CCS6354 Blockchain and Smart Contract
2510

AmenaFund DApp: Tokenized Crowdfunding Smart Contract and UI Integration

LECTURE SECTION: TC1L

TUTORIAL SECTION: TT1L

Student Name	Student ID
Mohammed Amena Mohammed Abdulkarem	1221305728

Submission Date:
30 May 2025

Table of Contents

Table of Figures

1. Introduction

Amena crowdfunding is a secure crowdfunding Decentralized Application (DApp) built with Solidity. It enables users to contribute ETH towards a funding goal, with automatic refunds if the goal is not reached by the deadline.

2. System Architecture

2.1 Frontend

The user interface is built using **HTML, CSS, and JavaScript**, providing real-time interactivity by syncing with the Ethereum blockchain. Key UI components are dynamically updated based on the contract state:

- walletAddress: Displays the connected wallet address
- walletBalance: Shows the user's current ETH balance
- goalAmount, raisedAmount, contributorCount, userContribution: Present live campaign statistics
- timeLeft, refundInfo: Indicate remaining time and refund eligibility
- transactionHistory: Logs all major events—contributions, refunds, and withdrawals

2.1.1 Core Functionalities

Wallet Connection
connectWallet(): Connects the user's MetaMask wallet, initializes the provider/signer/contract, fetches ETH balance, wallet address, and displays owner UI if applicable.
Load Contract Data
loadContractData(): Retrieves and displays campaign goal, deadline, total raised, number of contributors, user's own contribution, refund eligibility, and campaign status. Updates the progress bar and related visuals.
Contribution
contribute(): Enables users to contribute ≥ 0.01 ETH via contract.contribute. On success, updates contract data and the user's wallet balance.
Withdrawal (Owner Only)
withdraw(): Allows only the contract owner to withdraw funds. Upon success, updates both the UI and wallet balance.
Refunds
requestRefund(): Allows eligible contributors to claim a refund after the deadline if the funding goal is unmet. The UI updates accordingly after execution.

2.1.2 Smart Contract Event Integration

- setupEventListeners(): Subscribes to key smart contract events:
- FundReceived: Triggered on user contribution
- GoalReached: Fired when the goal is met
- Refunded: Indicates a refund was processed
- FundsWithdrawn: Triggered on owner withdrawal
  Each event is logged using addTransactionToHistory() for visibility in the transaction history panel.

2.1.3 Utility Functions

- **Alerts**:
  showInfo(), showSuccess(), showError() display contextual messages.
  hideAlert(), hideAllAlerts() manage visibility.
- **Loading Indicators**:
  showLoading(), hideLoading() handle visual loaders for actions like contributing, withdrawing, and refunding.

2.2 Backend (Smart Contract)

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

/**
 * @title AmyFund
 * @dev A decentralized crowdfunding smart contract with refund mechanism
 */
contract AmyFund {
    // State variables
    address public owner;
    uint256 public goal;
    uint256 public deadline;
    uint256 public amountRaised;
    uint256 public contributorCount;
    uint256 public constant MINIMUM_CONTRIBUTION = 0.01 ether;
    bool public goalReached;
    bool public fundingClosed;

    mapping(address => uint256) public contributions;
    address[] public contributors;

    // Events
    event FundReceived(address indexed contributor, uint256 amount, uint256 timestamp);
    event GoalReached(uint256 totalAmount, uint256 timestamp);
    event Refunded(address indexed contributor, uint256 amount, uint256 timestamp);
    event FundsWithdrawn(address indexed owner, uint256 amount, uint256 timestamp);

    // Modifiers
    modifier onlyOwner() {
        require(msg.sender == owner, "Only owner can call this function");
        _;
    }

    modifier beforeDeadline() {
        require(block.timestamp < deadline, "Deadline has passed");
        _;
```

Figure 1: Smart contract code snippet showing the main structure of AmyFund with state variables, events, and modifiers

```
    modifier goalNotReached() {
        require(!goalReached, "Goal has already been reached");
        _;
    }

    modifier validContribution() {
        require(msg.value >= MINIMUM_CONTRIBUTION, "Contribution must be at least 0.01 ETH");
        _;
    }
```

The backend is a Solidity smart contract deployed on an Ethereum-compatible network. It includes:

- getDetails(): Returns the campaign goal, deadline, raised amount, contributor count, goal status, and time left.

```
function getDetails()    infinite gas
    external
    view
    returns (
        uint256 _goal,
        uint256 _deadline,
        uint256 _amountRaised,
        uint256 _contributorCount,
        bool _goalReached,
        uint256 _timeLeft
    )
{
    _timeLeft = block.timestamp < deadline ? deadline - block.timestamp : 0;

    return (
        goal,
        deadline,
        amountRaised,
        contributorCount,
        goalReached,
        _timeLeft
    );
}
```

Figure 2: getDetails() function

- contribute(): Allows users to send ETH to the campaign before the deadline, updating state and emitting events.

```
function contribute() external payable beforeDeadline validContribution {    infinite gas
    require(!fundingClosed, "Funding is closed");

    if (contributions[msg.sender] == 0) {
        contributors.push(msg.sender);
        contributorCount++;
    }

    contributions[msg.sender] += msg.value;
    amountRaised += msg.value;

    if (amountRaised >= goal && !goalReached) {
        goalReached = true;
        emit GoalReached(amountRaised, block.timestamp);
    }

    emit FundReceived(msg.sender, msg.value, block.timestamp);
}
```

Figure 3: contribute() function

- refund(): Allows contributors to get their funds back if the campaign fails.

```
function refund() external afterDeadline goalNotReached {    infinite gas
    require(contributions[msg.sender] > 0, "No contribution found");

    uint256 contributionAmount = contributions[msg.sender];
    contributions[msg.sender] = 0;
    amountRaised -= contributionAmount;

    (bool success, ) = payable(msg.sender).call{value: contributionAmount}("");
    require(success, "Refund transfer failed");

    emit Refunded(msg.sender, contributionAmount, block.timestamp);
}
```

Figure 4: refund() function

- withdraw(): Allows the contract owner to withdraw funds if the goal is met.

```
function withdraw() external onlyOwner {    infinite gas
    require(goalReached, "Goal not reached");
    require(amountRaised > 0, "No funds to withdraw");

    uint256 amount = amountRaised;
    amountRaised = 0;
    fundingClosed = true;

    (bool success, ) = payable(owner).call{value: amount}("");
    require(success, "Transfer failed");

    emit FundsWithdrawn(owner, amount, block.timestamp);
}
```

Figure 5: withdraw() function

- getOwner(): Returns the contract owner's address.

```
function getOwner() external view returns (address) {    1267 gas
    return owner;
}
```

Figure 6: getOwner() function

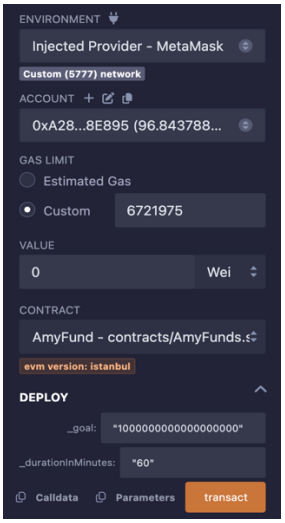

Figure 7: AmyFund contract deployment panel in Remix IDE

2.3 Blockchain Integration

The frontend connects to the Ethereum blockchain using ethers.js. It accesses the user's wallet via MetaMask (window.ethereum) and interacts with the smart contract to handle contributions, withdrawals, and refunds.

```
provider = new ethers.BrowserProvider(window.ethereum);
signer = await provider.getSigner();
contract = new ethers.Contract(CONTRACT_ADDRESS, CONTRACT_ABI, signer);
```

Figure 8: Ethers.js integration to interact with the smart contract using MetaMask

3. Results

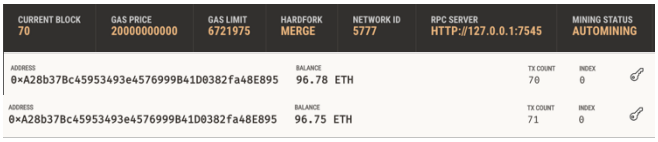

Figure 9: ETH Balance Decrease on Contribution (Ganache)

Figure 10: Successful Wallet Connection Displayed in App Interface

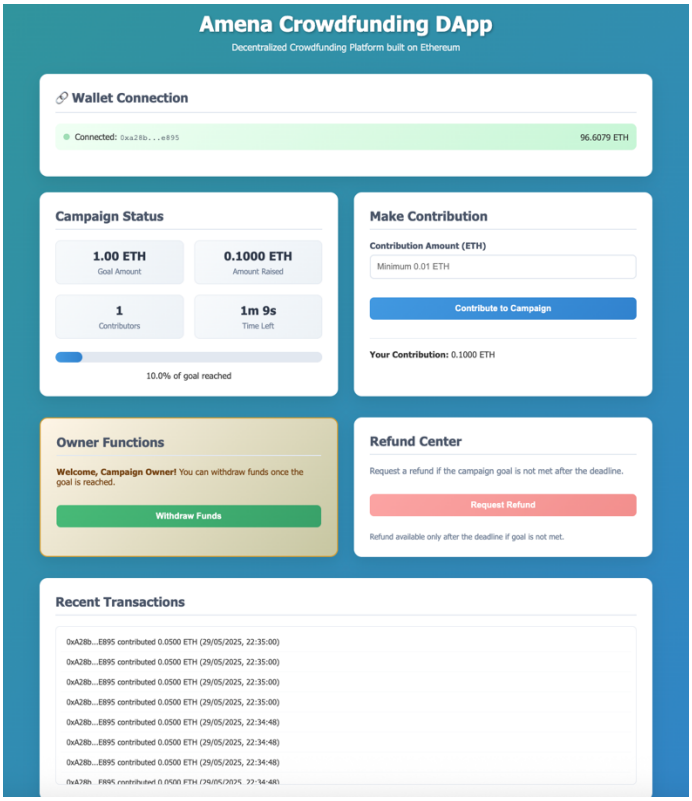

Figure 11: Crowdfunding DApp Interface — Wallet Connection, Campaign Status, Contribution, Withdrawal, Refund, and Transaction History

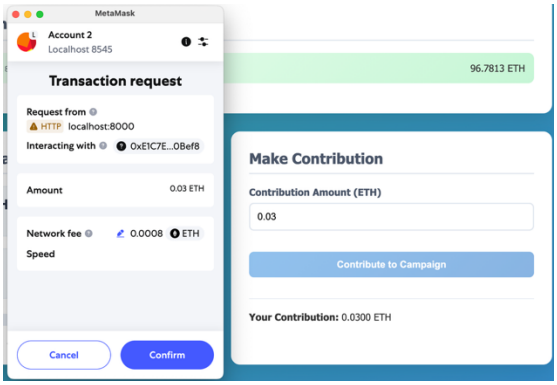

Figure 12: Contribution Interface — Enter ETH Amount, Submit Contribution, and View Updated Campaign Progress

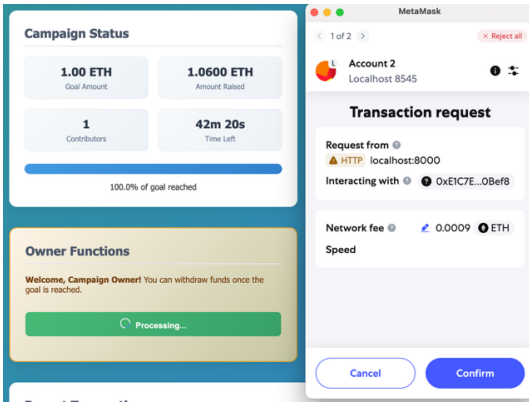

Figure 13: DApp Interface — Campaign Goal Reached, Allowing Owner to Withdraw Funds

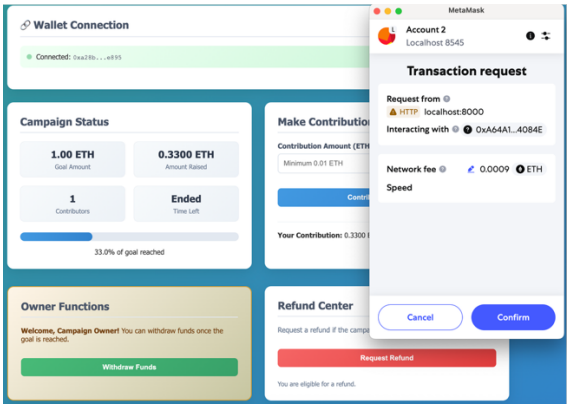

Figure 14: DApp Interface — Campaign Goal Not Reached and Deadline Passed, Allowing Contributors to Request Refunds

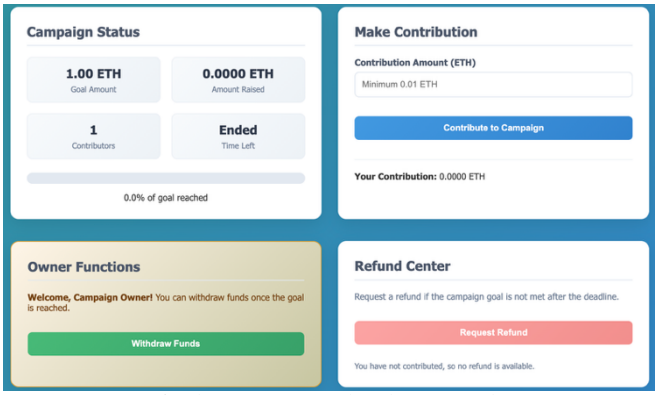

Figure 15: Refund Process Completed — Contributor's Amount Returned and UI Updated

4. Conclusion

This crowdfunding DApp enables a secure, transparent, and decentralized fundraising process. It allows users to track campaign progress, contribute funds, and reclaim their ETH if the goal isn't achieved. By synchronizing the UI with live blockchain data through MetaMask and ethers.js, the DApp ensures a smooth user experience. Full integration of alerts, balance updates, and event logging ensures reliability and user trust.