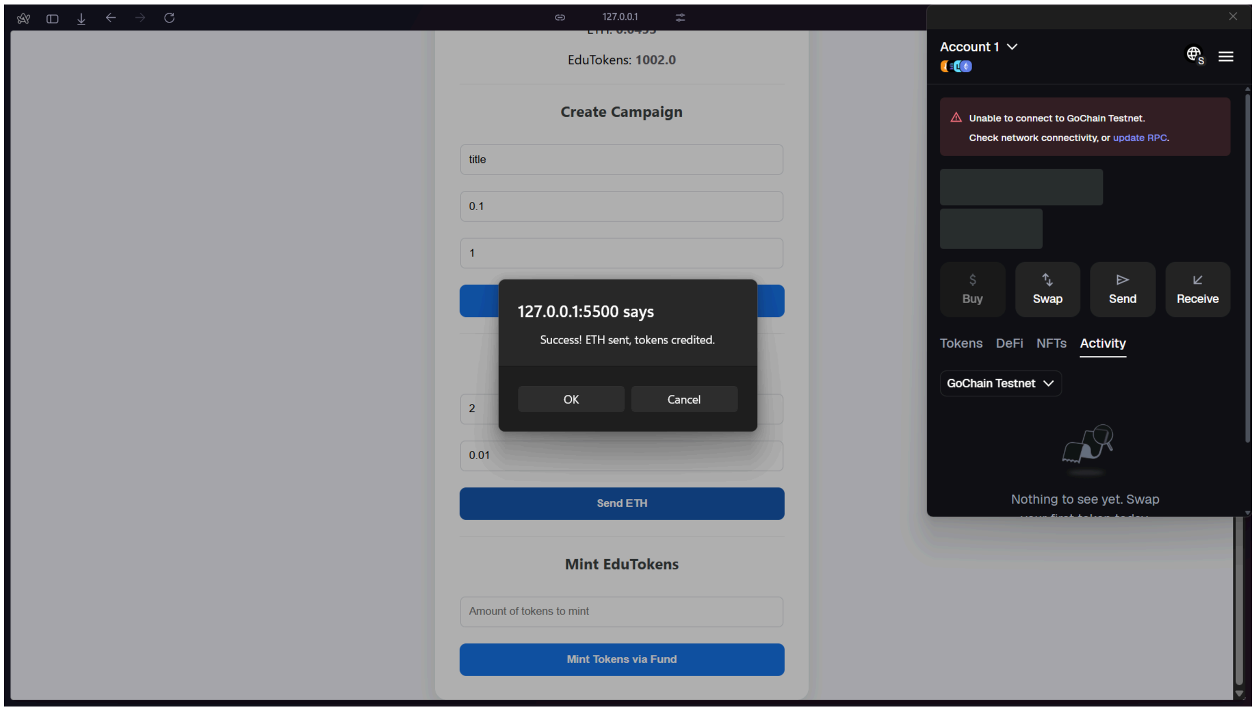
Mint EduTokens

Amount of tokens to mint

Transaction request

Estimated changes

You send	- 0.01 SepoliaETH
You receive	+ 1 0xfd47D...c765c
Network	Sepolia
Request from	HTTP 127.0.0.1:5500
Interacting with	0xA0b62...91920
Method	Contribute
Amount	0.01 SepoliaETH
Network fee	0.0002 SepoliaETH


EduTokens: 1002.0

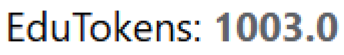
Create Campaign

127.0.0.1:5500 says

Success! ETH sent, tokens credited.

Unable to connect to GoChain Testnet.
Check network connectivity, or update RPC.

Nothing to see yet. Swap


EduTokens: 1003.0

6. Deployment Instructions

Prerequisites:

- Node.js
- MetaMask browser extension
- Ethereum test network account

Steps:

- Install dependencies:

  npm install

- Compile contracts:

  npx hardhat compile

- Deploy contracts to test network:

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

- Open live server for client:

  npx serve frontend/index.html

7. Obtaining Test ETH

Test ETH is required for interaction and can be obtained from official faucets (example link).

Users must connect MetaMask to a test network and request free test ETHs.