Introduction

This smart contract allows fundraisers and founders participate in a trustless and secure crowdfunding campaign without the need for 3rd Parties and intermediaries.

In the crowdfunding campaign, when participants contribute to their projects, the funds are locked temporarily in the smart contract until the pool of investment meet the predefined threshold. If the funding succeeds the contract unlocks certain predefined rights and corresponding actions for the fundraisers and founders, it then locks the funds in a DAO. In a situation where the set conditions are not met the crowdfunding fails and the contract unlocks the funds contributed, which are now returned back to the contributors.

Features

The Following functionalities are needed in the smart contract:

1. Ability to fund/contribute to projects
2. Ability to check if conditions are reached at the deadline of the fundraising.
3. Ability to refund fundraisers/contributors if the fundraising failed.
4. Ability to forward funds to the DAO if the fundraising is successful.
5. Issuance of smart tokens(rights) which binds fundraisers to their funds.
6. Voting system for beneficiaries based on issued tokens
7. Withdrawal tracking system

Requirements

The contract should be able to:

1. Have a way to detect Moderator and beneficiary
2. Be able to map the beneficiary address to their balance
3. Have the ability to mint tokens(rights) to beneficiary
4. Have the ability to assign crowdfund state as “Running, Success, Failed, Forwarded”

Specifications

The following specifications are desired in the smart contract

Parent Contract (Inheritance)

The contract requires some functions to be accessible only by the owner of the smart contract and thus it’s a requirement to inherit the **Ownable.sol** library of OpenZeppelin.

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/ownership/Ownable.sol>

**Note:** Ensure that the inheritance of openZepplin contract is also accounted for

Library Functions

The contract uses additions and subtractions in various functions, thus, it’s also a requirement to import and use the **SafeMath** library of OpenZeppelin.

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/math/SafeMath.sol>

**Note:** The SafeMath library provided only deals with uint256 functions. You are also required to create a SafeMath library for uint16 functions as the contract requires addition and subtraction functionality for that variable as well.

Recommended to create a **SafeMath16** library out of the provided openZeppelin link and import that in the contract as well

**For minting tokens:**

"openzeppelin-solidity/contracts/token/ERC20/DetailedERC20.sol"

"openzeppelin-solidity/contracts/token/ERC20/StandardToken.sol"

"openzeppelin-solidity/contracts/token/ERC20/MintableToken.sol"

Enums

The following enums are required:

|  |  |
| --- | --- |
| **Name** | **Values** |
| CrowdsaleState | Running, Success, Failed, Forwarded |

Variables

The following variables are recommended to be used:

|  |  |
| --- | --- |
| **Name** | **Type** |
| crowdsaleBeneficiary | *Address*  *The beneficiary of crowdsale* |
| crowdsaleModerator | *address*  The moderator of crowdsale |
| fundingGoal | *uint*  *The target amount of ETh for the crowdsale* |
| fundingCap | *uint*  The maximum amount of ETh for the crowdsale |
| duration | *uint*  Maximum duration of the crowdsale |
| tokenPriceNumerator | *uint*  Numerator of the token price Example: If I want to give 10 tokens for 1 ETH - numerator 10, denominator 1 |
| tokenPriceDenominator | *uint*  Denominator of the token price |
| tokenRewardAddress | *Address*  Token used for reward |

Events

|  |  |
| --- | --- |
| **Events** | **Purpose** |
| FundsRecieved | Log funds recieved |
| FundsWithdrawn | *Log funds withrawn* |
| CrowdfundSuccessful | Log that the crowdfund was successful |
| CrowdsaleFundsForwarded | Log the crowdsale funds fowarded to the DAO |

Modifiers

The following modifiers is recommended to ensure proper restrictions on the smart contract functions:

|  |  |
| --- | --- |
| **Modifiers** | **Purpose** |
| onlyCrowdSaleModerator | Throws error if funding goal is not reached |
| onlyRunningCrowdsale | Passes if crowdsale is successful |
| onlySuccessfulCrowdsale | Passes if crowdsale is successful |
| onlyFailedCrowdsale | Passes if crowdsale is failed |

Functions

The following functions should be present in the smart contract, The student should decide on the visibility of the functions and variables along with view and pure.

For Reference: <https://solidity.readthedocs.io/en/v0.4.24/contracts.html>

#### Constructor

|  |  |
| --- | --- |
| **Functions** | **Purpose** |
| constructor | * Set beneficiary * Set moderator * Set fundingGoal * Set fundingCap * Set deadline * Set tokenPriceNumerator * Set tokenPriceDenominator * Set tokenReward |

#### Other Functions

|  |  |
| --- | --- |
| **Functions** | **Purpose** |
| Fallback function | * Receive ETH and mint corresponding amount of tokens |
| finishCrowdsale | * Check if conditions are reached at the dedaline of the fundraising |
| withdraw | * Refund contributions if the fundraising failed |
| forwardCrowdsaleFunding | * Forward funds to the DAO if the fundraising is successful |