Final project report

Giulio Piva

July 5, 2022

1 Starting the application

The steps I use to start the application are the following:

- Launch ganache with the -m option:

```
ganache -m "mountain magic oven first used asthma
rookie green bonus giant ability welcome"
```

- Import the private keys of Ganache into Metamask
- Deploy the application

```
truffle migrate --reset --network development
```

- Run the lite-server

```
npm run dev
```

Starting *ganache* with the -m option allows to avoid to repeat the import of the private keys every time ganache is restarted. Therefore, once a private key has been imported once, it is not necessary to import it again. However, when restarting ganache, also the transactions of the account in metamask must be resetted, otherwise it will produce errors due to unsynchronized state.

2 Functionalities

2.1 Operator functionalities

All the functionalities of the operators are implemented in the file lottery_manager.js. The lottery can be managed from the page lottery_manager.html. Here the interface will change dinamically, according to the current state of the lottery:

- If a round is not yet started, the button "Start Round" will be enabled.
- If there is an active round, then the interface will show the number of remaining blocks to pass.
- If the round is not active, then a "Draw numbers" button will appear.

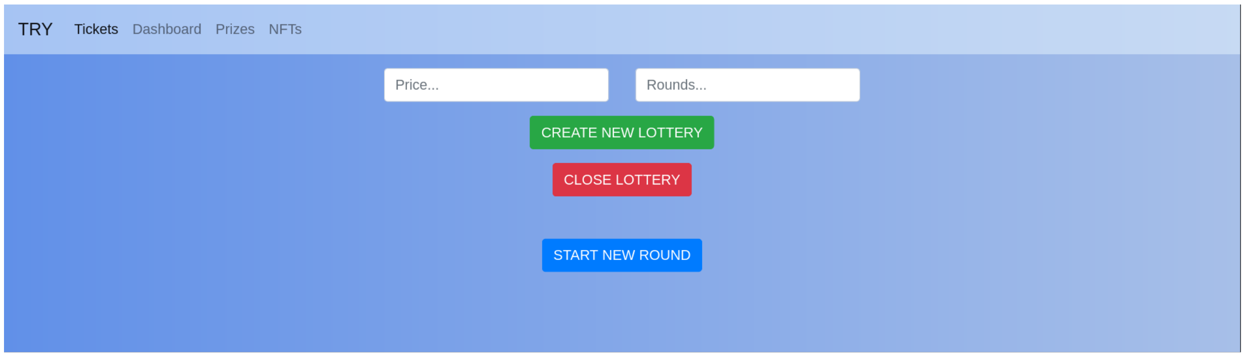


Figure 1: Interface of the lottery manager

Creating a new lottery

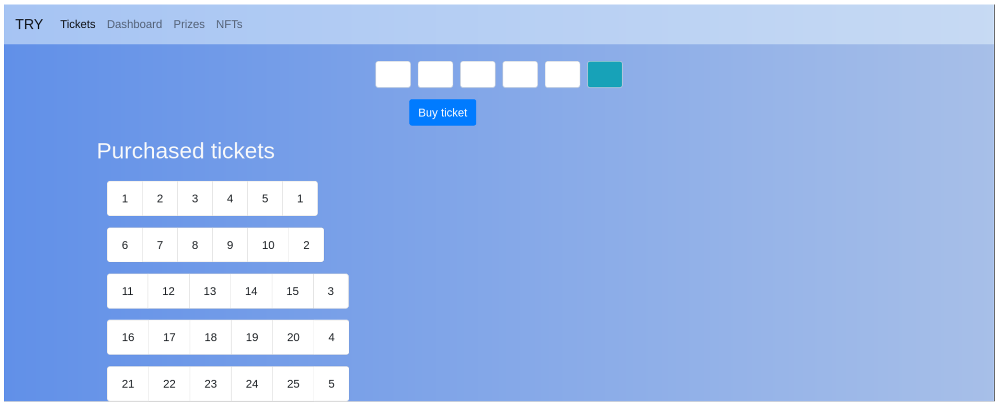
With the button "Create Lottery" is possible to create a new lottery. However, in order to perform this operation, the previous round must be finished.

Closing the lottery

With the button "Close Lottery" is possible to close the current lottery. Once a lottery has been deactivated, a warning message will be shown.

2.2 Tickets

The page buy.html is used to buy new tickets and show those previously bought. If an error occur when the user tries to buy a ticket (e.g numbers constraint not respected, Round not active, etc...), a red alert will appear above to flag the error. The cost of the ticket is retrieved automatically through the smart contract, so the user doesn't need to specify it and only need to confirm the transaction with metamask.



Purchased tickets

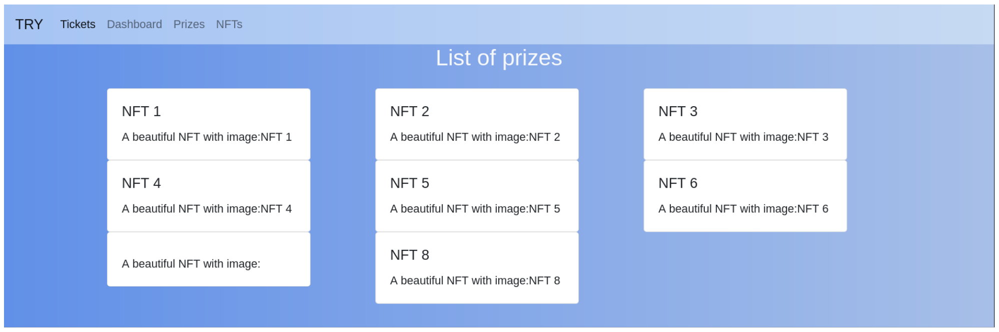
1	2	3	4	5	1
6	7	8	9	10	2
11	12	13	14	15	3
16	17	18	19	20	4
21	22	23	24	25	5

Figure 2: Buy tickets

2.3 NFTs

Prizes

The page prizes.html is used to show the prizes of the current round.



NFT 1
A beautiful NFT with image:NFT 1

NFT 2
A beautiful NFT with image:NFT 2

NFT 3
A beautiful NFT with image:NFT 3

NFT 4
A beautiful NFT with image:NFT 4

NFT 5
A beautiful NFT with image:NFT 5

NFT 6
A beautiful NFT with image:NFT 6

A beautiful NFT with image:

NFT 8
A beautiful NFT with image:NFT 8

Figure 3: Prizes

NFTs won

The page nft.html is used to show the NFTs won by the user. It uses the same format as the page for prizes.

3 Events

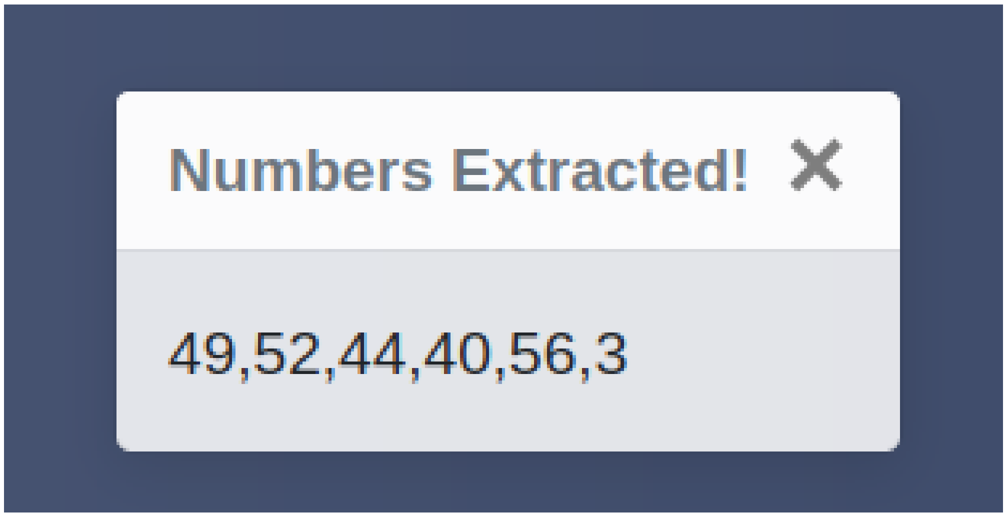
When an events is triggered, a toast message will be shown on the bottom right corner of the screen with the related event. the events managed by the lottery are:

- The creation of a new lottery
- The start of a new round
- The extraction of the numbers
- The winning of an NFT

4 Implementation details

4.1 Lottery Factory

I created a new contract to manage the creation and migration of new lotteries from the UI. It is called LotteryFactory. It has two main functions: CreateLottery and getLotteryAddress. The first one is used to create a new lottery and the second one is used to get the address of the the currently active lottery, which will be used to instantiate a TruffleContract in the JavaScript code to interact with the lottery. The LotteryFactory has to be deployed with the Truffle migration script, which it will also create the first lottery.


Numbers Extracted!

49,52,44,40,56,3

Figure 4: An example of toast to represent an event

4.2 Core.js

In the file core.js, I implemented the functions to initialize all the lottery machinery. The function init returns an object which will be used to interact with the blockchain. This module is imported in every other file, so I could avoid code duplication.

4.3 Files

For every html page, I created a JavaScript file with the necessary functions to implement the functionalities of that specific page. I preferred to separate the functionalities per file to have a better structured code and keep related methods together.

4.4 GUI

I implemented the guy using the Bootstrap CSS framework and JQuery. I added a navbar to access the different pages of application. ***Note***: The link to the Operator dashboard has been added to the navbar just for simplicity but in the real application that page would be only accessible to the operator.

4.5 Test

To facilitate the testing of the application, I automated the buy of the tickets in the migration script (1_initial_migration.js). Then the operator can immediately extract the numbers and the prizes are given. One can see the NFT won in the nft.html page. This avoid the burden of buying tickets every time from scratch to test the lottery. However, for the delivery of the project I commented those lines, so the lottery will start from scratch.

5 Improvements of the Lottery contract

I applied some modifications and corrections to the smart contract delivered in the final term.

Tickets

The tickets are now represented with a mapping (address => Ticket[]). This has been done to better support the retrieval of tickets that belong to a specific user and show them in the buy.html page.

Change

Now a change is sent back to the buyer when the the value sent is more than the cost of the ticket.

Initial minting

The initial minting of the NFTs is now made automatically in the constructor. Previously, it had to be done manually.

Drawing numbers

I corrected the bug where the numbers could be extracted before all the blocks passed. I added a require to check this condition to permit the drawing of the numbers.