Peer to Peer Systems and Blockchain

BitCollect:
A decentralized crowdfunding platform

Luca Roveroni - 606803

10 June 2020

1 Final Project - BitCollect

The goal of the project is to develop a smart contract to implement a crowdfunding platform. The crowdfunding campaign has three main actors: the **organizers**, those who open the campaign; the **donors**, those who donate to the campaign; the **beneficiaries**, the receivers of the funds collected during the campaign. The implemented platform need to follow specific requirements specified in the attached pdf assignment.

The smart contract, Truffle test files and other materials are available at my Github repository: **Crowdfunding DAPP**

2 Project choices

The developed smart contract represents a crowdfunding campaign that could be deployed on the Ethereum Blockchain. This single contract incapsulates all the attributes and methods to interact with it and modify its state. For every single important step of the campaign an event is emitted. A campaign can be in the following states:

- **CREATED**: When the contract is deployed on the blockchain
- **INITIALIZATED**: When the campaign is assigned a list of organizers and beneficiaries (also a list of rewards since I choose that free choice requirement)
- **READY**: When all the organizers make a donation to the campaign to let donors start donating and reporters reporting the campaign
- **REPORTED**: When the campaign is set as fraudulent (That's an alternative final step)

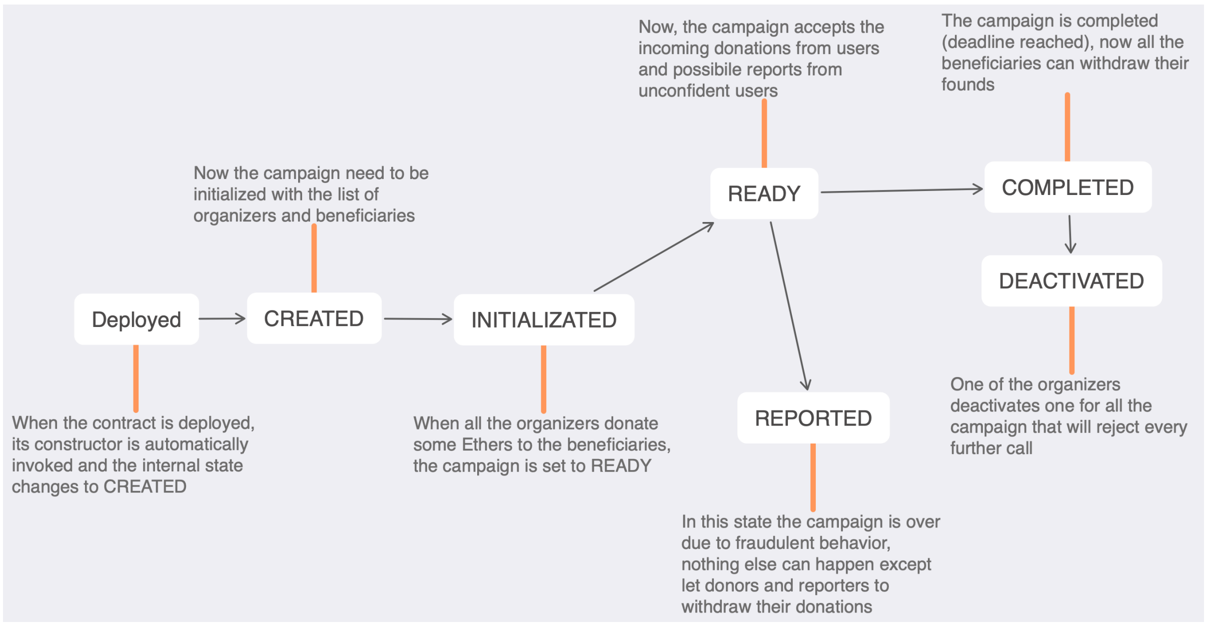
- **COMPLETED**: When the deadline is reached and all the beneficiaries can withdraw their donations
- **DEACTIVATED**: Final step when one of the organizers deactivate the contract

All of these steps are verified by several middleware functions called modifiers to check the integrity of the contract state and the permissions to modify it for each actor. Talking about the two free choice requirements I chose to implement the following:

- **RQ-REWARD**: Where the organizers set a number of rewards for the donors that reach a given threshold (The rewards are specified in the initialization phase)
- **RQ-REPORT**: Is a simple reporting strategy where users (called reporters) can report the campaign as fraudulent. My personal implementation of this requirement consists to set the campaign in the REPORTED state by checking if the number of reporters are greater or equal of the 50% of the donors + 1, but only if the total number of donors is greater than 15. I know this is not a very sophisticated strategy and could have some issues in a real case scenario but I find it a reasonable solution.

3 The smart contract

The crowdfunding campaign represented by the developed smart contract follows this workflow diagram:


Now, the campaign accepts the incoming donations from users and possibile reports from unconfident users
The campaign is completed (deadline reached), now all the beneficiaries can withdraw their founds
Now the campaign need to be initialized with the list of organizers and beneficiaries
READY
COMPLETED
DEACTIVATED
Deployed
CREATED
INITIALIZATED
REPORTED
When the contract is deployed, its constructor is automatically invoked and the internal state changes to CREATED
When all the organizers donate some Ethers to the beneficiaries, the campaign is set to READY
One of the organizers deactivates one for all the campaign that will reject every further call
In this state the campaign is over due to fraudulent behavior, nothing else can happen except let donors and reporters to withdraw their donations

Every macro-state emits an event that will be recorded on the blockchain. This strategy is used to implement the RQ-EVENT requirement.

3.1 Implementation phase

The contract has been developed in a local environment using some useful tools for deploy, testing and interact with it. These tools are Truffle and Ganache. The first phase consist to implement the smart contract in Solidity by defining the macro-state of the campaign (the ones in the image above) and the internal attributes like:

- **Deadline**: defined as unsigned integer that represents the number of seconds past from the base Unix Epoch (1 January 1970)
- **Some lists**: they are simple arrays storing addresses of each entity of the contract like organizers, beneficiaries, donors and reporters. These arrays are used as simple way to perform indexing and retrieve the total number of each of them
- **Some mappings**: are basically hash tables used for storing the real struct of each entity and perform a lookup in costant time for some operations

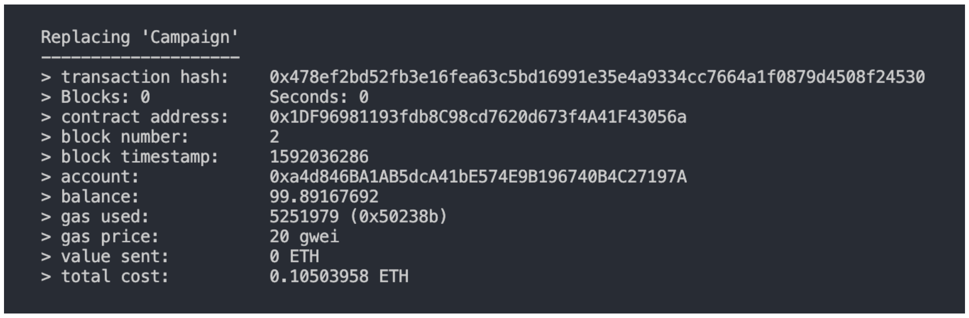
Then we have few modifiers functions used for checking the campaign state and implement some security checks. Lastly, all the contract functions that should be invoked by the contract entities to change its state and make the campaign "alive". All the details about the must requirements and the two free choices are inside the solidity smart contract. I list in briefly how all of the requirements have been implemented:

- **RQ-CREATE**: it is the constructor call made when the smart contract is deployed
- **RQ-PARAMS**: it is implemented by the setCampaignParams(...) function that takes the list of organizers, beneficiaries, rewards and the deadline
- **RQ-INIT**: it is implemented by the initCampaign() function which is payable and here all the organizers must donate to set the campaign to READY
- **RQ-DONS(2)**: it is basically implemented using a donate(...) function passing as parameter an array of struct to specify for each beneficiary his/her donation amount
- **RQ-HIST**: it's a mapping of Donors struct use to store all the information about the donor donations (total amount of donations and number of donations)
- **RQ-REJECT**: it is implemented using a modifier function

- **RQ-WITHDR**: it is another function callable only by the beneficiaries
- **RQ-CLOSE**: it's a function callable by only one of the organizers to set the campaign closed
- **RQ-REWARD**: in the RQ-PARAMS phase are listed all the possible rewards, then there is a mapping of ClaimReward struct that contains for each donors the assigned reward and if it has been already withdrawn
- **RQ-REPORT**: in the same way there is a mapping of reporters that are stored as different entity as the donors and when the campaign is set to fraudulent, reporters and donors can withdraw their donations.

3.2 Testing phase

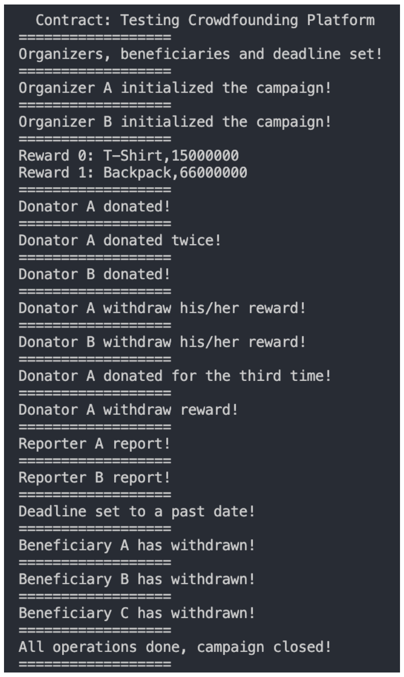
The tests are performed according to the assignment description, so using 2 organizers and 3 beneficiaries defined in the Truffle javascript test file. To start the testing phase the contract must be deployed on a blockchain, in this case I used Ganache as private blockchain. As we can see from the image below this is the contract deploy confirmation:



```
Replacing 'Campaign'
--------------------
> transaction hash:    0x478ef2bd52fb3e16fea63c5bd16991e35e4a9334cc7664a1f0879d4508f24530
> Blocks: 0            Seconds: 0
> contract address:    0x1DF96981193fdb8C98cd7620d673f4A41F43056a
> block number:        2
> block timestamp:     1592036286
> account:             0xa4d846BA1AB5dcA41bE574E9B196740B4C27197A
> balance:             99.89167692
> gas used:            5251979 (0x50238b)
> gas price:           20 gwei
> value sent:          0 ETH
> total cost:          0.10503958 ETH
```

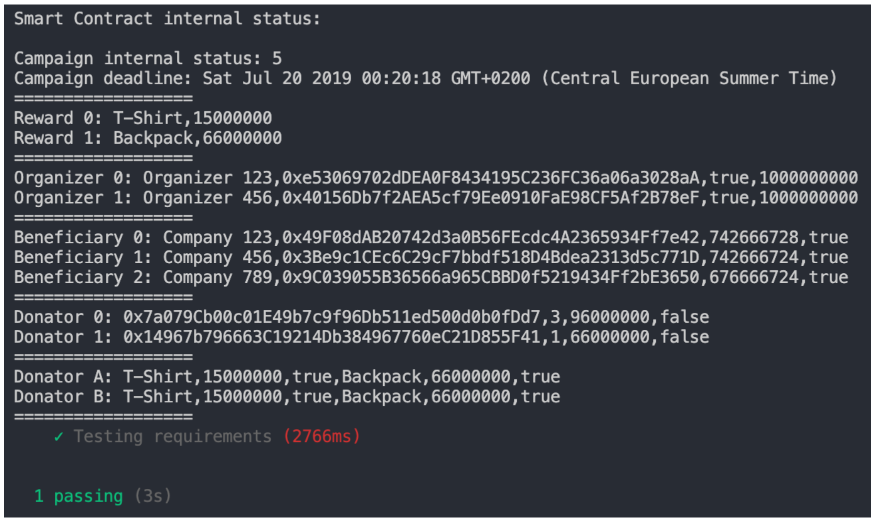
After the deploy, I wrote the testing file (all the files available on GitHub) testing the contract functionalities. To do so, I also wrote some useful functions in the end of the smart contract to retrieve the contract attributes to get the values of the internal attributes and show them on screen. During the test phase all the function calls has been logged with some messages in the console output to have a better understanding of the workflow.

At the end of the test I print the internal status of the campaign contract to have a complete overview of the contract internal state (the campaign state picture is here below).
As we can see from the picture at the right of the page, I made a test using 2 organizers, 3 beneficiaries, 2 donors and 2 reporters, all with (obviously) different Ganache account. This workflow is the one explained in the project assignment and the same procedure that will be shown in the live demo during the further online meeting. This test can be checked on the online repository in the path **test/test_campaign_func.js**.



```
  Contract: Testing Crowdfounding Platform
==================
Organizers, beneficiaries and deadline set!
==================
Organizer A initialized the campaign!
==================
Organizer B initialized the campaign!
==================
Reward 0: T-Shirt,15000000
Reward 1: Backpack,66000000
==================
Donator A donated!
==================
Donator A donated twice!
==================
Donator B donated!
==================
Donator A withdraw his/her reward!
==================
Donator B withdraw his/her reward!
==================
Donator A donated for the third time!
==================
Donator A withdraw reward!
==================
Reporter A report!
==================
Reporter B report!
==================
Deadline set to a past date!
==================
Beneficiary A has withdrawn!
==================
Beneficiary B has withdrawn!
==================
Beneficiary C has withdrawn!
==================
All operations done, campaign closed!
==================
```

Smart contract internal status after the tests:

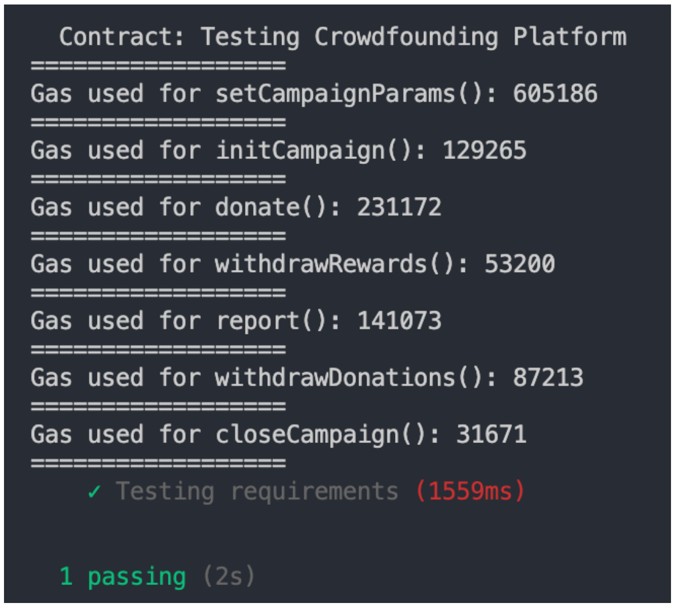


```
Smart Contract internal status:

Campaign internal status: 5
Campaign deadline: Sat Jul 20 2019 00:20:18 GMT+0200 (Central European Summer Time)
==================
Reward 0: T-Shirt,15000000
Reward 1: Backpack,66000000
==================
Organizer 0: Organizer 123,0xe53069702dDEA0F8434195C236FC36a06a3028aA,true,1000000000
Organizer 1: Organizer 456,0x40156Db7f2AEA5cf79Ee0910FaE98CF5Af2B78eF,true,1000000000
==================
Beneficiary 0: Company 123,0x49F08dAB20742d3a0B56FEcdc4A2365934Ff7e42,742666728,true
Beneficiary 1: Company 456,0x3Be9c1CEc6C29cF7bbdf518D4Bdea2313d5c771D,742666724,true
Beneficiary 2: Company 789,0x9C039055B36566a965CBBD0f5219434Ff2bE3650,676666724,true
==================
Donator 0: 0x7a079Cb00c01E49b7c9f96Db511ed500d0b0fDd7,3,96000000,false
Donator 1: 0x14967b796663C19214Db384967760eC21D855F41,1,66000000,false
==================
Donator A: T-Shirt,15000000,true,Backpack,66000000,true
Donator B: T-Shirt,15000000,true,Backpack,66000000,true
==================
    ✓ Testing requirements (2766ms)


  1 passing (3s)
```

Looking at the final status, we can
see that:

- The deadline is set to a past date to end the campaign
- The campaign status is set to 5 which represents the position of COMPLETED state inside the enum attribute
- The two rewards (T-Shirt and Backpack) with their threshold set by the organizer during the starting of the campaign
- List of organizers: name, eth. address, if has initializate the campaign and the initial donation
- List of beneficiaries: name, eth. address, total amount of donations and if has withdrawn his/her donations (this value is true since we are looking the campaign at the completed state)
- List of donors: eth. address, number of donations, total amount of donations and if withdraw his/her donations due to fraudolent campaign
- List of donors rewards: a list of reward name with its threshold reached
- There is also the list of users that report the campaign as fraudolent, but they are not show in the console output. Their structure is similar to the donator one (you can check in the code).

Last thing is the gas cost of every single functions as shown in this image:



```
  Contract: Testing Crowdfounding Platform
==================
Gas used for setCampaignParams(): 605186
==================
Gas used for initCampaign(): 129265
==================
Gas used for donate(): 231172
==================
Gas used for withdrawRewards(): 53200
==================
Gas used for report(): 141073
==================
Gas used for withdrawDonations(): 87213
==================
Gas used for closeCampaign(): 31671
==================
    ✓ Testing requirements (1559ms)


  1 passing (2s)
```

4 Security concerns

Talking about the security aspects, I'll try to evaluate if the developed smart contract can be affected by the most common known vulnerabilities. Here a list of the most common vulnerabilities with a personal comment about the application of that vulnerability on this smart contract:

- **Reentrancy**: it's considered to be one of the most severe vulnerability. It has been first recognised by the biggest attack ever made (The DAO hack). The reentrancy vulnerability relies on the interaction between two smart contracts so the absence of two or more contracts doesn't affect this specific case

- **Transaction ordering dependence**: it refers to the idea that the user can't be sure of the order of transactions. That's a common phenomenon and in this smart contract there could be this vulnerability. Looking at all the function calls, they are verified by modifier functions and they have additional require statements to assert if the actual campaign state, function parameters or entity that is calling has the proper privileges to do that operation.

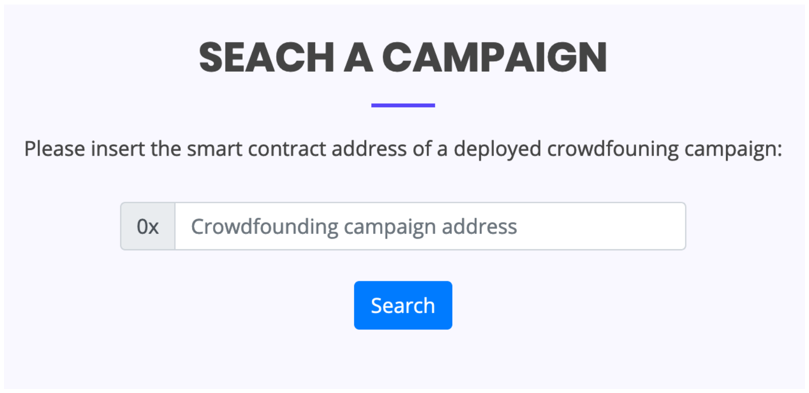
- **External calls & DoS**: like reentrancy in this smart contract there aren't any external calls except for the using of call.value() operation to transfer ether from the contract balance to the beneficiaries or donors/reporters in case of fraudolent campaign. Anyway this operations is safe like using send() since we can check and handle a possible transfer error

- **Blockhash usage**: it is not recommended to be used on crucial components because the miners, to some degree, can manipulate it and change the output to their favour. In the crowdfunding campaign contract it is used the "now" keyword that refers to the block timestamp, it is used to check if an operations is done after the deadline. Anyway, this is not a real problem since the deadline is set as a static value as private attribute, so the comparison between it and the block timestamp doesn't affect the correct workflow of the contract. For example if a miner postpone or anticipate the block mining, any operation done by the internal contract functions are check in advance if the deadline is reached or not.

5 The front-end application

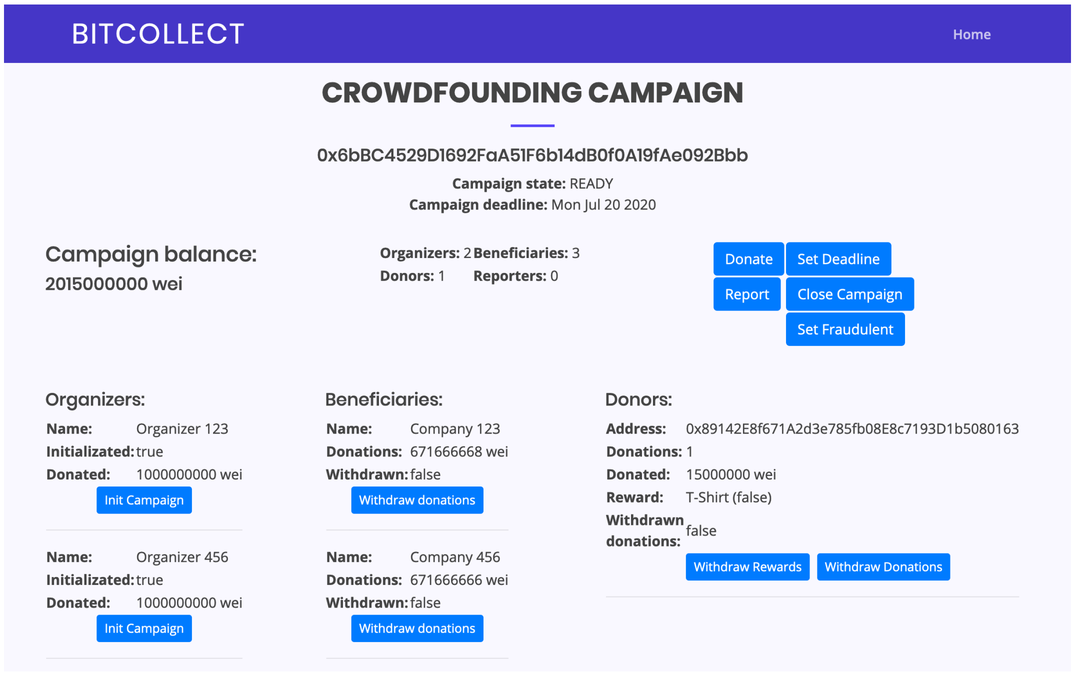
The front-end application has been developed using NodeJS and ExpressJS framework, to provide a reliable web user interface to interact with the smart contract. The developed Javascript application interacts with the smart contracts, that acts as back-end, using the Web3JS library to perform all the function calls. This choice let us to develop a more complex application around the smart contract to provide a scalable infrastructure, for example using off-chain technologies like databases. This let us to actually create a real DAPP like the ones in production on the Ethereum Blockchain, so performing additional features and operations like authenticating users, create more crowdfunding campaign, ecc.
Anyway, the front-end is split into two main pages:

- Home page: reachable at the localhost:3000/ path after starting the server. From this page we have a seach field where we can insert a deployed campaign address.


SEACH A CAMPAIGN

Please insert the smart contract address of a deployed crowdfouning campaign:

- Campaign page: it's a dynamic page use for show the contract status and interact with it. It is automatically opened when we search a contract from the home page. From this page we can see a lot of information about the campaign attributes, like: list of organizers, beneficiaries, donors, reporters and rewards. At the same time we can interact with the smart contract by simulating a donation, a report, set deadline to a past date to end the donations and many other operations. More explanations about the user interface and all the functionalities that can be performed using the Web UI will be given during the oral meeting. Finally, the campaign page appears like this:



CROWDFOUNDING CAMPAIGN

0x6bBC4529D1692FaA51F6b14dB0f0A19fAe092Bbb

Campaign state: READY
Campaign deadline: Mon Jul 20 2020

Campaign balance:
2015000000 wei

Organizers: 2 **Beneficiaries:** 3
Donors: 1 **Reporters:** 0

Organizers:

Name: Organizer 123
Initializated: true
Donated: 1000000000 wei

Name: Organizer 456
Initializated: true
Donated: 1000000000 wei

Beneficiaries:

Name: Company 123
Donations: 671666668 wei
Withdrawn: false

Name: Company 456
Donations: 671666666 wei
Withdrawn: false

Donors:

Address: 0x89142E8f671A2d3e785fb08E8c7193D1b5080163
Donations: 1
Donated: 15000000 wei
Reward: T-Shirt (false)
Withdrawn donations: false