

Development of a Dapp for COBrA

Academic Year 2017/2018

1 Introduction

COBrA is a decentralized application based on the Ethereum blockchain. This platform offers a service to both authors, who wants to share their contents, and to customers, who are interested in accessing those files.

In the final-term it was developed a basic implementation of the back-end. This project extends that structure with more advanced functionalities and a front-end application to interact easier with the contracts.

1.1 Tools and Libraries Used

The back-end part of the project is composed of Solidity contracts. The initial attempts were written and tested using **Remix**, an Ethereum IDE for the web. However, this tool does not allow to run scripts to check the correct implementation of functionalities. So, once the application grew in complexity, it was hard to test the code.

It was decided therefore to move the implementation to the Truffle Suite. This offers the framework **Truffle**, a development environment for testing applications based on the Ethereum blockchain, and **Ganache**, a personal Ethereum blockchain that can be used to deploy contracts.

In order to run, they require the presence of the package manager for JavaScript **NPM**. Moreover, it was used **Metamask**, a browser extension that allows you to run Ethereum DApps right in your browser without running a full Ethereum node.

Once these tools are installed, the back-end part of the project can be run and tested locally, miming the effect of working on a live blockchain.

As soon as the application was running smoothly locally we tried also to launch it on a real blockchain. We used **Ropsten**, a test net that exploits the same protocol used by Ethereum. Also, with **Infura**, a hosted Ethereum node, we were able to run the DApp without having to set up an Ethereum node or wallet. The last thing we needed was some sample ether to use. We took those from the **Matamask Faucet**.

Regarding the front-end, three Javascript libraries were mainly used: **Bootstrap**, to build a responsive and mobile-friendly application, **jQuery**, to select and manipulate elements and to handle events, and **web3.js**, used to interact with a local or remote ethereum node.

At last, the code editor Visual Studio Code was found very useful as it's very easy to switch between Solidity, Javascript, HTML and CSS since it offers a syntax checker and highlighter for all these languages.

For a reference to find these tools, there are the links in the bibliography in section 6.

1.2 Project Structure

The project is structured in three folders. The other files are just some utility scripts needed to run the truffle suite and a lite-server.

1. *contracts* - in this directory are kept all our Solidity contracts and an additional Migrations contract needed in order to use Truffle's Migrations feature.
2. *migrations* - Truffle uses a migration system to handle smart contract deployments. In a nutshell, these files are used to initialize the contracts and perform some initial operations on the blockchain.
3. *src* - the core of the front-end application is stored in this folder: all the Javascript, HTML and CSS files used to build the graphic interface of the DApp.

2 Extension of the Smart Contracts

2.1 BaseContent

Data Stored

The introduction of additional functionalities implied the creation of new variables.

The major requirement for this project was the implementation of a rating system. As shown in the code snippet below, we introduced three variables to store the data needed.

```
1  uint32[4] public feed; // 0: overall, 1: price, 2: quality, 3: details
2  uint32 public nVotes; // number of people who voted
3  mapping (address => bool) public has_consumed; // list of addresses who can vote
```

The first one is an array with four positions, one for each feedback type:

1. *overall* - it's a measure of general appreciation of the content;
2. *price* - it evaluates the price fairness, hence if the cost required for the content is worth of the product offered;
3. *quality* - it defines if the resolution or the audio quality, for example, are as expected;
4. *details* - it checks if there is enough information about the product to understand what it is sold.

For each category the customers can leave a vote from 1 to 5, that correspond respectively to minimum and maximum appreciation. Since we are not interested in the singular votes but just of the average in a category we simply store in each position the sum of all the votes left. Then, in order to compute the mean, we just need *nVotes* to retrieve the current number of votes.

However, only customers who has consumed the content are allowed to leave a feedback. So it was created a mapping to store the addresses authorized to vote.

Another assignment specification was to allow creators to define the price of their contents. Therefore the variable *price* was created and added to the parameters of the constructor.

Finally, the last variable inserted was *subgenre*. This is an optional field that an author can decide to add or not. Since the genre is a macro information about the content, we thought it was more realistic to insert also an additional information to give customers a more concrete idea of the product.

Events Triggered

Three events are triggered by the *BaseContent*. The first two were already present in the final-term to notify that the content has been consumed and that the minimum number of views to be paid has been reached. The last one was added to inform that a feedback has been left.

Actually, these don't really trigger an event, they call a function of the catalog called *EmitEvent()*. Later we will explain better how this function works and why we decided to not emit directly an event.

Functions

Also a set of functions has been added, they are quite trivial but they simplify the management of the contracts.

First of all, it was written a view function that checks if an account is allowed to vote. Indeed it must be checked that a customer has consumed the content first and that he hasn't vote already.

Another useful function is *ContentType()*. This simply returns the name of the contract or an empty string if the contract used is a *BaseContent*. Indeed we decided to extend the *BaseContent* with some subclasses, and, with this function, is very quick to understand which contract has been initialized.

Finally, *LeaveRate()* updates the feedback array and the number votes and *SetSubgenre()* can be called by the owner of the contract to insert this additional field.

2.2 Catalog

Data Stored

In this project, while the price for a single content can be set, the cost and duration of a premium subscription are still set once the *Catalog* is created. Since the release of the final-term the average block time hasn't changed much, so we decided to keep the duration of the premium subscription to 40000 blocks, which is on average a week. On the other hand, the value of an Ether has decreased a lot. Just 6 months ago the value of an Ether was about 400 Euro, now is just slightly above 100 Euro. We decided to raise adequately the cost of premium getting it up to 0.25 Ether, which is right now approximately 20 Euro.

However it's easy to see that setting a fixed price for premium is not a feasible solution. Computing dynamically a price based on the value of an Ether is not an easy task and still doesn't solve the problem as it does not consider the possible inflation of the currency considered. Another option could have been to insert a function that allows the catalog owner to change the price to a more fair one. Anyhow this solution goes against the project directions as the catalog owner should not have additional power besides creating and deleting the catalog.

In the end, since these solutions weren't satisfying, we left a fixed price for premium subscription but it could be a case of study to find a better way to solve the problem.

The other modifications to the *Catalog* were the introduction of two mappings: *notifications_to_see* and *notifications_preferences*. The first stores for each users' address the block number of the last event seen, while the second keeps an array of the settings chosen.

```
1 mapping (address => uint) public notifications_to_see; // map from account ...  
   to last block seen  
2 mapping (address => bytes32[]) public notifications_preferences; // map from ...  
   account to notification setting
```

They are required to save the data of an account regarding the events triggered. Indeed, in the front-end application a user can display all the events emitted since his last login. But, in order to have consistent values, this data cannot be saved locally in the browser. The best way would have been to store it in a central server instead of using the blockchain, as it is not sensitive information. However, for the purpose of the project, it was easier to store everything in the blockchain. Still, this means that every time these mappings are modified we have to pay for a transaction.

Events Triggered

The events triggered by the *Catalog* are the same as the final-term: *new_publication*, *content_acquired*, *premium_acquired* and *author_payed*. We just enriched the set of parameters given to each event, this way we are able to apply some sort of filter to each event so that an account receives only the ones concerning itself.

Functions

Since we have defined a rating system it is also necessary to create a set of functions to access these votes and perform some queries based on the value of the feedback.

In the project it was required to insert some functions that returns the content with highest rating, in particular the user must be able to decide whether to specify a precise feedback category or get the average of all, and to filter the research by a specific genre or author.

In Solidity there are no native floating point types such as float or double. So, before computing every division, in order to get one decimal digit we multiply the numerator by 10. It will be later, in the JavaScript files that handles the front-end, that the inverse operation will be performed in order to get again a value in the range [1, 5].

As we said before, we created a function *EmitEvent()* that can be called by other contracts. This function is characterized by an eventID and some other parameters to describe the event triggered. So, once this function is called it will simply check the eventID and will emit the appropriate event. This trick simplifies the management of events from the front-end. Indeed, instead of having to listen to the events emitted by all the contracts we just have to listen to the ones of the catalog.

In the end, besides adding some simple getters or setters, we have modified the function *PayAuthor()*. Compared to the final-term, the author can still be paid each n views but this time it is not rewarded with the whole price of the content but with a fraction of it. In particular, the price given for each view is computed as the cost for the content multiplied by its average rating over the maximum rating possible.

2.3 Additional Contracts

We decided to insert some other types of contracts for a content. In addition to the *BaseContent*, that contains only the basic functionalities, we wrote three subclasses: *MovieContent*, *PhotoContent* and *SongContent*.

These are just some examples of how the *BaseContent* can be extended. They contain a fixed string for the genre and they add some specific fields. For example in the *MovieContent* we inserted a field *movie_length*. The possibility of having specific functions and variables for a contract allow us to customized the front-end based on the type of contract initialized.

3 Implementation of the Front-End

The front-end is structured in a series of HTML files, each with a corresponding JavaScript to handle the dynamic behaviour of the interface.

The main page is *index.html* where the user can scroll through the contents in the catalog, see some general information or select them to open the page of each specific content. In the top part there is a navigation bar that allows the user to surf through the application. It is divided in different sections:

- *Author* - where an artist can have a look of all the contents he has uploaded or insert a new one in the catalog. To do so, he must fill the form with the four required fields (title, author, genre and price) and with some optional ones. Some additional fields may appear according to the

genre selected. For example, if it is set to *Movie*, it is possible also to set its length. Once the form is completed the author can deploy the contract and subsequently add it to the catalog.

- *Premium* - this page simply allows you to purchase or gift to someone a premium subscription.
- *Search* - the search of a content is divided in two: a simple one using the title of the content or a smart one using the catalog statistics. It is possible therefore to search for the most recent, popular or rated content and filter these results by different categories.
- *Personal Area* - in this page are collected a series of data regarding the user. First of all its address and whether it's a standard or premium account. In the latter case it will also be displayed the number of blocks before expiration. Subsequently there are two lists: one for the contents the user can consume and another for the ones he can rate. In this page are also shown the notification customized to the user. In the top right corner is inserted a badge with the number of unseen notification, while the description of the event is written below.

Other pages are not linked in the navigation bar but can still be reached. There is a page for each content that can be found by clicking on the content itself. From that we can find a page for leaving a rate if the user is allowed.

In a quest to make the interface more user-friendly, we decided to let some buttons visible only by the people who are authorized to press them. For example, the button for deleting the catalog is shown only if the account logged correspond to the catalog owner. The ones for consuming or rating a content are present only if the user has respectively purchased or consumed the content.

The JavaScript files have all the same structure: first there is the declaration of the variables, then the implementation of the render function, the management of buttons events and some functions just to make the code more readable. Only two files don't follow this scheme: *utilities.js* and *app.js*. Both are linked in all HTML pages. The first is just a collection of function to avoid duplication, the second is needed to set some variables as the web3Provider, the contracts and the current account. Once it has done that, it calls the render function that is implemented differently for each page. This function has the purpose of computing the operations required to render the dynamic part of the page on opening.

4 Malicious Strategies

The new payment system introduced in this project opens the door to some malicious behaviours. Since the gain of an author is directly proportional to the rating value of the content, an author may perform a lot of views on their contents and leave very high ratings. The cost of performing those views can be regained if the improvement of the rating is substantial.

In the same way, they may perform a lot of views on an opponent's content and leave low votes. This action reduces the amount gained by the other person. However, in order to do so you would have to spend the price of the content and not get any reward. The only way to gain the money back is indirectly. If you discredit an opponent that offers a content similar to yours you may end up receiving more views than you usually should.

5 Deployment

In order to test the DApp it is necessary first to download and install the tools described in Section 1.1.

We inserted in the first part of the Bibliography (Section 6.1) some tutorials to have some guidance. In particular, *The Ultimate Ethereum Dapp Tutorial*¹ of the DApp University is very useful to understand

¹ www.dappuniversity.com/articles/the-ultimate-ethereum-dapp-tutorial

how to install the tools for executing locally, while the one for Infura in the Truffle Suite is a complete guide for running the application on the Ropsten testnet.

The following set of actions must be done in order to install the project:

1. Either clone or download the project from my github page²;
2. Enter the project folder and open a terminal window;
3. Run the command `$npm install truffle` to install the folder `node_modules` needed to access node.js library and use its functionalities;
4. Run the command `$npm install` to install the dependencies required for the project.

5.1 Local Execution

Now that the project is ready you can either run it locally or on a real blockchain. If you want to test it locally you can:

1. Open Ganache;
2. Open a browser window and access to your Metamask account and connect to the local network `HTTP://127.0.0.1:7545`;
3. Import from Ganache an account and add it to Metamask;
4. In the terminal opened before run enter the command `$truffle migrate --reset` to compile and deploy the project. It will create a folder `build` to store a json image for each contract containing the ABI (Application Binary Interface) and other variables needed to perform contracts' calls;
5. Execute `$npm run dev` to launch the lite-server and run the application;
6. If required, accept the privacy agreement for Metamask;
7. If, while performing the first operation on the DApp, Metamask triggers an error connected to payload it means that the account must be reset. To do so click on the Metamask Icon > Settings > Reset Account.

5.2 Ropsten Execution

The execution on Ropsetn is a little different from the local one, you have to:

1. Execute the command `$npm install truffle-hdwallet-provider` to install Truffle's HDWalletProvider package;
2. Connect to you Metamask account and select the Ropsten Test Network;
3. Retrieve your Infura private key;
4. Retrieve the twelve words seed from your account Metamask;
5. Then you can either create inside the project folder a file json called `private.keys.json` with two fields: `infuraKey` and `mnemonic` that contains the data from the previous two points or change in the file `truffle.js` the value of the constants `infuraKey` and `mnemonic` with those values as shown in the Infura tutorial.

² github.com/fsbolgi/COBrA-DApp

6. Connect to the Metamask Faucet to request some ether;
7. Compile and deploy the application by running the command `$struffle migrate --network ropsten`;
8. Open the application by executing `$npm run dev`;
9. If you want you can check the transactions executed using Etherscan.

5.3 System Simulation

In Figure 1 is shown the author section of the DApp.

COBrA

CATALOG

AUTHOR

PREMIUM

SEARCH

PERSONAL AREA

#

TITLE

AUTHOR

GENRE

PRICE

VIEWS

RATING

1

Pretty Shining People

George Ezra

Song

20000000000000000000wei

2

★★★★☆

3

Harry Potter

David Yates

Movie

11000000000000000000wei

5

★★★★★

5

And Then There Where None

Agatha Christie

Book

42000000000000000000wei

3

★★★★★

6

Happy

Pharrell

Song

20000000000000000000wei

4

★★★★★

ADD A NEW CONTENT

Insert the informations about the content you want to deploy and add to the catalog:

TITLE*

The title of the content

1

AUTHOR*

The author's name

2

GENRE*

3

PRICE*

The price required to be payed

4

SUBGENRE

5

* Required Fields

DEPLOY CONTRACT

6

ADD TO CATALOG

7

Figura 1: Screenshot of the Author's page.

The blue arrows indicates a set of actions that can be performed:

1. By clicking on the Catalog tab you can have a look of all the contents present;
2. At the top right corner is stored the number of events you still haven't seen. You can access your personal page to show them or see other information relative to your account;
3. In this table there is the list of contents you have uploaded, you can click on one of them to access the content page;
4. You can insert also a new one. The red arrows show the procedure:
 - (a) Insert the title of the content;
 - (b) Add the author;
 - (c) Choose one of the genre available: Movie, Song, Photo and Other. If Other is selected an additional field will appear where you can insert the chosen one;
 - (d) Digit the price in wei;
 - (e) If you want, optionally, you can insert also a subgenre;
 - (f) Deploy the contract;
 - (g) Once the deployment is done you can add the content to the catalog.

6 Bibliography

6.1 Tutorials for getting started

Crypto Zombies - <https://cryptozombies.io/en/course/>

DApp University - dappuniversity.com

Truffle - truffleframework.com/tutorials/pet-shop

Infura - truffleframework.com/tutorials/using-infura-custom-provider

6.2 Tools for Running Smart Contracts

Remix - remix.ethereum.org

Truffle - truffleframework.com/truffle

Ganache - truffleframework.com/ganache

NPM - nodejs.org/en

Metamask - metamask.io

Ropsten - ropsten.etherscan.io

Infura - infura.io

Metamask Faucet - faucet.metamask.io

Etherscan - ropsten.etherscan.io

6.3 Tools for Creating the front-end

Bootstrap - getbootstrap.com

jQuery - jquery.com

web3.js - web3js.readthedocs.io/en/1.0

Visual Studio Code - code.visualstudio.com

7 Smart Contracts Code

7.1 BaseContent

```
1  pragma solidity ^0.4.19;
2
3  import "./Catalog.sol";
4
5  contract BaseContent {
6
7      /* VARIABLES DECLARATION */
8
9      /* data about the contract itself */
10     address public owner; // address of who uploads
11     address public content_address; // address of the contract
12     Catalog public catalog; // catalog where the content is published
13
14     /* data about the content */
15     bytes32 public title; // unique ID
16     bytes32 public author; // name of the author
17     bytes32 public genre; // indicates the type {song, video, photo, etc.}
18     bytes32 public subgenre = 0x00; // optionally can be set
19     uint public price; // the price of the content
20     uint32 public view_count = 0; // total number of views
21     uint32 public views_already_payed = 0; // number of views already payed by ...
22         the catalog
23
24     /* data about feedback */
25     uint32[4] public feed; // 0: overall, 1: price, 2: quality, 3: details
26     uint32 public nVotes; // number of people who voted
27     mapping (address => bool) public has_consumed; // list of addresses who can vote
28
29     /* data about the customers authorized to access the content */
30     mapping (address => bool) private authorized_std; // list of standard ...
31         accounts who can see
32     mapping (address => uint) private authorized_premium; // list of premium who ...
33         can see
34
35     /* modifiers that enforce that some functions can be called just by the ...
36         Catalog */
37     modifier byCatalog() {
38         require(msg.sender == address(catalog));
39         _;
40     }
41
42     modifier byOwner() {
43         require(msg.sender == owner);
44         _;
45     }
46
47     /* CONSTRUCTOR */
48
49     constructor (Catalog _catalog, bytes32 _title, bytes32 _author, bytes32 ...
50         _gen, uint _price) public {
51         owner = msg.sender;
52         content_address = this;
53         catalog = _catalog;
54         title = _title;
55         author = _author;
56         genre = _gen;
57         price = _price;
58     }
59
60     /* VIEW FUNCTIONS */
61
62     /* checks if an account is authorized to consume the content */
63     function IsAuthorized() public view returns (bool){
```

```

61     return authorized_std[msg.sender] || authorized_premium[msg.sender] > ...
        block.number;
62 }
63
64 /* checks if an account is authorized to leave a feedback */
65 function CanVote() public view returns (bool){
66     return has_consumed[msg.sender];
67 }
68
69 /* dummy function overwritten by subclasses to understant contract class */
70 function ContentType() public view returns (bytes32) {
71     return "";
72 }
73
74
75 /* STATE MODIFYING FUNCTIONS */
76
77 /* insert the customer address as authorized to access this content */
78 function Authorize(address _customer, uint _expiration_date) external byCatalog{
79     if (_expiration_date == 0) {
80         authorized_std[_customer] = true;
81     } else {
82         authorized_premium[_customer] = _expiration_date;
83     }
84 }
85
86 /* called by the catalog to mark the views already payed */
87 function Payed(uint32 _views_just_payed) external byCatalog {
88     views_already_payed = views_already_payed + _views_just_payed;
89 }
90
91 /* authorized customers can consume the content */
92 function ConsumeContent() external {
93     require(IsAuthorized());
94     if (authorized_std[msg.sender]) {
95         view_count ++;
96         if (view_count % catalog.min_v() == 0) {
97             catalog.EmitEvent(0, owner, title, (view_count - ...
views_already_payed)); // min_v_reached
98         }
99         delete authorized_std[msg.sender];
100     } else {
101         delete authorized_premium[msg.sender];
102     }
103     has_consumed[msg.sender] = true;
104     catalog.EmitEvent(1, msg.sender, title, 0); //content_consumed
105 }
106
107 /* customers who has consumed can leave a feedback */
108 function LeaveRate(uint32[] f) external {
109     require(CanVote());
110     for (uint32 i = 0; i < 4; i++) {
111         feed[i] = feed[i] + f[i];
112     }
113     nVotes ++;
114     delete has_consumed[msg.sender];
115     catalog.EmitEvent(2, msg.sender, title, 0); //rate_left
116 }
117
118 /* allows the owner to set a subgenre for the content */
119 function SetSubgenre (bytes32 _s) external byOwner {
120     subgenre = _s;
121 }
122 }

```

7.2 MovieContent

```

1 pragma solidity ^0.4.19;
2

```

```

3  import "./BaseContent.sol";
4
5  contract MovieContent is BaseContent {
6
7      /* VARIABLES DECLARATION */
8
9      uint32 public movie_length; // duration of the movie in minutes
10     bytes32 public g = ...
        0x4d6f766696500000000000000000000000000000000000000000000000000000; // ...
        string "Movie"
11
12     /* CONSTRUCTOR */
13
14     /* inherits the contractor from Base Content */
15     constructor(Catalog _catalog, bytes32 _title, bytes32 _author, uint _price) ...
        BaseContent (_catalog, _title, _author, g, _price) public {
16         movie_length = 0;
17     }
18
19     /* FUNCTIONS */
20
21     /* utility function to understand contract class */
22     function ContentType() public view returns (bytes32) {
23         return g;
24     }
25
26     /* allows the owner to insert an additional information */
27     function SetMovieLength(uint32 _l) external byOwner {
28         movie_length = _l;
29     }
30 }

```

7.3 PhotoContent

```

1  pragma solidity ^0.4.19;
2
3  import "./BaseContent.sol";
4
5  contract PhotoContent is BaseContent {
6
7      /* VARIABLES DECLARATION */
8
9      uint32 public n_pixel; // number of pixels of the larger side
10     bytes32 public g = ...
        0x50686f746f000000000000000000000000000000000000000000000000000000; // ...
        string "Photo"
11
12     /* CONSTRUCTOR */
13
14     /* inherits the contractor from Base Content */
15     constructor(Catalog _catalog, bytes32 _title, bytes32 _author, uint _price) ...
        BaseContent (_catalog, _title, _author, g, _price) public {
16         n_pixel = 0;
17     }
18
19     /* FUNCTIONS */
20
21     /* utility function to understand contract class */
22     function ContentType() public view returns (bytes32) {
23         return g;
24     }
25
26     /* allows the owner to insert an additional information */
27     function SetNPixel(uint32 _n) external byOwner {
28         n_pixel = _n;
29     }
30 }

```

7.4 SongContent

```
1 pragma solidity ^0.4.19;
2
3 import "./BaseContent.sol";
4
5 contract SongContent is BaseContent {
6
7     /* VARIABLES DECLARATION */
8
9     uint32 public track_length; // duration of the song in seconds
10    bytes32 public g = ...
        0x536f6e6700000000000000000000000000000000000000000000000000000000; // ...
        string "Song"
11
12    /* CONSTRUCTOR */
13
14    /* inherits the contractor from Base Content */
15    constructor(Catalog _catalog, bytes32 _title, bytes32 _author, uint _price) ...
        BaseContent(_catalog, _title, _author, g, _price) public {
16        track_length = 0;
17    }
18
19    /* FUNCTIONS */
20
21    /* utility function to understand contract class */
22    function ContentType() public view returns (bytes32) {
23        return g;
24    }
25
26    /* allows the owner to insert an additional information */
27    function SetTrackLength(uint32 _l) external byOwner {
28        track_length = _l;
29    }
30 }
```

7.5 Catalog

```
1 pragma solidity ^0.4.19;
2
3 import "./BaseContent.sol";
4
5 contract Catalog {
6
7     /* VARIABLES DECLARATION */
8
9     /* data about the contract itself */
10    address public owner; // address of who uploads
11    address private catalog_address; // address of the contract
12
13    /* data about the catalog */
14    BaseContent[] public contents_list; // list of all the contents in the catalog
15    mapping (bytes32 => uint) public position_content; // map from title to ...
        position + 1
16    uint private time_premium = 40000; // premium lasts approximately one week
17    uint public cost_premium = 0.25 ether; // premium costs about 20 euro
18    uint32 public min_v = 100; // number of views required before payment
19
20    /* data about the accounts */
21    mapping (address => uint) public premium_customers; // map from premium to ...
        expiration date
22    mapping (address => uint) public notifications_to_see; // map from account ...
        to last block seen
23    mapping (address => bytes32[]) public notifications_preferences; // map from ...
        account to notification setting
24
25    /* events triggered */
26 }
```

```

26 event new_publication (bytes32 _title, bytes32 _author, bytes32 _genre, ...
    address _owner);
27 event content_acquired (bytes32 _title, address _sender, address _receiver, ...
    uint32 _gifted);
28 event premium_acquired (address _sender, address _receiver, uint32 _gifted);
29 event author_payed (bytes32 _t, address _owner, uint _tot_money);
30 event min_v_reached (address _account, bytes32 _title, uint _v);
31 event content_consumed (address _customer, bytes32 _title);
32 event rate_left(address _customer, bytes32 _title);
33
34
35 /* CONSTRUCTOR */
36
37 constructor () public {
38     owner = msg.sender;
39     catalog_address = address(this);
40 }
41
42
43 /* VIEW FUNCTIONS */
44
45 /* returns the number of contents in the catalog */
46 function GetLengthCatalog() public view returns (uint) {
47     return contents_list.length;
48 }
49
50 /* returns the author of a content */
51 function GetAuthor(bytes32 _t) public view returns (bytes32) {
52     uint i = position_content[_t];
53     if (i != 0) {
54         return contents_list[i-1].author();
55     }
56 }
57
58 /* returns the author of a content */
59 function GetGenre(bytes32 _t) public view returns (bytes32) {
60     uint i = position_content[_t];
61     if (i != 0) {
62         return contents_list[i-1].genre();
63     }
64 }
65
66 /* returns the price of a content */
67 function GetPrice(bytes32 _t) public view returns (uint){
68     uint i = position_content[_t];
69     if (i != 0) {
70         return contents_list[i-1].price();
71     }
72 }
73
74 /* returns the number of views of a content */
75 function GetViews(bytes32 _t) public view returns (uint32){
76     uint i = position_content[_t];
77     if (i != 0) {
78         return contents_list[i-1].view_count();
79     }
80 }
81
82 /* returns the owner of a content */
83 function GetOwner(bytes32 _t) public view returns (address){
84     uint i = position_content[_t];
85     if (i != 0) {
86         return contents_list[i-1].owner();
87     }
88 }
89
90 /* returns the average rating for a content from 0 to 50 */
91 function GetRate(bytes32 _t) public view returns (uint32) {
92     uint i = position_content[_t];
93     uint32 s = 0;
94     BaseContent bc = contents_list[i-1];
95     for(uint32 j = 0; j < 4; j++){
96         s += bc.feed(j);

```

```

97     }
98     if (bc.nVotes() != 0 ) {
99         return ((s * 10) / (bc.nVotes() * 4));
100    } else {
101        return 0;
102    }
103 }
104
105 /* returns the list of notification preferences of an account */
106 function GetNotifPreferences(address _u) public view returns (bytes32[] ...
107     memory) {
108     return notifications_preferences[_u];
109 }
110
111 /* returns the number of views for each content */
112 function GetStatistics() public view returns (bytes32[] memory, uint32[] ...
113     memory){
114     uint l_length = GetLengthCatalog();
115     bytes32[] memory titles_list = new bytes32[](l_length);
116     uint32[] memory views_list = new uint32[](l_length);
117     for (uint i = 0; i < l_length; i++) {
118         titles_list[i] = contents_list[i].title();
119         views_list[i] = contents_list[i].view_count();
120     }
121     return (titles_list, views_list);
122 }
123
124 /* returns the list of contents without the number of views */
125 function GetContentList() public view returns (bytes32[] memory) {
126     uint l_length = GetLengthCatalog();
127     bytes32[] memory titles_list = new bytes32[](l_length);
128     for (uint i = 0; i < l_length; i++) {
129         titles_list[i] = contents_list[i].title();
130     }
131     return titles_list;
132 }
133
134 /* returns the list of x newest contents */
135 function GetNewContentsList(uint x) public view returns (bytes32[] memory) {
136     uint l_length = GetLengthCatalog();
137     require(x <= l_length);
138     bytes32[] memory titles_list = new bytes32[](x);
139     if (l_length > 0 && x > 0) {
140         for (uint i = l_length - 1; i >= l_length-x; i--) {
141             titles_list[l_length - 1 - i] = contents_list[i].title();
142             if (i == 0) {
143                 return titles_list;
144             }
145         }
146     }
147     return titles_list;
148 }
149
150 /* returns the most recent content with genre _g */
151 function GetLatestByGenre(bytes32 _g) public view returns (bytes32) {
152     uint l_length = GetLengthCatalog();
153     if (l_length > 0) {
154         for (uint i = l_length - 1; i >= 0; i--) {
155             if (contents_list[i].genre() == _g) {
156                 return contents_list[i].title();
157             }
158             if (i == 0) {
159                 return 0x0;
160             }
161         }
162     }
163     return 0x0;
164 }
165
166 /* returns the content with genre _g with max views */
167 function GetMostPopularByGenre(bytes32 _g) public view returns (bytes32) {
168     bytes32 most_popular = 0;
169     uint32 most_views = 0;

```

```

168     for (uint i = 0; i < GetLengthCatalog(); i++) {
169         if (contents_list[i].genre() == _g && contents_list[i].view_count() > ...
most_views) {
170             most_popular = contents_list[i].title();
171             most_views = contents_list[i].view_count();
172         }
173     }
174     return most_popular;
175 }
176
177 /* returns the most recent content of the author _a */
178 function GetLatestByAuthor(bytes32 _a) public view returns (bytes32) {
179     uint l_length = GetLengthCatalog();
180     if (l_length > 0) {
181         for (uint i = l_length - 1; i >= 0; i--) {
182             if (contents_list[i].author() == _a) {
183                 return contents_list[i].title();
184             }
185             if (i == 0) {
186                 return 0x0;
187             }
188         }
189     }
190     return 0x0;
191 }
192
193 /* returns the content with most views of the author _a */
194 function GetMostPopularByAuthor(bytes32 _a) public view returns (bytes32){
195     bytes32 most_popular = 0;
196     uint32 most_views = 0;
197     for (uint i = 0; i < GetLengthCatalog(); i++) {
198         if (contents_list[i].author() == _a && contents_list[i].view_count() > ...
most_views) {
199             most_popular = contents_list[i].title();
200             most_views = contents_list[i].view_count();
201         }
202     }
203     return most_popular;
204 }
205
206 /* returns the content with most views */
207 function GetMostPopular() public view returns (bytes32){
208     bytes32 most_popular = 0;
209     uint32 most_views = 0;
210     for (uint i = 0; i < GetLengthCatalog(); i++) {
211         if (contents_list[i].view_count() > most_views) {
212             most_popular = contents_list[i].title();
213             most_views = contents_list[i].view_count();
214         }
215     }
216     return most_popular;
217 }
218
219 /* returns the content with highest rating for feedback category _y
(or highest average if y is greater than 4) */
220 function GetMostRated(uint32 _y) public view returns (bytes32){
221     bytes32 most_rated = 0;
222     uint32 highest_rating = 0;
223     for (uint i = 0; i < GetLengthCatalog(); i++) {
224         BaseContent bc = contents_list[i];
225         if (bc.nVotes() != 0) {
226             uint32 r = (_y < 4)? ((bc.feed(_y) * 10) / bc.nVotes()) : ...
GetRate(bc.title());
227             if (r > highest_rating) {
228                 most_rated = bc.title();
229                 highest_rating = r;
230             }
231         }
232     }
233     return most_rated;
234 }
235 }
236
237 /* returns the content with highest rating for feedback category _y

```



```

238 (or highest average if y is greater than 4) with genre _g */
239 function GetMostRatedByGenre(bytes32 _g, uint32 _y) public view returns ...
240 (bytes32){
241     bytes32 most_rated = 0;
242     uint32 highest_rating = 0;
243     for (uint i = 0; i < GetLengthCatalog(); i++) {
244         BaseContent bc = contents_list[i];
245         if (bc.genre() == _g && bc.nVotes() != 0) {
246             uint32 r = (_y < 4)? ((bc.feed(_y) * 10) / bc.nVotes()) : ...
247             GetRate(bc.title());
248             if (r > highest_rating) {
249                 most_rated = bc.title();
250                 highest_rating = r;
251             }
252         }
253     }
254     return most_rated;
255 }
256
257 /* returns the content with highest rating for feedback category _y
258 (or highest average if y is greater than 4) with author _a */
259 function GetMostRatedByAuthor(bytes32 _a, uint32 _y) public view returns ...
260 (bytes32){
261     bytes32 most_rated = 0;
262     uint32 highest_rating = 0;
263     for (uint i = 0; i < GetLengthCatalog(); i++) {
264         BaseContent bc = contents_list[i];
265         if (bc.author() == _a && bc.nVotes() != 0) {
266             uint32 r = (_y < 4)? ((bc.feed(_y) * 10) / bc.nVotes()) : ...
267             GetRate(bc.title());
268             if (r > highest_rating) {
269                 most_rated = bc.title();
270                 highest_rating = r;
271             }
272         }
273     }
274     return most_rated;
275 }
276
277 /* returns true if _c is a valid premium account */
278 function isPremium (address _c) public view returns(bool) {
279     return premium_customers[_c] > block.number;
280 }
281
282 /* STATE MODIFYING FUNCTIONS */
283
284 /* allows to set the last block seen by an account */
285 function SetNotification(address _a, uint _x) external {
286     notifications_to_see[_a] = _x;
287 }
288
289 /* set a list of preferences for an account */
290 function SetNotifPreferences(address _u, bytes32[] memory _list) public {
291     notifications_preferences[_u] = _list;
292 }
293
294 /* allows other contracts to emit events but listen only to the catalog */
295 function EmitEvent(uint _eventID, address _account, bytes32 _t, uint _v) ...
296 public {
297     if (_eventID == 0) {
298         emit min_v_reached(_account, _t, _v);
299     } else if (_eventID == 1) {
300         emit content_consumed(_account, _t);
301     } else {
302         emit rate_left(_account, _t);
303     }
304 }
305
306 /* submits a new content _c to the catalog */
307 function AddContent(BaseContent _c) external {
308     BaseContent cm = _c;
309     require (cm.owner() == msg.sender && cm.catalog() == catalog_address);

```

```

306     contents_list.push(cm);
307     position_content[cm.title()] = contents_list.length;
308     emit new_publication (cm.title(), cm.author(), cm.genre(), cm.owner());
309 }
310
311 /* standard accounts pays to access content _t */
312 function GetContent(bytes32 _t) external payable {
313     require(msg.value == GetPrice(_t));
314     uint i = position_content[_t];
315     if (i != 0) {
316         contents_list[i-1].Authorize(msg.sender, 0);
317         emit content_acquired (_t, msg.sender, msg.sender, 0);
318     }
319 }
320
321 /* premium accounts can requests access to content _t without paying */
322 function GetContentPremium(bytes32 _t) external {
323     require(isPremium(msg.sender));
324     uint i = position_content[_t];
325     if (i != 0) {
326         contents_list[i-1].Authorize(msg.sender, premium_customers[msg.sender]);
327         emit content_acquired (_t, msg.sender, msg.sender, 0);
328     }
329 }
330
331 /* pays for granting access to a content _t to the user _u */
332 function GiftContent(bytes32 _t, address _u) external payable {
333     require(msg.value == GetPrice(_t));
334     uint i = position_content[_t];
335     if (i != 0) {
336         contents_list[i-1].Authorize(_u, 0);
337         emit content_acquired (_t, msg.sender, _u, 1);
338     }
339 }
340
341 /* buys a new premium subscription */
342 function BuyPremium() external payable {
343     require(msg.value == cost_premium);
344     premium_customers[msg.sender] = block.number + time_premium;
345     emit premium_acquired (msg.sender, msg.sender, 0);
346 }
347
348 /* pays for granting a premium account to the user _u */
349 function GiftPremium(address _u) external payable {
350     require(msg.value == cost_premium);
351     premium_customers[_u] = block.number + time_premium;
352     emit premium_acquired (msg.sender, _u, 1);
353 }
354
355 /* pay an author a multiple of v views*/
356 function PayAuthor(bytes32 _t) external {
357     uint i = position_content[_t];
358     if (i != 0) {
359         uint32 tot_views = contents_list[i-1].view_count();
360         uint32 to_pay = tot_views - contents_list[i-1].views_already_payed();
361         if (to_pay >= min_v) {
362             uint256 price_per_view = contents_list[i-1].price() * GetRate(_t)/50;
363             contents_list[i-1].owner().transfer(to_pay * price_per_view);
364             contents_list[i-1].Payed(to_pay);
365             emit author_payed (_t, contents_list[i-1].owner(), to_pay * ...
366                 price_per_view);
367         }
368     }
369 }
370
371 /* returns the total number of views and the total number of views that has ...
372    to payed */
373 function GetTotalViews() private view returns (uint) {
374     uint tot_views = 0;
375     for (uint i = 0; i < GetLengthCatalog(); i++) {
376         tot_views = tot_views + contents_list[i].view_count();
377     }
378     return tot_views;

```

```

377 }
378
379 /* the catalog can be destructed */
380 function KillCatalog() external {
381     require(msg.sender == owner);
382     uint tot_views = GetTotalViews();
383     uint money_per_view = catalog_address.balance / tot_views;
384     for (uint i = 0; i < GetLengthCatalog(); i++) {
385         uint proportional_money = contents_list[i].view_count() * money_per_view;
386         contents_list[i].owner().transfer(proportional_money);
387         emit author_payed (contents_list[i].title(), contents_list[i-1].owner(), ...
proportional_money);
388     }
389     selfdestruct(catalog_address);
390 }
391 }

```

8 DApp Code

The code of the front-end was too long to be inserted in the report. The complete source code can be found in the folder attached on moodle or on my github page³.

³ github.com/fsbolgi/COBrA-DApp