Raccolta fondi

UNO SMART CONTRACT

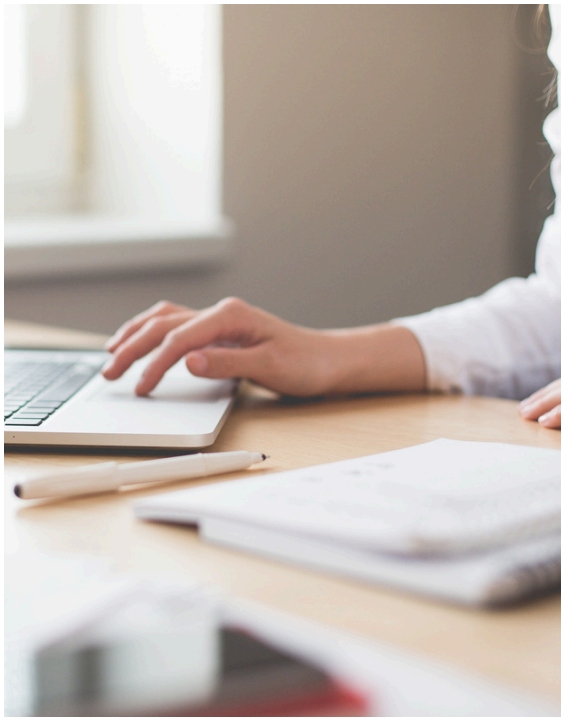
Un progetto di Carla Manavella

contents

Lavoro preliminare

Prima di inziare, si è proceduto a informarsi online circa la struttura migliore per scrivere smart contracts per le raccolte fondi. Le fonti principali:

- video YouTube “Learn Solidity: The COMPLETE Beginner’s Guide (Latest Version 0.8)” (https://www.youtube.com/watch?v=EhPeHeoKF88&feature=youtu.be);
- “Documentazione sullo sviluppo di Ethereum” (https://ethereum.org/it/developers/docs/);
- -“Solidity” (https://docs.soliditylang.org/en/v0.7.0/contracts.html#);
- “Solidity 0.6.x Features: Fallback and Receive Functions ()” (https://betterprogramming.pub/solidity-0-6-x-features-fallback-and-receive-functions-69895e3ffe)



L'inizio del contratto

La prima parte del codice, subito dopo l'esplicitazione della licenza (che, in questo caso, si è scelto di lasciare "**Unlicensed**") e la dichiarazione della versione Solidity utilizzata ("**pragma Solidity ^0.8.0;**"), è caratterizzata dall' "impalcatura" dell'intero lavoro: è qui, infatti, che si trovano le dichiarazioni di tutto ciò che servirà, più avanti, per l'esecuzione del contratto.

Dopo la dichiarazione della keyword “**contract**”, che indica l’inizio del codice che costituirà il contratto, si è subito esplicitata la funzione “vuota” chiamata “**receive ( ) external payable { }**”: questa permette al contratto di ricevere degli Ether.

Si continua, poi, con un “**enum**”, ovvero una “enumerazione” (un insieme di opzioni non modificabili), che permette di esplicitare i due stati possibili che avrà la campagna: “**Raising**” ed “**Ended**”. Subito dopo l'enumerazione si trova la dichiarazione della variabile “**currentStatus**”, che permette di usare lo stato nelle funzioni che seguiranno.

Troviamo, poi, la dichiarazione di tre variabili numeriche:

- “**uint public goalCampaign**” per l’obiettivo (espresso in Ether) della campagna, concordato in precedenza;
- “**uint totalRaised**” per il totale di Ether raccolti;
- “**uint totalDonor**” per il numero totale dei donatori.

Segue, quindi, un “**event**”, un evento che verrà notificato alla blockchain (va dichiarato esplicitamente all'inizio, ma servirà più avanti).

Proseguendo ancora, troviamo la struttura (*struct*) “**Donation**”, che servirà per “strutturare” le donazioni. Donation, al suo interno, è composto da:

- “**address contributor**”, che è l’indirizzo di chi ha effettuato la donazione;
- “**uint amount**”, ovvero il valore (in Ether) della donazione.

Donation[ ], poi, sarà anche una matrice di dimensione dimanica (o *array*) che servirà per archiviare e salvare all’interno del contratto le informazioni sulle donazioni che arriveranno.

Infine, a conclusione di questa sezione troviamo “**constructor**”: una funzione di Solidity che permette di creare *ad hoc* un elemento per il contratto corrente (“construct”, infatti, può essere dichiarato solo una volta); questo elemento verrà eseguito al momento della creazione del contratto stesso.

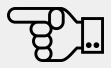
Nel constructor di questo contratto abbiamo due voci:

- “**campaignManager**”, ovvero chi gestisce la raccolta fondi (ed è suo l’indirizzo a cui i fondi verranno trasferiti);
- “**startTime = block.timestamp**”, che “fissa” il momento della creazione del contratto al timestamp del blocco in cui lo stesso viene salvato. Questa variabile servirà, più avanti, per permettere la distruzione del contratto solo alla scadenza della raccolta fondi (ovvero, a sei mesi dalla creazione).

La funzione contribute

La funzione “**contribute**” è quella che consente di effettuare tutte le operazioni che riguardano le contribuzioni e le donazioni che arriveranno al contratto.

Quando la funzione viene attivata, la prima azione che esegue è controllare che lo stato della campagna sia “Raising”; in seguito, provvede a raccogliere le donazioni e ad aggiungerle a “Donation” . L’array, i cui elementi rimangono salvati all’interno del contratto, si aggiorna con l’ultima donazione ogni volta che ne arriva una (“**contribution.push**”).



All’interno dell’array, Donation è formato a sua volta da una combinazione detta “***key-value***” (una combinazione “chiave-valore” - come le definizioni di un dizionario, dove a un lemma, la “*key*”, corrisponde la sua definizione, il “*value*”). Questo corrisponde a ciò che avevamo dichiarato in precedenza, ovvero un “Donation” composto da:

- l’indirizzo del contributor (“**msg.sender**”, ovvero chi ha mandato al contratto il messagio per donare Ether);
- il valore della sua donazione (“**msg.value**”, ovvero la quantità di Ether donati).

La funzione procede, poi, con l’aggiornamento del totale delle donazioni (“**totalRaised += msg.value**”) e del numero dei donatori (“**totalDonors = contribution.length**”).

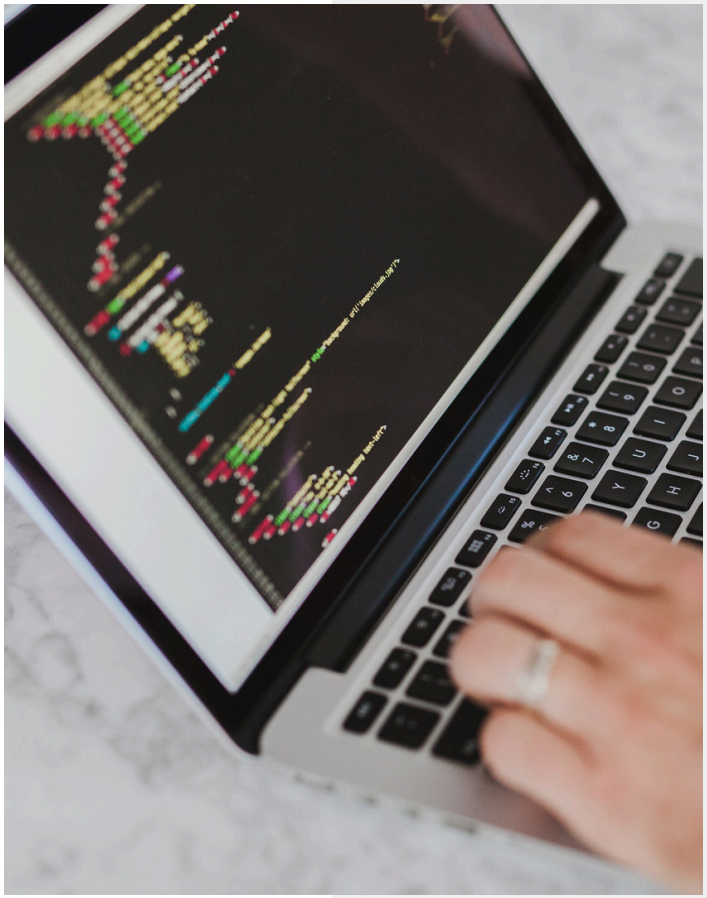
N.B.: il numero totale dei donatori viene calcolato chiedendo al contratto di dirci quante sono le “contribution” che compongono l’array “Donation”. Ciò significa che, se qualche utente facesse più di una donazione, questo verrebbe contato tante volte quante sono state le donazioni effettuate da quell'utente.

La funzione termina con “**emit**”, ovvero l’invio di un messaggio alla blockchain.

In questo messaggio ci saranno:

- “**msg.sender**”, ovvero chi ha mandato la donazione;
- “**msg.value**”, ovvero quanto è stato donato;
- “**totalRaised**”, ovvero il totale raccolto finora;
- “**totalDonors**”, ovvero il numero totale dei donatori.

La funzione checkFundingCompletion



La funzione “**checkFundingCompletion**” non è fondamentale per il funzionamento del contratto, ma può tornare utile per controllare se l’obiettivo della campagna di raccolta fondi sia stato raggiunto oppure no.

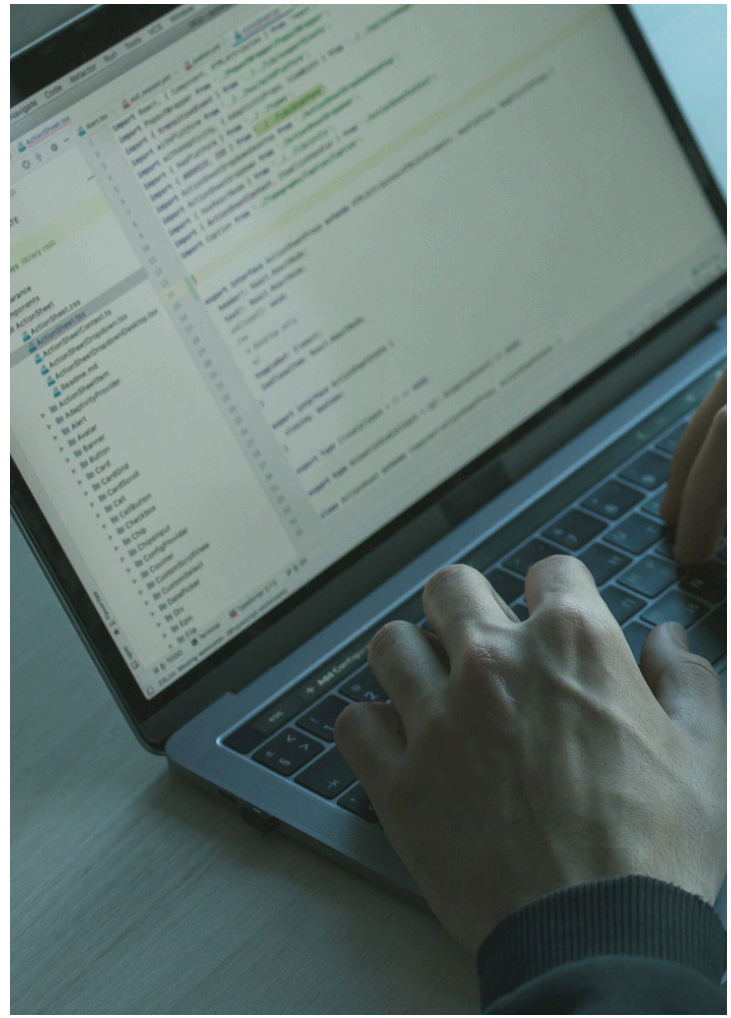
È composto da un semplice “*if-statement*”:
se l’obiettivo della campagna è stato raggiunto o superato (“**if (totalRaised >= goalCampaign) { }**”) , allora viene mostrato il messaggio “**We made it!**”; viceversa, se gli Ether raccolti sono meno rispetto all’obiettivo della campagna (“**else { }**”), viene mostrato il messaggio “**Not there, yet**”.

La funzione payOut

Il contratto termina con due funzioni:

- **payOut**;
- **removeContract**.

La prima di queste, ovvero “**payOut**”, non si può attivare prima che siano passati **sei mesi dalla pubblicazione del contratto sulla blockchain** (“**require(startTime+183 days < block.timestamp**”). Fa in modo che **lo stato della campagna passi da “Raising” a “Ended”** e, contestualmente, **trasferisce gli Ether raccolti dal contratto all'indirizzo del campaignManager**.

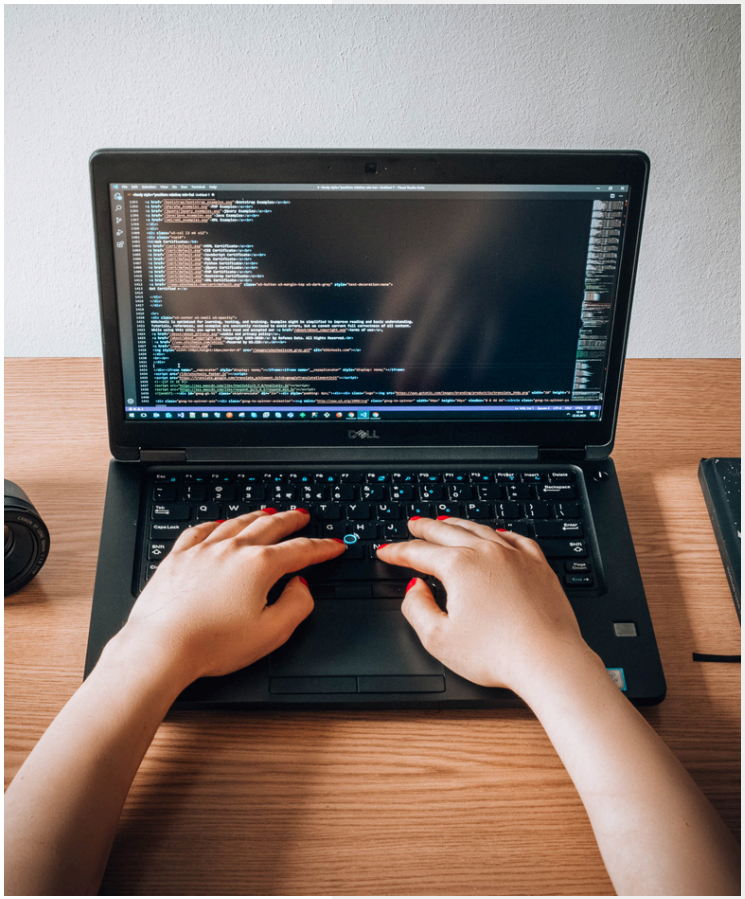


La funzione removeContract

La seconda funzione, “**removeContract**”, fa sì che **il contratto si auto-distrugga dopo 3 ore dal completamento della campagna** (triggerata dalla precedente funzione payOut).

La decisione di fissare un limite temporale alla campagna, e di rimuovere il contratto dalla blockchain alla sua scadenza, è stata presa per due ragioni:

1. Per non rischiare di utilizzare troppo gas, necessario per far funzionare il contratto stesso;
2. Per non occupare spazio sulla blockchain (che, per definizione, non è infinito) per più tempo del necessario.



Grazie
per l'attenzione