Università di Pisa

Peer to Peer Systems and Blockchains

A.A. 2021/2022

TRY - a nfT lotteRY

Final Project

Autori:
Francesco Kotopulos De Angelis
July 15, 2022

Contents

1 DApp - Back end

The Back end of the distributed app is divided into two main components:

- **Smart contracts:** are stored in the contract's folder and they are compiled and deployed on the testing blockchain thanks to Truffle Suite;
- **App.js:** javascript application that serves requests invoking smart contract functions.

1.1 Lottery Smart Contract - Updates

In this section we will list all major updates applied to smart contracts in this project.

- **Lottery Start:** first of all, it was necessary to move the initialisation of the smart contract from the constructor to a dedicated function. Now in the smart contract, there is a "StartLottery" function that does exactly the same operations executed in the constructor of the contract. This update has been necessary to allow the deployment of the smart contract without executing the constructor each time;
- **WonNFTs data structure:** a new data structure used to store a list of won prizes for all players. This data structure is now required to be able to show to the end-user a list of all won NFTs;
- **getNFTList():** function used to return the list of all NFTs won by a specific user;
- **getURI():** function used to get URI linked to a specific token.

1.2 Lottery Mint Smart Contract - Updates

The Lottery Mint contract is the same as the previous version except for NFT's URI. In the previous version of the contract, all images were uploaded on IPFS but in this implementation, in order to show NFTs thumbnails on the webpage, it was necessary to upload images on a different provider due to availability problems.

1.3 App.js

App.js is the main javascript application developed for this project. First of all the script is executed each time a user reloads the current page. The first time is executed, the app tries to connect to the MetaMask wallet and then redirects the user to the correct interface. When the account is connected, the App starts listening for events emitted by the lottery contract or by the web interface. If the user or the lottery operator clicks on a button on the respective interface, the button emits a click event and calls the linked function. Otherwise, all events coming from the smart contract are reported to the user as notifications.

2 DApp - Front end

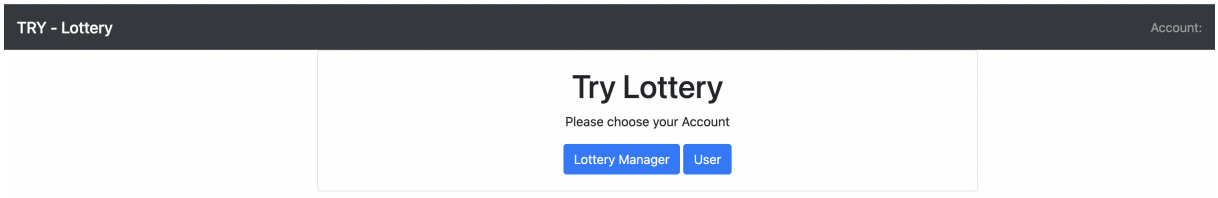

Figure 1: Lottery Homepage

The distributed application of TRY lottery is divided into two main interfaces to meet the needs of the user and the lottery manager. When the DApp it's started it will show the homepage where the user can choose to log in as a Lottery Manager or as a standard user. To access the lottery manager interface it's necessary to own the lottery manager account address and connect it to MetaMask.

2.1 Lottery Manager interface

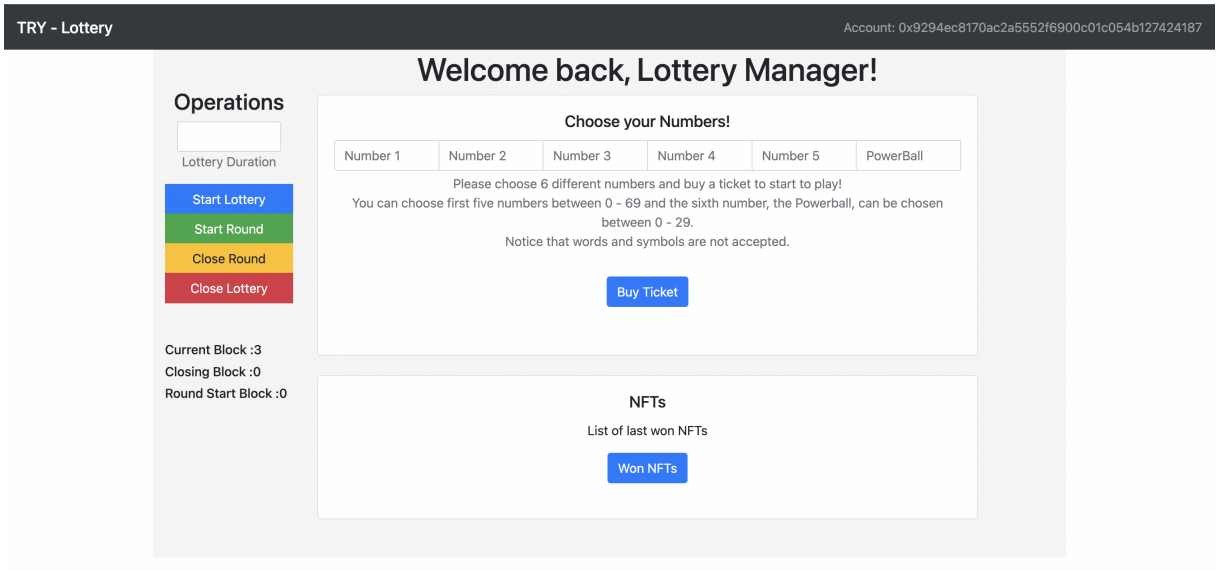

Figure 2: Lottery Manager Interface

The lottery manager is the only one allowed to create a lottery, start a new round and close an active round awarding winners. For this purpose, the lottery manager has a private interface accessible only if the current account connected on Metamask is the one of the lottery manager. The lottery manager interface provides the following functions:

- **Lottery Duration:** text box used to insert and define the duration of a single lottery round;
- **Start Lottery:** this function it's called only once to initiate the lottery. The lottery manager sends a transaction to the contract that calls a function declared inside the lottery contract. This function creates an instance of the Lottery Mint contract, sets up the lottery round duration and assigns the address of the lottery manager inside the contract.
- **Start Round:** after closing an active round and awarding winners, the lottery manager must be able to start a new round. In this way, every data structure related to the previous round will be cleared and a new round will be initialised.
- **Close Round:** when the current block number in the blockchain it's equal to the block number of the closure of the current round, the lottery operator can call the close round function to check if there are any winners in the current round. In that case, the function will award winners and will send them the correspondent prize.
- **Close Lottery:** the close lottery function can be called only once by the lottery operator. When this happens the lottery contract and all related functionalities will be no longer available and the contract will be self-destructed. When a contract it's destructed, a user can still send money to the contract or pay gas to execute functions. To avoid this, when the lottery manager closes the lottery, the user will be automatically redirected to a special page that says that the lottery it's no longer active and the user won't be able to interact with the contract.

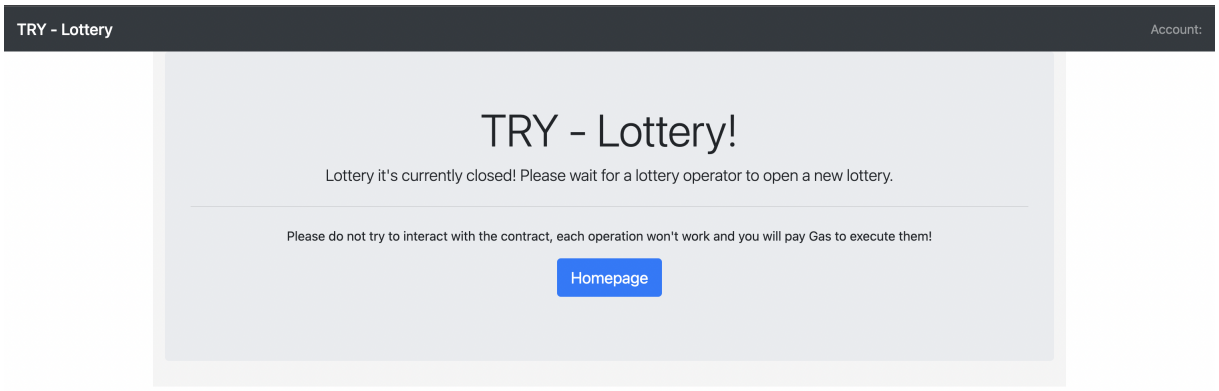

Figure 3: Lottery Closed Page

2.2 User interface

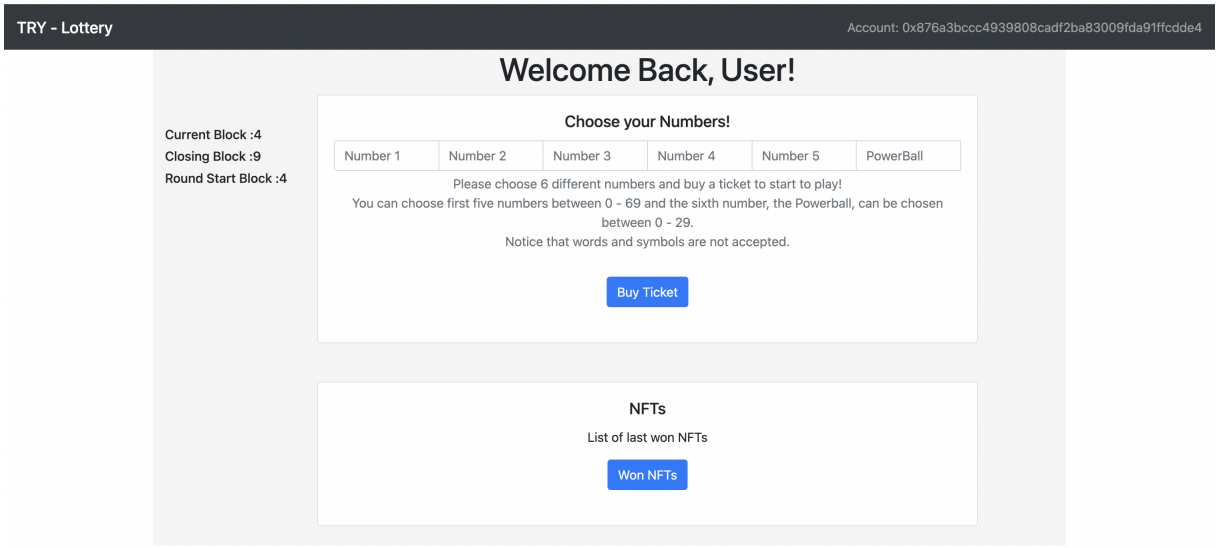

Figure 4: User Interface

A user has a dedicated interface to interact with the smart contract. The user has limited access to smart contract functionalities. For this reason, the user interface provides the following functions:

- **Buy a Ticket:** it's the main functionality of the user interface. By choosing six different numbers and clicking on the "Buy Ticket" button, the user will receive a confirmation from Metamask for the transaction and so will be able to buy a new ticket.
- **Show last won NFTs:** under the "Buy Ticket" button in the user interface, there is a section where the user can find all the last won NFTs. When the user refreshes the current page, a function will get from past events all won NFTs and it will show them in the correct section.
- **Show all NFTs:** the user can call this function ad see all won NFTs. When the function it's called, the user it's redirected to the NFT Storage page and will be able to see all last won NFTs.

3 Design choices and implementation

3.1 Lottery Start

To create a new lottery the lottery manager has to call a function available in its dedicated interface. From the previous version of the Lottery smart contract, there is a new function, "Start Lottery", to initialise a new instance of the lottery Mint contract, mint the first eight NFTs, set up the lottery round duration and assign the address of the lottery manager to a specific variable in the smart contract. In the previous version, all these operations were executed directly at contract deployment in the constructor. The problem with this implementation is that the deployer address at the beginning is "0x0" and setting the lottery operator directly in the constructor would have assigned the address "0x0" to the lottery operator. In this way, each function that must be executed by a lottery operator would have been no longer available.

3.2 Session Management

For this implementation of the distributed app, it's necessary to set up Local Session management due to the two different interfaces defined inside the application. The application must be able to detect the current account address and understand if it's a lottery manager or a user. After creating a lottery, the distributed application initialises a connection to MetaMask accounts and sets up two variables in the Local Session Storage of the browser:

- **Lottery Operator:** variable used to store locally the address of the lottery operator retrieved by the Lottery smart contract after the creation of a new lottery;
- **Current User:** address of the current user logged in MetaMask and connected to the distributed application.

After setting these variables, each time a user reloads the page, a function it's executed and if the current user it's not in the correct interface, it will be redirected immediately. In this way, it's possible to assign automatically the correct interface to the specified user.

3.3 Events Management

In order to notify the user of what's happening in the lottery contract, we manage events emitted by the lottery contract. First of all, to explain how notifications work, we start listing all events defined in the Lottery Contract:

- **TicketBuy(address player):** event emitted each time a user buys a new ticket;

- **LotteryStart():** event emitted only once when the lottery operator creates a new Lottery;
- **RoundStart():** event emitted each time the lottery operator decides to start a new round after closing the one still active;
- **RoundClosed():** event emitted when lottery operator, after a specific number of blocks, decides to close the round and award winners;
- **AwardPlayer(address player, uint256 prize):** event emitted after closing the lottery, each time a user wins a new NFT;
- **NumbersDrawn(uint256[6] winningNumbers):** event emitted each time the lottery operator closes the current round. In this case the Lottery contract has to draw winning numbers;
- **NFTMinted(address NFTOwner, uint256 newNFTId):** event emitted each time a new NFT it's minted;
- **LotteryClosed():** event emitted only once when the lottery manager decides to definitely close the lottery and destruct the contract.

In order to manage all events emitted by the lottery, we defined an event listener for each event declared inside the smart contract. When a new event it's emitted, the correspondent listener catches the event and sends a notification to the user or the lottery operator.

4 How to start the DApp

4.1 Requirements

In order to start our distributed App, we have to install all dependencies and initialise the project. For this reason in the following section we will list all requirements to execute the DApp:

- **Node JS:** javascript runtime environment used to serve our DApp;
- **Lite Server:** lightweight *development only* node server that serves a web app;
- **Ganache:** part of the Truffle Suite. It's used to create our testing Ethereum blockchain;
- **Truffle:** used to compile and deploy smart contracts on Ganache testing blockchain;

- **MetaMask:** used to manage accounts and to confirm transactions to the Lottery smart contract;
- **OpenZeppelin libraries:** libraries used in the smart contract to implement the ERC721 standard for NFTs.

4.2 Instructions to run the DApp

After installing all dependencies, there is a bash script "run.sh" built to automatically start all services necessary to run our DApp. The script simply runs four different commands:

```
#! /bin/bash
echo "Starting Ganache..."
ganache -p 8545 &
echo "Compiling Contracts..."
truffle compile
echo "Deploying Contracts..."
truffle migrate --reset
echo "Starting App..."
npm run dev
```

- **ganache -p 8545:** used to start ganache blockchain;
- **truffle compile:** used to compile smart contracts inside the contracts directory;
- **truffle migrate –reset:** used to deploy smart contracts on ganache testing blockchain;
- **npm run dev:** used to start lite server and to serve the DApp;

Running the bash script will start the entire application and the system will show to the user the lottery homepage. The entire source code of the project is available at this Link.