Subject: Blockchain Technologies 1
Students: Sadykov Danial, Kuan Akerke, Omar Aktoty
Teacher: Sayakulova Zarina

UrbanImprover - crowdfunding dapp for urban development

February 2026

Astana IT University

1. project introduction

the **UrbanImprover** project is a decentralized application (dapp) created to help people collect money for city projects like parks, playgrounds, or street lights. traditional crowdfunding has many fees and sometimes it is not transparent. our dapp uses blockchain technology to solve these problems. the main idea is that the money stays safe in a smart contract. the creator can only take the money (withdraw) if they reach the financial goal. this makes the system honest and safe for everyone who gives money.

2. technical architecture

the project uses three main parts to work:

- **solidity smart contract:** this is the "brain" of the project. it counts the money, saves the project details, and checks who is the creator.

- **hardhat networ**k: we use this to create a local blockchain on the computer. it is fast and good for testing transactions.
- f**rontend (web interface):** we used javascript and the ethers.js library. this allows the website to "talk" to the blockchain and show data to the user.

contracts: UrbanImprover.sol Contract

```
contracts > UrbanImprover.sol > UrbanImprover > withdrawFunds
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "./UrbanToken.sol";

contract UrbanImprover {
    struct Campaign {
        string title;
        uint256 goal;
        uint256 deadline;
        uint256 currentAmount;
        address creator;
        bool completed;
    }

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

    constructor(address _tokenAddress) {
        rewardToken = UrbanToken(_tokenAddress);
    }

    function createCampaign(string memory _title, uint256 _goal, uint256 _durationInDays) external {
        campaignCount++;
        campaigns[campaignCount] = Campaign({
            title: _title,
            goal: _goal,
            deadline: block.timestamp + (_durationInDays * 1 days),
            currentAmount: 0,
            creator: msg.sender,
            completed: false
        });
    }

```

```
36
37      function contribute(uint256 _campaignId) external payable {
38          Campaign storage campaign = campaigns[_campaignId];
39          require(block.timestamp < campaign.deadline, "Campaign ended");
40          require(msg.value > 0, "Contribution must be > 0");
41
42          campaign.currentAmount += msg.value;
43          contributions[_campaignId][msg.sender] += msg.value;
44
45          rewardToken.mint(msg.sender, msg.value);
46      }
47
48      function finalizeCampaign(uint256 _campaignId) external {
49          Campaign storage campaign = campaigns[_campaignId];
50          require(block.timestamp >= campaign.deadline, "Not ended yet");
51          require(!campaign.completed, "Already finalized");
52
53          campaign.completed = true;
54
55          if (campaign.currentAmount >= campaign.goal) {
56              payable(campaign.creator).transfer(campaign.currentAmount);
57          }
58      }
59
60      function withdrawFunds(uint256 id) external {
61          Campaign storage campaign = campaigns[id];
62          require(msg.sender == campaign.creator, "Only creator can withdraw");
63          require(campaign.currentAmount >= campaign.goal, "Goal not reached");
64          require(!campaign.completed, "Already withdrawn");
65
66          campaign.completed = true;
67          payable(campaign.creator).transfer(campaign.currentAmount);
68      }
69  }
```

token:

3. main features of the smart contract

the contract has several important functions:

- **createCampaign:** a user can start a new project with a name, a goal (how much money they need), and a deadline.
- **contribute:** any user can send **GO** tokens to a project. the contract updates the total amount automatically.
- **withdrawFunds:** this is a security feature. only the person who created the project can call this function, and only if the goal is reached.

4. implementation

4.1. contract deployment

first, we compiled the code and deployed it to the local network. the terminal shows the address of our contract.

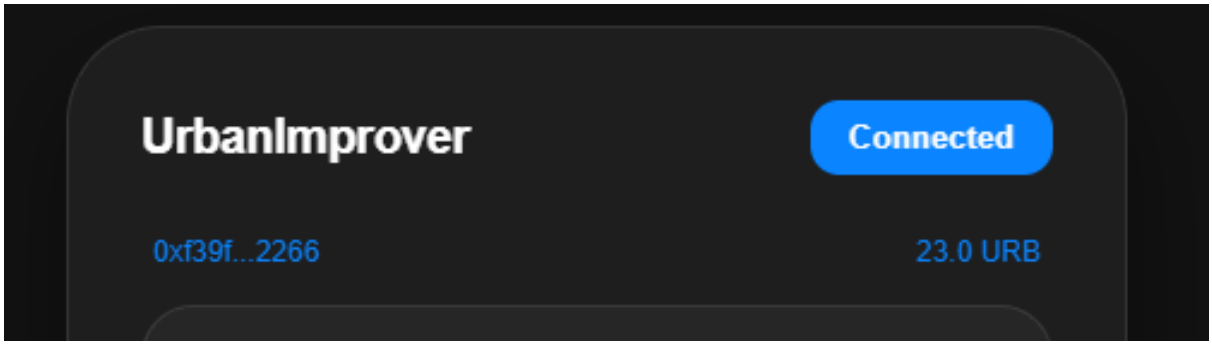
```
PS C:\Users\Admin\Documents\UrbanImprover> npx hardhat run scripts/deploy.js --network localhost
Compiled 1 Solidity file successfully (evm target: paris).
Deploying contracts with the account: 0xf39Fd6e51aad88F6F4ce6aB8827279cfffFb92266
UrbanToken deployed to: 0xa85233C63b9Ee964Add6F2cffe00Fd84eb32338f
UrbanImprover deployed to: 0x4A679253410272dd5232B3Ff7cF5dbB88f295319
Minter role given to UrbanImprover
```

scripts > deploy.js > ...

```
const hre = require("hardhat");

async function main() {
  const [deployer] = await hre.ethers.getSigners();
  console.log("Deploying contracts with the account:", deployer.address);

  const Token = await hre.ethers.getContractFactory("UrbanToken");
  const token = await Token.deploy();
  await token.waitForDeployment();
  const tokenAddress = await token.getAddress();
  console.log("UrbanToken deployed to:", tokenAddress);

  const Improver = await hre.ethers.getContractFactory("UrbanImprover");
  const improver = await Improver.deploy(tokenAddress);
  await improver.waitForDeployment();
  const improverAddress = await improver.getAddress();
  console.log("UrbanImprover deployed to:", improverAddress);

  const setMinterTx = await token.setMinter(improverAddress);
  await setMinterTx.wait();
  console.log("Minter role given to UrbanImprover");
}

main().catch((error) => {
  console.error(error);
  process.exitCode = 1;
});
```

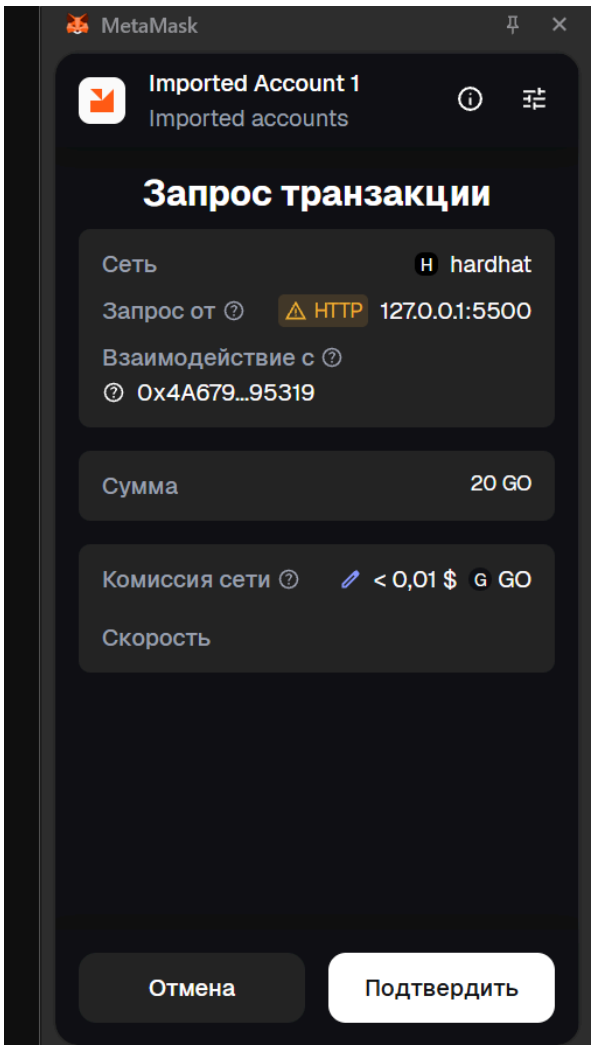
4.2. metamask connection

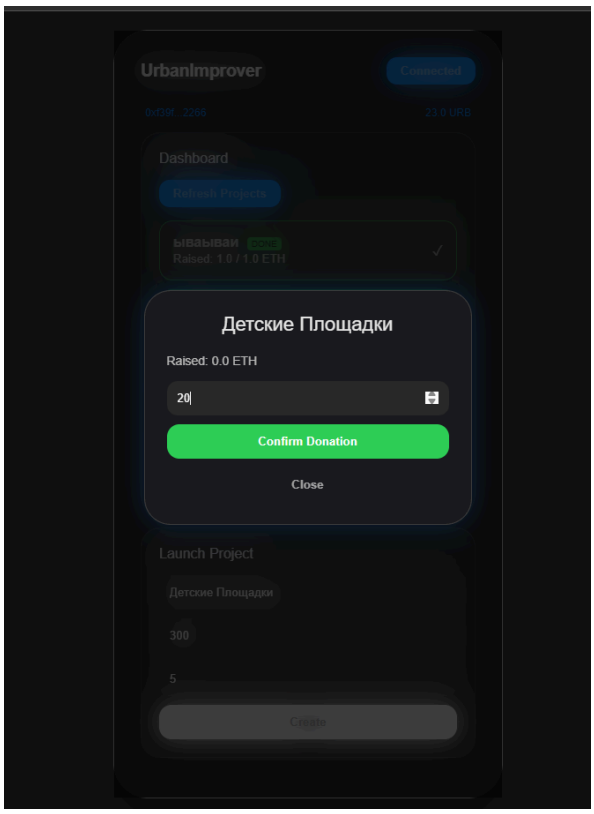
the dapp connects to the metamask wallet. we can see the user's address and their balance on the screen.



4.3. funding a project

when a user decides to help, they send tokens. the smart contract receives the tokens and stores them safely.





5. automated testing

to ensure the dapp is 100% secure, we wrote automated tests using the **mocha** framework and **chai** library. these tests check the logic of the smart contract without using the frontend.

5.1. test coverage

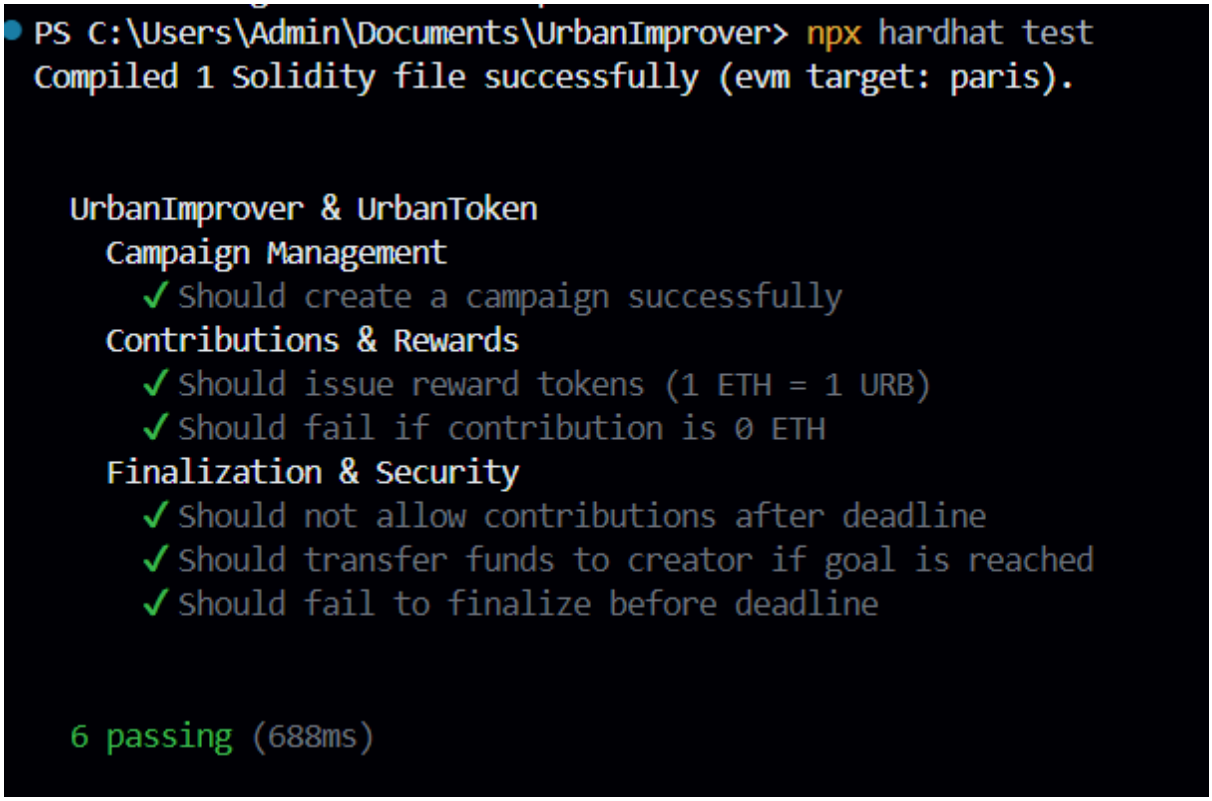
the test suite includes the following checks:

- **campaign management:** verifies that a new project (e.g., "test park") is saved correctly in the blockchain.
- **contributions & rewards:** checks that the system gives **1 URB token** for every **1 ETH** donated. it also blocks "zero" donations.
- **security & timing:** ensures that nobody can donate after the deadline and that the creator can only receive funds if the goal is reached.

- **finalization:** confirms that a campaign cannot be closed earlier than the official deadline.

5.2. running the tests

we use the command npx hardhat test to run all checks. all tests passed successfully, which means the smart contract logic is solid.

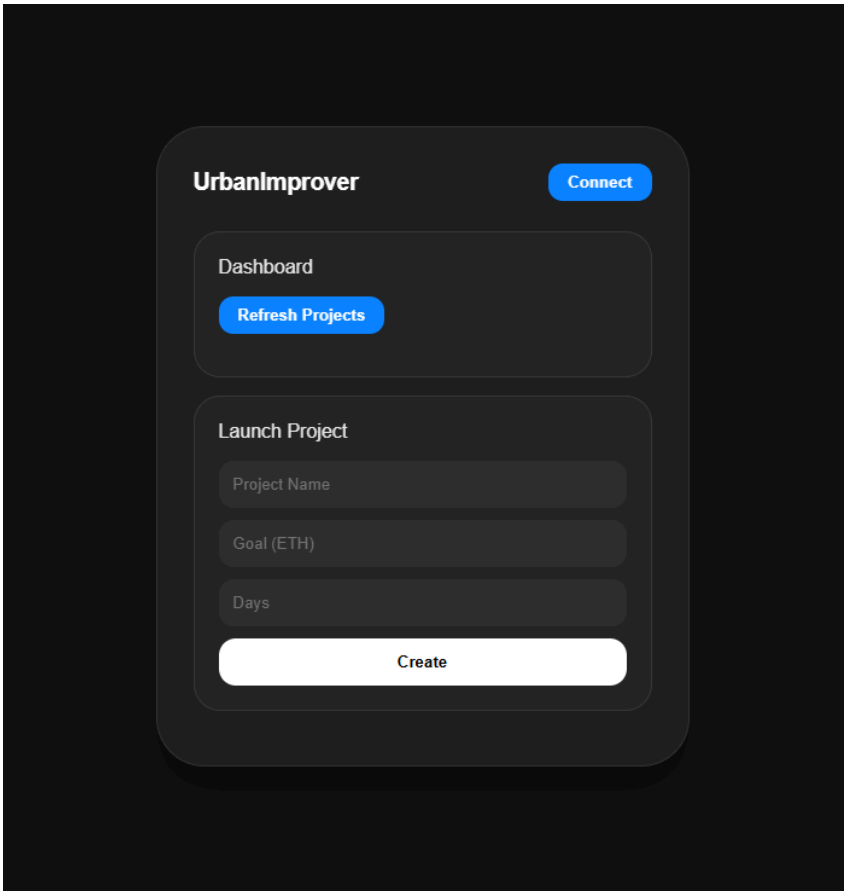

PS C:\Users\Admin\Documents\UrbanImprover> npx hardhat test
Compiled 1 Solidity file successfully (evm target: paris).
UrbanImprover & UrbanToken
Campaign Management
✓ Should create a campaign successfully
Contributions & Rewards
✓ Should issue reward tokens (1 ETH = 1 URB)
✓ Should fail if contribution is 0 ETH
Finalization & Security
✓ Should not allow contributions after deadline
✓ Should transfer funds to creator if goal is reached
✓ Should fail to finalize before deadline
6 passing (688ms)

```js
const { expect } = require("chai");
const { ethers } = require("hardhat");
const { time } = require("@nomicfoundation/hardhat-network-helpers");

describe("UrbanImprover & UrbanToken", function () {
  let token, crowdfunding, owner, addr1, addr2;

  beforeEach(async function () {
    [owner, addr1, addr2] = await ethers.getSigners();

    const UrbanToken = await ethers.getContractFactory("UrbanToken");
    token = await UrbanToken.deploy();

    const UrbanImprover = await ethers.getContractFactory("UrbanImprover");
    crowdfunding = await UrbanImprover.deploy(await token.getAddress());

    await token.setMinter(await crowdfunding.getAddress());
  });

  describe("Campaign Management", function () {
    it("Should create a campaign successfully", async function () {
      await crowdfunding.createCampaign("Test Park", ethers.parseEther("10"), 7);
      const campaign = await crowdfunding.campaigns(1);
      expect(campaign.title).to.equal("Test Park");
    });
  });

  describe("Contributions & Rewards", function () {
    it("Should issue reward tokens (1 ETH = 1 URB)", async function () {
      await crowdfunding.createCampaign("Test Park", ethers.parseEther("10"), 7);
      await crowdfunding.connect(addr1).contribute(1, { value: ethers.parseEther("1") });
      const balance = await token.balanceOf(addr1.address);
      expect(balance).to.equal(ethers.parseEther("1"));
    });

    it("Should fail if contribution is 0 ETH", async function () {
      await crowdfunding.createCampaign("Zero Test", ethers.parseEther("10"), 7);
      await expect(
        crowdfunding.connect(addr1).contribute(1, { value: 0 })
      ).to.be.revertedWith("Contribution must be > 0");
    });
  });
```

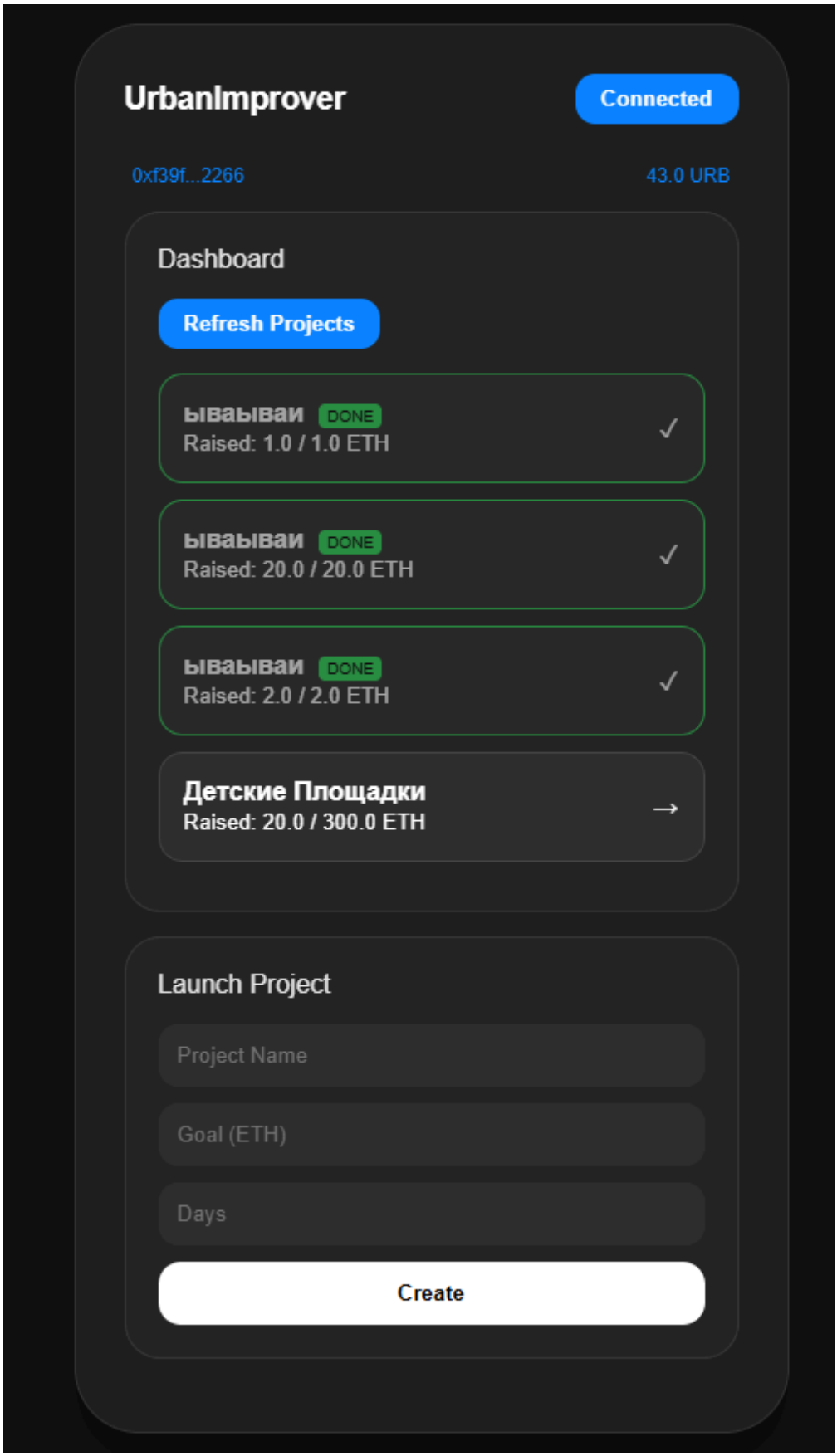
```
  describe("Finalization & Security", function () {
    it("Should not allow ibutions after deadline", async function () {
      await crowdfunding.createCampaign("Expired Test", ethers.parseEther("10"), 1);
      await time.increase(2 * 24 * 60 * 60);
      await expect(
        crowdfunding.connect(addr1).contribute(1, { value: ethers.parseEther("1") })
      ).to.be.revertedWith("Campaign ended");
    });

    it("Should transfer funds to creator if goal is reached", async function () {
      const goal = ethers.parseEther("2");
      await crowdfunding.connect(addr1).createCampaign("Fund Test", goal, 1);
      await crowdfunding.connect(addr2).contribute(1, { value: goal });
      await time.increase(2 * 24 * 60 * 60);
      const initialBalance = await ethers.provider.getBalance(addr1.address);
      await crowdfunding.finalizeCampaign(1);
      const finalBalance = await ethers.provider.getBalance(addr1.address);
      expect(finalBalance).to.be.gt(initialBalance);
    });

    it("Should fail to finalize before deadline", async function () {
        await crowdfunding.createCampaign("Early Test", ethers.parseEther("10"), 1);
        await expect(
            crowdfunding.finalizeCampaign(1)
        ).to.be.revertedWith("Not ended yet");
    });
  });
});
```

6. frontend

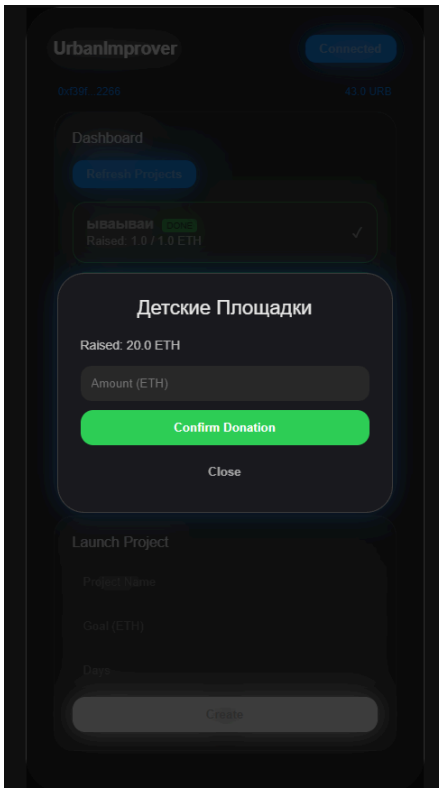
main page:



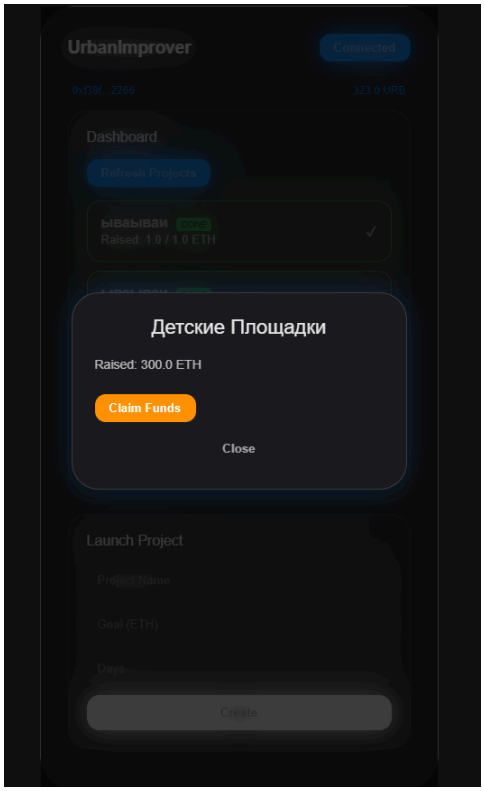
page with connected:



donate page:



claim funds (only creator can):



```html
<head>
    <link href="https://fonts.googleapis.com/css2?family=SF+Pro+Display:wght@300;400;600&display=swap" rel="stylesheet">
</head>
<body>
    <div class="glass-container">
        <header>
            <div class="logo">Urban<span>Improver</span></div>
            <button id="connect-wallet" class="ios-btn-primary">Connect</button>
        </header>

        <div id="status-bar" class="hidden">
            <span id="user-address"></span>
            <span id="user-balance">0 URB</span>
        </div>

        <main>
            <div class="ios-card">
                <h3>Dashboard</h3>
                <button onclick="loadCampaigns()" class="ios-btn-primary">Refresh Projects</button>
                <div id="campaign-list" style="margin-top: 15px;"></div>
            </div>

            <div id="donate-modal" class="hidden">
                <div class="ios-card modal-content">
                    <h3 id="modal-title">Project</h3>
                    <p id="modal-info" style="font-size: 14px; opacity: 0.7; margin-bottom: 15px;"></p>

                    <div id="donate-inputs">
                        <input type="number" id="contribution-amount" placeholder="Amount (ETH)">
                        <button onclick="contributeToCampaign()" class="ios-btn-reward">Confirm Donation</button>
                    </div>

                    <button id="claim-btn" onclick="withdrawFunds()" class="ios-btn-primary hidden" style="margin-top: 10px; background: #FF9500;">Claim Funds</button>

                    <button onclick="closeModal()" class="ios-btn" style="margin-top: 10px; background: none; color: white; opacity: 0.6;">Close</button>
                </div>
            </div>

            <div class="ios-card">
                <h3>Launch Project</h3>
                <input type="text" id="cam-title" placeholder="Project Name">
                <input type="number" id="cam-goal" placeholder="Goal (ETH)">
                <input type="number" id="cam-duration" placeholder="Days">
                <button onclick="createNewCampaign()" class="ios-btn">Create</button>
            </div>
        </main>
    </div>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/ethers/5.7.2/ethers.umd.min.js"></script>
    <script src="app.js"></script>
</body>
</html>
```

public > app.js > withdrawFunds

```javascript
const IMPROVER_ADDRESS = "0x4A679253410272dd5232B3Ff7cF5dbB88f295319";
const TOKEN_ADDRESS = "0xa85233C63b9Ee964Add6F2cffe00Fd84eb32338f";

const IMPROVER_ABI = [
    "function createCampaign(string title, uint256 goal, uint256 duration) external",
    "function contribute(uint256 id) external payable",
    "function withdrawFunds(uint256 id) external",
    "function campaignCount() public view returns (uint256)",
    "function campaigns(uint256) public view returns (string title, uint256 goal, uint256 deadline, uint256 currentAmount, address creator, bool completed)"
];

const TOKEN_ABI = [
    "function balanceOf(address) public view returns (uint256)"
];

let signer;
let improverContract;
let tokenContract;
let selectedCampaignId = null;

async function connect() {
    if (!window.ethereum) return alert("Install MetaMask");
    try {
        const accounts = await window.ethereum.request({ method: 'eth_requestAccounts' });
        const provider = new ethers.providers.Web3Provider(window.ethereum);
        signer = provider.getSigner();
        const address = accounts[0];

        improverContract = new ethers.Contract(IMPROVER_ADDRESS, IMPROVER_ABI, signer);
        tokenContract = new ethers.Contract(TOKEN_ADDRESS, TOKEN_ABI, signer);

        document.getElementById('connect-wallet').innerText = "Connected";
        document.getElementById('status-bar').classList.remove('hidden');
        document.getElementById('user-address').innerText = address.slice(0,6) + '...' + address.slice(-4);

        await updateBalance(address);
        await loadCampaigns();
    } catch (err) {
        console.error(err);
    }
}

async function updateBalance(address) {
    try {
        const balance = await tokenContract.balanceOf(address);
        document.getElementById('user-balance').innerText = `${ethers.utils.formatEther(balance)} URB`;
    } catch (e) { console.error(e); }
}
```

```
async function updateBalance(address) {
    try {
        const balance = await tokenContract.balanceOf(address);
        document.getElementById('user-balance').innerText = `${ethers.utils.formatEther(balance)} URB`;
    } catch (e) { console.error(e); }
}

async function loadCampaigns() {
    const listElement = document.getElementById('campaign-list');
    if (!improverContract) return;

    listElement.innerHTML = "Updating...";

    try {
        const countBN = await improverContract.campaignCount();
        const count = countBN.toNumber();

        listElement.innerHTML = "";
        for (let i = 1; i <= count; i++) {
            const cam = await improverContract.campaigns(i);
            const isGoalReached = cam.currentAmount.gte(cam.goal);

            const div = document.createElement('div');
            div.className = `campaign-item ${isGoalReached || cam.completed ? 'completed' : ''}`;
            div.innerHTML = `
                <div style="display:flex; justify-content:space-between; align-items:center;">
                    <div>
                        <strong>${cam.title}</strong>
                        ${isGoalReached ? '<span class="status-tag" style="background:#30D158; color:black; margin-left:5px; border-radius:4px; padding:2px 5px; font-size:10px;">DONE</span>' : ''}
                        <br>
                        <small>Raised: ${ethers.utils.formatEther(cam.currentAmount)} / ${ethers.utils.formatEther(cam.goal)} ETH</small>
                    </div>
                    <span>${isGoalReached ? '✓' : '+'}</span>
                </div>
            `;
            div.onclick = () => openDonateModal(i, cam);
            listElement.appendChild(div);
        }
    } catch (e) {
        listElement.innerHTML = "Error";
    }
}
```

```
async function openDonateModal(id, cam) {
    selectedCampaignId = id;
    const userAddress = await signer.getAddress();
    const isCreator = cam.creator.toLowerCase() === userAddress.toLowerCase();
    const isGoalReached = cam.currentAmount.gte(cam.goal);

    document.getElementById('modal-title').innerText = cam.title;
    document.getElementById('modal-info').innerText = `Raised: ${ethers.utils.formatEther(cam.currentAmount)} ETH`;

    const donateInputs = document.getElementById('donate-inputs');
    const claimBtn = document.getElementById('claim-btn');

    if (isGoalReached) {
        donateInputs.classList.add('hidden');
        if (isCreator && !cam.completed) {
            claimBtn.classList.remove('hidden');
        } else {
            claimBtn.classList.add('hidden');
        }
    } else {
        donateInputs.classList.remove('hidden');
        claimBtn.classList.add('hidden');
    }
    document.getElementById('donate-modal').classList.remove('hidden');
}

function closeModal() {
    document.getElementById('donate-modal').classList.add('hidden');
}

async function createNewCampaign() {
    try {
        const title = document.getElementById('cam-title').value;
        const goal = ethers.utils.parseEther(document.getElementById('cam-goal').value);
        const duration = document.getElementById('cam-duration').value;
        const tx = await improverContract.createCampaign(title, goal, duration);
        await tx.wait();
        loadCampaigns();
    } catch (e) { alert("Error"); }
}

async function contributeToCampaign() {
    try {
        const amount = document.getElementById('contribution-amount').value;
        const tx = await improverContract.contribute(selectedCampaignId, {
            value: ethers.utils.parseEther(amount)
        });
        await tx.wait();
        closeModal();
        loadCampaigns();
        updateBalance(await signer.getAddress());
    } catch (e) { alert("Failed"); }
}
```

```
async function withdrawFunds() {
    try {
        const currentAddress = await signer.getAddress();
        const balanceBefore = await signer.getBalance();
        console.log("Balance BEFORE:", ethers.utils.formatEther(balanceBefore));

        const tx = await improverContract.withdrawFunds(selectedCampaignId);
        await tx.wait();

        const balanceAfter = await signer.getBalance();
        console.log("Balance AFTER:", ethers.utils.formatEther(balanceAfter));

        const diff = balanceAfter.sub(balanceBefore);
        alert("Actual change: " + ethers.utils.formatEther(diff) + " GO");

        loadCampaigns();
    } catch (e) {
        console.error("Error details:", e);
    }
}

document.getElementById('connect-wallet').onclick = connect;

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

7. reward system and future utility

the **urbanimprover** dapp includes a special reward system to motivate donors. instead of just giving money, every contributor becomes a part of the ecosystem through **urban tokens (urb)**.

- **current reward logic:** for every **1 eth** (or go token) donated to a project, the smart contract automatically mints and sends **1 urb token** to the donor's wallet. this process is instant and verified by our automated tests.

- **future "governance" & voting:** in the next version, users with more **urb tokens** will have "voting power". they can help choose which city projects are the most important.
- **government privileges:** we plan to connect the dapp with city services. people who have many **urb tokens** could get real-life privileges from the government. for example:
  - lower city taxes for active citizens.
  - priority access to public events or parks. this makes the **urb**

token a bridge between the digital blockchain and real city life.

8. conclusion

the **UrbanImprover dapp** project successfully shows how blockchain can help city development. during the work, we created a secure smart contract, a reward system with **urb tokens**, and a clear web interface.

this dapp is a great example of a transparent crowdfunding system. it is more honest than traditional websites because the code, not a person, controls the money. in the future, this project can be launched on a real testnet like **sepolia or Hoodi or Haskely or Mainnet** to allow more people to participate in improving their cities.