FINAL EXAMINATION PROJECT

Duisenbek Bekzat Nur-Adilet Mustafa Aknur Ondasyn

SE-2438

link:https://github.com/duesenbek/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.

```
pragma solidity ^0.8.20;

import "./RewardToken.sol";

interface ITrustLayer {
    function mintReward(address to, uint256 amount) external;
}

contract Campaign {
    struct Milestone {
        string description;
        uint256 amount;
        bool isApproved;
        bool isReleased;
        uint256 votingDeadline; // When voting ends for this milestone
    }

    address public creator;
    string public title;
    string public description;
    uint256 public goal;
    uint256 public deadline;
    uint256 public totalRaised;
    bool public isFunded;

    Milestone[] public milestones;
    mapping(address => uint256) public contributions;
    // Mapping from milestone ID -> voter address -> has voted
    mapping(uint256 => mapping(address => bool)) public approvals;
    // Mapping from milestone ID -> total approval weight (in wei)
    mapping(uint256 => uint256) public approvalWeights;

    address public factory;

    event Contributed(address contributor, uint256 amount);
    event MilestoneCreated(uint256 id, string description, uint256 amount);
    event VoteCast(address voter, uint256 milestoneId, bool approve);
    event FundsReleased(uint256 milestoneId, uint256 amount);

    modifier onlyCreator() {
        require(msg.sender == creator, "Only creator can call this");
        _;
    }

    modifier onlyContributor() {
        require(contributions[msg.sender] > 0, "Only contributors can call this");
        _;
    }

    constructor(
        address _creator,
        string memory _title,
        string memory _description,
        uint256 _goal,
        uint256 _duration,
        address _factory
    ) {
        creator = _creator;
        title = _title;
        description = _description;
        goal = _goal;
        deadline = block.timestamp + _duration;
        factory = _factory;
    }

```

```
    function contribute() public payable {
        require(block.timestamp < deadline, "Campaign has ended");
        contributions[msg.sender] += msg.value;
        totalRaised += msg.value;

        if (totalRaised >= goal) {
            isFunded = true;
        }

        // Mint 1000 tokens per 1 ETH via Factory
        // amount * 1000
        ITrustLayer(factory).mintReward(msg.sender, msg.value * 1000);

        emit Contributed(msg.sender, msg.value);
    }

    /**
     * @dev Creator adds a milestone.
     */
    function createMilestone(string memory _desc, uint256 _amount) public onlyCreator {
        require(isFunded, "Campaign must be met goal first");
        require(address(this).balance >= _amount, "Insufficient balance");

        Milestone memory newMilestone = Milestone({
            description: _desc,
            amount: _amount,
            isApproved: false,
            isReleased: false,
            votingDeadline: 0 // Set when voting starts? Or simplified to open voting.
        });

        milestones.push(newMilestone);
        emit MilestoneCreated(milestones.length - 1, _desc, _amount);
    }

    /**
     * @dev Contributors vote to approve a milestone.
     * Voting power is proportional to contribution.
     */
    function vote(uint256 _milestoneId) public onlyContributor {
        Milestone storage milestone = milestones[_milestoneId];
        require(!milestone.isReleased, "Milestone already released");
        require(!approvals[_milestoneId][msg.sender], "Already voted");

        approvals[_milestoneId][msg.sender] = true;
        approvalWeights[_milestoneId] += contributions[msg.sender];

        // Check if > 50% of total raised funds have approved
        if (approvalWeights[_milestoneId] > totalRaised / 2) {
            milestone.isApproved = true;
        }

        emit VoteCast(msg.sender, _milestoneId, true);
    }

    /**
     * @dev Creator withdraws funds for an approved milestone.
     */
    function withdraw(uint256 _milestoneId) public onlyCreator {
        Milestone storage milestone = milestones[_milestoneId];
        require(milestone.isApproved, "Milestone not approved");
        require(!milestone.isReleased, "Already released");

        milestone.isReleased = true;
        payable(creator).transfer(milestone.amount);

```

RewardToken.sol

This is a custom ERC-20 token.

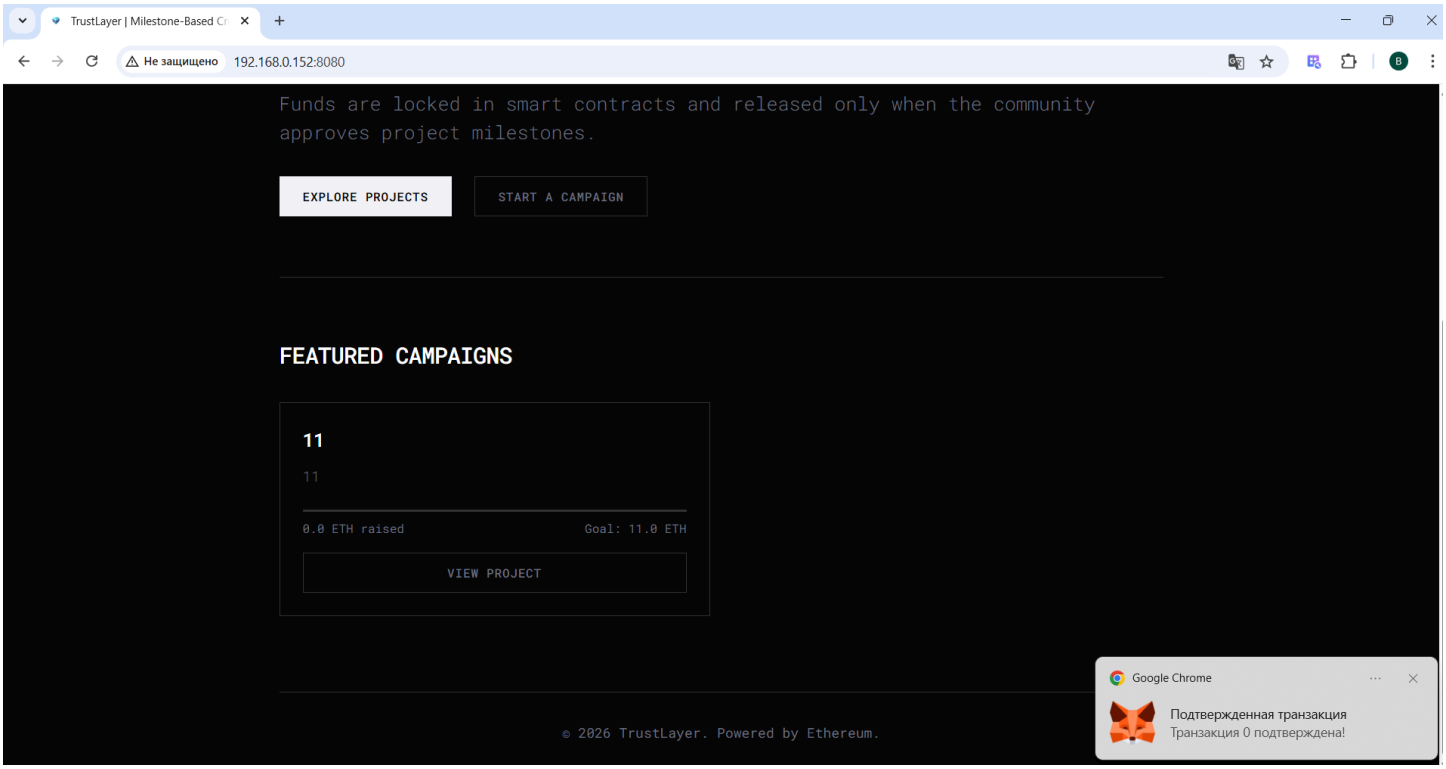
The token:is minted automatically when users contribute,has no real value,is used only for learning purposes.

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

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

contract RewardToken is ERC20, Ownable {
    constructor() ERC20("TrustLayer Token", "TSL") Ownable(msg.sender) {}

    function mint(address to, uint256 amount) public onlyOwner {
        _mint(to, amount);
    }
}
```

Application Workflow

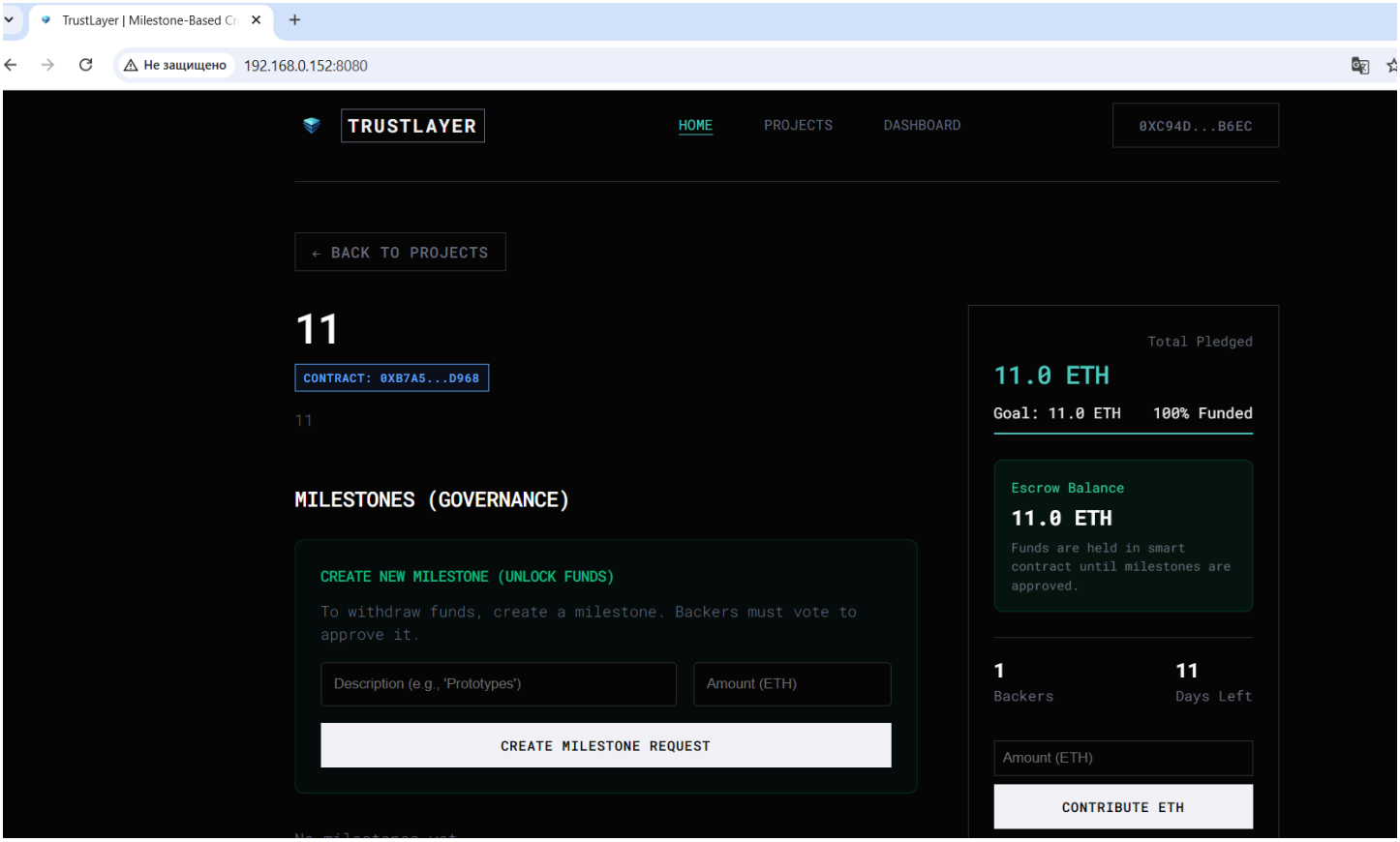
Step 1: Create Campaign

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



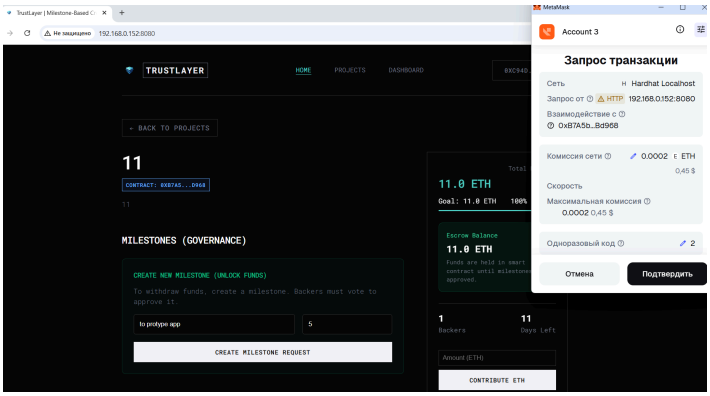
Step 2: Contribute ETH

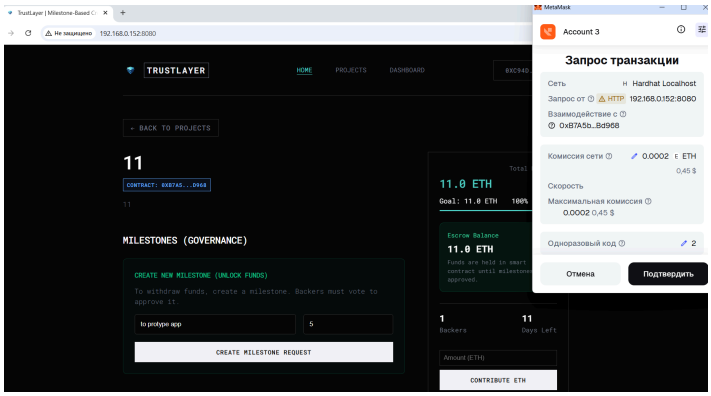
Users contribute test ETH using MetaMask.ETH is locked in the smart contract.



Step 3: Create Milestone

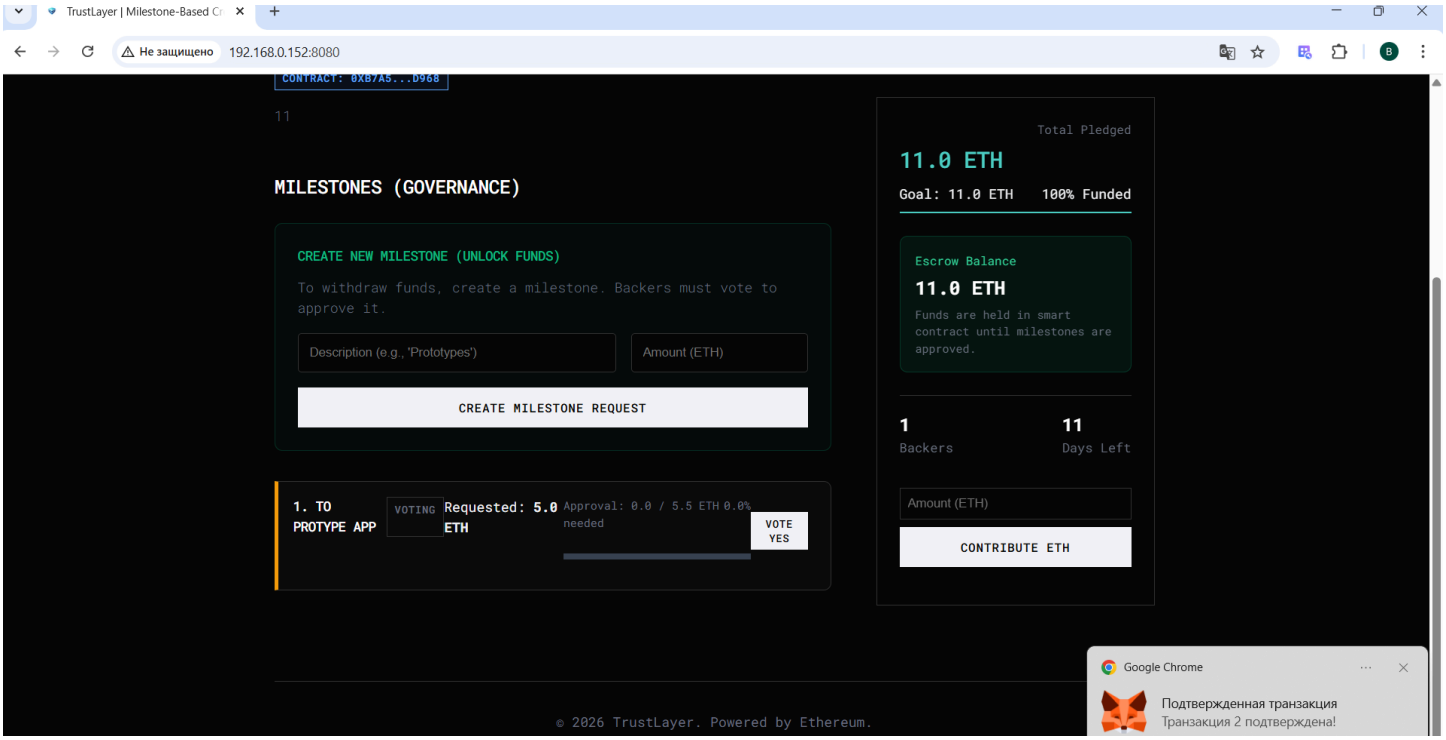
The campaign creator requests a milestone to unlock part of the funds.





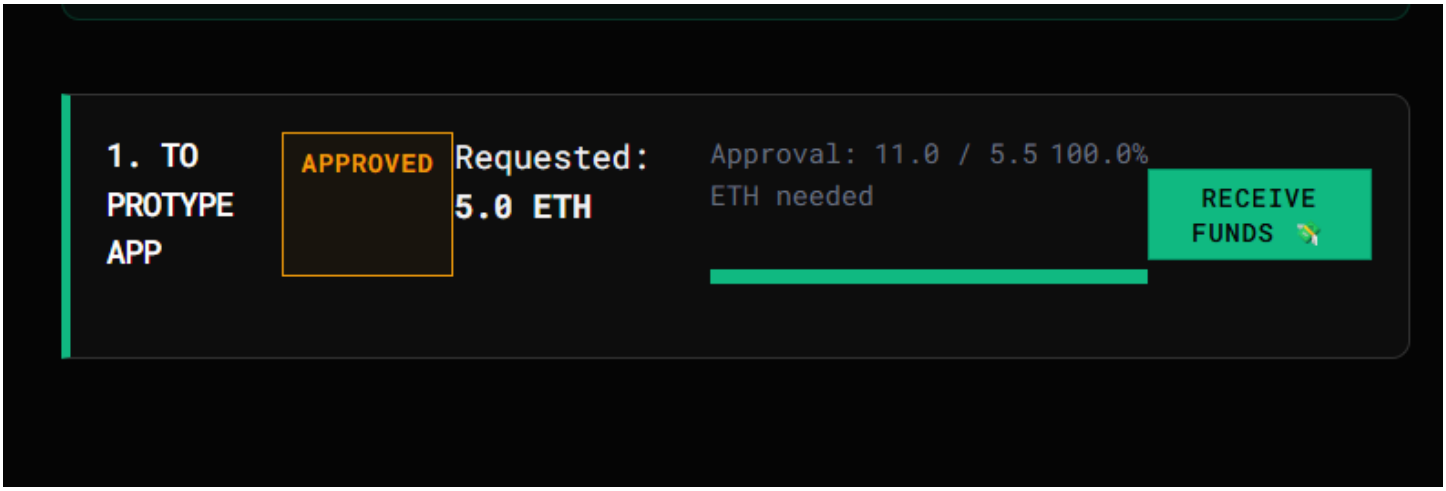
Step 4: Voting

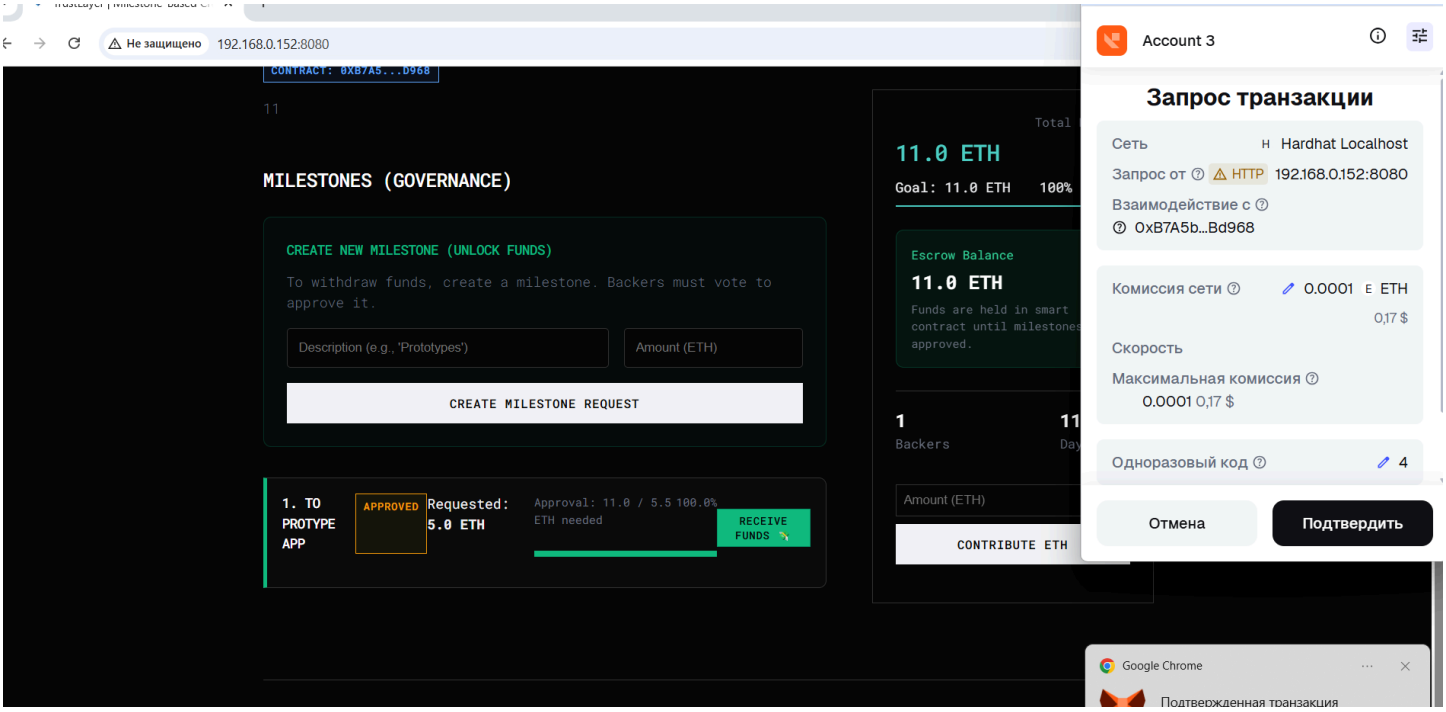
Contributors vote to approve the milestone.
 Voting power depends on the amount contributed.



Step 5: Release Funds

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



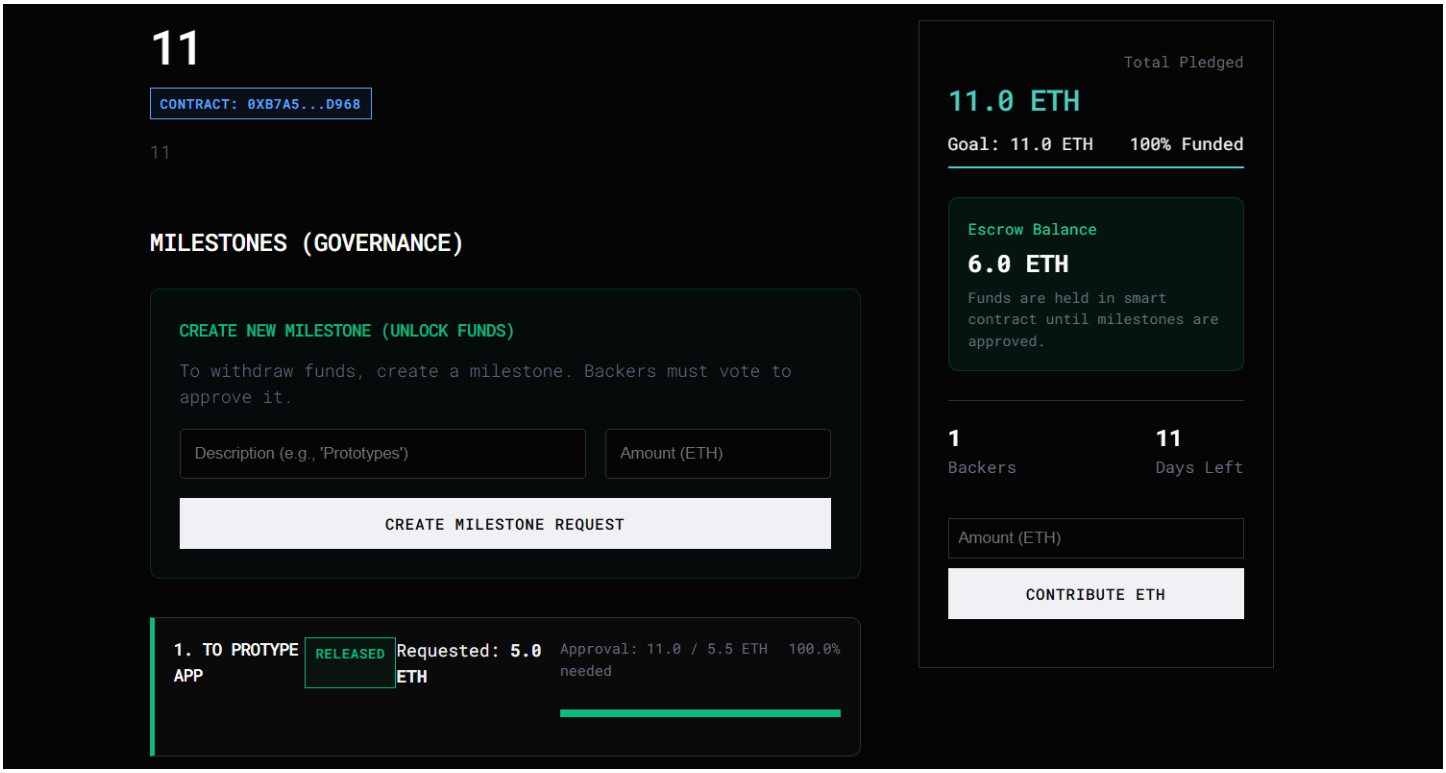


6. MetaMask Integration

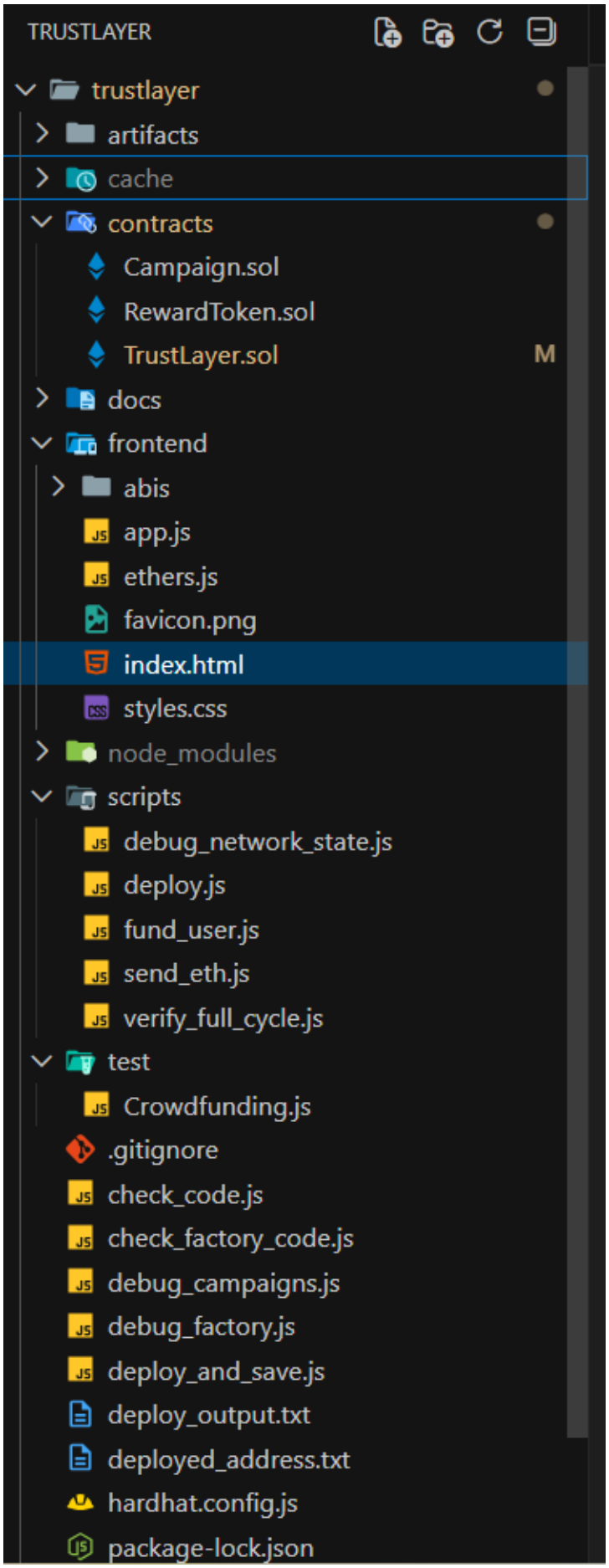
MetaMask is used to:

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

All transactions are executed on Ethereum test networks only.



Project Structure


TRUSTLAYER
trustlayer
artifacts
cache
contracts
Campaign.sol
RewardToken.sol
TrustLayer.sol
M
docs
frontend
abis
app.js
ethers.js
favicon.png
index.html
styles.css
node_modules
scripts
debug_network_state.js
deploy.js
fund_user.js
send_eth.js
verify_full_cycle.js
test
Crowdfunding.js
.gitignore
check_code.js
check_factory_code.js
debug_campaigns.js
debug_factory.js
deploy_and_save.js
deploy_output.txt
deployed_address.txt
hardhat.config.js
package-lock.json

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.