COBrA Relationship

Aldo D’Aquino • a.y. 2017/18

# Aim of the project

COBrA is a university project that aims to implement a decentralized content publishing service.

Some authors will be able to publish their contents on a catalog and users will be able to use them by purchasing them or subscribing to a premium subscription.

Contents can be song, video, photo or something else created by the artists and they will be rewarded accordingly to customers’ fruition.

The system will have to rely entirely on the blockchain, so as to be completely decentralized and not having to depend on third-party servers that entail high running costs, are single points of failure and can often be unreliable.The purpose of the project is to deepen the use of blockchain for decentralized and secure systems, other than the mere exchange of money.

# Project specifications

Users of the system are divided into Customers and Authors. An author can also be a customer of the system as he may be interested in content from other authors.

Customers are divided into two types: Standard and Premium.

All the accounts start as Standard accounts: they must buy contents before accessing them and they are entitled to a single fruition of them. For further access they need to buy it again.

Standard account can subscript a Premium access that last for x blocks number on the blockchain. During the Premium period they can watch all the contents for free as many times they want.

# Implementation specifications

The system must be implemented a single Catalog Contract and some Content Management Contracts.

The number of Authors, Customers and Contents is not fixed and can change dynamically.

Each Content Management Contract controls exactly one unique content.

An author submits a new content by deploying a new Content Management Contract that inherit a predefined set of functionalities by extending the Base Content Management Contract. The Content Management Contract must include the name of the content and the content data and may include other features for content management at the discretion of the author.

Once the Content Management Contract is deployed the author has to submit a new content publishing request to the Catalog Contract to link the new contract to the Catalog. This linking is provided by a method call with which the Catalog can find the necessary information about the content from the Content Management Contract.

The Catalog Smart Contract acts as an intermediary between Authors and Customers.

Customers can consult the library of all the contents published in the Catalog by Content Management Contracts.

Customers access the Catalog Contract to request access to contents. To access the content, the Customer must send to the contract an amount of ether equal to the cost of the content, which for simplicity is the same for each content. Premium accounts do not have to pay and can directly access all the contents for free until the Premium subscription expiration. When the access is granted the Customer can consume the content at the Content Management Contract.

All the payments from the users are collected and redistributed among authors by the Catalog Contract, according to the number of views of each content.

For the sake of simplicity Premium Account content fruitions are not considered in the view count.

To ensure decentralization of the system the Catalog Contract deployer cannot receiver any rewards: at the end of the Catalog Contract the remaining budget must be redistributed among the authors according to the number of views obtained by their contents.

Both the Catalog Contract, the Base Content Management Contract and any Content Management Contract must be written in solidity and deployed on the Ropsten network.

It is mandatory to implement at least the following functions:

* Public views
  + GetStatistics(): returns the number of views for each content.
  + GetContentList(): returns the list of contents without the number of views.
  + GetNewContentsList(): returns the list of newest contents.
  + GetLatestByGenre(g): returns the most recent content with genre g.
  + GetMostPopularByGenre(g): returns the content with genre g, which has received the maximum number of views
  + GetLatestByAuthor(a): returns the most recent content of the author a.
  + GetMostPopularByAuthor(a): returns the content with most views of the author a.
  + IsPremium(x): returns true if x holds a still valid premium account, false otherwise.
* Public actions (modify the state of the contracts):
  + GetContent(x): pays for access to content x.
  + GetContentPremium(x): requests access to content x without paying, premium accounts only.
  + GiftContent(x,u): pays for granting access to content x to the user u.
  + GiftPremium(u): pays for granting a Premium Account to the user u.
  + BuyPremium(): starts a new premium subscription.

# Implementation

## Coding conventions choices

We choose to write all the code in compliance of the official stylish guidelines (1). In particular we refer to the Solidity version 0.4.24 (2), which is the last release at the moment. The Ethereum docs are continuously under construction, their progress as well as the latest version that still under construction (3) can be found in the *docs* path in the GitHub repository (4).

Among others, we consider in particular the following conventions.

### Encoding

The stylish guidelines recommend UTF-8 or ASCII. We choose UTF-8.

### Indentation

Indentation is 4 spaces wide as suggested; spaces are preferred instead of tabs.

### Line breaks

There must be 2 empty lines between contracts, 1 empty line between the functions implementation and 0 between functions and variables declaration. For greater clarity refer to the example below (figure 1), taken from the official documentation (1).

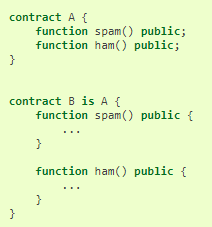


Figure 1 - Line breaks example.

le funzioni sono mixed case invece che camel case come richiesto per rimanere conformi alla guida

Comment styles in solidity code

\_suicide to not override the default function?

Assert, Require, Throw and Revert

<https://medium.com/blockchannel/the-use-of-revert-assert-and-require-in-solidity-and-the-new-revert-opcode-in-the-evm-1a3a7990e06e>

Cambiate alcune method signature per rendere più omogeneo e chiaro (from x to u, etc)

<https://ethereum.stackexchange.com/questions/28813/how-to-write-an-optimized-gas-cost-smart-contract>

Dynamically sized bytes or arrays vs static

I permessi per accedere al contenuto li salvo nel catalogo e non come contratti a se stanti nella blockchain per non spammare troppo

<https://ethereum.github.io/yellowpaper/paper.pdf> page 25 gas costs

define the parameters of the system (for instance, the amount to pay for each content, the lasting time of a premium account, etc.). Their choice must be discussed, when critical.

Reentrancy: <https://medium.com/@gus_tavo_guim/reentrancy-attack-on-smart-contracts-how-to-identify-the-exploitable-and-an-example-of-an-attack-4470a2d8dfe4>

Reentrancy: <http://solidity.readthedocs.io/en/develop/security-considerations.html#re-entrancy>

Bytes32 instead of string:

* + - String is not acceptable
    - Bytes potentially infinite, so infinite gas consumption
    - Bytes32 usually used for strings, and is enough
    - This is a 32 letters long title.

GAS COST:

Deploy catalog: transaction 2470605 - execution 1859489

Deploy content: transaction 1190425 - execution 877537