// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

//This section declares an interface named IERC20.

//An interface is like a contract blueprint that defines the functions that another

//contract must implement. Here, IERC20 specifies two functions: transfer and transferFrom,

//which are commonly used in ERC20 token contracts to transfer tokens between addresses.

interface IERC20 {

function transfer(address, uint) external returns (bool);

function transferFrom(address, address, uint) external returns (bool);

}

//This line begins the definition of a new Solidity contract named CrowdFund.

contract CrowdFund {

//This line declares an event named Launch, which is triggered when a new

//crowdfunding campaign is launched. It records details such as the campaign ID,

// the creator's address, the fundraising goal, and the start and end times of the campaign.

event Launch(

uint id,

address indexed creator,

uint goal,

uint32 startAt,

uint32 endAt

);

//This line declares an event named Cancel, which is triggered when a crowdfunding

//campaign is canceled. It records the ID of the canceled campaign.

event Cancel(uint id);

//This line declares an event named Pledge, which is triggered when someone pledges tokens

//to a crowdfunding campaign. It records the campaign ID, the address of the pledger, and the amount of tokens pledged.

event Pledge(uint indexed id, address indexed caller, uint amount);

//This line declares an event named Unpledge, which is triggered when someone withdraws their pledge from a crowdfunding campaign.

//It records the campaign ID, the address of the caller, and the amount of tokens withdrawn.

event Unpledge(uint indexed id, address indexed caller, uint amount);

//This line declares an event named Claim, which is triggered when the creator of a crowdfunding campaign

// claims the pledged tokens after the campaign ends successfully. It records the ID of the claimed campaign.

event Claim(uint id);

//This line declares an event named Refund, which is triggered when a pledger receives a refund of their

//pledged tokens after a campaign ends unsuccessfully. It records the campaign ID, the address of the caller, and the amount of tokens refunded.

event Refund(uint id, address indexed caller, uint amount);

struct Campaign {

// Creator of campaign

address creator;

// Amount of tokens to raise

uint goal;

// Total amount pledged

uint pledged;

// Timestamp of start of campaign

uint32 startAt;

// Timestamp of end of campaign

uint32 endAt;

// True if goal was reached and creator has claimed the tokens.

bool claimed;

}

IERC20 public immutable token;

// Total count of campaigns created.

// It is also used to generate id for new campaigns.

uint public count;

// Mapping from id to Campaign

mapping(uint => Campaign) public campaigns;

// Mapping from campaign id => pledger => amount pledged

mapping(uint => mapping(address => uint)) public pledgedAmount;

constructor(address \_token) {

token = IERC20(\_token);

}

function launch(uint \_goal, uint32 \_startAt, uint32 \_endAt) external {

//the time of the campaign is in the future or equals to the current blocks timestamp

//if its not in the future it will revert the erroe message "start at < now"

require(\_startAt >= block.timestamp, "start at < now");

//the time you will end the programme will be greater than the starting time or equal to

// the \_startAt, else it will revert the error message "end at <start at

require(\_endAt >= \_startAt, "end at < start at");

//this ensures that the difference between \_endAt and the current block timestamp is not more

//than 90 days, if it exceeds 90days it will revert the error message "end at > max duration

require(\_endAt <= block.timestamp + 90 days, "end at > max duration");

// keeps track of the number of campaigns

count += 1;

//it initializes a new campaign and stores it in the campaigns mapping at the index count.

//It creates a Campaign struct with properties creator, goal, pledged, startAt, endAt

//and claimed. msg.sender represents the address of the account that called the launch

//function. goal is set to \_goal, pledged is initialized to 0, startAt and endAt are

//set to the provided values, and claimed is set to false.

campaigns[count] = Campaign({

creator: msg.sender,

goal: \_goal,

pledged: 0,

startAt: \_startAt,

endAt: \_endAt,

claimed: false

});

emit Launch(count, msg.sender, \_goal, \_startAt, \_endAt);

}

//This function allows the creator of a campaign to cancel it before it starts.

function cancel(uint \_id) external {

//It retrieves the campaign information based on the \_id provided.

Campaign memory campaign = campaigns[\_id];

//It ensures that the caller of the function is the creator of the campaign and that the campaign hasn't started yet.

require(campaign.creator == msg.sender, "not creator");

require(block.timestamp < campaign.startAt, "started");

//If conditions are met, it deletes the campaign from the campaigns mapping and emits a Cancel event.

delete campaigns[\_id];

emit Cancel(\_id);

}

//This function allows users to pledge funds to a campaign.

function pledge(uint \_id, uint \_amount) external {

//It retrieves the campaign information based on the \_id provided.

Campaign storage campaign = campaigns[\_id];

//It ensures that the campaign has started and has not ended yet.

require(block.timestamp >= campaign.startAt, "not started");

require(block.timestamp <= campaign.endAt, "ended");

//It increases the pledged amount for the campaign, updates the

//pledged amount for the user, and transfers tokens from the user to the contract.

campaign.pledged += \_amount;

pledgedAmount[\_id][msg.sender] += \_amount;

token.transferFrom(msg.sender, address(this), \_amount);

//It emits a Pledge event.

emit Pledge(\_id, msg.sender, \_amount);

}

//This function allows users to retract their pledged funds from a campaign.

function unpledge(uint \_id, uint \_amount) external {

//It retrieves the campaign information based on the \_id provided.

Campaign storage campaign = campaigns[\_id];

//It ensures that the campaign has not ended yet.

require(block.timestamp <= campaign.endAt, "ended");

//It decreases the pledged amount for the campaign, updates the

// amount for the user, and transfers tokens back to the user.

campaign.pledged -= \_amount;

pledgedAmount[\_id][msg.sender] -= \_amount;

token.transfer(msg.sender, \_amount);

//It emits an Unpledge event.

emit Unpledge(\_id, msg.sender, \_amount);

}

//This function allows the creator

//of a campaign to claim the pledged funds after the campaign ends

function claim(uint \_id) external {

//It retrieves the campaign information based on the \_id provided.

Campaign storage campaign = campaigns[\_id];

// It ensures that the caller is the creator of the campaign, the

//campaign has ended, the pledged amount is greater than or equal

//to the goal, and the funds haven't been claimed yet.

require(campaign.creator == msg.sender, "not creator");

require(block.timestamp > campaign.endAt, "not ended");

require(campaign.pledged >= campaign.goal, "pledged < goal");

require(!campaign.claimed, "claimed");

//It marks the campaign as claimed, transfers the pledged funds to the

// creator, and emits a Claim event.

campaign.claimed = true;

token.transfer(campaign.creator, campaign.pledged);

emit Claim(\_id);

}

//This function allows users to claim a refund if the campaign

//ends without reaching its goal.

function refund(uint \_id) external {

//It retrieves the campaign information based on the \_id provided.

Campaign memory campaign = campaigns[\_id];

//It ensures that the campaign has ended and the pledged amount is less than the goal.

require(block.timestamp > campaign.endAt, "not ended");

require(campaign.pledged < campaign.goal, "pledged >= goal");

//It calculates and retrieves the pledged amount for the user, resets the pledged amount,

//transfers tokens back to the user, and emits a Refund event.

uint bal = pledgedAmount[\_id][msg.sender];

pledgedAmount[\_id][msg.sender] = 0;

token.transfer(msg.sender, bal);

emit Refund(\_id, msg.sender, bal);

}

}