

Универзитет у Крагујевцу
Факултет инжењерских наука



Пројектни задатак – документација

Тема:

Фер игра на срећу имплементирана коришћењем Ethereum
паметних уговора

Студент:
Ђорђе Гачић 626/2018

Предметни професор:
др. Владимир Миловановић

Крагујевац, септембар 2021. године

Садржај

1. Увод.....	2
2. Опис функционисања лутрије	2
3. Приказ рада апликације	3
4. Опис паметног уговора.....	7
5. Структура пројекта и опис дијелова кода.....	14
6. Закључак	34
7. Литература	35

1. Увод

У новије вријеме све више се говори о blockchain технологији која има огроман потенцијал за примјену у разним софтверским пројектима. Најраспрострањенија примјена до сада је примјена у криптовалутама. Поред тога све чешће се помиње увођење blockchain технологије у коцкарску индустрију јер омогућава фер и провјерљиве резултате клађења. У овом пројекту је за израду једне такве фер игре на срећу коришћена Ethereum технологија паметних уговора.

Да бисмо приближили кориснику начин функционисања паметних уговора потребно је напоменути да се читав њихов смисао заснива на blockchain технологији и да оно што је једном записано у ланац блокова не може бити промијењено нити обрисано захваљујући прије свега хеш функцијама. Најпростије објашњење паметног уговора било би то да је то код уграђен у blockchain који има властиту адресу која треба да буде опште позната јер ако покренемо код са те адресе бићемо сигурни да је то није измијењен код.

У овом пројекту биће имплементиран један такав уговор у програмском језику Solidity, који је иначе намијењен за писање паметних уговора, као и Web апликација коришћењем Django фрејмворка и Python програмског језика како би се кориснику пружио одговарајући графички интерфејс.

2. Опис функционисања лутрије

Да би се схватио начин функционисања апликације потребно је упознати се са принципом на којем се одвија лутрија и извлачење. Ова апликација замишљена је и имплементирана тако да обезбједи кориснику да креира своју лутрију или да се прикључи са одређеним улогом већ креираној лутрији. При креирању сваке лутрије покреће се исти паметни уговор на истој адреси и свакој лутрији се додијаљујњ идентификатор који представља редни број лутрије у низу свих креираних лутрија.

Извлачење добитника се одвија на основу индекса који опклада учесника има у низу опклада које учествују у истој лутрији. На примјер, када корисник креира лутрију са одређеном количином у Ether валути тада ће његов индекс бити 0, следећа опклада која се појави имаће индекс 1, затим 2 итд. Сваки корисник мора да уложи најмање просјечну до тада уложену вриједност Ether валуте у ту лутрију којој хоће да се прикључи. Максималан број учесника у некој лутрији задаје креатор и извлачење добитника се дешава онда када се прикључи тачно онолико учесника колико је креатор дефинисао (никако раније). Тада се лутрија на тој адреси затвара и није више доступна за улагање, али се и даље чувају информације о њој у паметном уговору тако да су увијек доступне за провјеру. Случајан број на основу кога се извлачи добитник је излаз хеш функције чији

улаз представљају подаци о свим опкладама које су учествовале, адреса рудара тог блока и timestamp тог блока у коме ће да се налази трансакција на адресу добитника.

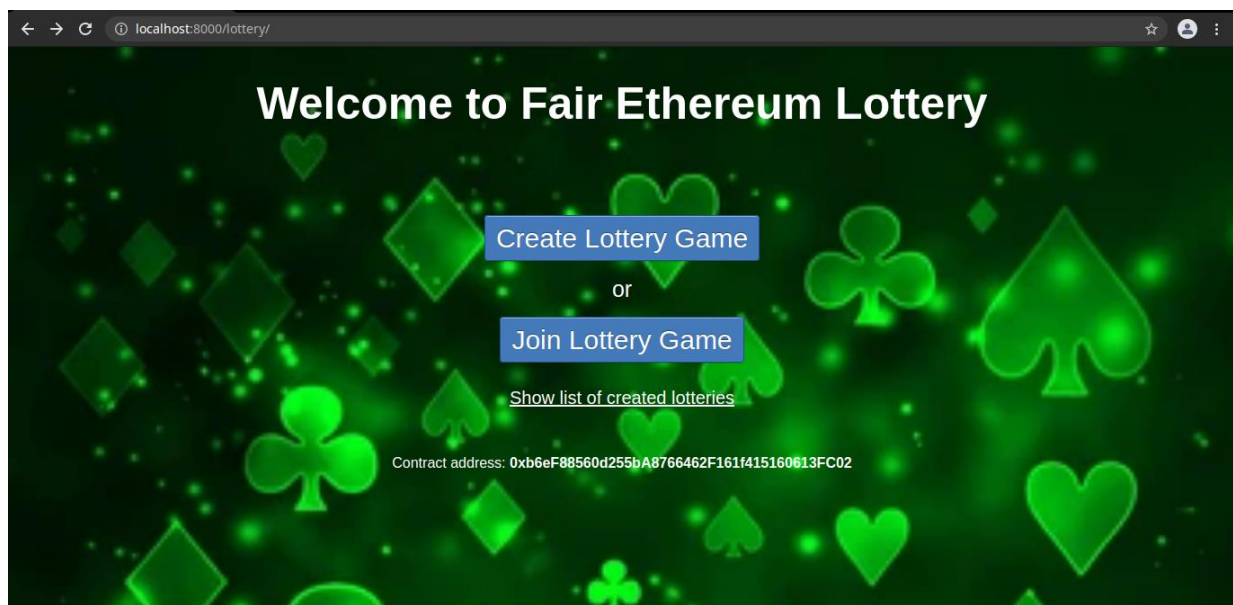
Идентификација корисника врши се преко адресе корисника која се представља хексадецималном вриједношћу дужине 40 хексадецималних цифара, односно 160 бинарних цифара. Свакој исправној адреси одговара приватни кључ који се користи за потврду идентитета корисника као и за потписивање трансакција. Дужина приватног кључа у хексадецималном запису је 64 цифре, односно 256 цифара у бинарном запису. Примјер адресе и приватног кључа који јој одговара дат је испод:

Адреса: 0x9b341e66c23ee794e2df72843884817ee819ad02

Приватни кључ: 0e3a1ce09d529274a38dd863fcd7e904b62e462d0386efdfdc99b925791a3b21

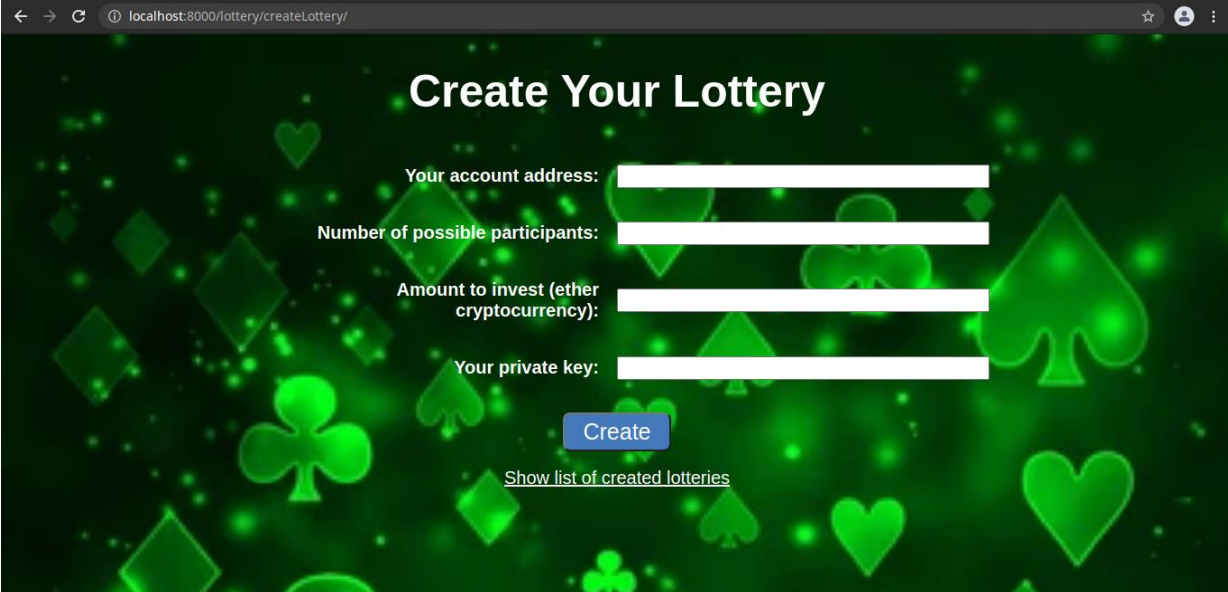
3. Приказ рада апликације

На почетној страни кориснику се нуди опција да креира лутрију, да се придружи некој од већ креираних лутрија или да погледа листу већ креираних лутрија.



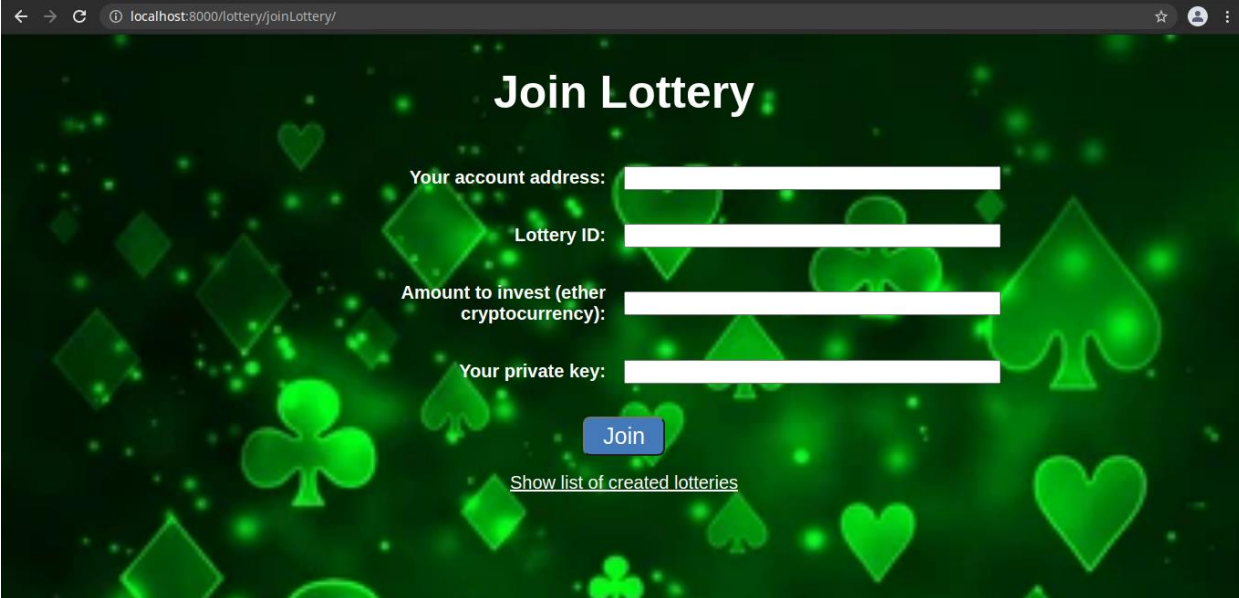
Слика 1: Почетна страна

При креирању лутрије потребно је да корисник унесе своју Ethereum адресу, вриједност коју улаже у лутрију (која ће уједно бити и почетна минимална вриједност за улагање осталих учесника), број учесника и на крају приватни кључ који одговара претходно унијетој адреси.



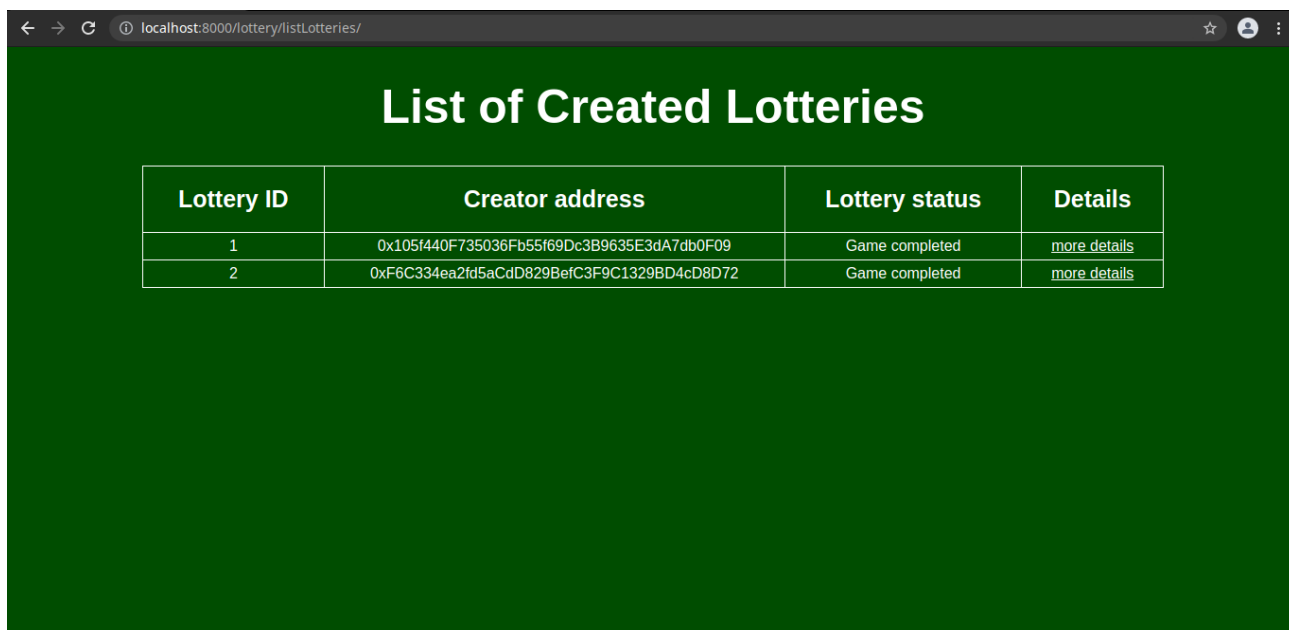
Слика 2: Страна за креирање лутрије

При придруживању некој од лутрија потребно је да корисник унесе своју Ethereum адресу, идентификатор жељене лутрије, вриједност коју улаже у лутрију (која не може бити мања од просјечне вриједности до тада уложене у ту лутрију), и на крају приватни кључ који одговара претходно унијетој адреси.



Слика 3: Страна за придруживање лутрији

Листа већ креираних лутрија садржи идентификатор сваке од лутрија, адресу креатора, статус лутрије (комплетирана или доступна за улагање). Такође, за сваку од лутрија оставља се могућност приказа детаља те лутрије.

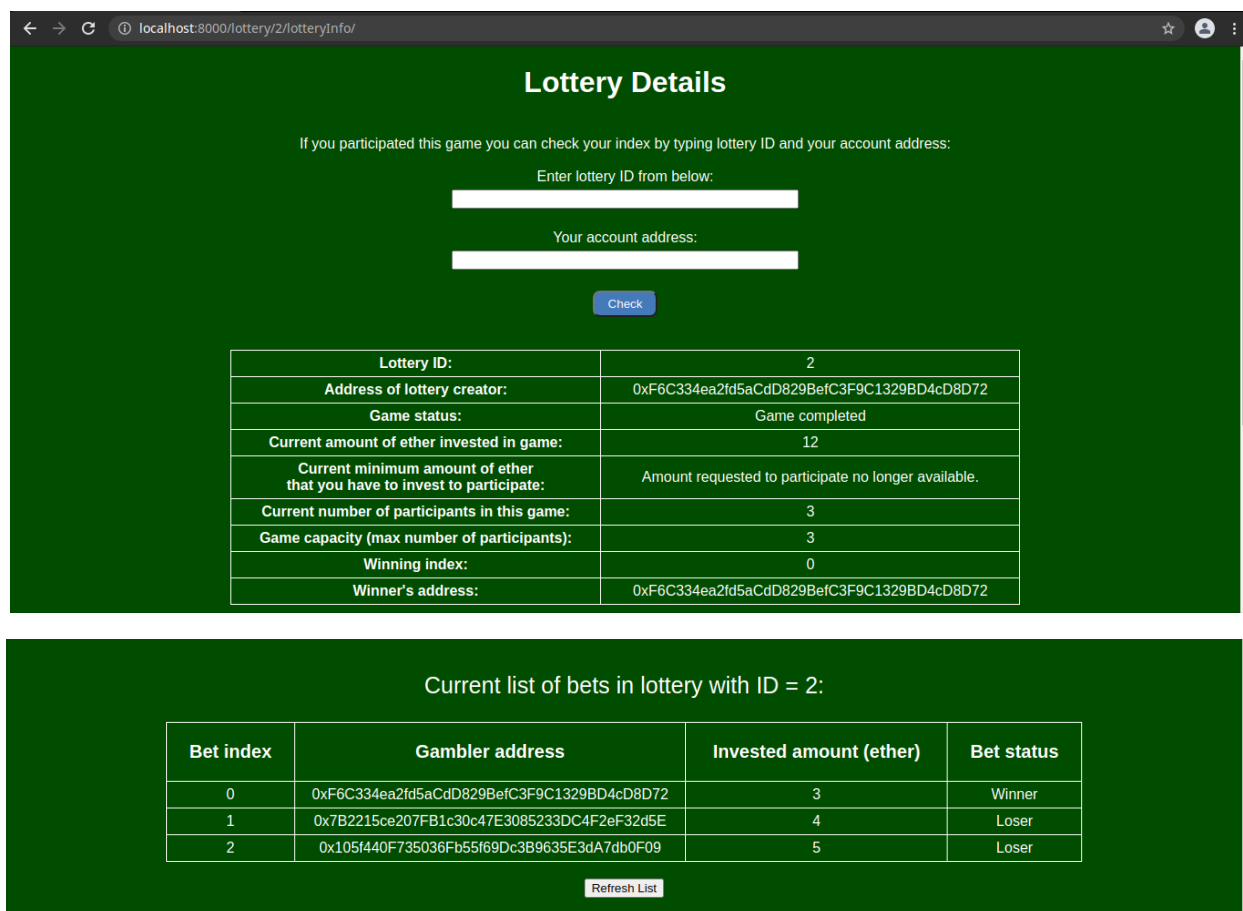


Lottery ID	Creator address	Lottery status	Details
1	0x105f440f735036fb55f69dc3b9635e3da7db0f09	Game completed	more details
2	0xf6c334ea2fd5acd829befc3f9c1329bd4cd8d72	Game completed	more details

Слика 4: Страна са листом креираних лутрија

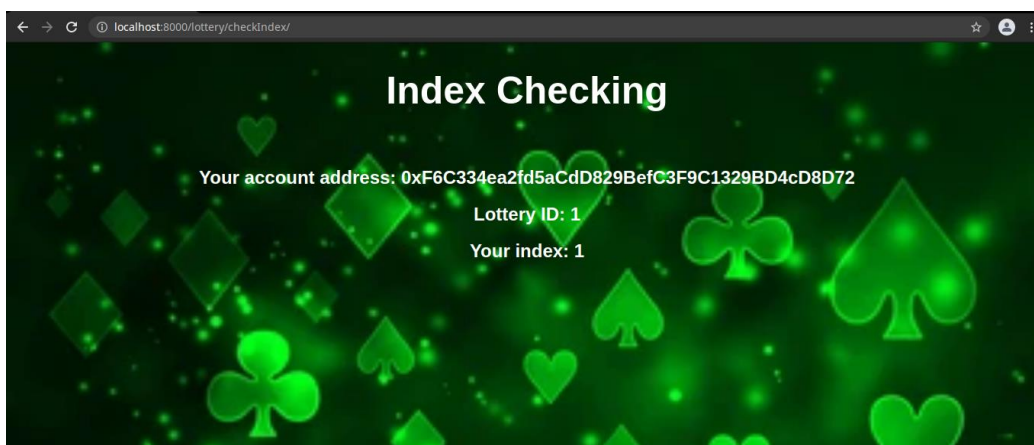
Страница са детаљима лутрије садржи следеће податке:

- Идентификатор лутрије
- Адресу креатора лутрије
- Статус лутрије (отворена или завршена)
- Тренутну укупну вриједност до тада уложену у ту лутрију (у Ether валути)
- Тренутну захтјевану вриједност улога за учешће (у Ether валути)
- Тренутни број опклада (учесника) у тој лутрији
- Предвиђени број учесника (капацитет лутрије)
- Побједнички индекс (уколико је лутрија завршена)
- Адреса добитника (уколико је лутрија завршена)
- Тренутну листу опклада гдје свака опклада садржи: индекс опкладе, адресу учесника (коцкара), уложену количину криптовалуте, статус опкладе (Winner, Loser или Pending)



Слика 5: Страна са детаљима лутрије

Поред горе поменутих података кориснику је омогућено да провјери вриједност властитог индекса уколико је учествовао у лутрији тако што ће унијети своју адресу и идентификатор лутрије за коју жели провјеру. Након тога приказује се страница са исписаним индексом или са грешком уколико нисте учесник те лутрије.



Слика 6: Страна са приказом индекса учесника

4. Опис паметног уговора

Адреса уговора: 0xb6eF88560d255bA8766462F161f415160613FC02

У наставку ће бити приказан код паметног уговора у програмском језику Solidity (верзија 0.8.7) уз постепено објашњење дијелова кода. Код паметног уговора налази се у датотеци „smart_contract.sol“.

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.8.0;
contract Gambling {

    // bet status:
    uint constant STATUS_WIN = 1;
    uint constant STATUS_LOSE = 2;
    uint constant STATUS_PENDING = 3;

    //game status:
    uint constant STATUS_STARTED = 1;
    uint constant STATUS_COMPLETE = 2;

    //the 'bet' structure
    struct Bet {
        uint betValue;
        address addr;
        uint status;
    }

    //the 'game' structure
    struct Game {
        uint256 gameValue;
        uint outcome;
        uint status;
        uint numOfBets;
        Bet[] bets;
        uint id;
        address creatorAddress;
    }

    mapping(uint => Game) games;
    uint gameID = 0;
    uint lastID;
    Game game;

    fallback() external {} //fallback function
```


Паметни уговор је представљен класом Gambling. На почетку се иницијализују константе за статус чија ће примјена бити објашњена касније. Затим се креирају 2 структуре: Bet и Game. Прва представља појединачну опкладу у једној лутрији, а друга представља саму лутрију. Свака опклада садржи вриједност улога, адресу учесника и статус те опкладе. Свака лутрија садржи:

- Вриједност тј. укупну количину Ether валуте уложену у лутрију
- Исход лутрије (добитни индекс и адреса добитника)
- Статус лутрије
- Капацитет лутрије (број учесника потребан за генерисање исхода лутрије)
- Низ опклада које учествују у тој лутрији
- Идентификатор лутрије
- Адресу креатора те лутрије

Затим се декларише низ лутрија (објеката Game) у који ће бити смјештене све креиране лутрије. Промјенљива gameId служи да се увећа за 1 сваки пут прије додавања нове лутрије у низ. lastID је промјенљива у којој се налази последњи идентификатор (служи да бисмо у сваком тренутку знали колико има лутрија у низу). Промјенљива game којој ће при креирању лутрије бити додијељене унесене вриједности и такав објекат тима Game биће додат у низ лутрија.

Fallback функција се позива ако се позове функција чији идентификатор не одговара ни једној од дефинисаних функција у класи паметног уговора. Та функција нема име, не прима аргументе нити враћа било коју вриједност.

```
//function to create game. It have to be called one time per constructed contract a
nd right after construction
function createGame(uint _numOfBets) public payable {
    require(msg.value != 0, "You have to invest more than 0 eth!");
    require (_numOfBets >= 2, "Number of participants have to be 2 or more!");
    game.gameValue = msg.value;
    game.outcome = 0;
    game.status = STATUS_STARTED;
    game.numOfBets = _numOfBets;
    delete game.bets;
    game.bets.push(Bet(msg.value, msg.sender, STATUS_PENDING));
    gameId += 1;
    game.id = gameId;
    lastID = gameId;
    game.creatorAddress = msg.sender;
    games[gameID] = game;
}
```

Функција `createGame` служи за креирање лутрије. Функција прима један аргумент који представља број учесника и захтева се да тај број буде већи или једнак 2. Вриједност `msg.value` представља количину у криптовалуту са којом се врши трансакција, а у овом случају представља улог опкладе креатора лутрије и захтева се да тај улог буде различит од 0 тј већи од 0. Вриједност лутрије се подешава баш на ту вриједност улога, исход на 0 (подразумјевана вриједност – поново се подешава при генерисању исхода). Статус лутрије подешава се на `STATUS_STARTED`. Капацитет лутрије је број прослијеђен као аргумент. У низ опклада улази опклада креатора при чему је вриједност опкладе `msg.value`, адреса креатора је `msg.sender` (чита адресу онога ко позива функцију), а статус се подешава на `STATUS_PENDING`.

```
// function to participate game
function takeBet(uint _gameID) public payable {
    //requires the taker to make the same bet amount
    require(games[_gameID].status == STATUS_STARTED, "Game didn't start or complete
d!");
    require(msg.value >= games[_gameID].gameValue/games[_gameID].bets.length, "You
investment have to be greater then or equal to average of former investments!");
    games[_gameID].gameValue += msg.value;
    games[_gameID].bets.push(Bet(msg.value, msg.sender, STATUS_PENDING));
    if(games[_gameID].bets.length == games[_gameID].numOfBets){
        generateGameOutcome(_gameID);
        payout(_gameID);
    }
}
```

Функција `takeBet` служи за придруживање лутрији тј креирање опкладе. Прима један аргумент који представља идентификатор лутрије којој се придружује учесник. Да би се креирала опклада захтева се да лутрија још није завршена, и да је вриједност улога већа од просјечне до тада уложене вриједности у лутрију. Вриједност лутрије се увећава за вриједност улога, а низ опклада се проширује опкладом учесника који позива функцију тј. који се придружује лутрији. Ако је након придруживања корисника лутрији капацитет лутрије испуњен тј. ако је учесник последњи предвиђени учесник тада се позивају функције за генерисање исхода лутрије и врши се исплата добитнику.

```
//function to transfer game value to winner
function payout(uint _gameID) private {
    checkPermissions(msg.sender, _gameID);
    require(games[_gameID].status == STATUS_COMPLETE, "Game didn't complete yet");
    for (uint b = 0; b < games[_gameID].bets.length; b++) {
        if (games[_gameID].bets[b].status == STATUS_WIN) {
```

```

        payable(games[_gameID].bets[b].addr).transfer(games[_gameID].gameValue)
    ;
    }
}

```

Функција `payout` служи да се исплати укупна вриједност лутрије извученом добитнику. Прима један аргумент који представља идентификатор лутрије за коју се врши исплата. На почетку се врши провјера дозвола тј. провјера да ли се функција позива од стране учесника лутрије (предвиђено је да функцију технички позове последњи учесник у лутрији). Захтијева се да статус лутрије буде `STATUS_COMPLETE`. Затим се у низу опклада тражи она чији је статус `STATUS_WIN` и на адресу која се налази у тој опклади шаље се укупна вриједност уложена у лутрију.

```

// function to randomly generate game outcome
function generateGameOutcome(uint _gameID) private {
    checkPermissions(msg.sender, _gameID);
    games[_gameID].status = STATUS_COMPLETE;
    Bet[] memory tempArrBets = games[_gameID].bets;
    // generate random number: (array of bets, current block miner's address and cu
    rrent block timestamp as input for hash function)
    games[_gameID].outcome = uint(keccak256(abi.encode(tempArrBets, block.coinbase,
    block.timestamp)))%games[_gameID].numOfBets;
    // winner is address in bet with index equal to outcome
    for(uint j = 0; j < games[_gameID].bets.length; j++){
        if (j==games[_gameID].outcome){
            games[_gameID].bets[j].status = STATUS_WIN;
        }
        else {
            games[_gameID].bets[j].status = STATUS_LOSE;
        }
    }
}
}

```

Функција `generateGameOutcome` служи за генерисање исхода лутрије. Као аргумент прима идентификатор лутрије чији се исход генерише. На почетку се провјеравају дозволе као у претходној функцији. Затим се статус лутрије поставља на `STATUS_COMPLETE`. Исход лутрије се генерише тако што се хеш функцији `keccak256` као улаз прослиједи читав копиран низ опклада, адреса рудара у чијем блоку ће да се налази трансакција (има више смисла на стварној Ethereum мрежи него на локалној) и `timestamp`

блока. Излаз хеш функције који смо доволно насумичним конвертује се у (велики) цијели број и исход лутрије постаје његов остатак при дијељењу са бројем учесника. Тако ће бити одабран само један индекс из низа опклада. Након тога пролазимо кроз низ опклада и провјеравамо чији је индекс једнак исходу опкладе и статус такве опкладе постаје STATUS_WIN. Статус осталих опклада је STATUS_LOSE.

```
// function to check if request sender is game participant
uint signalExists;
function checkPermissions(address sender, uint _gameID) private {
    //only the originator or taker can call this function
    signalExists = 0;
    for (uint i = 0; i < games[_gameID].bets.length; i++) {
        if (games[_gameID].bets[i].addr == sender){
            signalExists = 1;
        }
    }
    require(signalExists == 1, "Your address is not participant address!");
}
```

Функција checkPermissions служи за провјеравање дозвола. Прима два аргумента и то адресу учесника за кога се провјеравају дозволе и идентификатор лутрије за коју се дозволе провјеравају. Односи се на провјару да ли је адреса позиваоца функције адреса везана за неку од опклада и ако није враћа одговарајући изузетак.

Слиједи функције за добијање података о лутрији и опкладама чиме се завршава код паметног уговора:

```
function getLastID() public view returns(uint){
    return lastID;
}

// function to get total game value invested until the moment of the function call

function getGameValue(uint _gameID) public view returns(uint){
    return games[_gameID].gameValue;
}

// function to get number of bets until the moment of the function call
function getCurrentNumOfBets(uint _gameID) public view returns(uint){
    return games[_gameID].bets.length;
}

// function to get amount that we have to invest to enter game
function getAmountToEnterGame(uint _gameID) public view returns (uint){
```

```
        require(games[_gameID].status == STATUS_STARTED, "Amount requested to participate is unavailable to view, because the game is no longer open.");
        if (games[_gameID].bets.length > 0){
            return games[_gameID].gameValue/games[_gameID].bets.length;
        }
        else{
            return 0;
        }
    }

    // function to get maximum number of bets that can participate game
    function getGameCapacity(uint _gameID) public view returns (uint) {
        return games[_gameID].numOfBets;
    }

    // function to get currently status of game (completed or still open)
    function getGameStatus(uint _gameID) public view returns (string memory){
        if (games[_gameID].status == STATUS_COMPLETE){
            return "Game completed";
        }
        else{
            return "Game is still open";
        }
    }

    // function to get address of game creator
    function getCreatorAddress(uint _gameID) public view returns (address){
        return games[_gameID].creatorAddress;
    }

    // function to get current game bets
    function getCurrentGameBets(uint _gameID) public view returns (Bet[] memory){
        return games[_gameID].bets;
    }

    // function to get index of bet with request sender address
    function getMyIndex(uint _gameID) public view returns (uint myIndex){
        myIndex = games[_gameID].numOfBets; //impossible index as default
        for (uint p = 0; p < games[_gameID].bets.length; p++){
            if (games[_gameID].bets[p].addr == msg.sender){
                myIndex = p;
            }
        }
        if (myIndex == games[_gameID].numOfBets){
            require(1 == 0, "You are not participant!");
        }
    }
}
```

```
    }  
  }  
  
  // function to get winning index and address of bet with that index  
  function getGameOutcome(uint _gameID) public view returns (uint winningIndex, address winnerAddress){  
    //checkPermissions(msg.sender);  
    require(games[_gameID].status == STATUS_COMPLETE, "Game didn't complete yet");  
    for (uint k = 0; k < games[_gameID].bets.length; k++){  
      if (games[_gameID].bets[k].status == STATUS_WIN){  
        winningIndex = k;  
        winnerAddress = games[_gameID].bets[k].addr;  
      }  
    }  
  }  
}  
  
}
```

Опис функција:

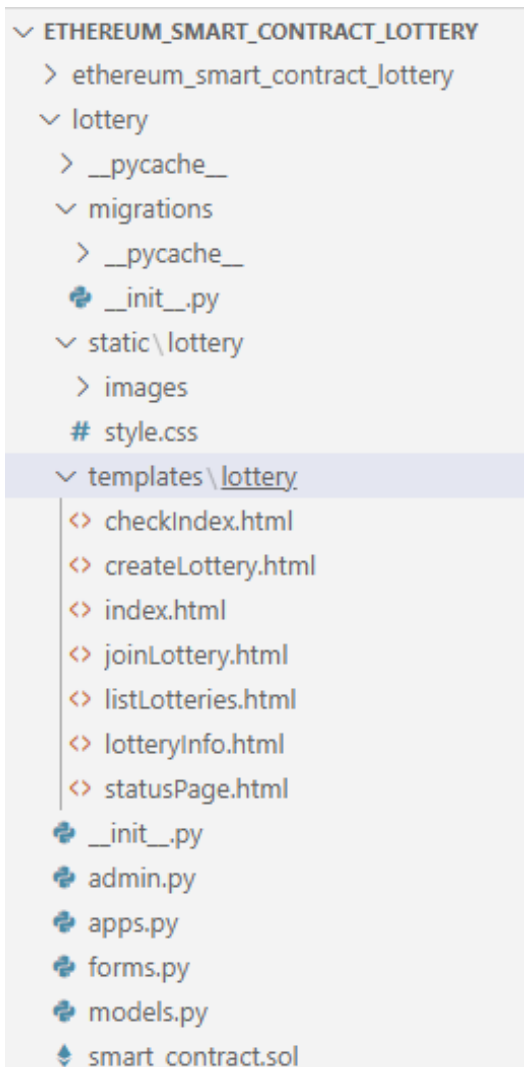
- getLastID – враћа идентификатор последње лутрије (укупан број лутрија)
- getGameValue – враћа тренутну вриједност лутрије чији се идентификатор прослиједи као аргумент
- getCurrentNumOfBets - враћа тренутни број опклада у лутрији чији се идентификатор прослиједи као аргумент
- getAmountToEnterGame – враћа тренутну захтијевану количину Ether валуте за учешће у лутрији чији се идентификатор прослиједи као аргумент
- getGameCapacity – враћа предвиђени број учесника (опклада) у лутрији чији се идентификатор прослиједи као аргумент
- getGameStatus – враћа тренутни статус лутрије чији се идентификатор прослиједи као аргумент
- getCreatorAddress – враћа адресу креиране лутрије чији се идентификатор прослиједи као аргумент
- getCurentGameBets – враћа тренутни низ опклада у лутрији чији се идентификатор прослиједи као аргумент
- getMyIndex – враћа индекс опкладе позиваоца функције ако је он учесник у лутрији чији се идентификатор прослиједи као аргумент, а ако није присилно се подиже изузетак

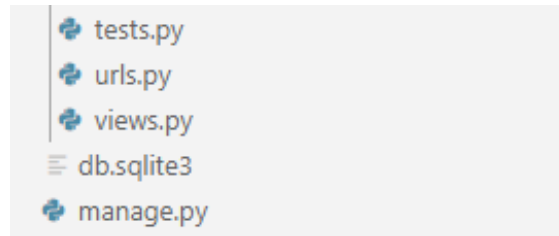
- `getGameOutcome` – враћа добитни индекс и адресу добитника уколико је лутрија чији се идентификатор проследи као аргумент завршена, а ако није враћа одговарајуће упозорење.

5. Структура пројекта и опис дијелова кода

У наставку ће бити појашњена структура пројекта тј. Django апликације као и садржај појединих датотека како би се схватио начин функционисања апликације и комуникација са локалним Ethereum blockchain-ом. Локални Ethereum blockchain обезбијеђен је преко софтвера Ganache који се може преузети са сајта <https://www.trufflesuite.com/ganache>. За повезивање на поменути локални Ethereum blockchain Ganache софтвер користи адресу локалног хоста и порта <http://127.0.0.1/7545>.

Структура пројекта:





Слика 7: Структура пројекта

Напомена: неке датотеке за Django фрејмворк као што су tests.py, manage.py, db.sqlite3, settings.py, apps.py, admin.py неће бити обрађиване у овој документацији већ ће се она базирати искључиво на појединостима везаним за начин функционисања апликације lottery.

Назив Django пројекта у овом случају је ethereum_smart_contract_lottery. У оквиру њега налази се апликација lottery.

Директоријум lottery/migrations је подразумевани директоријум у коме би се налазила иницијална датотека базе података (нпр. 0001_initial.py). Међутим, у овом пројекту не правимо сопствену базу података већ су сви подаци садржани у blockchain-у тј. на имплементираном паметном уговору.

Такође, садржај датотеке lottery/models.py био би представљен имплементацијом базе података и табела од којих би се база састојала (и које би биле имплементиране као класе). Међутим, датотека је празна из истог разлога као и датотека изнад јер не правимо сопствену базу података већ су сви подаци садржани у blockchain-у тј. на имплементираном паметном уговору па не правимо сопствену базу података.

У директоријуму lottery/static/lottery налазе се датотеке потребне за стилизацију графичког приказа имплементиране интернет апликације и садржај ових датотека неће бити детаљно обрађен у документацији како се не би много удаљавали од теме фер лутрије.

Директоријум lottery/templates/lottery садржи html датотеке потребне за функционисање апликације а то су:

Датотека index.html:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>ETH Lottery - Home Page</title>
    {% load static %}
    <link rel="stylesheet" type="text/css" href="{% static 'lottery/style.css' %}">
```



```

</head>
<body>
  <h1 class="pageTitle">Welcome to Fair Ethereum Lottery</h1>
  <div id="homeChoice">
    <a class="indexLink" href="{% url 'lottery:createLottery' %}">Create Lottery Game</a><br>
    <p style="color: white; font-size: 25px">or</p>
    <a class="indexLink" href="{% url 'lottery:joinLottery' %}">Join Lottery Game</a><br><br>
    <a class="homeList" href="{% url 'lottery:listLotteries' %}">Show list of created lotteries</a>
    <br><br><br>
    <p style="color: white; text-align: center">Contract address: <b>0xb6eF88560d255bA8766462F161f415160613FC02</b></p>
  </div>
</body>
</html>

```

Важно: Ради лакшег разумијевања кратања кроз апликацију преко линкова треба напоменути да се сегмент listLotteries из “{% url 'lottery:listLotteries' %}” односи на „name” дио у одређеној путањи која представља једну од путања које се користе у апликацији а које се налазе у датотеци lottery/urls.py у низу који се назива urlpatterns. Иста ствар важи са све линкове идентичног формата у апликацији.

Датотека createLottery.html:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>ETH Lottery - Create</title>
    {% load static %}
    <link rel="stylesheet" type="text/css" href="{% static 'lottery/style.css' %}">
  </head>
  <body>
    <h1 class="pageTitle">Create Your Lottery</h1>
    <form action="{% url 'lottery:create_form_handling' %}" method="post">
      {% csrf_token %}
      <div class="formCreateJoin">
        <table class="tableCreateJoin">
          {{ form }}
        </table><br>

```

```

        <input class="submitForm" type="submit" value="Create">
        <br><br>
        <a class="homeList" href="{% url 'lottery:listLotteries' %}">Show list of creat
ed lotteries</a>
    </div>
</form>
</body>
</html>

```

Датотека joinLottery.html:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>ETH Lottery - Join</title>
    {% load static %}
    <link rel="stylesheet" type="text/css" href="{% static 'lottery/style.css' %}">
  </head>
  <body>
    <h1 class="pageTitle">Join Lottery</h1>
    <form action="{% url 'lottery:join_form_handling' %}" method="post">
      {% csrf_token %}
      <div class="formCreateJoin">
        <table class="tableCreateJoin">
          {{ form }}
        </table><br>
        <input class="submitForm" type="submit" value="Join">
        <br><br>
        <a class="homeList" href="{% url 'lottery:listLotteries' %}">Show list of created
lotteries</a>
      </div>
    </body>
</html>

```

Датотека listLotteries.html:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>ETH Lottery - List</title>
    {% load static %}
    <link rel="stylesheet" type="text/css" href="{% static 'lottery/style.css' %}">

```

```

</head>
<body style="background-image: none; background-color: #004d00; text-align: center;">
  <h1 class="pageTitle">List of Created Lotteries</h1>
  {% if list_of_lotteries %}
    <table class="tableList" border="1" style="border: 1px solid white; margin: auto; width: 80%; color: white; border-collapse: collapse;">
      <tr>
        <th style="font-size: 25px;">Lottery ID</th>
        <th style="font-size: 25px;">Creator address</th>
        <th style="font-size: 25px;">Lottery status</th>
        <th style="font-size: 25px;">Details</th>
      </tr>
      {% for lottery in list_of_lotteries %}
        <tr>
          <td>{{ lottery.lottery_id }}</td>
          <td>{{ lottery.creator_address }}</td>
          <td>{{ lottery.lottery_status }}</td>
          <td><a style="color: white" href="{% url 'lottery:lotteryInfo' lottery.lottery_id %}">more details</a></td>
        </tr>
      {% endfor %}
    </table>
  {% else %}
    <p style="color: white">No lotteries are available.</p>
  {% endif %}
</body>
</html>

```

Оно што се разликује у претходном коду је присуство сегмената у двоструким витичастим заградама а користе се за како би се дохватиле вриједности промјенљивих које се прослијеђују страници преко одређеног линка (за то је задужен код исписан у датотеци `lottery/views.py` и биће детаљније објашњен касније).

Датотека `lotteryInfo.html`:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>ETH Lottery - Details</title>
    {% load static %}
    <link rel="stylesheet" type="text/css" href="{% static 'lottery/style.css' %}">

```

```

</head>
<body style="background-image: none; background-color: #004d00; text-align: center">
  <h1 style="color: white; text-align: center">Lottery Details</h1>

  <form id="getIndexForm" action="{% url 'lottery:checkIndex' %}" method="post">
    {% csrf_token %}
    <br>
    <label id="index_info">If you participated this game you can check your index b
y typing lottery ID and your account address:</label><br><br>
    <label for="lottery_id">Enter lottery ID from below:</label><br>
    <input style="margin: 5px; width: 40%" id="lottery_id" type="number" name="lott
ery_id" min="1" required><br><br>
    <label for="your_address">Your account address:</label><br>
    <input style="margin: 5px; width: 40%" id="your_address" type="text" name="your
_address" maxlength=100 required><br><br>
    <input id="checkButton" type="submit" value="Check">
  </form>

  <br><br>

  <table id="tableInfo" border="1">
    <tr>
      <th>Lottery ID:</th>
      <td>{{ lottery_details.lotteryID }}</td>
    </tr>
    <tr>
      <th>Address of lottery creator:</th>
      <td>{{ lottery_details.creatorAddress }}</td>
    </tr>
    <tr>
      <th>Game status:</th>
      <td>{{ lottery_details.gameStatus }}</td>
    </tr>
    <tr>
      <th>Current amount of ether invested in game:</th>
      <td>{{ lottery_details.currentGameValue }}</td>
    </tr>
    <tr>
      <th>Current minimum amount of ether <br>that you have to invest to particip
ate:</th>
      <td>{{ lottery_details.amountToInvest }}</td>
    </tr>
    <tr>
      <th>Current number of participants in this game:</th>
      <td>{{ lottery_details.currentNumOfBets }}</td>

```

```

    </tr>
    <tr>
      <th>Game capacity (max number of participants):</th>
      <td>{{ lottery_details.gameCapacity }}</td>
    </tr>
    <tr>
      <th>Winning index:</th>
      <td>{{ lottery_details.winnerIndex }}</td>
    </tr>
    <tr>
      <th>Winner's address:</th>
      <td>{{ lottery_details.winnerAddress }}</td>
    </tr>
  </table>

  <br>
  <p style="color: white; text-align: center; font-size: 25px">Current list of bets in lottery with ID = {{ lottery_details.lotteryID }}:</p>

  <table class="betList" border="1" style="border: 1px solid white; margin: auto; width: 75%; color: white; border-collapse: collapse; text-align: center">
    <tr>
      <th style="font-size: 20px; padding: 1">Bet index</th>
      <th style="font-size: 20px; padding: 1">Gambler address</th>
      <th style="font-size: 20px; padding: 1">Invested amount (ether)</th>
      <th style="font-size: 20px; padding: 1">Bet status</th>
    </tr>
    {% for bet in lottery_details.currentBets %}
      <tr>
        <td>{{ bet.bet_index }}</td>
        <td>{{ bet.gambler_address }}</td>
        <td>{{ bet.invested_amount }}</td>
        <td>{{ bet.bet_status }}</td>
      </tr>
    {% endfor %}
  </table>
  <br>
  <button onClick="window.location.reload();">Refresh List</button>
  <br><br>
</body>
</html>

```

Датотека checkIndex.html:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>ETH Lottery - Check Index</title>
    {% load static %}
    <link rel="stylesheet" type="text/css" href="{% static 'lottery/style.css' %}">
  </head>
  <body id="bodyIndexCheck">
    <h1 class="pageTitle">Index Checking</h1><br>
    <h2>Your account address: {{ data_and_index.account_addr }}</h2>
    <h2>Lottery ID: {{ data_and_index.lottery_id }}</h2>
    <h2>Your index: {{ data_and_index.index }}</h2>
  </body>
</html>
```

Датотека statusPage.html:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>ETH Lottery - Status</title>
    {% load static %}
    <link rel="stylesheet" type="text/css" href="{% static 'lottery/style.css' %}">
    <script>
      function goBack() {
        window.history.back();
      }
    </script>
  </head>
  <body id="bodyStatus" style="text-align: center; color: white">
    <br><br><br><br><br><br>
    {% if status_id == 0 %}<h1>Status:<br>Successfully created lottery</h1>{% endif %}
    {% if status_id == 5 %}<h1>Status:<br>Successfully joined lottery</h1>{% endif %}
    {% if status_id == 1 %}<h1>Status: Error 1</h1><p>Invalid account address!</p>{% en
dif %}
    {% if status_id == 2 %}<h1>Status: Error 2</h1><br><p>Invalid number of participant
s!<br>(Please enter integer between 2 and 1000)</p>{% endif %}
    {% if status_id == 3 %}<h1>Status: Error 3</h1><br><p>Invalid value to invest<br>(P
lease check minimum value that you have to invest)</p>{% endif %}
    {% if status_id == 4 %}<h1>Status: Error 4</h1><br><p>Invalid private key!</p>{% en
dif %}
```

```

    {% if status_id == 7 %}<h1>Status: Error 7</h1><br><p>Invalid lottery ID!</p>{% end
if %}
    {% if status_id == 10 %}<h1>Status: Error 10</h1><br><p>Game didn't start or alread
y completed!</p>{% endif %}
    {% if status_id == 11 %}<h1>Status: Error 11</h1><br><p>Failed to send your transac
tion!</p>{% endif %}
    {% if status_id == 12 %}<h1>Status: Error 12</h1><br><p>Your account address is inv
alid!</p>{% endif %}
    {% if status_id == 13 %}<h1>Status: Error 13</h1><br><p>Invalid contract address!</
p>{% endif %}
    {% if status_id == 14 %}<h1>Status: Error 14</h1><br><p>Contract not found!</p>{% e
ndif %}
    <br>
    <button id="backButton" onclick="goBack()">Go Back</button>

</body>
</html>

```

Датотека statusPage.html служи да се обезбједи страница која ће корисника да обавјештава о статусу акције коју жели да изврши. Примарна намјена јој је та да обавијести о каквој врсти грешке се ради уколико се она појави али корисник се обавјештава и ако је акција успјешно обављена. Статус се одређује преко промјенљиве status_id чија вриједност се прослијеђује страници.

Датотека lottery/forms.py садржи Python код који имплементира форме за креирање лутрије и придруживање лутрији кроз двије класе: FormCreate и FormJoin и обје наслијеђују класу Form из пакета forms из Django библиотеке.

```

from django import forms
from decimal import Decimal
#from django.template.defaultfilters import linebreaks

class FormCreate(forms.Form):
    account_address = forms.CharField(label='Your account address:', max_length=100)
    num_of_participants = forms.IntegerField(label='Number of possible participants:',
min_value=2, max_value=1000)
    amount_to_invest = forms.DecimalField(label='Amount to invest (ether cryptocurrency
):', widget=forms.NumberInput(attrs={'step': 0.0000001}), min_value=Decimal(0.0000001))
    private_key = forms.CharField(label='Your private key:', widget=forms.PasswordInput
, max_length=100)

class FormJoin(forms.Form):
    account_address = forms.CharField(label='Your account address:', max_length=100)

```

```
lottery_id = forms.CharField(label='Lottery ID:', max_length=100)
amount_to_invest = forms.DecimalField(label='Amount to invest (ether cryptocurrency):', widget=forms.NumberInput(attrs={'step': 0.0000001}), min_value=Decimal(0.0000001))
private_key = forms.CharField(label='Your private key:', widget=forms.PasswordInput, max_length=100)
```

Датотека `lottery/urls.py` садржи путање које су неопходне за кретање кроз апликацију и међусобну комуникацију тј. размјену података између страница којима корисник има приступ. Програмер је задужен да све путање упише у `urlpatterns` низ. Садржај датотеке је следећи:

```
from django.urls import path

from . import views

app_name = 'lottery'

urlpatterns = [
    path('', views.index, name='index'),
    path('createLottery/', views.createLottery, name='createLottery'),
    path('joinLottery/', views.joinLottery, name='joinLottery'),
    path('listLotteries/', views.listLotteries, name='listLotteries'),
    path('<int:lottery_id>/lotteryInfo/', views.lotteryInfo, name='lotteryInfo'),
    path('creating/', views.create_form_handling, name='create_form_handling'),
    path('joining/', views.join_form_handling, name='join_form_handling'),
    path('<int:status_id>/statusPage/', views.statusPage, name='statusPage'),
    path('checkIndex/', views.checkIndex, name='checkIndex'),
]
```

Први аргумент у `path` објекту је испис који ће кориснику бити приказан као линк и он не мора да одговара стварном називу `html` датотека, други аргумент је позив одговарајуће функције из `lottery/views.py` датотеке, а трећи је име путање које ће да служи као њена идентификација.

Датотека `lottery/views.py` је датотека у којој се налази имплементација функционалности апликације као и повезивања на локалну Ethereum blockchain мрежу и комуникацију с њом. Садржај ове датотеке ће због обимности бити појашњаван постепено уз коментар после дијелова кода.

```
from django.http import HttpResponseRedirect
from django.shortcuts import get_object_or_404, render
from django.urls import reverse
from django.views import generic
from django.template import loader

import json
from web3 import Web3
from solcx import compile_source
from decimal import Decimal
from django.utils import timezone
import sys
import ast

from .forms import FormCreate, FormJoin
```

На почетку се увозе потребне функционалности из одређених django библиотека, затим функционалности и класе потребни претежно за комуникацију са blockchain-ом и на крају се увозе класе базе података и форме за креирање и придруживање лутрији.

```
def compile_source_file(file_path):
    with open(file_path, 'r') as f:
        source = f.read()

    return compile_source(source)

def index(request):
    return render(request, 'lottery/index.html')

def createLottery(request):
    form = FormCreate()
    return render(request, 'lottery/createLottery.html', {'form': form})

def joinLottery(request):
    form = FormJoin()
```

```

    return render(request, 'lottery/joinLottery.html', {'form': form})

class LotteryGame():
    lottery_id = ''
    creator_address = ''
    lottery_status = ''

def listLotteries(request):
    list_of_lotteries = []

    ganache_url = "HTTP://127.0.0.1:7545"
    web3 = Web3(Web3.HTTPProvider(ganache_url))

    compiled_contract = compile_source_file('/home/djordje/ethereum_smart_contract_lottery2/lottery/smart_contract.sol')
    contract_id, contract_interface = compiled_contract.popitem()
    gambling_contract = web3.eth.contract(address = '0xb6eF88560d255bA8766462F161f415160613FC02', abi = contract_interface['abi'])

    lastID = gambling_contract.functions.getLastID().call()
    for i in range(1, lastID+1):
        lottery_game = LotteryGame()
        lottery_game.lottery_id = str(i)
        lottery_game.creator_address = str(gambling_contract.functions.getCreatorAddress(i).call())
        lottery_game.lottery_status = str(gambling_contract.functions.getGameStatus(i).call())
        list_of_lotteries.append(lottery_game)

    return render(request, 'lottery/listLotteries.html', {'list_of_lotteries': list_of_lotteries})

def statusPage(request, status_id):
    return render(request, 'lottery/statusPage.html', {'status_id': status_id})

```

Претходно су дефинисане следеће функције:

- `compile_source_file` – служи да се изврши код (у овом случају паметни уговор) тако што јој се као аргумент прослијеђује путања до датотеке која садржи код.
- `index` – функција која обезбјеђује линк за учитавање почетне стране
- `createLottery` – функција која обезбјеђује страницу за креирање лутрије тако што се празна форма прослјеђује поменутој страници

- joinLottery - функција која обезбјеђује страницу за придруживање лутрији тако што се празна форма прослјеђује поменутој страници
- listLotteries – у њој се дешава комуникација са blockchain-ом и читају се креиране лутрије поредане према идентификатору и додијељују се као вриједност промјенљиве list_of_lotteries која се прослјеђује страници за испис креираних лутрија
- statusPage - функција која обезбјеђује страницу за испис статуса акције коју корисник врши тако што се идентификатор статуса прослјеђује поменутој страници

```
def create_form_handling(request):
    # if this is a POST request we need to process the form data
    if request.method == 'POST':
        # create a form instance and populate it with data from the request:
        form = FormCreate(request.POST)
        status_id = 0
        # check whether it's valid:
        if form.is_valid():
            # process the data in form.cleaned_data as required
            ganache_url = "HTTP://127.0.0.1:7545"
            web3 = Web3(Web3.HTTPProvider(ganache_url))

            accountAddr = form.cleaned_data['account_address']
            if accountAddr[:2] != '0x':
                accountAddr = '0x'+accountAddr
            try:
                accountAddress = web3.toChecksumAddress(accountAddr)
                web3.eth.defaultAccount = accountAddress
            except:
                status_id = 1
            return render(request, 'lottery/statusPage.html', {'status_id': status_
id})

        compiled_contract = compile_source_file('/home/djordje/ethereum_smart_contr
act_lottery2/lottery/smart_contract.sol')
        contract_id, contract_interface = compiled_contract.popitem()

        gambling_contract = web3.eth.contract(address = '0xb6eF88560d255bA8766462F1
61f415160613FC02', abi = contract_interface['abi'])

        numOfParticipants = form.cleaned_data['num_of_participants']
        try:
            numOfParticipants = int(numOfParticipants)
```

```

        if (numOfParticipants < 2 and numOfParticipants > 1000):
            status_id = 2
            return render(request, 'lottery/statusPage.html', {'status_id': sta
tus_id})
        except:
            status_id = 2
            return render(request, 'lottery/statusPage.html', {'status_id': status_
id})

        valueToInvest = form.cleaned_data['amount_to_invest']
        try:
            valueToInvest = float(valueToInvest)
        except:
            status_id = 3
            return render(request, 'lottery/statusPage.html', {'status_id': status_
id})

        nonce = web3.eth.getTransactionCount(web3.eth.defaultAccount)

        tx_hash2 = gambling_contract.functions.createGame(numOfParticipants).buildT
ransaction({'nonce':nonce, 'value':web3.toWei(Decimal(valueToInvest), 'ether')})

        private_key = form.cleaned_data['private_key']
        if private_key[:2] != '0x':
            private_key = '0x'+private_key
        try:
            signed_txn = web3.eth.account.sign_transaction(tx_hash2, private_key=pr
ivate_key)
        except:
            status_id = 4
            return render(request, 'lottery/statusPage.html', {'status_id': status_
id})

        try:
            tx_hash3 = web3.eth.send_raw_transaction(signed_txn.rawTransaction)
        except:
            status_id = 11
            return render(request, 'lottery/statusPage.html', {'status_id': status_
id})

        return render(request, 'lottery/statusPage.html', {'status_id': status_id})
    # if a GET (or any other method) we'll create a blank form
    else:
        form = FormCreate()

```

```
return render(request, 'lottery/createLottery.html', {'form': form})
```

Функција `create_form_handling` преузима податке из форме за креирање лутрије и извршава креирање лутрије са одређеним идентификатором тако што се повезује на локални сервер и комуницира са локалним blockchain-ом. Позива се функција за креирање лутрије гдје се подешавају почетне вриједности лутрије као и опклада креатора лутрије. Прије слања такве трансакције она се потписује са приватним кључем креатора. Све вријеме при проласку кроз код функције врше се провјере да ли се успјешно одвијају све акције и у случају неке грешке врши се преусмјерење на страницу која ће исписати обавјештење о грешци са одређеним идентификатором статуса. Ако је све прошло како треба функција враћа преусмјерење на исту страницу само са идентификатором статуса (`status_id`) успјешно креиране лутрије.

```
def join_form_handling(request):
    # if this is a POST request we need to process the form data
    if request.method == 'POST':
        # create a form instance and populate it with data from the request:
        form = FormJoin(request.POST)
        status_id = 5
        # check whether it's valid:
        if form.is_valid():
            # process the data in form.cleaned_data as required
            ganache_url = "HTTP://127.0.0.1:7545"
            web3 = Web3(Web3.HTTPProvider(ganache_url))

            accountAddr = form.cleaned_data['account_address']
            if accountAddr[:2] != '0x':
                accountAddr = '0x'+accountAddr
            try:
                accountAddress = web3.toChecksumAddress(accountAddr)
                web3.eth.defaultAccount = accountAddress
            except:
                status_id = 1
            return render(request, 'lottery/statusPage.html', {'status_id': status_
id})

        compiled_contract = compile_source_file('/home/djordje/ethereum_smart_contr
act_lottery2/lottery/smart_contract.sol')
        contract_id, contract_interface = compiled_contract.popitem()

        lotteryID = form.cleaned_data['lottery_id']
        try:
```

```

        lotteryID = int(lotteryID)
    except:
        status_id = 7
        return render(request, 'lottery/statusPage.html', {'status_id': status_
id})

    gambling_contract = web3.eth.contract(address = '0xb6eF88560d255bA8766462F1
61f415160613FC02', abi = contract_interface['abi'])

    valueToInvest = form.cleaned_data['amount_to_invest']
    try:
        valueToInvest = float(valueToInvest)
    except:
        status_id = 3
        return render(request, 'lottery/statusPage.html', {'status_id': status_
id})

    nonce = web3.eth.getTransactionCount(web3.eth.defaultAccount)

    try:
        tx_hash2 = gambling_contract.functions.takeBet(lotteryID).buildTransact
ion({'nonce':nonce, 'value':web3.toWei(Decimal(valueToInvest), 'ether')})
    except:
        status_id = 10
        return render(request, 'lottery/statusPage.html', {'status_id': status_
id})

    private_key = form.cleaned_data['private_key']
    if private_key[:2] != '0x':
        private_key = '0x'+private_key
    try:
        signed_txn = web3.eth.account.sign_transaction(tx_hash2, private_key=pr
ivate_key)
    except:
        status_id = 4
        return render(request, 'lottery/statusPage.html', {'status_id': status_
id})

    try:
        tx_hash3 = web3.eth.send_raw_transaction(signed_txn.rawTransaction)
    except:
        status_id = 11
        return render(request, 'lottery/statusPage.html', {'status_id': status_
id})

```

```
        return render(request, 'lottery/statusPage.html', {'status_id': status_id})
    else:
        form = FormJoin()

    return render(request, 'lottery/joinLottery.html', {'form': form})
```

Функција `join_form_handling` преузима податке из форме за придруживање лутрији и извршава придруживање лутрији са одређеним идентификатором тако што се повезује на локални сервер и комуницира са локалним blockchain-ом. Дохвата се вриједност паметног уговора са адресе уговора и смјешта се у промјенљиву `gambling_contract`. Затим се врши генерисање трансакције. На основу те трансакције корисник ће направити властиту опкладу у лутрији којој се придружује и то према одређеним правилима. Прије слања такве трансакције она се потписује са приватним кључем учесника. Све вријеме при проласку кроз код функције врше се провјере да ли се успјешно одвијају све акције и у случају неке грешке врши се преусмјерење на страницу која ће исписати обавјештење о грешци са одређеним идентификатором статуса. Ако је све прошло како треба функција враћа преусмјерење на исту страницу само са идентификатором статуса успјешног придруживања лутрији.

```
class LotteryDetails():
    lotteryID = ''
    creatorAddress = ''
    gameStatus = ''
    currentGameValue = ''
    amountToInvest = ''
    currentNumOfBets = ''
    gameCapacity = ''
    winnerIndex = ''
    winnerAddress = ''
    currentBets = []

class Bet():
    bet_index = ''
    invested_amount = ''
    gambler_address = ''
    bet_status = ''

def lotteryInfo(request, lottery_id):

    contractAddress = '0xb6eF88560d255bA8766462F161f415160613FC02'
```

```
ganache_url = "HTTP://127.0.0.1:7545"
web3 = Web3(Web3.HTTPProvider(ganache_url))

compiled_contract = compile_source_file('/home/djordje/ethereum_smart_contract_lottery2/lottery/smart_contract.sol')
contract_id, contract_interface = compiled_contract.popitem()

gambling_contract = web3.eth.contract(address = contractAddress, abi = contract_interface['abi'])

lottery_id = int(lottery_id)
lottery_details = LotteryDetails()
lottery_details.lotteryID = str(lottery_id)
lottery_details.creatorAddress = str(gambling_contract.functions.getCreatorAddress(lottery_id).call())
lottery_details.gameStatus = str(gambling_contract.functions.getGameStatus(lottery_id).call())
lottery_details.currentGameValue = str(web3.fromWei(int(gambling_contract.functions.getGameValue(lottery_id).call()), 'ether'))
    try:
        lottery_details.amountToInvest = str(web3.fromWei(int(gambling_contract.functions.getAmountToEnterGame(lottery_id).call()), 'ether'))
    except:
        lottery_details.amountToInvest = "Amount requested to participate no longer available."
    lottery_details.currentNumOfBets = str(gambling_contract.functions.getCurrentNumOfBets(lottery_id).call())
    lottery_details.gameCapacity = str(gambling_contract.functions.getGameCapacity(lottery_id).call())
    try:
        gameOutcomeArr = ast.literal_eval(str(gambling_contract.functions.getGameOutcome(lottery_id).call()))
        lottery_details.winnerIndex = str(gameOutcomeArr[0])
        lottery_details.winnerAddress = str(gameOutcomeArr[1])
    except:
        lottery_details.winnerIndex = "You cannot view winner's index because game didn't complete yet!"
        lottery_details.winnerAddress = "You cannot view winner's address because game didn't complete yet!"

currentGameBets = gambling_contract.functions.getCurrentGameBets(lottery_id).call()

lottery_details.currentBets = []
cntIndex = 0
for b in currentGameBets:
```



```

bet = Bet()
bet.bet_index = str(cntIndex)
cntIndex += 1
bet.invested_amount = str(web3.fromWei(int(b[0]), 'ether'))
bet.gambler_address = str(b[1])
if str(b[2]) == "1":
    strStatus = 'Winner'
if str(b[2]) == "2":
    strStatus = 'Loser'
if str(b[2]) == "3":
    strStatus = 'Pending'
bet.bet_status = strStatus
lottery_details.currentBets.append(bet)

return render(request, 'lottery/lotteryInfo.html', {'lottery_details': lottery_details})

```

Класа LotteryDetails садржи податке о лутрији а класа Bet податке о опклади. Функција lotteryInfo поред уобичајеног request параметра има и lottery_id параметар преко кога из паметног уговора преузима вриједност са идентичним инентификатором као што је вриједност тог параметра. Затим се одвија повезивање на локални сервер како би се остварила комуникација са blockchain-ом. Дохвата се вриједност кода паметног уговора са адресе уговора и смјешта у промјенљиву gambling_contract. Затим се редом позивају функције паметног уговора од којих је свака претходно објашњена и инстанца класе LotteryDetails се прослијеђује страници за испис детаља лутрије.

```

class DataAndIndex():
    lottery_id = ''
    account_addr = ''
    index = ''

def checkIndex(request):
    if request.method == 'POST':
        your_address = request.POST['your_address']
        lottery_id = request.POST['lottery_id']

        ganache_url = "HTTP://127.0.0.1:7545"
        web3 = Web3(Web3.HTTPProvider(ganache_url))

        if your_address[:2] != '0x':
            your_address = '0x'+your_address
    try:

```

```

        your_address = web3.toChecksumAddress(your_address)
        web3.eth.defaultAccount = your_address
    except:
        status_id = 12
        return render(request, 'lottery/statusPage.html', {'status_id': status_id})

    try:
        contract_address = web3.toChecksumAddress('0xb6eF88560d255bA8766462F161f415
160613FC02')
    except:
        status_id = 13
        return render(request, 'lottery/statusPage.html', {'status_id': status_id})

    compiled_contract = compile_source_file('/home/djordje/ethereum_smart_contract_
lottery2/lottery/smart_contract.sol')
    contract_id, contract_interface = compiled_contract.popitem()

    try:
        gambling_contract = web3.eth.contract(address = contract_address, abi = con
tract_interface['abi'])
    except:
        status_id = 14
        return render(request, 'lottery/statusPage.html', {'status_id': status_id})

    lastID = int(gambling_contract.functions.getLastID().call())
    if not ((int(lottery_id) >= 1) and (int(lottery_id) <= lastID)):
        status_id = 7
        return render(request, 'lottery/statusPage.html', {'status_id': status_id})

    try:
        your_index = str(gambling_contract.functions.getMyIndex(int(lottery_id)).ca
ll())
    except:
        your_index = "You cannot have index as long as you are not a participant!"

    data_and_index = DataAndIndex()
    data_and_index.lottery_id = lottery_id
    data_and_index.account_addr = your_address
    data_and_index.index = your_index
    return render(request, 'lottery/checkIndex.html', {'data_and_index': data_and_i
ndex})

```

Функција `checkIndex` на основу унијете адресе корисника и идентификатора уговора врши комуникацију са локалним blockchain-ом при чему се провјерава индекс корисника

у датој лутрији или се избацује обавјештење ако корисник није учесник лутрије. Класа `DataAndIndex` служи да се њена инстанца са подешеним подацима прослиједи страници за испис индекса.

6. Закључак

Након објашњења начина функционисања апликација у пројекту и описа саме идеје развоја коцкарске индустрије на blockchain технологији уз помоћ паметних уговора неизбјежно је закључити да овај вид технологије у комбинацији са децентрализованом p2p мрежом и неким сјајним криптографским рјешењима као што је то, поред осталих, у овом случају кесак256 хеш функција даје заиста широке могућности за примјену.

Треба напоменути да се у овом пројекту није вршило повезивање на стварни Ethereum blockchain већ на локални који је предвиђен за развојне програмере ради поједностављеног тестирања кода. Ако бисмо хтјели да се повежемо на стварни Ethereum blockchain требали бисмо да испунимо следеће захтјеве:

- Проналазак адресе преко које бисмо се повезали на Ethereum blockchain (То се може обезбиједити регистрањем и покретањем пројекта на сајту <https://infura.io/>). Адресе које би нам биле доступне за повезивање преко нашег регистрованог пројекта у коду бисмо уписали умјесто адресе локалног сервера <http://127.0.0.1/7545>.
- Посједовање адреса са стварном количином Ether валуте (највећи разлог зашто се тестирање одвија на локалном blockchain-у преко Ganache софтвера, поготово кад се у обзир узму „провизије“ (fees) које се плаћају за све трансакције и ризик да смо при имплементацији учинили грешку па та средства можемо заувјек да изгубимо).

Разлог зашто би такав вид технологије требао ући у употребу јесте тај што обичан корисник може имати пуно повјерење у такву технологију јер прије свега ради на принципу децентрализације и имплементације паметних уговора који нам омогућавају да увијек покренемо без сумње идентичан код. Још један битан разлог за повјерење јесте тај да сваки учесник дијелом утиче на генерисање резултата једног извлачења, тако да нпр. без учешћа једног од корисника, резултат једног извлачења би био сасвим другачији.

7. Литература

1. <https://www.coursera.org/learn/cryptocurrency>
2. <https://ethereum.org/en/developers/docs/programming-languages/python/>
3. <https://www.youtube.com/watch?v=SAi5rYFh7yw>
4. <https://web3py.readthedocs.io/en/stable/>
5. <https://docs.soliditylang.org/en/v0.8.7/>
6. <https://www.coursera.org/learn/django-database-web-apps?specialization=django>
7. <https://www.coursera.org/learn/django-build-web-apps?specialization=django>
8. <https://docs.djangoproject.com/en/3.2/>
9. <https://www.dj4e.com/>
10. Др Владимир М. Миловановић, Компонување рачунарских програма, Факултет инжењерских наука, Крагујевац, 2021
11. Dr. Charles R. Severance, Python for Everybody, University of Michigan, 2016
12. <https://stackoverflow.com/>
13. <https://remix.ethereum.org/>