

Lab Exercise 1


Mohammed Yousef Mohammed Abdulkarem

1221305727

Lecture: TC1L

Tutorial: TT1L

i. Code Explanation

```
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);
}
```

The contribute() function allows any outside user to transfer ETH to the contract to become a participant in the crowdfunding campaign. It first checks whether the current time is still before the campaign termination date and whether the sent amount is greater than zero. If the sender is a new contributor, the number of contributors is incremented. The value contributed is logged to a map for the sender's address, and the overall raised amount is increased.

```
function checkBalance() external view returns (uint256) {  2476 gas
    return amountRaised;
}
```

The checkBalance() function returns the contract's current ETH balance. This is useful for debugging, audits, or displaying the total amount of money presently locked in the campaign. it is a view function.

```
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);
}
```

The withdraw() function should only be called by the contract owner and only after the campaign has closed successfully. It verifies that the deadline has been met and whether the amount received is equal to or greater than the target of the fundraising. If these conditions are fulfilled, the function

draws the full balance of the contract and attempts to send it to the owner in a safe call fashion. On a failed transfer, the function rolls back the transaction. This makes it so that only a successful campaign allows access to funds by the owner.

```
function refund() external afterDeadline goalNotReached {
    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");

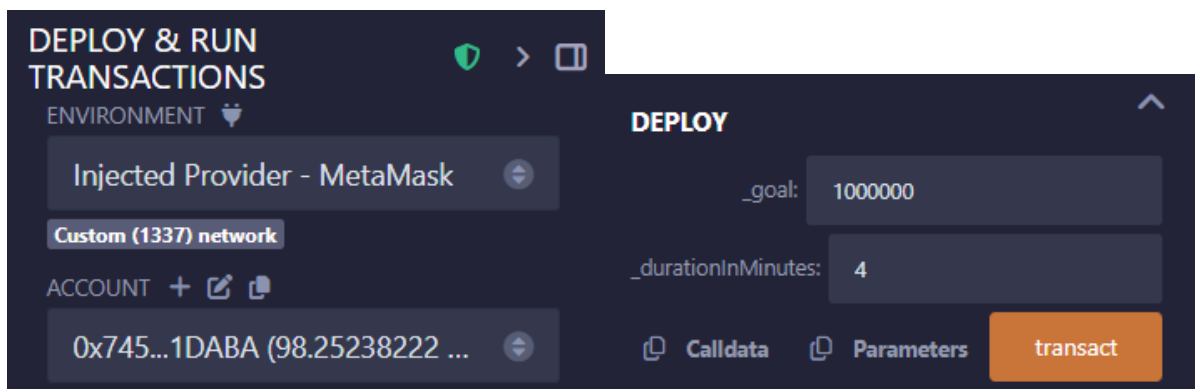
    emit Refunded(msg.sender, contributionAmount, block.timestamp);
}
```

The refund() function is to repay contributions to users in the event the campaign is unable to raise its goal at the deadline of the campaign. It can only be called after the deadline, and only when the total amount raised falls short of the goal. The function ensures that the sender of the contribution has a recorded contribution. If so, it reverts the sender's contribution amount to zero (to prevent reentrancy) and refunds the ETH back using low-level call. In case of success, it sends a Refunded event so users and frontend can track refund activity.

```
function getDetails()
    external
    view
    returns (
        uint256 _goal,
        uint256 _deadline,
        uint256 _amountRaised,
        uint256 _contributorCount,
        bool _goalReached,
        uint256 _timeLeft
    )
```

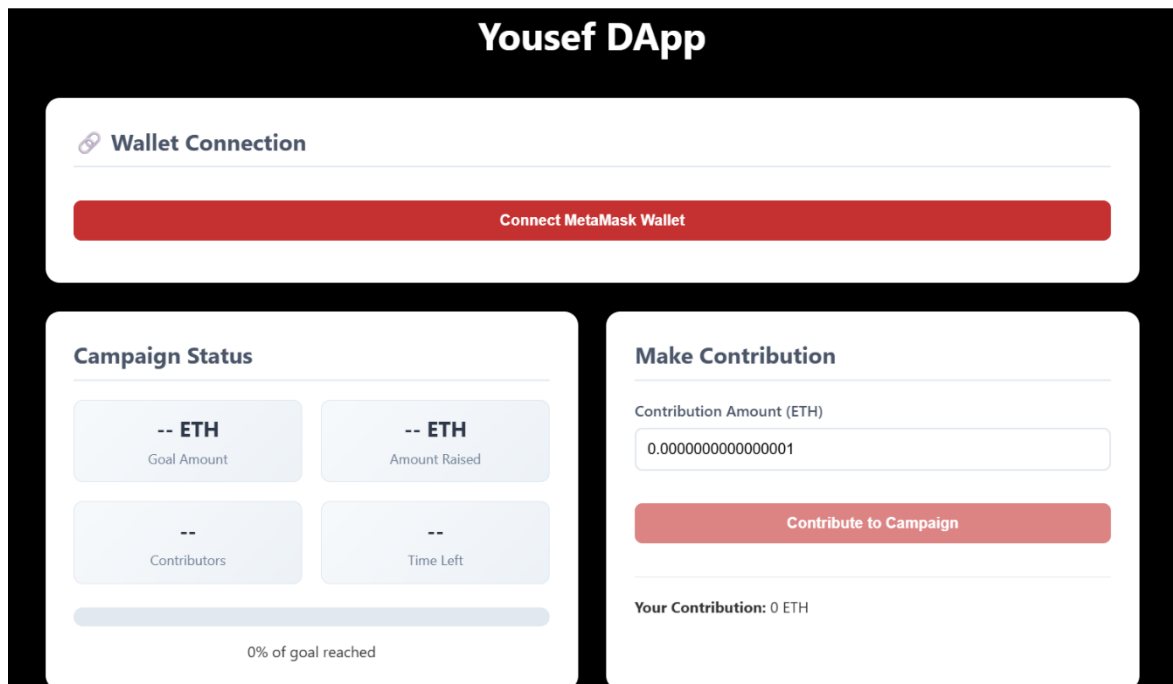
The getDetails() function is a read-only function that gives a summary of the present status of the campaign. Specifically, it returns the target amount for fundraising, the campaign's deadline in the format of a Unix timestamp, how much has been raised so far, and the number of unique contributors. It is all easily pulled in by the frontend application to inform users with real-time data.

ii. Deployment on Remix



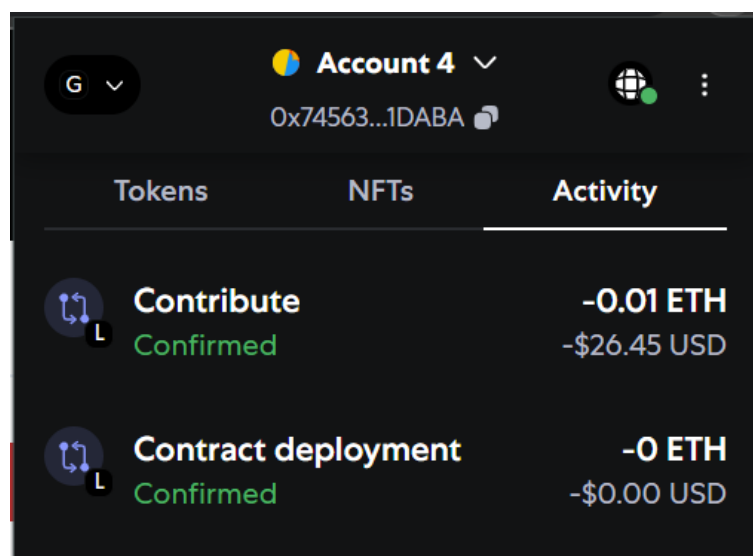
The smart contract was deployed on the MetaMask environment built in Remix IDE connected to MetaMask wallet that uses the localhost (Ganache). After deployment, the contract address was used in the frontend to enable interaction Functions like contribute(), refund(), and withdraw() were tested through both Remix and the DApp interface to ensure the contract performed as expected. Events and state changes confirmed successful deployment and integration.

iii. Frontend



Frontend was developed in JavaScript, CSS, and HTML, and Ethers.js facilitated interaction between deployed contract and web interface. Users can connect their MetaMask wallet to view live campaign data, contribute, and refunds. If they are connected as the contract owner, users are shown an option to withdraw money when campaign requirements are met. The interface is refreshed on its own whenever blocks are mined, allowing for accurate and updated information.

iv. MetaMask Integration



MetaMask was added to the DApp to allow people to securely connect their Ethereum wallets and interact with the smart contract. Upon clicking the connect button, the DApp requests access to accounts using MetaMask. After successful connection, the user address is displayed, and the user can perform actions such as contributing, requesting refunds, or withdrawal using their MetaMask Accounts. The integration offers secure transaction signing and simple interaction with the Ethereum blockchain.