FINAL EXAMINATION PROJECT
Decentralized Academic Research Funding Platform

Course Name: Blockchain Technologies 1

Course Teacher: Zarina Sayakulova

Group: SE-2428

Team: Assem Rakhmanova, Aisana Kuanyshbek, Nurassyl Nurdilda

1. Project Purpose

The purpose of this project is to design and implement a decentralized crowdfunding application operating on an Ethereum test network.

The application demonstrates:

- Smart contract development using Solidity
- ERC-20 token implementation
- Frontend–blockchain interaction using JavaScript
- Integration with MetaMask wallet
- Real blockchain transaction execution

The project operates exclusively on a local Ethereum test network (Hardhat) and uses only test ETH.

2. Project Overview

Decentralized Academic Research Funding Platform

This platform allows:

- A user to create a research project
- Other users to contribute test ETH
- Contributors to receive internal ERC-20 reward tokens
- The project to receive a final status after the deadline (Successful or Failed)

That is the full system logic.

What the Platform Supports

The system allows:

- Creating a crowdfunding project
- Sending test ETH to a project
- Storing funds inside the smart contract
- Minting ERC-20 tokens as a reward
- Finalizing a project after deadline

- Displaying project status
- Interaction through MetaMask

This fully satisfies the functional requirements of the assignment

3. System Architecture

The application consists of three main components:

ResearchFunding.sol – main crowdfunding logic

ResearchToken.sol – ERC-20 reward token

Frontend (HTML + CSS + JavaScript) – user interface and MetaMask integration

Files Structure:

```
Blockchain-Final/
│
├── contracts/
│   ├── ResearchFunding.sol
│   └── ResearchToken.sol
│
├── frontend/
│   ├── index.html
│   ├── app.js
│   └── style.css
│
├── scripts/
│   └── deploy2.js
│
├── .gitignore
├── hardhat.config.js
├── package.json
├── package-lock.json
```

Explanation of Structure

contracts/

Contains Solidity smart contracts:

- **ResearchFunding.sol** – Main crowdfunding logic
- **ResearchToken.sol** – ERC-20 reward token

frontend/

Contains client-side application:

- **index.html** – User interface
- **app.js** – Blockchain interaction (ethers.js + MetaMask)
- **style.css** – UI styling

scripts/

- **deploy2.js** – Contract deployment script

Root Files

- **hardhat.config.js** – Hardhat configuration
- **package.json** – Project dependencies
- **package-lock.json** – Dependency lock file
- **.gitignore** – Ignored files

4. Smart Contract Architecture

4.1 ResearchFunding.sol

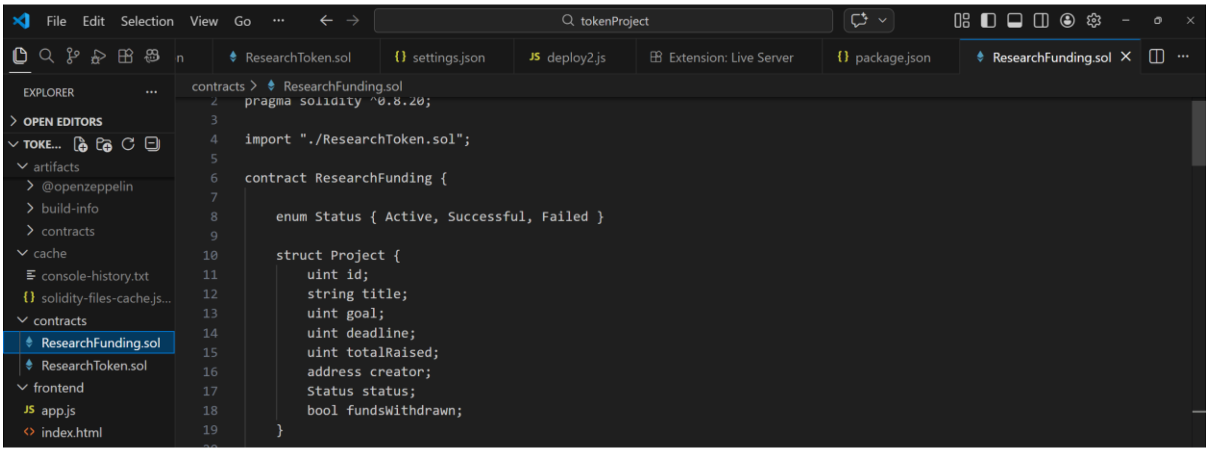
This contract contains the main crowdfunding logic.

Project Structure

```
struct Project {
    uint id;
    string title;
    uint goal;
    uint deadline;
    uint totalRaised;
    address creator;
    Status status;
    bool fundsWithdrawn;
}
```

Status Enum

```
enum Status { Active, Successful, Failed }
```

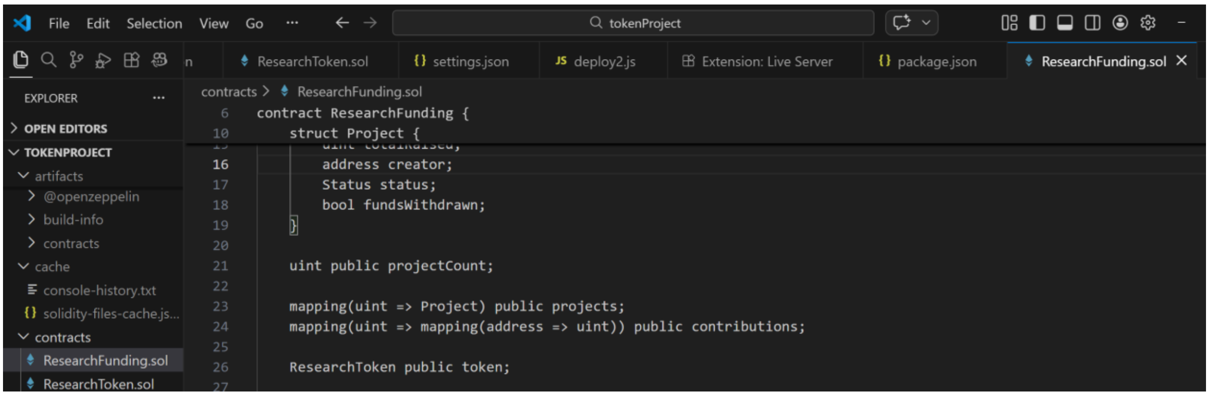


```
pragma solidity ^0.8.20;

import "./ResearchToken.sol";

contract ResearchFunding {

    enum Status { Active, Successful, Failed }

    struct Project {
        uint id;
        string title;
        uint goal;
        uint deadline;
        uint totalRaised;
        address creator;
        Status status;
        bool fundsWithdrawn;
    }
```

Contribution Mapping

```
mapping(uint => mapping(address => uint)) public contributions;
```

This mapping stores individual contributions for each project.

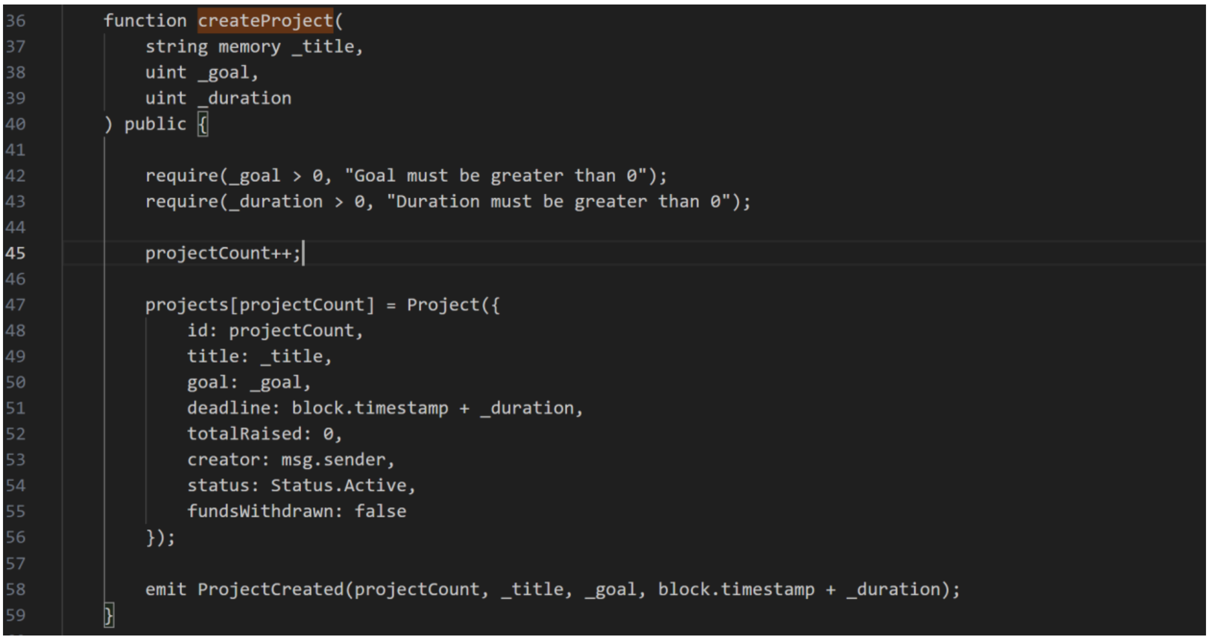


```
contract ResearchFunding {
    struct Project {
        uint totalRaised;
        address creator;
        Status status;
        bool fundsWithdrawn;
    }

    uint public projectCount;

    mapping(uint => Project) public projects;
    mapping(uint => mapping(address => uint)) public contributions;

    ResearchToken public token;
```

Core Functions

createProject()

- Accepts title
- Accepts funding goal
- Accepts duration
- Creates a new project
- Sets status to Active



```
    function createProject(
        string memory _title,
        uint _goal,
        uint _duration
    ) public {

        require(_goal > 0, "Goal must be greater than 0");
        require(_duration > 0, "Duration must be greater than 0");

        projectCount++;

        projects[projectCount] = Project({
            id: projectCount,
            title: _title,
            goal: _goal,
            deadline: block.timestamp + _duration,
            totalRaised: 0,
            creator: msg.sender,
            status: Status.Active,
            fundsWithdrawn: false
        });

        emit ProjectCreated(projectCount, _title, _goal, block.timestamp + _duration);
    }
```

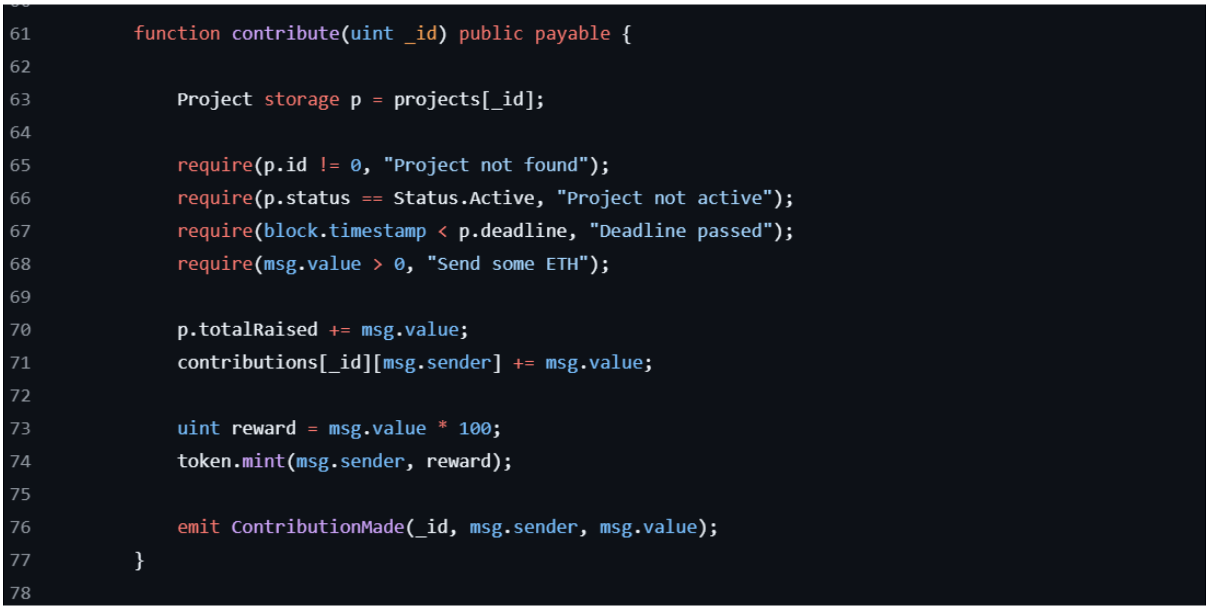
contribute(uint id)

- Accepts test ETH
- Checks:
    - Project exists
    - Project is Active
    - Deadline not passed

- Increases totalRaised
- Stores contribution
- Calls token.mint()

Reward formula implemented:

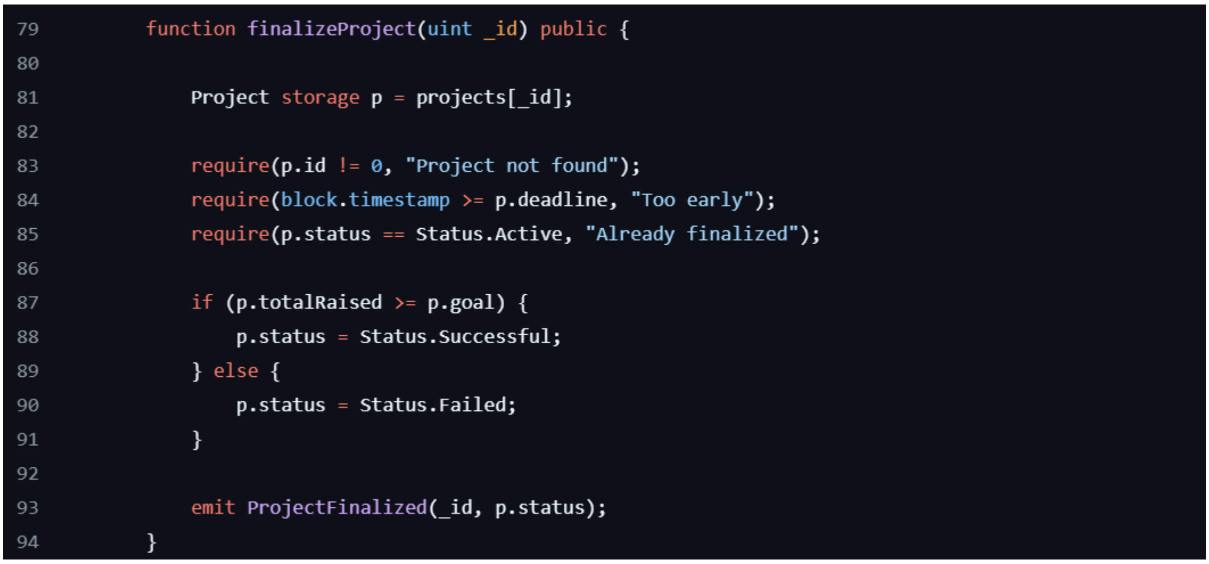
```
1 ETH = 100 RST tokens
```



```
    function contribute(uint _id) public payable {

        Project storage p = projects[_id];

        require(p.id != 0, "Project not found");
        require(p.status == Status.Active, "Project not active");
        require(block.timestamp < p.deadline, "Deadline passed");
        require(msg.value > 0, "Send some ETH");

        p.totalRaised += msg.value;
        contributions[_id][msg.sender] += msg.value;

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

        emit ContributionMade(_id, msg.sender, msg.value);
    }

```

finalizeProject(uint id)

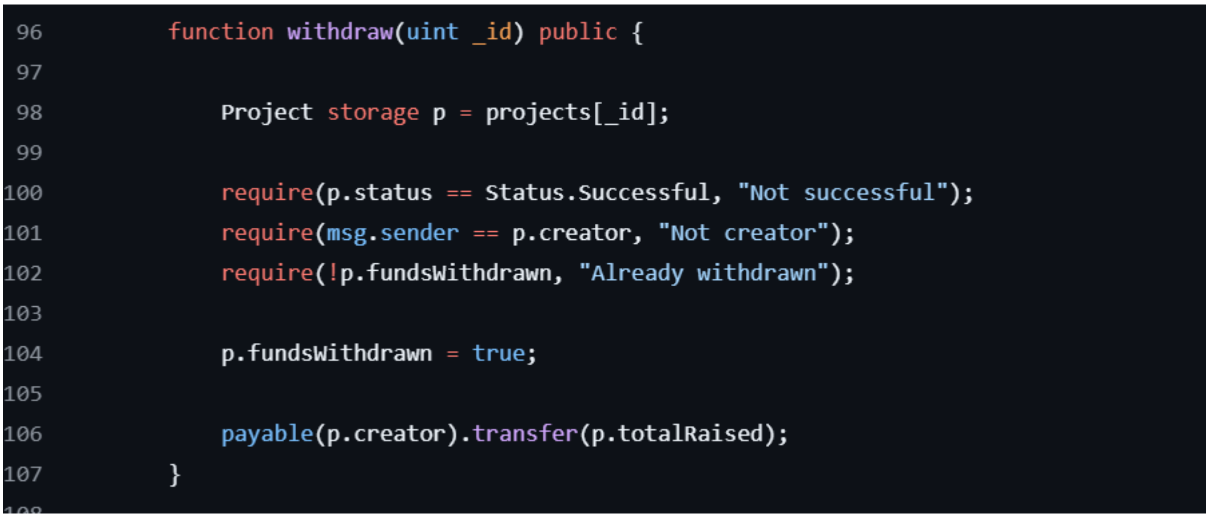
- Can be called after deadline
- If goal reached → Successful
- Otherwise → Failed



```
    function finalizeProject(uint _id) public {

        Project storage p = projects[_id];

        require(p.id != 0, "Project not found");
        require(block.timestamp >= p.deadline, "Too early");
        require(p.status == Status.Active, "Already finalized");

        if (p.totalRaised >= p.goal) {
            p.status = Status.Successful;
        } else {
            p.status = Status.Failed;
        }

        emit ProjectFinalized(_id, p.status);
    }
```

withdraw(uint id)

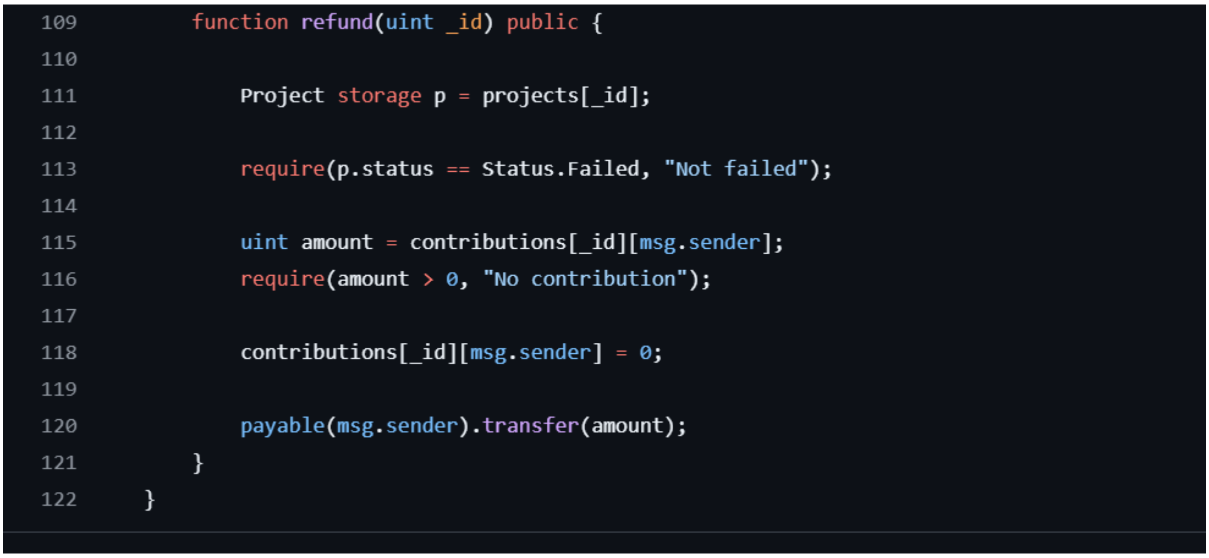
- Allows project creator to withdraw funds
- Only if project is Successful



```
    function withdraw(uint _id) public {

        Project storage p = projects[_id];

        require(p.status == Status.Successful, "Not successful");
        require(msg.sender == p.creator, "Not creator");
        require(!p.fundsWithdrawn, "Already withdrawn");

        p.fundsWithdrawn = true;

        payable(p.creator).transfer(p.totalRaised);
    }
```

refund(uint id)

- Allows contributors to withdraw their ETH
- Only if project Failed



```
    function refund(uint _id) public {

        Project storage p = projects[_id];

        require(p.status == Status.Failed, "Not failed");

        uint amount = contributions[_id][msg.sender];
        require(amount > 0, "No contribution");

        contributions[_id][msg.sender] = 0;

        payable(msg.sender).transfer(amount);
    }
}
```

4.2 ResearchToken.sol

This contract implements a custom ERC-20 token used as an internal reward token.

It includes:

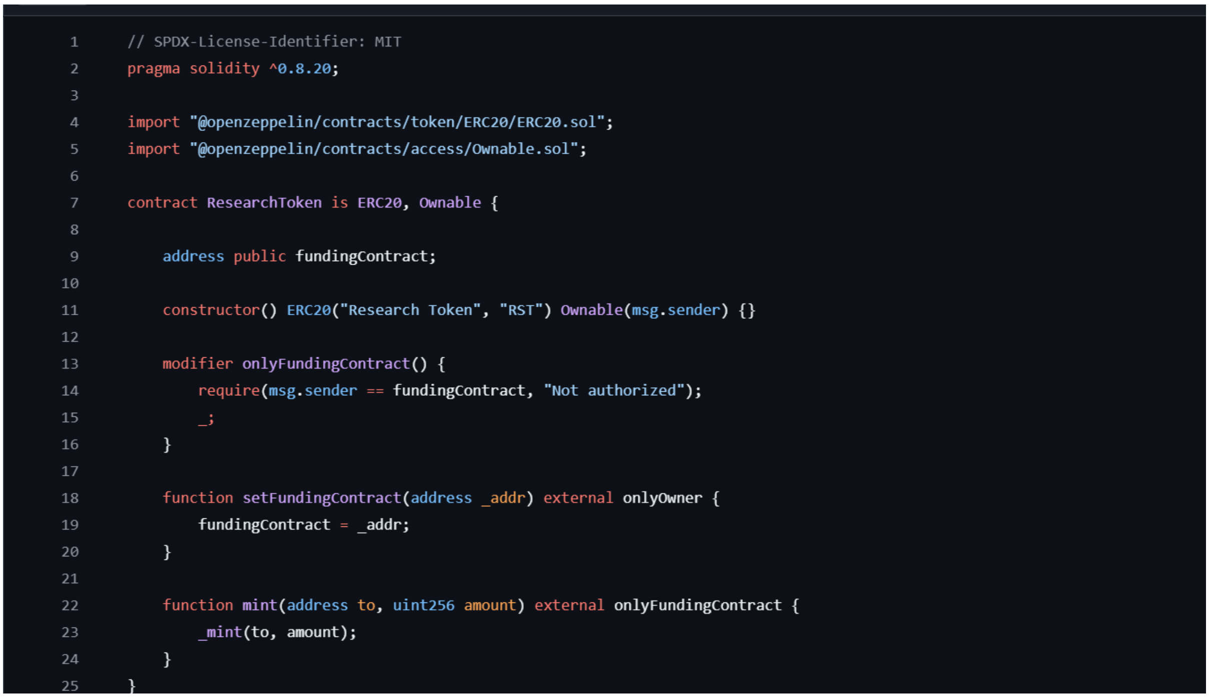
- name
- symbol
- decimals
- mint() function
- fundingContract address
- setFundingContract()

The mint function can only be called by the ResearchFunding contract.

This prevents unauthorized token issuance.

The token:

- Has no monetary value
- Is used only for educational purposes
- Demonstrates ERC-20 minting logic

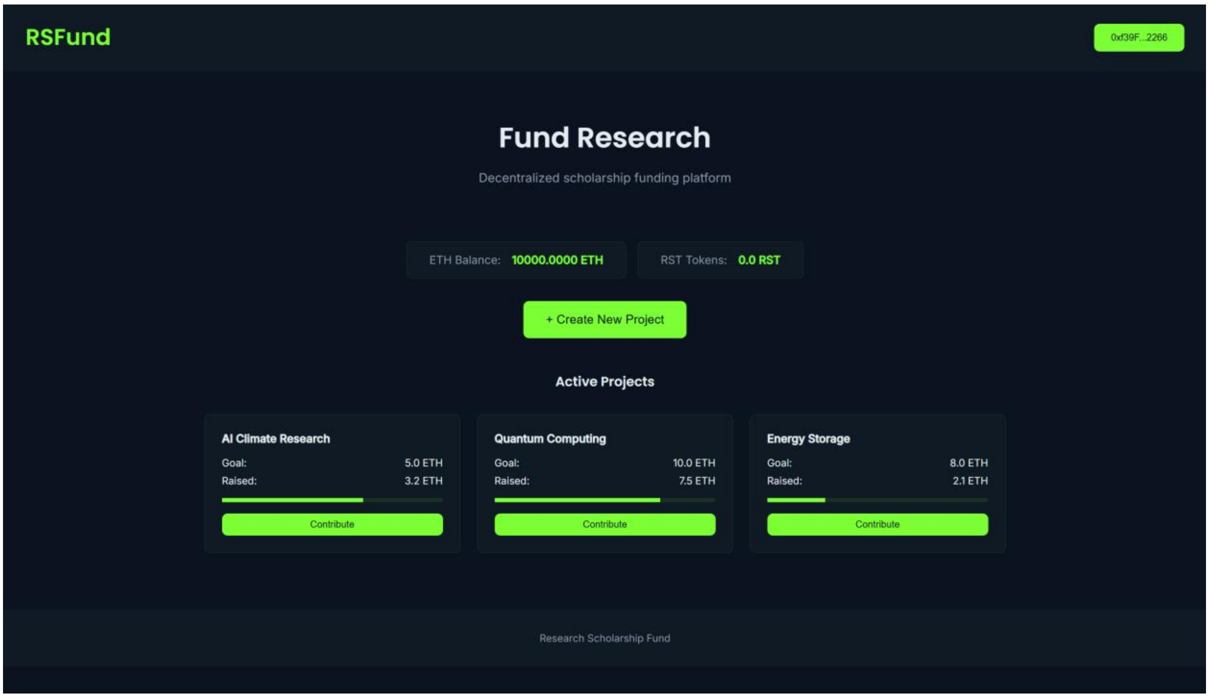


```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

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

contract ResearchToken is ERC20, Ownable {

    address public fundingContract;

    constructor() ERC20("Research Token", "RST") Ownable(msg.sender) {}

    modifier onlyFundingContract() {
        require(msg.sender == fundingContract, "Not authorized");
        _;
    }

    function setFundingContract(address _addr) external onlyOwner {
        fundingContract = _addr;
    }

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

5. Frontend Architecture

The frontend is built using:

- HTML
- CSS
- JavaScript
- Ethers.js



RSFund

0xf39F...2266

Fund Research

Decentralized scholarship funding platform

ETH Balance: 10000.0000 ETH

RST Tokens: 0.0 RST

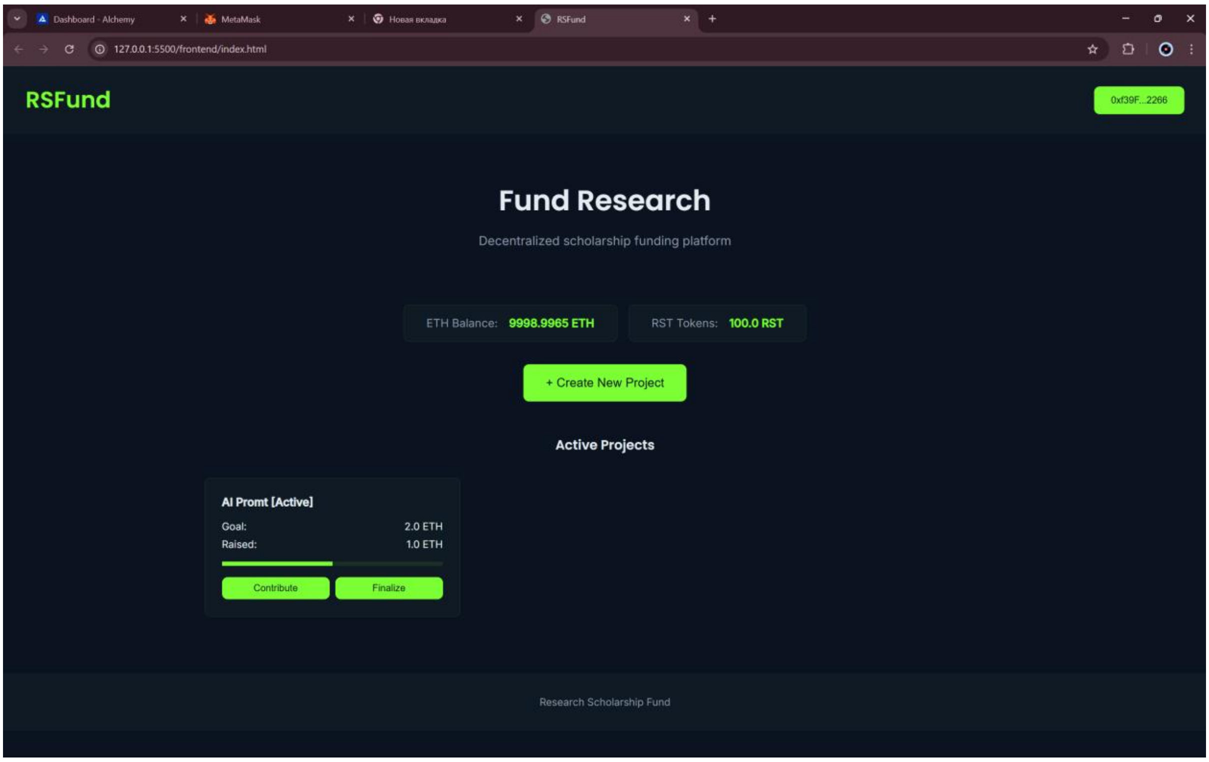
Active Projects

AI Climate Research
Goal: 5.0 ETH
Raised: 3.2 ETH

Quantum Computing
Goal: 10.0 ETH
Raised: 7.5 ETH

Energy Storage
Goal: 8.0 ETH
Raised: 2.1 ETH

Research Scholarship Fund



Dashboard - Alchemy | MetaMask | Новая вкладка | RSFund

127.0.0.1:5500/frontend/index.html

RSFund

0xf39F...2266

Fund Research

Decentralized scholarship funding platform

ETH Balance: 9998.9965 ETH

RST Tokens: 100.0 RST

Active Projects

AI Promt [Active]
Goal: 2.0 ETH
Raised: 1.0 ETH

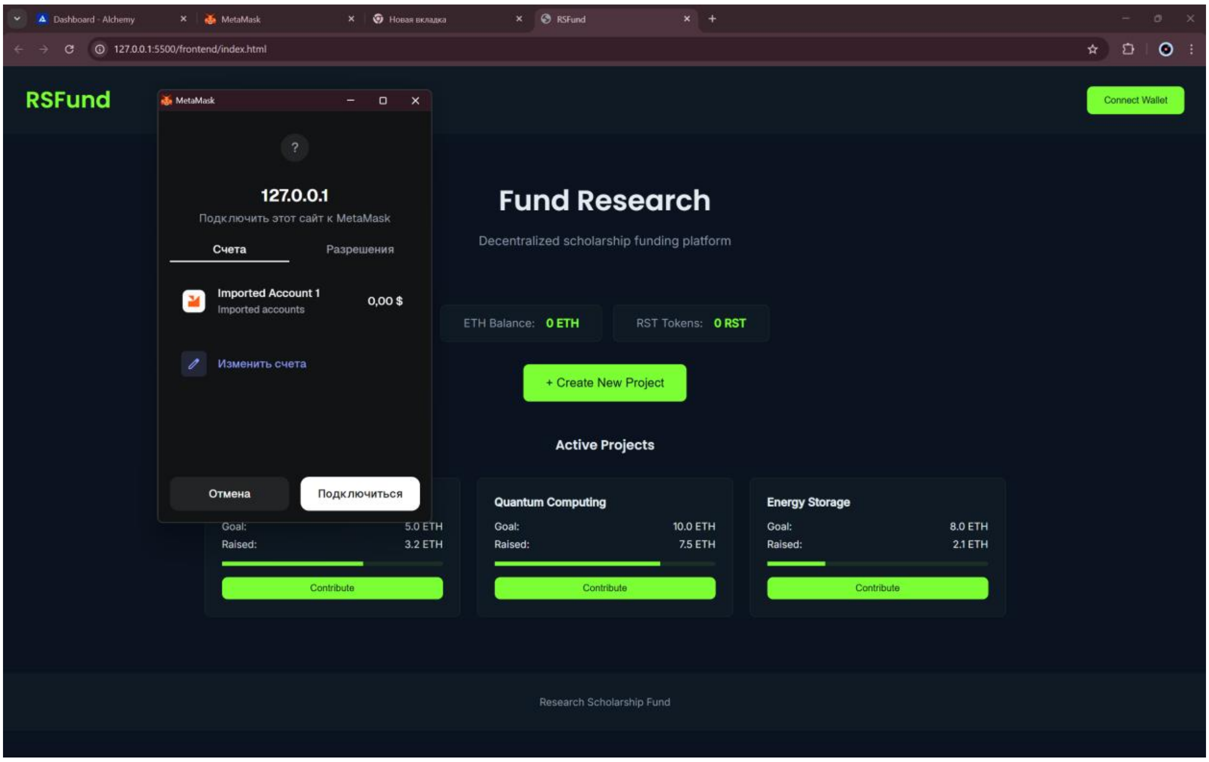
Research Scholarship Fund

MetaMask Integration

The frontend:

- Requests wallet connection
- Displays connected wallet address
- Verifies selected blockchain network (Hardhat local network)

- Sends transactions through MetaMask



127.0.0.1

Подключить этот сайт к MetaMask

Imported Account 1
Imported accounts
0,00 $

Fund Research

Decentralized scholarship funding platform

ETH Balance: 0 ETH

RST Tokens: 0 RST

+ Create New Project

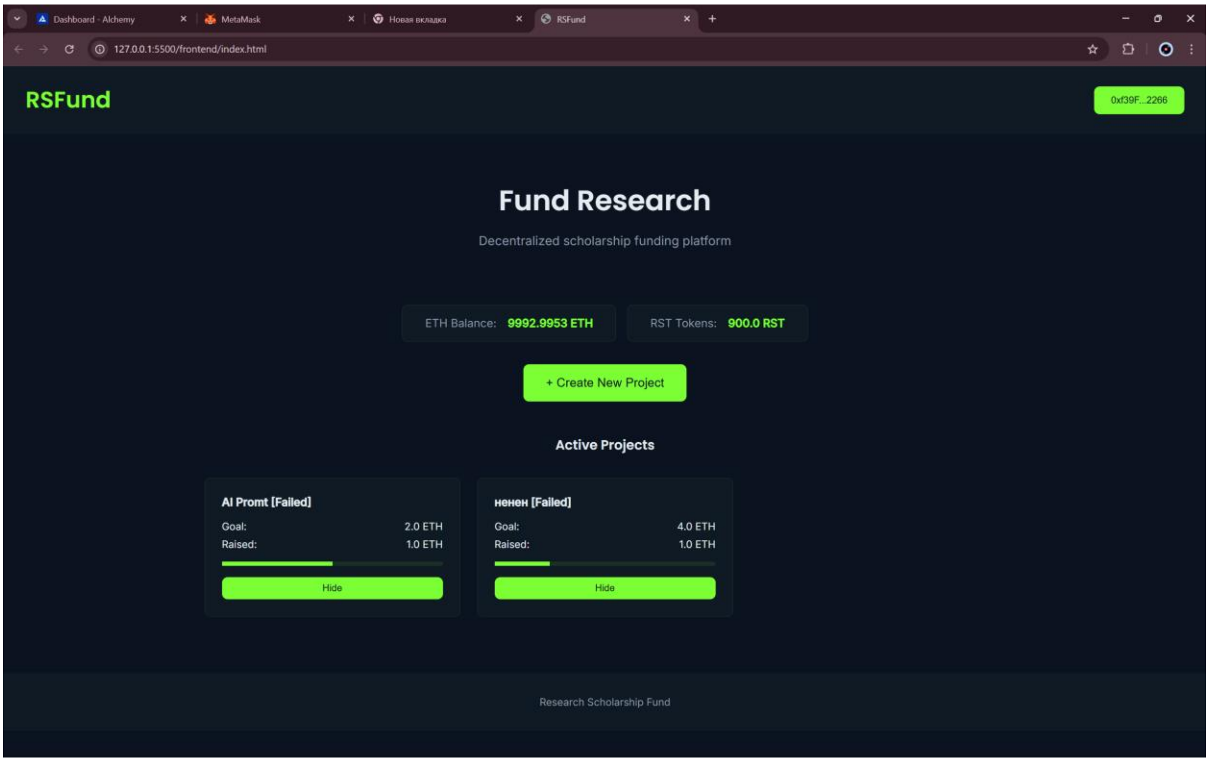
Active Projects

Goal: 5.0 ETH
Raised: 3.2 ETH

Quantum Computing
Goal: 10.0 ETH
Raised: 7.5 ETH

Energy Storage
Goal: 8.0 ETH
Raised: 2.1 ETH

Research Scholarship Fund



Fund Research

Decentralized scholarship funding platform

ETH Balance: 9992.9953 ETH

RST Tokens: 900.0 RST

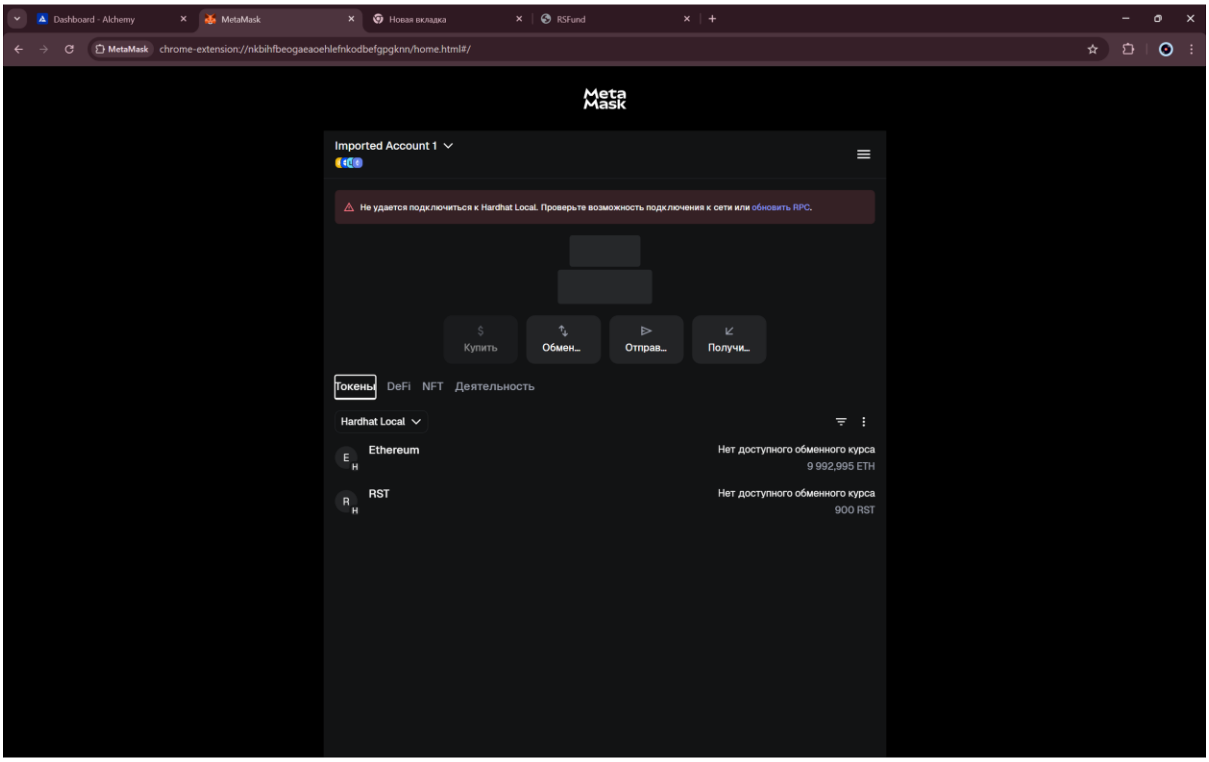
+ Create New Project

Active Projects

AI Promt [Failed]
Goal: 2.0 ETH
Raised: 1.0 ETH

ненен [Failed]
Goal: 4.0 ETH
Raised: 1.0 ETH

Research Scholarship Fund



Не удается подключиться к Hardhat Local. Проверьте возможность подключения к сети или обновить RPC.

Ethereum — Нет доступного обменного курса — 9 992,995 ETH

RST — Нет доступного обменного курса — 900 RST

Frontend Functionalities

The interface allows users to:

- Connect MetaMask
- Create new projects
- Contribute test ETH
- Finalize projects
- Withdraw funds (if successful)
- Request refunds (if failed)
- View ETH balance
- View token balance
- See project status and progress

All blockchain interactions are performed through ethers.js.

6. Frontend–Blockchain Interaction

Interaction flow:

1. User connects MetaMask
2. Frontend creates provider and signer

3. Contract instances are initialized
4. User triggers a function (e.g., contribute)
5. MetaMask requests transaction confirmation
6. Transaction is executed on local test network
7. UI updates after confirmation

This demonstrates real blockchain interaction.

7. Deployment & Execution Guide

Step 1 – Install Dependencies

```
npm install
```

Step 2 – Start Local Network

```
npx hardhat node
```

Step 3 – Deploy Contracts

In a separate terminal:

```
npx hardhat run scripts/deploy2.js --network localhost
```

This deploys:

- ResearchToken
- ResearchFunding
- Sets funding contract inside token

Step 4 – Configure MetaMask

Add local network:

- Network Name: Hardhat Local

- RPC URL: http://127.0.0.1:8545
- Chain ID: 31337
- Currency: ETH

Import one of the private keys from Hardhat into MetaMask.

Step 5 – Run Frontend

Open index.html in browser.

Click **Connect Wallet** and start interacting.

8. Test ETH

Test ETH is automatically provided by Hardhat local node.

No real cryptocurrency is used.

Deployment on Ethereum mainnet is strictly prohibited and not used in this project.

9. Team Responsibilities

Participant 1 – Assem Rakhmanova

- Frontend development
- MetaMask integration
- ResearchToken.sol implementation

Participant 2 – Aisana Kuanyshbek

- ResearchFunding.sol implementation
- Crowdfunding logic

Participant 3 – Nurassyl Nurdilda

- ResearchFunding.sol implementation

- Crowdfunding logic

All participants collaborated on:

- Deployment
- Testing
- Documentation preparation

10. Conclusion

The Decentralized Academic Research Funding Platform successfully demonstrates:

- Smart contract implementation
- Correct crowdfunding lifecycle logic
- ERC-20 token minting
- Secure MetaMask integration
- Operation on Ethereum test network
- Real blockchain transaction execution

The system satisfies all functional and technical requirements of the final examination project.

Github repo link: https://github.com/satoyakiii/Blockchain-Final.git