Collectibles.sol

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.8.0 <0.9.0;

contract Collectibles {


    uint public immutable collectible_prize=1000000000000000000;
    uint8 public immutable collectibles_number = 8;
    mapping(uint8 => string  ) public collectibles_list;
    bool [9] public collectibles_sold;

    event collectible_sold(uint8);

    constructor(){

        collectibles_list[1]="Hello_Kitty_1";
        collectibles_list[2]="Hello_Kitty_2";
        collectibles_list[3]="Hello_Kitty_3";
        collectibles_list[4]="Hello_Kitty_4";
        collectibles_list[5]="Hello_Kitty_5";
        collectibles_list[6]="Hello_Kitty_6";
        collectibles_list[7]="Hello_Kitty_7";
        collectibles_list[8]="Hello_Kitty_8";
    }

    function get_address() view public returns (address){
        return address(this);
    }

    function buy_collectible(uint8 collectible_id) external payable returns(string
memory) {
        require(collectible_id <= collectibles_number, "Wrong Collectible_ID");
        require( msg.value >= collectible_prize, "Value too small");
        require(!collectibles_sold[collectible_id], "Collectible already sold");

        uint money = msg.value;
        uint change;

        if(money > collectible_prize) {
            change = money - collectible_prize;
            // Reimbourse the change
            payable(msg.sender).transfer(change);
        }

        emit collectible_sold(collectible_id);
        collectibles_sold[collectible_id] = true;
        return collectibles_list[collectible_id];
    }



}
```

newNFT.sol

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.8.0 <0.9.0;

import "./../node_modules/@openzeppelin/contracts/access/Ownable.sol";
import "./../node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol";

    contract newNFT is Ownable, ERC721("NFT", "TRY_NFT"){
        uint8 private tokenId;
        mapping(uint8=>NFT_info) public ownershipRecord;

        struct NFT_info{
            address nft_owner;
            uint8 token_Id;
            string  description_collectible; // collectible description
            uint8 class; //class value
        }

        event NFT_mint(address owner, uint8 token_id, string description_collectible,
uint8 class );

        //mint a new token and store locally its information
        function mintToken(address recipient, string memory
description_collectible_given, uint8 given_class) onlyOwner external returns (uint8) {
            _safeMint(recipient, tokenId);
            ownershipRecord[tokenId]=(NFT_info(recipient,tokenId,
description_collectible_given, given_class));
```

```
            uint8 token_return = tokenId; //save the token_id of the actual NFT minted,
then return it

            tokenId = tokenId + 1;
            return token_return;
        }

        //trasnfer the ownership to a new owner
        function give_to_winner(address owner,address winner, uint8 _tokenId) external
{
            super._transfer(owner, winner, _tokenId); //change the NFT owner class
attribute
            ownershipRecord[_tokenId].nft_owner=winner; //change the NFT owner local
attribute
        }

        //return all the info about a NFT identified by its token_id
        function get_NFT_informations(uint8 tokenId_given) public view returns
(address, string memory, uint8, uint8){
            return (ownershipRecord[tokenId_given].nft_owner,
ownershipRecord[tokenId_given].description_collectible,
ownershipRecord[tokenId_given].class, tokenId_given);
        }

        //return the string description about a NFT identified by its token_id
        function get_NFT_desc(uint8 tokenId_given) public view returns (string memory){
            return (ownershipRecord[tokenId_given].description_collectible);
        }



}
```

Lottery.sol

```
/// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.8.0 <0.9.0;
import "./Collectibles.sol";
import "./newNFT.sol";
contract Lottery{

    Collectibles CL;
    newNFT newnft;
    address private CL_address; //address of the Collectibles deployed contract

    address public operator;
    address public balance_receiver;

    mapping(uint8=>string) private collectibles_bought; // id_collectible =>
collectible description
    uint8 [] private collectibles_bought_id;

    uint8 [6] public drawn_numbers;
    mapping (uint8 => bool) private drawn_numbers_mapping; // drawn_number => true

    mapping (address => uint8) private reward_list; //address winner => class of the
reward nft
    address[] private winners;

    uint8[] private NFT_minted; //take track of token id minted so far

    struct Ticket{
        address owner;
        uint8 [6] chosen_numbers;
    }
    Ticket[] private tickets_sold;

    uint64 public immutable price_tricket = 1000000000000000000;
    uint64 private block_number_init; //
    uint64 private immutable M = 5;

    uint8 private immutable collectibles_number = 8;

    string public lottery_state = "Closed";
    string public lottery_phase_operator = "Not_created";

    bool private first_init_round = true; //used in order to detect the case in
startNewRound

    uint8 public contatore =0;

    constructor () {
        //init the lottery operator
        operator = msg.sender;
        init_side_contracts();
    }

    //EVENTS

    event bal_initialized(address receiver);

    event phase_change(string phase_name);

    event ticket_bought(address owner, uint8 [6] return_chosen_numbers);
    event values_drawn(uint8[6] values_drawn);

    event prize_assigned(address winner, uint8[6] value_provided_user, uint8[6]
draw_numbers, uint8 class_prize);
    //address of the winner, values provided by the winner, drawn numbers, class of the
prize

    event collectible_bought(uint8 id, string description);

    event NFT_minted_now (address owner,string description, uint8 NFT_class, uint8 id);

    event NFT_transfered (address owner, string description, uint8 NFT_class, uint8
id);

    event lottery_created(bool lottery_closed);

    event lottery_closed(bool lottery_closed);


    //MODIFIERS

    modifier onlyOperator { //verify that only the lottery operator can execute a
function
        if (msg.sender == operator) _;
    }

    modifier initPhaseActive{
        if(keccak256(abi.encodePacked(lottery_phase_operator)) ==
keccak256(abi.encodePacked("Init_phase")))
        _;
    }

    modifier buyPhaseActive { //verify that the buy phase is start, so now the users
can buy tickets
```

```
        if(keccak256(abi.encodePacked(lottery_phase_operator)) ==
keccak256(abi.encodePacked("Buy_phase")))
         _;
    }

    modifier  extractionPhaseActive { //verify that the buy phase is closed, so now the
operator can extract the random numbers
        if(keccak256(abi.encodePacked(lottery_phase_operator)) ==
keccak256(abi.encodePacked("Extraction_phase")))
         _;
    }

    modifier givePrizesPhaseActive{ //verify that the random numbers are extracted, so
now the operator can assign and compute the prizes
        if(keccak256(abi.encodePacked(lottery_phase_operator)) ==
keccak256(abi.encodePacked("Give_prizes_phase")))
        _;
    }

    modifier computePrizesPhaseActive{ //verify that the prizes are computed, so now
the operator can give them to the winner users
        if(keccak256(abi.encodePacked(lottery_phase_operator)) ==
keccak256(abi.encodePacked("Compute_prizes_phase")))
            _;
    }

    modifier lotteryOpen{ //verify that the lottery is open. Once the close_lottery()
fun is invoked the value of lottery_open is set to false, so the function can't be
invoked anymore.
        if(keccak256(abi.encodePacked(lottery_state)) ==
keccak256(abi.encodePacked("Open")))
            _;
    }

    //GETTER FUNCTIONS

    function get_max_collectible_id() view public returns (uint8){
        return CL.collectibles_number();
    }

    //returns the block.number registered at the start round time
    function get_block_initRound() view public returns (uint){
        return block_number_init;
    }

    function get_num_tickets_sold() view public returns (uint8){
        return uint8(tickets_sold.length);
    }

    //receive the ticket_id and retrieve the ticket informations (owner, submitted
values)
    function get_ticket_information(uint8 index) view public returns(address, uint8 [6]
memory){
        require(index < tickets_sold.length, "Wrong ticket_id inserted");
        return (tickets_sold[index].owner, tickets_sold[index].chosen_numbers);
    }

    function get_contract_balance () view public returns (uint){
        return address(this).balance;
    }

    function get_num_collectibles_bought() view public returns (uint8){
        return uint8(collectibles_bought_id.length);
    }

    function get_collectible_info(uint8 index) view public returns (string memory){
        if(index < uint8(collectibles_bought_id.length))
            return string(collectibles_bought[collectibles_bought_id[index]]);
        else
            return ("No collectible bought so far");
    }

    function get_num_NFTs_minted() view public returns (uint8){
        return uint8(NFT_minted.length);
    }

        //receive the index in NFT_Minted, then retrieve the NFT informations (owner,
description, class)
    function get_NFT_information(uint8 index) view public returns(address, string
memory, uint8, uint8){
        return newnft.get_NFT_informations(NFT_minted[index]);
    }

    function get_last_ticket_bought() view public returns (address, uint8 [6] memory){
        //require (tickets_sold.length > 0, "No ticket bought so far");
        if(tickets_sold.length > 0){
```

```solidity
            uint8 ticket_id = uint8(tickets_sold.length-1);
            return get_ticket_information(ticket_id);
        }
        else{
            uint8 [6] memory ret = [0,0,0,0,0,0];
            return (address(0x0), ret);
        }
    }

    function get_drawn_numbers() view public returns (uint8 [6] memory){
        return drawn_numbers;
    }

    /*
    function get_last_prize_assigned() view public returns (uint){
        return address(this).balance;
    }
    */

    //drawn numbers gestita da app.js

    //BUSINESS LOGIC FUNCTIONS

    function init_side_contracts() internal returns (bool res){
        newnft = new newNFT();

        CL = new Collectibles();
        CL_address = CL.get_address();
        res = true;
        return res;
    }

    function create_lottery() public {
        lottery_phase_operator="Created";
        lottery_state="Open";
        emit phase_change(lottery_phase_operator);
        emit lottery_created(true);
    }

    function check_initPhase() internal returns (bool res){
        res=false;
        if(collectibles_bought_id.length==8 && (balance_receiver!= address(0x0)) ){
            lottery_phase_operator="Init_phase";
            emit phase_change(lottery_phase_operator);
            res = true;
        }
        return res;
    }



    function set_balance_receiver(address receiver) onlyOperator external returns (bool
res){
        require(address(0x0)!= receiver, "Invalid address given");
        balance_receiver=receiver;

        emit bal_initialized(receiver);

        check_initPhase(); // verify if startNewRound can be invoked

        return res = true;
    }

        //buy collectibles through the Collectibles object
    function buy_collectibles(uint8 id_collectible) onlyOperator payable external
returns(bool res){
        require(id_collectible <= collectibles_number, "Wrong Collectible_id
inserted");
        require( id_collectible >0 , "Wrong Collectible_id inserted");
        collectibles_bought[id_collectible] =
string(Collectibles(CL_address).buy_collectible{value:msg.value}(id_collectible));
        collectibles_bought_id.push(id_collectible);

        emit collectible_bought(id_collectible,collectibles_bought[id_collectible] );

        check_initPhase(); // verify if startNewRound can be invoked

        return res = true;
    }

    //operator can close the lottery in any moment
    function close_lottery() onlyOperator lotteryOpen external returns (bool res){
        //first disable all the other function calls
        lottery_state="Closed"; //no way to set it Open again => no way of getting
reentrancy
        lottery_phase_operator = "Closed";
        emit phase_change(lottery_phase_operator);

        for(uint8 i=0;i<tickets_sold.length;i++)
            payable(tickets_sold[i].owner).transfer(price_tricket);

        emit lottery_closed (true);

        return res = true;
    }

    //verify the correctness of the numbers provided by the users
    //invoked by buy_ticket()
    function check_given_numbers(uint8 [6] memory given_numbers) pure private returns
(bool){

        for(uint8 i=0; i<5; i++){
            if(given_numbers[i] < 1 || given_numbers[i] > 69)
                return false;
        }
        if(given_numbers[5] < 1 || given_numbers[5] > 26)
            return false;

        return true;
    }

    //verify the correctness of the numbers provided by the users
    //invoked by buy_ticket()
    function check_duplicate(uint8 [6] memory input) private pure returns(bool){
        for(uint8 i=0; i<5;i++){
            for(uint8 j=i+1; j<5;j++){
                if(input[i]==input[j])
                    return false;
            }
        }
        return true;
    }



    function buy_ticket(uint8 [6] calldata  chosen_numbers) buyPhaseActive payable
external returns (bool res){

        uint64 money=uint64(msg.value);
        address owner = msg.sender;
        //checks for input data
        require(msg.sender != balance_receiver, "The balance Receiver can't be a
player");
        require(chosen_numbers.length == 6, "You have to choose 6 numbers");
        require(check_duplicate(chosen_numbers), "You can't insert duplicate values in
the standard numbers");

        require(check_given_numbers(chosen_numbers), "Wrong values inserted");

        require(money>=price_tricket, "You have not enough ether");

        Ticket memory  ticket =  Ticket(owner, chosen_numbers);
        tickets_sold.push(ticket);

        if(money >= price_tricket) {
            uint64 change = uint64(money) - price_tricket;
            // send the change
            payable(owner).transfer(change);
        }

        uint8 [6] memory return_chosen_numbers = chosen_numbers;

        emit ticket_bought(address(msg.sender), return_chosen_numbers);

        check_round_is_active();

        return res = true;
    }

    //after each ticket bought, verify if the number of blocks M is reached or not
    //invoked by buy_ticket()
    function check_round_is_active() private {
        if( (block.number - block_number_init)>= M){ //round concluded
            lottery_phase_operator = "Extraction_phase";
            emit phase_change(lottery_phase_operator);
        }
    }

    //extract the numbers randomly, done outside of the blockchain
    //invoked by draw_numbers()
    function draw_numbers_compute() private view returns (uint8 [6] memory ){ //returns
the extracted value
```

```
        uint8 [6] memory  numbers;
        uint seed =block.difficulty;
        uint8 extracted_value;
        bytes32 bhash;
        bytes32 rand;


        for(uint8 i=0;i<6;i++){

            bhash = (keccak256(abi.encodePacked(block.number,extracted_value,seed)));

            rand = keccak256(abi.encodePacked(bhash));

            if(i==5)
                extracted_value=uint8(uint(rand) % 27);
            else
                extracted_value=uint8(uint(rand) % 70);

            if(extracted_value==0) //since value range start from 1
                extracted_value++;

            numbers[i]=extracted_value;
        }

        bhash=0;
        rand=0;

        return (numbers);
    }


    function draw_numbers() onlyOperator extractionPhaseActive external returns(bool
res) {

        uint8 [6] memory  computed_values; //mi permette di non accedere allo storage
nel for
        computed_values = draw_numbers_compute();

        require(check_duplicate(computed_values), "Duplicate values in drawn numbers");

        emit values_drawn (computed_values);
        //aggiorno il mapping
        for(uint8 i =0; i<5;i++)
            drawn_numbers_mapping[computed_values[i]]=true;

        //scrivo su storage i valori estratti
        drawn_numbers = computed_values;

        //cancello variabile locale
        delete computed_values;

        lottery_phase_operator = "Compute_prizes_phase";
        emit phase_change(lottery_phase_operator);

        return res = true;
    }

    //compute, for each ticket bought, the number of matches obtained
    //invoked by compute_prizes()
    function compute_matches(uint8[6] memory num_picked_usr ) view private returns
(uint8 , uint8 ){
        //input= array numeri scelti da utente
        uint8 normal_matches;
        uint8 powerball_match;
        for(uint8 i=0; i<5;i++){ //for each value submitted by user
            if(drawn_numbers_mapping[num_picked_usr[i]]==true)
                normal_matches++;
        }

        if(drawn_numbers[5]==num_picked_usr[5])
            powerball_match++;

        return (normal_matches, powerball_match);
    }


    //compare the drawn_numbers with the values provided by the users in the tickets
and elect the winners
    function compute_prizes() computePrizesPhaseActive onlyOperator external returns
(bool res){

        uint8 powerball_match; //counter
        uint8 normal_matches; //counter
        uint8 class_value;
        uint8 i;

        uint8 [6] memory num_picked_usr;

        for( i=0; i<tickets_sold.length;i++){ //for each ticket
            num_picked_usr = tickets_sold[i].chosen_numbers; //take values submitted

            (normal_matches, powerball_match) = compute_matches(num_picked_usr);

            class_value=compute_class_value_prize(powerball_match, normal_matches);
            //modifica
            if(class_value>0){
                reward_list[tickets_sold[i].owner] = class_value;
                winners.push(tickets_sold[i].owner);
                emit prize_assigned(tickets_sold[i].owner, num_picked_usr,
drawn_numbers, class_value);
            }

        }
        lottery_phase_operator = "Give_prizes_phase";
        emit phase_change(lottery_phase_operator);
        res = true;
        return res;
    }

    //compute, for each ticket bought, the class of the NFT given as reward based on
the number of matches
    function compute_class_value_prize(uint8 powerball_match, uint8 normal_matches)
pure private returns (uint8){
        bool done=false;
        uint8 class_value=0;
        uint8 sum_values;
        //classes 1,2,7,8 handled handmade
        if(powerball_match==1){
            if(normal_matches==0){
                class_value=8;
                done=true;
            }
            if(normal_matches==5){
                class_value=1;
                done=true;
            }
        }
        if(powerball_match==0){
            if(normal_matches==1){
                class_value=7;
                done=true;
            }
            if(normal_matches==5){
                class_value=2;
                done=true;
            }
        }

        //classes 3,4,5,6 computed algorithmically
        if(!done){
            sum_values = powerball_match+normal_matches;
            if(sum_values>0)
                class_value= 8 - sum_values;
            else
                class_value=0;
        }

        return class_value;
    }

    //assign the NFTs to the winners: if a NFT is avaible it just transfer it to the
winner, else it mints a new NFT
    function give_prizes() onlyOperator givePrizesPhaseActive external returns (bool
res){
        lottery_phase_operator = "Init_phase"; // set here, ANTI REENTRANCY
        uint8 i;
        uint8 NFT_class;

        address address_temp;
        string memory description_temp;
        uint8 class_temp;
        uint8 token_id_temp;
        for(i=0;i<winners.length;i++){

            NFT_class = reward_list[winners[i]]; // take the class of the nft to give
as reward

            // initially, 8 NFTs (one for each class) were minted with owner =
operator, and their token_id are in the array NFT_Minted
```

```
            // so now check:
            // if the NFT minted initially with the class = class of the reward  is
owned by the operator yet (this means that was not assigned as reward) => transfer this
NFT to the winner user
            // if the NFT minted initially with the class = class of the reward is
owned by a user (this means that was already assigned as reward) => mint another with
the same class

            (address_temp, description_temp, class_temp, token_id_temp) =
get_NFT_information(NFT_class-1);
            // get_NFT_information takes the token_id of the NFT in input.
            // Since the first 8 NFT minted are one for each class and they're stored
in NFT_Minted, an NFT of class 1 is stored in NFT_Minted[0] (because the index of the
array NFT_Minted starts from 0, while the NFT class start from 1)
            // So generally for the first 8 NFT minted is true that a NFT of class x is
stored in NFT_Minted[x-1].

            if(address_temp == operator){
                newnft.give_to_winner(operator,winners[i],token_id_temp); //class-1
perche i primi 8 nft, memorizzati in indici [0,7] corrispondo alle 8 classi
                emit NFT_transfered (winners[i], collectibles_bought[NFT_class],
NFT_class, token_id_temp);
            }
            else
                mint(NFT_class, winners[i], true);
        }

        emit phase_change(lottery_phase_operator);
        return res = true;
    }


    function mint(uint8 NFT_class, address owner, bool print) private returns (bool) {

        uint8 tokenId = newnft.mintToken(owner, collectibles_bought[NFT_class],
NFT_class);

        NFT_minted.push(tokenId);

        if(print)
            emit NFT_minted_now (owner, collectibles_bought[NFT_class], NFT_class,
tokenId);
```

```
        return true;
    }





    function start_New_Round() onlyOperator initPhaseActive  external returns (bool
res) { //non deve avere modifier rewardPhase perche viene invocato anche nel
construttore

        uint8 i;
        if(first_init_round){ //IF A LOTTERY'S ENDING => CLEAN ALL DATA STRUCTURE AND
RESTART

            //mint the first 8 NFTs, one for each class
            for(i =1; i< 9; i++)
                mint(i, operator, false); //(class, owner, bool that def if print or
not the minting)

            first_init_round = false;

        }
        else{
            delete tickets_sold;

            for( i=0; i<5;i++)
                delete drawn_numbers_mapping[drawn_numbers[i]];

            delete drawn_numbers;

            for(i=0; i<winners.length;i++)
                delete reward_list[winners[i]];

            delete winners;

            payable(balance_receiver).transfer(address(this).balance);
        }


        block_number_init = uint64(block.number); //SET THE NEW INITIAL BLOCK NUMBER

        lottery_phase_operator="Buy_phase";
        emit phase_change(lottery_phase_operator);
        res = true;
        return res;
    }
```

index.html

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
crossorigin="anonymous">
    <link rel="stylesheet" href="./css/Style.css">

    <title>TRY Lottery</title>
</head>

<body>

    <!-- COMMON INTERFACE FIRST PART START-->


    <!-- Main title -->
    <div class="box" id="title">
        <h1 class="text-center">TRY Lottery Dashboard</h1>
    </div>

    <!-- First block of common info displayed -->
    <div class='parent'>
        <div class='child'>
            <h3 class="text-left" id="accountId">Your address: </h3>
        </div>

        <div class='child'>
            <h3 class="text-left" id="accountType">Account type: </h3>
        </div>

        <div class='child'>
            <h3 class="text-left" id="accountBalance">Account balance: </h3>
        </div>
    </div>

    <!-- COMMON INTERFACE FIRST PART END-->


    <!-- OPERATOR INTERFACE INIZIO -->
    <div id="operator_interface">

        <!-- dividing line -->
        <hr
style="height:40%;width:70%;border-width:1%;color:gray;background-color:gray;size:
2px;">

        <!-- Commands string -->
        <div class='parent'>
            <div class='child'>
                <h2 class="text-center">Commands for the lottery handling: </h2>
            </div>
        </div>

        <!-- First 5 Buttons -->
        <div class="cards">
            <button id="createLottery" class="card"
onclick="App.createLottery()">Create Lottery</button>
            <button class="card" onclick="App.closeLottery()">Close Lottery</button>
            <button class="card" onclick="App.startNewRound()">Start New Round</button>
            <button class="card" onclick="App.drawNumbers()">Draw Numbers</button>
            <button class="card" onclick="App.computePrizes()">Compute Prizes</button>
            <button class="card" onclick="App.givePrizes()">Give Prizes</button>
        </div>

        <!-- Commands string -->
        <div class='parent'>
            <div class='child'>
```

```
                <h2 class="text-center">Set here the balance receiver: </h2>
            </div>
        </div>

        <!-- Last button + textarea for BALANCE RECEIVER -->
        <div class="cards-second">
            <input  type="text" id="balanceReceiver" placeholder="Insert Address">
            <input type="button" value="Set Balance receiver" class="card-second"
onclick="App.setBalancerReceiver()"></button>
        </div>

        <!-- Commands string -->
        <div class='parent'>
            <div class='child'>
                <h2 class="text-center">Buy here the collectibles: </h2>
            </div>
        </div>

        <!-- button + textarea for BUY COLLECTIBLE -->
        <div class="cards-second" >
            <input  type="number" id="collectible_input" min="1" max="8" size="10"
maxlength="1" placeholder="Insert ID">
            <button class="card-second" onclick="App.buyCollectible()">Buy Single
Collectible </button>
            <button class="card-second" onclick="App.buyAllCollectibles()">Buy All
Collectibles </button>
        </div>

        <!-- dividing line -->
        <hr
style="height:40%;width:70%;border-width:1%;color:gray;background-color:gray">


        <!-- Second block of info displayed -->
        <div class='parent'>

            <div class='child'>
                <h3 class="text-left" id="balanceReceiverFieldOperator">Balance
Receiver: </h3>
            </div>

            <div class='child'>
```

```html
                <h3 class="text-left" id="listCollectiblesBoughtOperator">List of 
Collectibles bought: </h3>
            </div>

            <div class='child'>
                <h3 class="text-left" id="listNFTsMintedOperator">List of NFTs Minted: 
</h3>
            </div>
        </div>

        <!-- dividing line -->
        <hr 
style="height:40%;width:70%;border-width:1%;color:gray;background-color:gray">


        <!-- Thrid block of info displayed -->
        <div class='parent'>

            <div class='child'>
                <h3 class="text-left" id="lotteryBalanceOperator">Lottery balance: 
</h3>
            </div>

            <div class='child'>
                <h3 class="text-left" id="lotteryPhaseOperator">Lottery Phase: </h3>
            </div>

            <div class='child'>
                <h3 class="text-left" id="ticketsSoldOperator">List of Tickets Sold: 
</h3>
            </div>

            <div class='child'>
                <h3 class="text-left" id="lotteryStateOperator">Lottery State: </h3>
            </div>

            <div class='child'>
                <h3 class="text-left" id="drawnNumbersOperator">Drawn numbers: </h3>
            </div>

        </div>
```

```html
    </div> <!-- OPERATOR INTERFACE FINE -->




    <!-- USER INTERFACE INIZIO -->
    <div id="user_interface">

        <!-- dividing line -->
        <hr
style="height:40%;width:70%;border-width:1%;color:gray;background-color:gray">

        <!-- Commands string -->
        <div class='parent'>
            <div class='child'>
                <h2 class="text-center">Insert the 6 numbers and buy a ticket: </h2>
            </div>
        </div>

        <!-- button + textarea for BUY TICKET -->
        <div class="cards-second" >
            <input  type="number" id="number_input1"  min="1" max="69"
placeholder="First" required>
            <input  type="number" id="number_input2"  min="1" max="69"
placeholder="Second" required>
            <input  type="number" id="number_input3"  min="1" max="69"
placeholder="Third" required>
            <input  type="number" id="number_input4"  min="1" max="69"
placeholder="Fourth" required>
            <input  type="number" id="number_input5"  min="1" max="69"
placeholder="Fifth" required>
            <input  type="number" id="number_input6"  min="1" max="26"
placeholder="Powerball" required>
            <button class="card-second" onclick="App.buyTicket()">Buy ticket</button>
        </div>




        <!-- dividing line -->
        <hr
style="height:40%;width:70%;border-width:3%;color:gray;background-color:gray">

        <!-- Second block of info displayed -->
        <div class='parent'>
```

```html
            <div class='child'>
                <h3 class="text-left" id="lotteryPhaseUser">Lottery Phase: </h3>
            </div>

            <div class='child'>
                <h3 class="text-left" id="ticketsBoughtUser">List of Tickets Bought:
</h3>
            </div>

            <div class='child'>
                <h3 class="text-left" id="NFTWonUser">List of rewards obtained: </h3>
            </div>


            <div class='child'>
                <h3 class="text-left" id="lotteryStateUser">Lottery State: </h3>
            </div>

            <div class='child'>
                <h3 class="text-left" id="drawnNumbersUser">Drawn numbers: </h3>
            </div>

        </div>


    </div> <!-- USER INTERFACE FINE -->






    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js"
integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49"
crossorigin="anonymous"></script>
    <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js"
```

```html
integrity="sha384-ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE241rYiqJxyMiZ6OW/JmZQ5stwEULTy"
crossorigin="anonymous"></script>
    <!-- Web3 -->
    <script src="./dist/web3.min.js"></script>
    <script src="./dist/truffle-contract.js"></script>
    <!-- App -->
    <script src="./js/app.js"></script>

</body>
</html>
```

app.js

```javascript
App = { // Object

    //Object's attributes
    contracts: {},
    web3Provider: null,             // Web3 provider
    url: 'http://localhost:8545',   // Url for web3
    account: '0x0',                 // address of the account detected on Metamask
    account_balance: 0,             // balance of that address detected
    current_account_type : "operator",
    operator: '0x0',                // address of the lottery operatore
    lottery_state : "Closed",         // state of the lottery saved locally
    lottery_phase : "Not_created",  // phase of the lottery saved locally
    price : 1000000000000000000n,   //price of a collectible/ticket
    max_collectible_id : 0,         // max num of collectibles to be bought
    lottery_created:false,


    /* 0 */
    init: function() {
        return App.initWeb3();
    },

    /* 1 */
    /* INITIALIZE THE WEB3 */
    initWeb3: function() {
        console.log("Entered")

        if(typeof web3 != 'undefined') {
            App.web3Provider = window.ethereum;
            web3 = new Web3(App.web3Provider);
            try {
                    ethereum.request({ method: 'eth_requestAccounts' }).then(async() =>
{ //before was ethereum.enable. Used to connect the DApp with Metamask
                        console.log("DApp connected to Metamask");
                    });
            }
            catch(error) {
                console.log(error);
            }
        } else {
            App.web3Provider = new Web3.providers.HttpProvider(App.url);
            web3 = new Web3(App.web3Provider);
        }
        return App.initContract();
    },

    /* 2 */
    /* UPLOAD THE CONTRACT ABSTRACTION */
    initContract: function() {
        console.log("Dentro initcontract");
        // Get current account
        web3.eth.getCoinbase(function(err, account) {
            if(err == null) {
                App.account = account.toLowerCase(); //set App.account
            }
        });

        // Load content's abstractions, taking the compiled contract
        $.getJSON("Lottery.json").done(function(c) {
            App.contracts["Contract"] = TruffleContract(c);
            App.contracts["Contract"].setProvider(App.web3Provider);
            return App.initDApp();
        });
    },

    /* 3 */
    /* INITIALIZE DAPP STATE VARIABLES & UI */
    initDApp: function(){
        console.log("Dentro initOperator");

            //get the lottery operator (set by the contract)
            get_lottery_operator();

            //set the max collectible id (set by the contract)
            set_max_collectible_id();

            //get the contract balance
            get_contract_balance();


            //get the balance receiver (if set, else ret the 0x0 address)
            get_balance_receiver();

            //get the list of all collectibles bought
            get_list_collectibles_bought();


            //get the list of NFTs minted (if set)
            get_list_NFTs();


            //get the lottery phase
            get_lottery_phase();

            //get the list of all tickets bought so far
            get_list_tickets();

            //get the lottery state
            get_lottery_state();

            //get the numbers extracted randomly
            get_drawn_numbers();

        return App.listenForEvents();
    },


    /* 4 */
    /*SET OF FUNCTION HANDLERS FOR EACH EVENT EMITTED */
    listenForEvents: function() {
        console.log("Dentro listen");
        App.contracts["Contract"].deployed().then(async (instance) => {

            //event that notifies lottery creation
            instance.lottery_created().on('data', function (event) {
                App.lottery_phase="Created";
                App.lottery_state="Open";
                $("#lotteryPhaseOperator").html("Lottery Phase: Created");
                $("#lotteryPhaseUser").html("Lottery Phase: Created");
                $("#lotteryStateUser").html("Lottery State: Open");
                $("#lotteryStateOperator").html("Lottery State: Open");
                console.log("eventoPReso2"+event);
            });

            //event that notifies when the balance receiver address is set
            instance.bal_initialized().on('data', function (event) {
                $("#balanceReceiverFieldOperator").html("Balance Receiver: 
"+event.args.receiver);
                console.log("eventoPReso2"+event.args.receiver);
            });

            //event that notifies when a collectible is bought
            instance.collectible_bought().on('data', function (event) {
                $("#listCollectiblesBoughtOperator").append("<br> "+event.args[1]);
                console.log(event);
            });

            //event that notifies when a NFT is minted
            instance.NFT_minted_now().on('data', function (event) {
                $("#listNFTsMintedOperator").append("<br>Owner: 
"+event.returnValues[0]+"<br>NFT description: "+event.returnValues[1]+"<br> NFT class: 
"+event.returnValues[2]+"<br> NFT ID: "+event.returnValues[3]+"<br>");
                if(App.account == event.returnValues[0].toLowerCase() && App.account != 
App.operator){
                    $("#NFTWonUser").append("<br>NFT description: 
"+event.returnValues[1]+"<br> NFT class: "+event.returnValues[2]+"<br> NFT ID: 
"+event.returnValues[3]+"<br>");
                }
                console.log(event);
            });

            //event that notifies when a NFT ownership is transfered (usually from the 
operator to a winner)
            instance.NFT_transfered().on('data', function (event) {
                $("#listNFTsMintedOperator").append("<br> Last NFT Transfered: Owner: 
"+event.returnValues[0]+"<br>NFT description: "+event.returnValues[1]+"<br> NFT class: 
"+event.returnValues[2]+"<br> NFT ID: "+event.returnValues[3]+"<br>");
```

```
                if(App.account == event.returnValues[0].toLowerCase()){
                    $("#NFTWonUser").append("<br>NFT description:
"+event.returnValues[1]+"<br> NFT class: "+event.returnValues[2]+"<br> NFT ID:
"+event.returnValues[3]+"<br>");
                }
                console.log(event);
            });

            //event that notifies when the phase of the lottery changes (the value of
the variable "lottery_phase_operator")
            instance.phase_change().on('data', function (event) {
                App.lottery_phase = event["returnValues"][0];
                $("#lotteryPhaseOperator").html("Lottery Phase:
"+event["returnValues"][0]);
                $("#lotteryPhaseUser").html("Lottery Phase:
"+event["returnValues"][0]);
            });

            //event that notifies when a ticket is bought
            instance.ticket_bought().on('data', function (event) {
                $("#ticketsSoldOperator").append("<br> Owner:
"+event["returnValues"][0]+ "<br> values: ["+event["returnValues"][1]+"]<br>");
                if(App.account == event.returnValues[0].toLowerCase()){

$("#ticketsBoughtUser").append("["+event["returnValues"][1]+"]"+"<br> ");
                }
                console.log("TICKET"+event["returnValues"][0]+"
"+event["returnValues"][1]);
            });

            //console.log(Object.getOwnPropertyNames(object1)); -> useful to inspect
inside the event obj

            //event that notifies when the lottery is definetively closed
            instance.lottery_closed().on('data', function (event) {
                App.lottery_state = "Closed";
                $("#lotteryStateOperator").html("Lottery State: Closed");
                $("#lotteryPhaseOperator").html("Lottery Phase: Closed");
                $("#lotteryStateUser").html("Lottery State: Closed");
                $("#lotteryPhaseUser").html("Lottery Phase: Closed");
                console.log(event);
            });
            //event that notifies when the values are drawn randomically
            instance.values_drawn().on('data', function (event) {
                $("#drawnNumbersOperator").html("Drawn numbers: 
"+event.returnValues[0]);
                $("#drawnNumbersUser").html("Drawn numbers: "+event.returnValues[0]);
                console.log(event);
            });

        });

        /* detect the change in the metamask account -> then update the UI */
        ethereum.on('accountsChanged', function (accounts) {
            console.log("MetaMask account changed");
            App.setAccountType(1); //arg is 1 since the Metamask acc changed
        });
    },


    /* SET OF FUNCTIONS DIRECTLY INVOKED BY THE DAPP (through the buttons, except for 
the first two) */

    //Invoked when the account on MetaMask change (whyIsInvoked=1) or when a page is 
reloaded (whyIsInvoked=0)
    //This fun analyze the account on metamask, set some account variables and update 
the UI regarding the account information
    setAccountType: async function(whyIsInvoked){

        //get the metamask account address
        let accounts = await window.ethereum.request({ method: 'eth_requestAccounts' 
});
        App.account = accounts[0].toLowerCase();

        //get its balance
        App.account_balance = await web3.utils.fromWei(await 
web3.eth.getBalance(accounts[0]))

        // check if the account is the lottery operator or not
        if(App.account == App.operator)
            App.current_account_type="operator";
        else
            App.current_account_type="user";
        console.log("setAccountType detect: " + App.account + " of type: " +
App.current_account_type);

        // update the info about the account
        $("#accountId").html("Your address: " + App.account);
        $("#accountType").html("Account type: " + App.current_account_type);
        $("#accountBalance").html("Account balance: " + App.account_balance);

        // based on the account type, display a specific interface (Operator vs User)
        set_interface(whyIsInvoked);
    },

    //Set the address of the account of the balance Receiver (this account will not be
able to play/buy tickets)
    setBalancerReceiver: function() {
        if(App.lottery_phase != "Created"){
            alert("Wrong phase for this action, actual phase is: "+App.lottery_phase);
            return;
        }
        App.contracts["Contract"].deployed().then(async(instance) =>{
            try{
                let address_string =
document.getElementById('balanceReceiver').value.toLowerCase(); // <input name="one">
element
                let address = web3.utils.toChecksumAddress(address_string);

                if(App.operator.toLowerCase() == address.toLowerCase()){
                    alert("Operator can't be a balance Receiver");
                    return;
                }

                await instance.set_balance_receiver(address,{from: App.account});
            }
            catch(err){
                alert(err);
            }
        });
    },

    //start a new lottery round
    startNewRound: function() {
        if(App.lottery_phase != "Init_phase"){
            alert("Wrong phase for this action, actual phase is: "+App.lottery_phase);
            return;
        }
        App.contracts["Contract"].deployed().then(async(instance) =>{
            try{
                await instance.start_New_Round({from: App.account});
                //set/reset some UI information
                $("#ticketsBoughtUser").html("List of Tickets Bought: ");
                $("#ticketsSoldOperator").html("List of Tickets Sold: ");
                $("#drawnNumbersOperator").html("Drawn numbers: Will be drawn later");
                $("#drawnNumbersUser").html("Drawn numbers: Will be drawn later");
                get_contract_balance();
                get_list_NFTs();
            }
            catch(err){
                alert(err);
            }
        });
    },

    //create the lottery
    createLottery: function() {
        if(App.lottery_phase != "Not_created"){
            alert("Wrong phase for this action, actual phase is: "+App.lottery_phase);
            console.log(App.lottery_phase == "Not_created");
            console.log("PHASE: "+App.lottery_phase);
            return;
        }
        App.contracts["Contract"].deployed().then(async(instance) =>{
            try{
                await instance.create_lottery({from: App.account});
            }
            catch(err){
                alert(err);
            }
        });

    },

    //draw the number randomically
    drawNumbers: function() {
        if(App.lottery_phase != "Extraction_phase"){
            alert("Wrong phase for this action, actual phase is: "+App.lottery_phase);
            return;
        }
        App.contracts["Contract"].deployed().then(async(instance) =>{
            try{
                await instance.draw_numbers({from: App.account});
            }
            catch(err){
                alert(err);
            }
        });
    },

    //compare the drawn number with the ticket numbers and declare the winners
    computePrizes: function() {
        if(App.lottery_phase != "Compute_prizes_phase"){
            alert("Wrong phase for this action, actual phase is: "+App.lottery_phase);
            return;
        }
        App.contracts["Contract"].deployed().then(async(instance) =>{
            try{
                await instance.compute_prizes({from: App.account});
            }
            catch(err){
                alert(err);
            }
        });
    },

    //assign the prizes (NFTs) to the winners
    givePrizes: function() {
        if(App.lottery_phase != "Give_prizes_phase"){
            alert("Wrong phase for this action, actual phase is: "+App.lottery_phase);
            return;
        }
        App.contracts["Contract"].deployed().then(async(instance) =>{
            try{
                await instance.give_prizes({from: App.account});
            }
            catch(err){
                alert(err);
            }
        });
    },
    //close defintetively the lottery, once done it can't be opened again
    closeLottery: function() {
        if(App.lottery_state != "Open"){
            alert("Wrong state for this action, actual state is: "+App.lottery_state);
            App.contracts["Contract"].deployed().then(async(instance) =>{
            let res = await instance.lottery_state({from: App.account});
            console.log("STATE: "+res);
            });
            return;
        }
        App.contracts["Contract"].deployed().then(async(instance) =>{
            try{
                await instance.close_lottery({from: App.account});
            }
            catch(err){
                alert(err);
            }
        });
    },

    //buy a single collectible
    buyCollectible: function() {
        if(App.lottery_phase != "Created"){
            alert("Wrong phase for this action, actual phase is: "+App.lottery_phase);
            return;
        }
        App.contracts["Contract"].deployed().then(async(instance) =>{
            try{
                let collectible_id =
document.getElementById('collectible_input').value; // <input name="one"> element
                await instance.buy_collectibles(collectible_id, {from: App.account,
value: App.price.toString()});
            }
            catch(err){
                alert("Wrong collectible or collectible already bought");
            }
            let accounts = await window.ethereum.request({ method:
'eth_requestAccounts' });
            App.account_balance = await web3.utils.fromWei(await
web3.eth.getBalance(accounts[0]));
            $("#accountBalance").html("Account balance: " + App.account_balance);
        });
    },
    //buy all the 8 collectibles in a row
    buyAllCollectibles: function() {
        if(App.lottery_phase != "Created"){
            alert("Wrong phase for this action, actual phase is: "+App.lottery_phase);
            return;
        }
        App.contracts["Contract"].deployed().then(async(instance) =>{
            try{
                for(i=1; i<9; i++)
                    await instance.buy_collectibles(i, {from: App.account, value:
App.price.toString()});
            }
            catch(err){
                alert("Wrong collectible or collectible already bought");
            }
            let accounts = await window.ethereum.request({ method:
'eth_requestAccounts' });
            App.account_balance = await web3.utils.fromWei(await
web3.eth.getBalance(accounts[0]));
            $("#accountBalance").html("Account balance: " + App.account_balance);
        });
    },

    //buy a single ticket (only fun invoked by a User account)
    buyTicket: function() {
        if(App.lottery_phase != "Buy_phase"){
            alert("Wrong phase for this action, actual phase is: "+App.lottery_phase);
            return;
        }
        if(App.account == App.balance_receiver_address.toLowerCase()){
            alert("You can't buy tickets ");
            return;
        }
        App.contracts["Contract"].deployed().then(async(instance) =>{
            try{
                let input_values = [];
                for(let i =1; i< 7; i++)

input_values.push((document.getElementById('number_input'+i).value));

                console.log("VALORI "+input_values);
                await instance.buy_ticket(input_values, {from: App.account, value: 
App.price.toString()});
            }
            catch(err){
                //console.log(Object.getOwnPropertyNames(err));
                alert(err);
            }
            let accounts = await window.ethereum.request({ method: 
'eth_requestAccounts' });
            App.account_balance = await web3.utils.fromWei(await 
web3.eth.getBalance(accounts[0]));
            $("#accountBalance").html("Account balance: " + App.account_balance);
            get_contract_balance();

        });
    },

}



/* SET OF FUNCTIONS USED TO GET INFORMATION FROM THE SMART CONTRACT  */

function get_lottery_operator(){
    App.contracts["Contract"].deployed().then(async(instance) =>{
        let res = await instance.operator({from: App.account});
        App.operator = res.toLowerCase();
        //set the account informations
        App.setAccountType(0); //arg is 0 since the page is reloaded/load for the first 
time
    });
}

function set_max_collectible_id(){
    App.contracts["Contract"].deployed().then(async(instance) =>{
        App.max_collectible_id = await instance.get_max_collectible_id({from: 
App.account});
    });
}

function get_contract_balance(){
    App.contracts["Contract"].deployed().then(async(instance) =>{
        let res = await instance.get_contract_balance({from: App.account});
            $("#lotteryBalanceOperator").html("Lottery balance: "+res);
    });
}



function get_balance_receiver(){
    App.contracts["Contract"].deployed().then(async(instance) =>{
        let res = await instance.balance_receiver({from: App.account});
        $("#balanceReceiverFieldOperator").html("Balance Receiver: "+res);
        App.balance_receiver_address = res.toLowerCase();
    });
}

function get_list_collectibles_bought(){
    App.contracts["Contract"].deployed().then(async(instance) =>{
        let res;
        let num = await instance.get_num_collectibles_bought({from: App.account});
            if(num>0){ //if some collectibles are bought
                $("#listCollectiblesBoughtOperator").html("List of Collectibles bought:
"); //clean the UI field
                for(let i = 0; i < num; i++){ //add all of them in the UI
                    res = await instance.get_collectible_info(i, {from: App.account});
                    $("#listCollectiblesBoughtOperator").append("<br> "+res);
                }
            }
            console.log("LCB "+res);
    });
}

function get_list_NFTs(){
    App.contracts["Contract"].deployed().then(async(instance) =>{
        let res;
        let num = await instance.get_num_NFTs_minted({from: App.account});
            if(num>0){ //if some already minted

                //clean the UI field
                $("#listNFTsMintedOperator").empty();
                $("#listNFTsMintedOperator").html("List of NFTs Minted: ");

                $("#NFTWonUser").empty();
                $("#NFTWonUser").html("List of rewards obtained: ");
                for(let i = 0; i < num; i++){
                    res = await instance.get_NFT_information(i, {from: App.account});
                    console.log("NTFMINTATO "+res[0]);
                    //add all of them to the UI of the Operator
                    $("#listNFTsMintedOperator").append("<br>Owner: "+res[0]+"<br>NFT
description: "+res[1]+"<br> NFT class: "+res[2]+"<br> NFT ID: "+res[3]+"<br>");

                    //add all of them to the UI of the User IFF the owner of the NFT is
the actual user account on MetaMask
                    if(App.account == res[0].toLowerCase() && App.account !=
App.operator){
                        $("#NFTWonUser").append("<br>NFT description: "+res[1]+"<br>
NFT class: "+res[2]+"<br> NFT ID: "+res[3]+"<br>");
                    }
                }
            }
    });
}

function get_lottery_phase(){
    App.contracts["Contract"].deployed().then(async(instance) =>{
        let res = await instance.lottery_phase_operator({from: App.account});
            App.lottery_phase = res;
            $("#lotteryPhaseOperator").html("Lottery Phase: "+res);
            $("#lotteryPhaseUser").html("Lottery Phase: "+res);


    });
}

function get_list_tickets(){
    App.contracts["Contract"].deployed().then(async(instance) =>{
        let res;
        try{
            let num = await instance.get_num_tickets_sold({from: App.account});
            if(num>0){//if some already bought

                //clean the UI field
                $("#ticketsSoldOperator").empty();
                $("#ticketsSoldOperator").html("List of Tickets Sold: ");

                $("#ticketsBoughtUser").empty();
                $("#ticketsBoughtUser").html("List of Tickets Bought: ");
                for(let i = 0; i < num; i++){
                    res = await instance.get_ticket_information(i, {from:
App.account});
```

```
                    //add to the UI operator
                    if(res[1][0]==0){ //res[0] = address, res[1] = array di valori
scelti
                        res = "No ticket bought so far";
                        $("#ticketsSoldOperator").html("Last Ticket Bought: "+res);
                    }
                    else
                        $("#ticketsSoldOperator").append("<br> Owner: "+res[0]+"<br>
values: ["+res[1]+"] <br>");
                    //add to the UI User IFF the owner of the ticket is the same
account of the MetaMask user
                    if(App.account == res[0].toLowerCase())
                        $("#ticketsBoughtUser").append("<br>["+res[1]+"] ");
                }
            }
        }
        catch{
            res = "No ticket bought so far";
            $("#ticketsSoldOperator").html("Last Ticket Bought: "+res);
            $("#ticketsSoldUser").html("Last Ticket Bought: "+res);
        }

    });
}

function get_lottery_state(){
    App.contracts["Contract"].deployed().then(async(instance) =>{
        let res = await instance.lottery_state({from: App.account});
            $("#lotteryStateUser").html("Lottery State: "+res);
            $("#lotteryStateOperator").html("Lottery State: "+res);
            App.lottery_state=res;
            console.log("STATE2: "+res);
    });
}

function get_drawn_numbers(){
    App.contracts["Contract"].deployed().then(async(instance) =>{
            //get the drawn numbers
            let res;
            try{
                res = await instance.get_drawn_numbers({from: App.account});
            }
            catch{
                res = "Will be drawn later";
            }
            if(res[0]==0)
                res = "Will be drawn later";
            $("#drawnNumbersOperator").html("Drawn numbers: "+res);
            $("#drawnNumbersUser").html("Drawn numbers: "+res);



    });
}



/* Set the specific UI (Operator or User) based on the MetaMask account address  */
function set_interface(whyIsInvoked){
    console.log("dentro change");

    //Based on account type display a specific interface
    if(App.current_account_type == "operator"){
        $("#operator_interface").show();
        $("#user_interface").hide();
    }
    if(App.current_account_type == "user"){
        $("#user_interface").show();
        $("#operator_interface").hide();
    }

    //if whyIsInvoked==0 => this fun is invoked on a page load/refresh, so the function
App.initDApp set the right interface => no more UI add here
    //if whyIsInvoked==1 => this fun is invoked on an eth account change, so just the
NFTs and Ticket must be recomputed for the specific account => UI add
    if(whyIsInvoked==1){
        get_list_NFTs();
        get_list_tickets();
        get_lottery_phase();
        get_lottery_state();
    }

}



// Call init whenever the window loads
$(function() {
    $(window).on('load', function () {
        App.init();
    });
});
```