Peer-to-peer and blockchain project report

Luca Di Mauro - 477943

October 7, 2018

Contents

1 Introduction

This brief report will describe the work behind the realization of COBrA project. This project provides the user with a graphic interface to use the smart contracts. The smart contracts were developed for the final term and were subsequently extended.

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

The first section will describe the extension of smart contracts, which was made specifically for this project.

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

The third section will describe how to deploy Catalog contract with Truffle framework.

Finally, the fourth section will provide an overview to helpful script.

2 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 the 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;

To use the added feature, the requested functions are provided: they compute, with different parameter, different type of averages about feedback types. Do note that, since solidity does not support float or double values, these functions return an integer type, even if computed value would be a float value.

The second added feature is the possibility for the contents creators to define the price that customers will need to pay to access that content. This value is stored inside only in the catalog smart contract, because it has to manage all payments system.

3 COBrA DApp development

3.1 Libraries and frameworks

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

[1] https://electronjs.org/

To interact with the Ethereum blockchain, *Web3js*[2] library was integrated with this framework. 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 defined inside of them than *Web3js*.

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]https://www.npmjs.com/, 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.

3.2 Folders 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. So the two different parts of the application needs to be placed into different folders.

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:

- *build* folder, which contains all '.sol' contracts compiled into '.json' artifact used by truffle to instantiate a specific contract object in the javascript code. These json files are created by typing the command `truffle compile --reset` and they can be recompiled with the same command;
- *contracts* folder, which contains solidity source files of contracts;

[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]
[7]https://semantic-ui.com/

- *migrations* folder, which contains javascript files needed by truffle to deploy specified contracts into a predefined Ethereum network (see folowing points). To perform a migration, type this command on a terminal `truffle migrate`. See 4 for more information about compiling and deploying contracts with *Truffle*;
- *script* folder, which contains two scripts to deploy the Catalog contract on an Ethereum network and to fill catalog with some contents. See 5 for more information;
- *truffle-config.js* file, which describes the variables needed by truffle to compile and deploy contracts. In this case, they are defined 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, just add the option `--network <network_name>`;
- *package.json* file, which has the same function of the *package.json* file contained into *electron* folder;

3.3 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, which defines the dependencies, to install them. Once you have installed *Nodejs* version 10.10.0 and *npm* version 6.4.1 [8], go into root directory and type the command `./install.sh`: this bash script installs automatically all needed dependencies and frameworks.

3.4 Application execution

Assuming that you followed steps at 3.3 to install the framework, you can run the application by easily opening a terminal into *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:

- –catalog-address <addr>: it uses given address as catalog contract address;
- –testnet : it runs program to support local test network (e.g. ganache);
- –infura : it uses the default Infura node as ethereum provider;

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

- –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, otherwise it uses the default account private key;

For example, typing `npm start -- --help`, will be shown a brief help message.

3.5 Application structure

Once you executed the application, a login module will be prompted: 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.

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

If you click on the author tab, you can see a section to create new contents on the left providing information about content type, title and price, and on the right there is a section which shows that you published contents as author.

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 to consume and contents that you have already consumed. To rate a consumed content, you can click on the green button on the left of name of content contained in this list. Or you can simply rate it by clicking on "Yes" button contained into the modal window shown after having received the event triggered by Catalog due to having consumed content;
- 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 name of a content.

On the top of the window the username, its balance and, eventually, "Premium user!" string will shown. Under them, alongside customer and author tab, there are three elements:

- 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;

4 Compile and deploy contracts using Truffle

Assuming that you followed steps at 3.3 to install the frameworks, you can deploy Catalog contract to the Ropsten Ethereum network using *Truffle* going into *truffle* directory and typing the command
`truffle migrate --network <eth_network>`, where "eth_netwok" could be:

- dev: it uses the local test network, assuming is running a program like **ganache** which creates a local ethereum test network;
- ropsten : it uses the global test network *Ropsten* to deploy Catalog contract, assuming is running a client 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.

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.

5 Helpful script

Moreover, to facilitate the Catalog management, another script, contained into `truffle/script` directory, is developed with which you can deploy the Catalog on an Ethereum network and, when this script receives a SIGINT or SIGTERM signal, it terminates its execution destroying the contract from the blockchain.

It allows to notify clients which has the application running that the Catalog died and they have to delete created contracts on the blockchain. Do note that, since clients listen to events from the latest block when they are started, they receive the events if and only if they are running during the Catalog destruction.

The script accepts some command line arguments. To show them, go into script directory and type `nodejs createCatalog --help`.

[9] https://infura.io/