Peer-to-peer and blockchain project report

Luca Di Mauro - 477943

November 1, 2018

Contents

Introduction

This brief report describes the work behind the realization of COBrA project. This project provides the user with a graphic interface to interact with the smart contracts, which were developed for the final term and were subsequently extended.

In order to do so, this paper is divided into three main sections.

The first section will describe the extension of smart contracts, i.e. the categories and user's feedback functionalities.

The second section describes the actual COBrA project, its features, its structure and its execution.

The third section describes how to deploy Catalog contract with Truffle framework and how to deploy or destroy Catalog contract with *catalogManager* javascript program.

1 Smart contracts extensions

First of all, smart contracts are extended to support new requested features: a user, who already bought and consumed a content, optionally can leave a feedback about that content.

To perform this feature, three feedback types are defined:

- **content appreciation**, representing how much the customer enjoyed the content;
- **price fairness**, representing how fair the requested price is considered compared to the content;
- **suitable for children**, representing how much the content is suitable for children;

The user leaves the feedback as an integer value but, since solidity does not support float values, the functions which compute a feedback average of a specific content return an integer value, even if computed value would be a float value.

Finally the content's creators have the possibility to choose the price of their own contents. The author rewards is computed starting from the chosen price and it is multiplied by the content's average feedback: thus, more a content is appreciated, more the author will be rewarded.

Since the payment system is based on the average feedback and a premium user can take any content without pay, a malicious author can increase its revenues by becoming a premium user and starting repeatedly to buy and to leave the maximum feedback to its contents. Obviously the price of the content have to be as higher as possible to maximize the revenues and, if it isn't enough high, it would be possible to spent more ether than those earned. On the other hand, a malicious customer can decrease other authors' revenue thanks to the same scheme: become a premium user, repeatedly buy contents and leave them the minimum feedback.

2 COBrA DApp development

2.1 Libraries and frameworks

The DApp is developed using *Electron* [1], a framework to create desktop application with HTML, CSS and Javascript. It provides two different processes which manage the same application: one renders the graphic and the other one interacts with file system and other functionalities.
These two processes communicate through asynchronous events.

To interact with the Ethereum blockchain, *Web3js*[2] library is used. It provides an interface to the blockchain implementing the **JSON RPC API**[3] thus providing a set of functions to interact with an RPC endpoint (like *geth* program).

Along with this library, the *Truffle* [4] suite was used because it provides a more powerful abstraction of smart contracts and functions.

The explained frameworks work thanks to the *NodeJS* engine[5]. It allows to run a Javascript engine locally without using a web browser and, through *Node Package Manager* program [6], to download and run libraries, frameworks and programs written in Javascript which require NodeJS.

For the user interface side, *Semantic UI* framework [7] was used. It helps to create web user interface with HTML, CSS and Javascript.

2.2 Project structure

The project is divided into two main directories: *electron* and *truffle.* This separation is needed because the Electron framework needs to rebuild some components to run with a particular version of NodeJS, while Truffle framework does not support this recompilation.

In *electron* folder we find:

- *css* folder, containing a css file which describes the formatting style of the html page;
- *img* folder, which contains images shown in the application;
- *js* folder, which contains the javascript files that describe the behavior of the render and main Electron processes;
- *index.html* file, which describes the structure of the interface;
- *package.json* file, which contains basic information about the developed application, its dependencies (e.g. electron package) and commands that perform something (e.g. the command to start application). This file is used by *npm* program to describe/define its possible operations;

The *truffle* folder contains:

[1] https://electronjs.org/
[2] https://github.com/ethereum/web3.js/
[3] https://github.com/ethereum/wiki/wiki/JSON-RPC
[4] https://truffleframework.com/
[5] https://nodejs.org
[6] https://www.npmjs.com/
[7] https://semantic-ui.com/

- *build* folder, which contains all '.sol' contracts compiled into '.json' artifact used by truffle to instantiate a specific contract object in the javascript code. See section 3.1 for more information about compiling contracts;
- *contracts* folder, which contains smart contract's source files;
- *migrations* folder, which contains javascript files needed by truffle to deploy specified contracts into a predefined Ethereum network (see following points). See section 3.2 for more information about deploying contracts with *Truffle*;
- *js* folder, which contains a script to deploy or delete the Catalog contract on an Ethereum network. See section 3.3 for more information about deploying contracts;
- *truffle-config.js* file, which describes the variables needed by truffle to compile and deploy contracts. This file defines the solc compiler, some private keys to allow correct contract deployment through Infura provider using a specific account and three different Ethereum network on which contracts could be deployed. To execute a migration (contract deployment) on a specific network, see section 3.2;
- *package.json* file, which has the same purpose of the *package.json* file contained into *electron* folder;

2.3 Requirements

As mentioned before, the application uses *Electron*, *Web3js* and *Truffle* frameworks so, to run application, they have to be installed. The installation process is simplified thanks to **npm** program: it leverages the *package.json* file to install them. Once you have installed *Nodejs* version 10.10.0 and *npm* version 6.4.1 [8], go into *truffle* folder with a terminal and type `npm install` to install truffle and its dependencies. Once it has finished the installation, go into *electron* directory and type in a terminal `npm install && npm run rebuild` to install electron and its dependencies.

Furthermore, to simplify the installation process, a bash script is created to perform the above commands. To execute it, go into root directory and type the command `./install.sh`.

2.4 Application execution

Assuming that you followed instructions exposed at section 2.3 to install the frameworks, you can run the application by easily opening a terminal inside *electron* directory and typing the command `npm start` and the DApp will be executed with defaults arguments: the address "0x49c32605a1aab14ecb6fbb180b4b3aae530a2701" as catalog address, the address "d4f293fe249a5b025361545f253d20e723c61072453e11ccedf92d4253abf167" as private account key and using the local ethereum client (e.g. *geth*).

Eventually you can provide these command line aguments to specify some execution parameters:

[8] Both of them can be installed following these instructions https://nodejs.org/en/download/package-manager

- –catalog-address <addr>: it uses given address as catalog contract address;
- –testnet : it runs program to support local test network (e.g. ganache). If you use this option, ”private-key” and ”mnemonic” options will be ignored;
- –infura : it uses the default Infura node as ethereum provider;
- –infura-key <key>: it uses given string as API key for infura Node;
- –mnemonic <words>: it uses given words to identify the Ethereum account;
- –private-key <key>: it uses given string to identify the Ethereum account;

Below are listed some examples to execute the application.

```
# This command shows an help message
npm start -- --help

# With this command, you can execute the application with default values
npm start

# With this command, you can execute the application using an Infura node as
provider instead the local provider. The default Infura key is used.
npm start -- --infura.

# With this command, you can execute the application using local Ethereum provider
#(i.e geth) defining a different catalog adderss and a different private key
npm start -- --catalog-address 0x9a6e2ea7f61d320fe16b664d65eda48489d0e6ce
  --private-key 2235f2146a5ca3ae0661ef4ba0023eb272781671635d9c24732d141e384fb902
```

2.5 Application interface

Once you executed the application, a login module will be prompted where you can chose your username and your address: if you are already registered, you must fill in the fields with correct information, otherwise insert a new usename and chose the address and you will be registered. Pay attention that at each address corresponds only one username.

Once you logged in, you can see on the top of the window the username, its balance and, eventually, ”Premium user!” string. Under them, alongside customer and author tab, there are commons functionalities independent from the ”author” or ”customer” profiles:

- the first one, ”Perform a query”, is a list of actions that you can perform on the Catalog to query some information about published contents;
- the second one, a cog icon, if clicked, shows a modal window with which you can specify the author or the genres you are interested in, separated by a comma;

- the last one, a bell icon, if clicked, shows the list of events that the DApp receives from the Catalog: when an event is received, the icon starts to blink with a red color for three times to notify the received event. When you close the event list after seeing it, events are deleted from the list;

The remaining application window is logically divided in two different parts: one collects information and operations for authors and the other for customer.

The author interface is splitted in two major areas: the left one provides the form to create a new content inserting information such a price, titles and type; the right one displays your published contents.

If you click on the customer tab, you can see many different sections:

- a section to buy or gift a premium account. To gift a premium account you have to indicate the username to which you want to gift it;
- a section to buy or gift a content. To gift a content to another one user, you have to indicate the username to which you want to gift it;
- a section containing list of contents that you have access granted (but not yet consumed) and contents that you have already consumed. To rate a consumed content, you can click on the green button on the left of the content's title contained in this list;
- a section on the right which shows all the available contents published on the Catalog. Moreover, you can request more info about a content by clicking on the orange button on the left of a content's name.

3 Catalog anagement

In all the following sections, we assumed that you have already followed instructions exposed at section 2.3 to install the frameworks.

3.1 Compile contracts

To compile contracts into json files used by *Web3js* library, go into *truffle* directory and type the command `truffle compile --reset` ;

3.2 Compile and deploy contracts using Truffle

You can deploy Catalog contract to the Ropsten Ethereum network using *Truffle* going into *truffle* directory and typing the command
`truffle migrate --reset --network <eth_network>` , where "eth_netwok" could be:

- ropsten : it uses the global test network *Ropsten* to deploy Catalog contract, assuming is running a client already connected to Ropsten, like *geth*;
- infura: it uses the global test netwok *Ropsten* to deploy Catalog contract using an *Infura*[9] node to access it.

[9] https://infura.io/

Notice that both private address key and Infura developer key are stored into *truffle-config.js* file, so if you want to use different values you have to change it in the file.

3.3 Manage Catalog using *catalogManager* program

To easly manage the Catalog lifecycle, the **catalogManager** javascript program was developed: it allows you to create or destroy a new Catalog contract on the Ropsten or local test networks. By default it runs with these default parameters: the address "d4f293fe249a5b025361545f253d20e723c61072453e11ccedf92d4253abf167" as private account key and using the local ethereum client (e.g. *geth*). It takes also these optional arguments:

- –testnet : it runs program to support local test network (e.g. ganache);
- –infura : it uses an Infura node as ethereum provider with default key;
- –infura-key <key>: it uses given string as API key for infura Node;
- –menomic <words>: it uses given words to identify the Ethereum account;
- –private-key <key>: it uses given string to identify the Ethereum account;

Moreover, it takes as non optional argument, one of them values:

- –create : it create a new Catalog contract depending on the previous parameters;
- –delete <address>: it destroys the Catalog contract corresponding to given address;

Do note that, since clients listen to events starting from the block they have started, they receive the events if and only if they are running during the Catalog destruction.

Below are listed some examples to execute the manager.

```
# With this command, you can create a Catalog with default values
nodejs catalogManager --create

# With this command, you can destroy a Catalog using an Infura node as
provider instead the local provider. The default Infura key is used.
nodejs catalogManager --infura
   --destroy 0x9a6e2ea7f61d320fe16b664d65eda48489d0e6ca

# With this command, you can execute the application using
local Ethereum provider (i.e geth) defining a different private key.
nodejs catalogManager --create
--private-key 2235f2146a5ca3ae0661ef4ba0023eb272781671635d9c24732d141e384fb902
```

Pay attention that only the user which creates the Catalog contract can destroy it.