



FINAL EXAMINATION PROJECT

Duisenbek Bekzat Nur-Adilet Mustafa Aknur Ondasyn

SE-2438

link:<https://github.com/duisenbek/trustlayer>

Project Goal

This project is all about creating a crowdfunding app that's decentralized using blockchain tech. The main idea is to fix the trust issues you see in normal crowdfunding. Right now, creators get all the cash at once, and backers don't really have a say after they donate. TrustLayer makes sure the money is handed out bit by bit, only when the backers give the thumbs up.

Project Idea

TrustLayer is a crowdfunding system where money is released as goals are reached.

Here's how it works:

- Money is kept safe using smart contracts.
- The creator doesn't get all the money upfront.
- Funds are only released when the community votes yes after a milestone is achieved.
- This way, you don't have to just blindly trust the creator; everything is based on blockchain rules instead.

System Architecture

The project consists of:

- Smart contracts written in Solidity,
- Frontend written in HTML, CSS, and JavaScript,
- MetaMask for wallet connection,
- Ethereum test network (Hardhat / Sepolia).

User → MetaMask → Frontend → Smart Contracts → Blockchain

Frontend

Smart Contracts

TrustLayer.sol

This is a factory contract. It creates new Campaign contracts and stores their addresses. Each campaign is deployed as a separate smart contract.

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

import "./Campaign.sol";
import "./RewardToken.sol";

contract TrustLayer {
    // Array of all deployed campaigns
    Campaign[] public deployedCampaigns;
    RewardToken public token;

    // Event emitted when a new campaign is created
    event CampaignCreated(address campaignAddress, address creator, string title, uint256 goal);

    constructor() {
        token = new RewardToken();
    }

    /**
     * @dev Creates a new Campaign contract.
     * @param _title Title of the project.
     * @param _description Brief description.
     * @param _goal Funding goal in wei.
     * @param _duration Duration in seconds.
     */
    function createCampaign(
        string memory _title,
        string memory _description,
        uint256 _goal,
        uint256 _duration
    ) public {
        Campaign newCampaign = new Campaign(
            msg.sender,
            _title,
            _description,
            _goal,
            _duration,
            address(this) // Pass Factory address instead of Token address
        );
        deployedCampaigns.push(newCampaign);
        isCampaign[address(newCampaign)] = true;

        emit CampaignCreated(address(newCampaign), msg.sender, _title, _goal);
    }

    mapping(address => bool) public isCampaign;

    function mintReward(address to, uint256 amount) external {
        require(isCampaign[msg.sender], "Only campaigns can mint");
        token.mint(to, amount);
    }

    /**
     * @dev Returns the entire list of deployed campaigns.
     */
    function getDeployedCampaigns() public view returns (Campaign[] memory) {
        return deployedCampaigns;
    }
}
```

Campaign.sol

This contract contains the main crowdfunding logic. Funds stay inside the contract until voting conditions are met.

```

2     pragma solidity ^0.8.20;
3
4     import "./RewardToken.sol";
5
6     interface ITrustLayer {
7         function mintReward(address to, uint256 amount) external;
8     }
9
10    contract Campaign {
11        struct Milestone {
12            string description;
13            uint256 amount;
14            bool isApproved;
15            bool isReleased;
16            uint256 votingDeadline; // When voting ends for this milestone
17        }
18
19        address public creator;
20        string public title;
21        string public description;
22        uint256 public goal;
23        uint256 public deadline;
24        uint256 public totalRaised;
25        bool public isFunded;
26
27        Milestone[] public milestones;
28        mapping(address => uint256) public contributions;
29        // Mapping from milestone ID -> voter address -> has voted
30        mapping(uint256 => mapping(address => bool)) public approvals;
31        // Mapping from milestone ID -> total approval weight (in wei)
32        mapping(uint256 => uint256) public approvalWeights;
33
34        address public factory;
35
36        event Contributed(address contributor, uint256 amount);
37        event MilestoneCreated(uint256 id, string description, uint256 amount);
38        event VoteCast(address voter, uint256 milestoneId, bool approve);
39        event FundsReleased(uint256 milestoneId, uint256 amount);
40
41        modifier onlyCreator() {
42            require(msg.sender == creator, "Only creator can call this");
43            _;
44        }
45
46        modifier onlyContributor() {
47            require(contributions[msg.sender] > 0, "Only contributors can call this");
48            _;
49        }
50
51        constructor(
52            address _creator,
53            string memory _title,
54            string memory _description,
55            uint256 _goal,
56            uint256 _duration,
57            address _factory
58        ) {
59            creator = _creator;
60            title = _title;
61            description = _description;
62            goal = _goal;
63            deadline = block.timestamp + _duration;
64            factory = _factory;
65        }
66

```

```

71     function contribute() public payable {
72         require(block.timestamp < deadline, "Campaign has ended");
73         contributions[msg.sender] += msg.value;
74         totalRaised += msg.value;
75
76         if (totalRaised >= goal) {
77             isFunded = true;
78         }
79
80         // Mint 1000 tokens per 1 ETH via Factory
81         // amount * 1000
82         ITrustLayer(factory).mintReward(msg.sender, msg.value * 1000);
83
84         emit Contributed(msg.sender, msg.value);
85     }
86
87 /**
88 * @dev Creator adds a milestone.
89 */
90 function createMilestone(string memory _desc, uint256 _amount) public onlyCreator {
91     require(isFunded, "Campaign must be met goal first");
92     require(address(this).balance >= _amount, "Insufficient balance");
93
94     Milestone memory newMilestone = Milestone({
95         description: _desc,
96         amount: _amount,
97         isApproved: false,
98         isReleased: false,
99         votingDeadline: 0 // Set when voting starts? Or simplified to open voting.
100    });
101
102     milestones.push(newMilestone);
103     emit MilestoneCreated(milestones.length - 1, _desc, _amount);
104 }
105
106 /**
107 * @dev Contributors vote to approve a milestone.
108 * Voting power is proportional to contribution.
109 */
110 function vote(uint256 _milestoneId) public onlyContributor {
111     Milestone storage milestone = milestones[_milestoneId];
112     require(!milestone.isReleased, "Milestone already released");
113     require(!approvals[_milestoneId][msg.sender], "Already voted");
114
115     approvals[_milestoneId][msg.sender] = true;
116     approvalWeights[_milestoneId] += contributions[msg.sender];
117
118     // Check if > 50% of total raised funds have approved
119     if (approvalWeights[_milestoneId] > totalRaised / 2) {
120         milestone.isApproved = true;
121     }
122
123     emit VoteCast(msg.sender, _milestoneId, true);
124 }
125
126 /**
127 * @dev Creator withdraws funds for an approved milestone.
128 */
129 function withdraw(uint256 _milestoneId) public onlyCreator {
130     Milestone storage milestone = milestones[_milestoneId];
131     require(milestone.isApproved, "Milestone not approved");
132     require(!milestone.isReleased, "Already released");
133
134     milestone.isReleased = true;
135     payable(creator).transfer(milestone.amount);
136
137     ...
138 }

```

RewardToken.sol

This is a custom ERC-20 token.

The token:is minted automatically when users contribute,has no real value,is used only for learning purposes.

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.20;
3
4 import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
5 import "@openzeppelin/contracts/access/Ownable.sol";
6
7 contract RewardToken is ERC20, Ownable {
8     constructor() ERC20("TrustLayer Token", "TSL") Ownable(msg.sender) {}
9
10    function mint(address to, uint256 amount) public onlyOwner {
11        _mint(to, amount);
12    }
13}
```

Application Workflow

Step 1: Create Campaign

A user creates a campaign by setting a goal and duration.

The screenshot shows a web browser window for 'TrustLayer | Milestone-Based Cr'. The address bar shows the URL '192.168.0.152:8080'. The page content includes a message: 'Funds are locked in smart contracts and released only when the community approves project milestones.' Below this are two buttons: 'EXPLORE PROJECTS' and 'START A CAMPAIGN'. A section titled 'FEATURED CAMPAIGNS' displays a card for a campaign titled '11' with a progress bar showing '0.0 ETH raised' and 'Goal: 11.0 ETH'. A 'VIEW PROJECT' button is at the bottom of the card. At the bottom of the page is a footer with the text '© 2026 TrustLayer. Powered by Ethereum.' To the right of the browser window is a separate screenshot of a Google Chrome extension window titled 'Подтверждена транзакция' (Transaction confirmed) with the message 'Транзакция 0 подтверждена!' (Transaction 0 confirmed!).

Step 2: Contribute ETH

Users contribute test ETH using MetaMask. ETH is locked in the smart contract.

The screenshot shows the TrustLayer web application interface. At the top, there's a navigation bar with links for HOME, PROJECTS, and DASHBOARD, and a session ID 8XC94D...B6EC. Below the navigation is a button labeled 'BACK TO PROJECTS'. The main content area displays a project summary: '11' (Contract: 0XB7A5...D968). To the right, a summary box shows 'Total Pledged 11.0 ETH' (Goal: 11.0 ETH, 100% Funded), 'Escrow Balance 11.0 ETH' (Funds held in smart contract until milestones are approved), '1 Backers', and '11 Days Left'. A large button at the bottom right says 'CONTRIBUTE ETH'. On the left, a modal window titled 'CREATE NEW MILESTONE (UNLOCK FUNDS)' contains fields for 'Description (e.g., "Prototypes")' and 'Amount (ETH)', with a 'CREATE MILESTONE REQUEST' button at the bottom.

Step 3: Create Milestone

The campaign creator requests a milestone to unlock part of the funds.

This section shows three sequential screenshots of the TrustLayer interface with MetaMask integrated. In the first screenshot, a transaction dialog in MetaMask shows the creation of a 'Запрос транзакции' (Transaction Request) for a milestone. It details the amount (11.0 ETH), goal (11.0 ETH, 100%), escrow balance (11.0 ETH), fees (0.0002 ETH, 0.45 \$), and gas limit (2). The second screenshot shows the transaction being signed in MetaMask, with the status 'Подтверждено' (Confirmed). The third screenshot shows the transaction confirmed in the TrustLayer interface, with the status 'Подтверждено' (Confirmed) and a green checkmark icon.

Step 4: Voting

Contributors vote to approve the milestone.
Voting power depends on the amount contributed.

TrustLayer | Milestone-Based Crowdfunding

CONTRACT: 0xb7a5...d988

11

MILESTONES (GOVERNANCE)

CREATE NEW MILESTONE (UNLOCK FUNDS)

To withdraw funds, create a milestone. Backers must vote to approve it.

Description (e.g., 'Prototypes') Amount (ETH)

CREATE MILESTONE REQUEST

1. TO PROTOTYPE APP VOTING ETH Requested: 5.0 Approval: 0.0 / 5.5 ETH 0.0% needed VOTE YES

Total Pledged 11.0 ETH Goal: 11.0 ETH 100% Funded

Escrow Balance 11.0 ETH Funds are held in smart contract until milestones are approved.

1 Backers 11 Days Left

Amount (ETH) CONTRIBUTE ETH

© 2026 TrustLayer. Powered by Ethereum.

Google Chrome Подтвержденная транзакция Транзакция 2 подтверждена!

11.0 ETH

Goal: 11.0 ETH 100% Funded

Escrow Balance 11.0 ETH

Funds are held in smart contract until milestones are approved.

1 Backers 11 Days Left

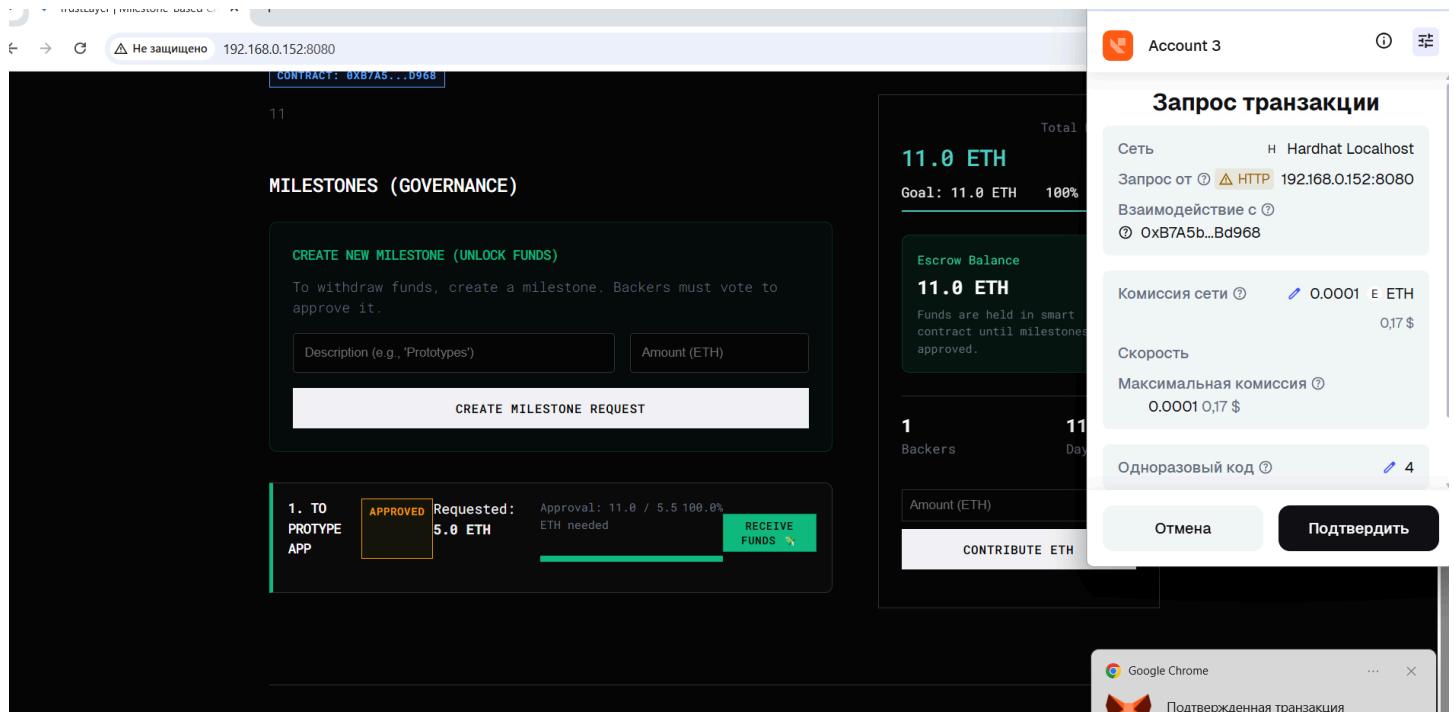
Amount (ETH) CONTRIBUTE ETH

Step 5: Release Funds

When more than 50% approve, funds are released automatically by the smart contract.

1. TO PROTOTYPE APP APPROVED Requested: 5.0 ETH Approval: 11.0 / 5.5 100.0% ETH needed RECEIVE FUNDS

RECEIVE FUNDS

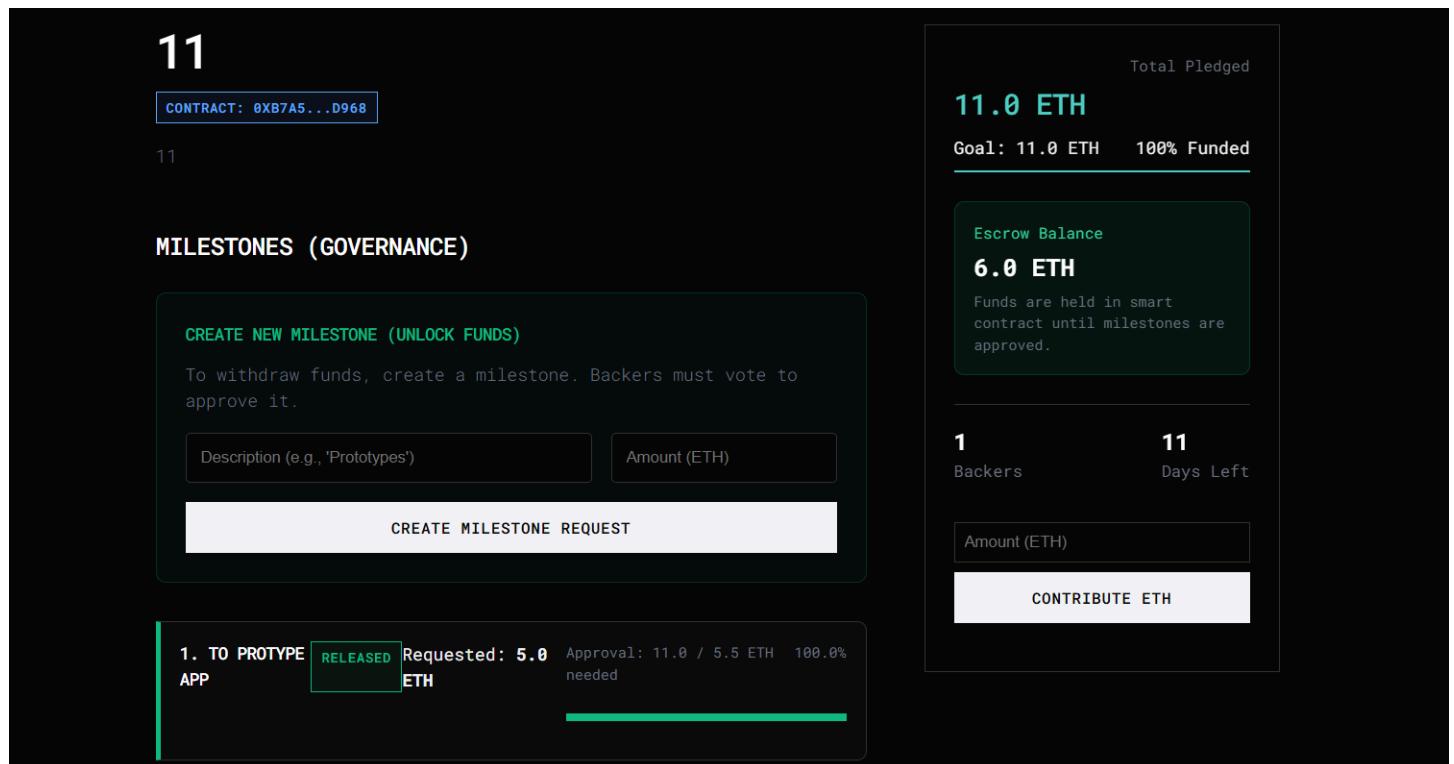


6. MetaMask Integration

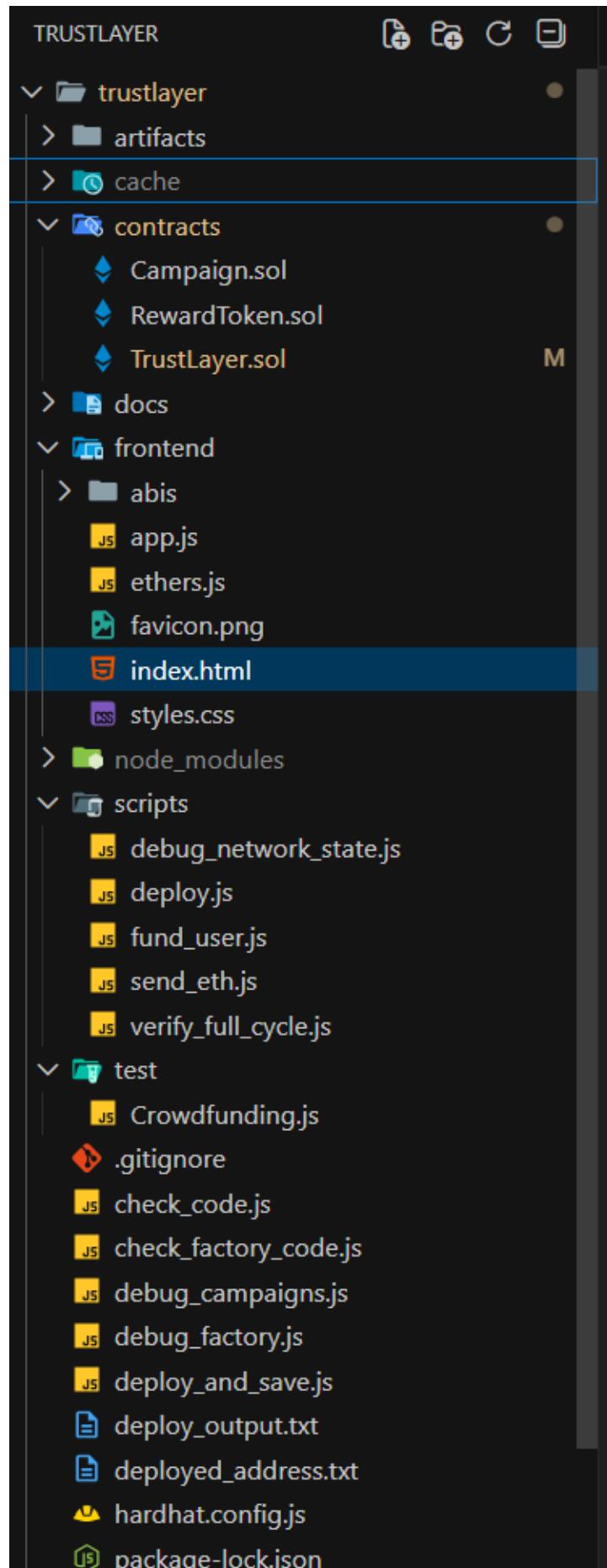
MetaMask is used to:

- connect wallets,
- sign transactions,
- show gas fees and confirmations.

All transactions are executed on Ethereum test networks only.



Project Structure



Conclusion

TrustLayer demonstrates how blockchain can be used to control funds without relying on trust. The project uses smart contracts, voting, and MetaMask integration to create a secure and transparent crowdfunding system. All requirements of the Blockchain 1 course are fully satisfied.