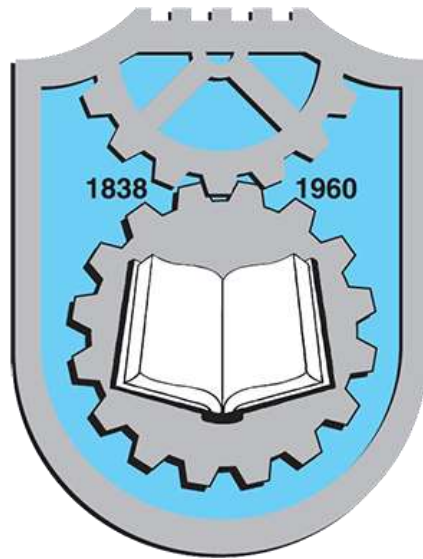


Univerzitet u Kragujevcu
Fakultet inženjerskih nauka



Projektni zadatak iz predmeta „Programiranje sistema u realnom vremenu“

Tema projektnog zadatka

Lutrija implementirana na Ethereum blockchain-u uz pomoć pametnih ugovora (Smart Contracts)

Profesor: Vladimir M. Milovanović

Student: Uroš Stanojkov 601/2018

Sadržaj

1. Uvod i opis zadatka.....	3
2. Opis funkcionisanja zadatka.....	4
3. Prikaz ugovora na Rinkeby Testnet-u.....	5
4. Pametni ugovori.....	11
5. Deploy-ovanje ugovora i pozivanje raznih funkcija korišćenjem Brownie framework-a.....	17
6. Testiranje pomoću Brownie framework-a.....	23
7. Zaključak.....	26
8. Literatura.....	27

Uvod i opis zadatka

Sama blockchain tehnologija se pojavila još 2008. godine napravljena od strane čoveka (tj. od grupice ljudi – nije poznato) koji je sebe nazivao **Satoshi Nakamoto**. Blockchain koristi blokove, kao što nam samo ime i govori, koji su povezani međusobno koristeći razne kriptografske funkcije.

Blockchain predstavlja decentralizovanu peer-to-peer mrežu koja je raširena po celom internetu čiji su članovi Node-ovi (čvorovi). Kreiranjem blockchain-a je omogućeno da **Bitcoin** postane prva digitalna kriptovaluta koja je rešila problem duplog plaćanja. Pojam duplo plaćanje se odnosi na to kada jedan korisnik iskoristi potencijalnu priliku da recimo jednu određenu sumu pošalje na dve različite adrese, što i nije toliko bitno za ovaj projektni zadatak.

Vremenom su se pojavljivale razne kriptovalute, koje nisu predstavljale nikakvu prekretnicu, sve do pojave **Ethereum**-a 2015. godine. Velika novina je što je bilo omogućeno programirati na Ethereum-ovom blockchainu pomoću raznih pametnih ugovora i ostalih tehnologija, što nije bilo moguće na Bitcoinu, on je samo predstavljao kriptovalutu.

Najbitnije za naš zadatak je sam pojam **decentralizacije**. Decentralizacija se odnosi na to da ne može jedan korisnik (admin) da izmeni bilo šta bitno u datom problemu, projektu, zadatku... Da li je na primer banka decentralizovana? Generalno jeste, ali kako da mi u to budemo sigurni da sutradan nećemo, iz nama nepoznatog razloga, videti da je stanje na računu jednako nuli. To je čisto poverenje u sistem.

Još jedan primer je kada uđete na sajt za online kockanje. Kako korisnik može da bude siguran da igre nisu nameštene tako da se pobednik uvek izvlači po istom principu? Nemoguće je to znati. Najveća vrlina Ethereum-ovog blockchain-a je da svi ugovori koji se deploy-uju na blockchain su svakom korisniku vidljivi, oni su javni i svako može da vidi kod i da ga izmeni po svojoj želji.

Konkretan slučaj, pošto se za programiranje na Ethereum-u koriste test kriptovalute (**faucet**-i), koje nemaju pravu vrednost, ja sam testiranje i deploy-ovanje mojih pametnih ugovora vršio na Rinkeby TESTNET-u. Na ovom testnetu je moguće naći moj ugovor pod adresom **0x592043C8faf8c364e353Bb57028111090A4D2260** (više o tome u daljem tekstu).

Poenta je da bilo koji korisnik na svetu, koji pristupi Rinkeby testnetu i pretraži moj ugovor, moći će da ga istraži, isproba, vidi sam source kod i još mnogo stvari.

Link do Rinkeby Testneta je: <https://rinkeby.etherscan.io/>

Opis funkcionisanja zadatka

Bitno je razumeti kako funkcioniše lutrija, šta treba uraditi ako neko želi da isproba aplikaciju?

Važno je napomenuti da je svaki korisnik obeležen svojom jedinstvenom javnom adresom. Postoje određene funkcije koje samo čitaju stanja ili funkcije koje menjaju stanja.

Na primer, svaki korisnik će moći da pročita kolika je potrebna vrednost da bi uplatio učestvovanje u lutriji, vidi adresu prethodnog pobednika lutrije. Dok sa druge strane postoje funkcije koje menjaju stanja, neke od njih su da korisnik pristupi lutriji, da se započne, završi lutrija, itd.

Lutrija ima tri stanja: Zatvorena, Otvorena i Izračunavanje dobitnika.

Pre početka lutrije, dok je njeno stanje još uvek zatvoreno korisnici mogu glasati da li žele duplirati ulog ili ne žele. Ako ima više korisnika koji žele da se duplira to će se i desiti.

Nakon toga ide startovanje lutrije koju može izvršiti samo vlasnik ugovora. Ova funkcionalnost je centralizovana ali ne utiče na biranje dobitnika, što je i najbitnije u ovom zadatku.

Kada se lutrija započne, korisnici mogu platiti učešće i pristupiti lutriji. Ovu funkciju mogu izvršiti svi korisnici. Svaki korisnik koji plati učešće, šalje tu vrednost na adresu samog ugovora.

Na kraju je potrebno završiti lutriju i izvući slučajnog dobitnika. Vlasnik ugovora može da pozove ovu funkciju ali to ne utiče na izvlačenje dobitnika i tada nije moguće da novi igrači pristupe lutriji, jer ona prelazi u stanje Izračunavanje pobednika.

Kada se izvuče nasumični pobednik sva vrednost koja se nalazi na adresi ugovora će biti prebačena na adresu dobitnika lutrije. Tada se igra završava, prelazi se u stanje Zatvoreno i čeka se da vlasnik ponovo pokrene lutriju.

Prikaz ugovora na Rinkeby Testnet-u

Slika 1 će prikazati kako korišćenjem brownie se deploy-uje pametan ugovor na Rinkeby testnet i šta prikazuje konzola.

```
PS C:\Users\Uros\Desktop\eth-projekat-lottery\lottery> brownie run scripts/deploy.py --network rinkeby
INFO: Could not find files for the given pattern(s).
Brownie v1.16.3 - Python development framework for Ethereum

LotteryProject is the active project.

Running 'scripts\deploy.py::main'...
Transaction sent: 0x7eb23f78cf3e05102a7ce13c175fb28a4c7626ca233ebb570b679dea765885f1
Gas price: 1.000000009 gwei Gas limit: 1251547 Nonce: 238
Lottery.constructor confirmed Block: 9354865 Gas used: 1137770 (90.91%)
Lottery deployed at: 0x592043C8faf8c364e353Bb57028111090A4D2260

Waiting for https://api-rinkeby.etherscan.io/api to process contract...
Verification submitted successfully. Waiting for result...
Verification pending...
Verification complete. Result: Pass - Verified
Kontrakt je deploy-ovan
Transaction sent: 0x64547332a5a21db30e15b54a3b4266fd2cfee6e8f0666060ab422e146dd59615
Gas price: 1.000000009 gwei Gas limit: 38772 Nonce: 239
Lottery.startLottery confirmed Block: 9354869 Gas used: 35156 (90.67%)

Lottery.startLottery confirmed Block: 9354869 Gas used: 35156 (90.67%)

Pokrenuta lutrija
Transaction sent: 0x72c8d2b689e6fffed1b41465c4a9f8427e26085b86199e95d3a38dea328ab040
Gas price: 1.000000009 gwei Gas limit: 101010 Nonce: 240
Lottery.enter confirmed Block: 9354870 Gas used: 91828 (90.91%)

Lottery.enter confirmed Block: 9354870 Gas used: 91828 (90.91%)

Uspesno ste se platili ucesce za lutriju
Transaction sent: 0x9858e54a3c0afbc1d0b405586a5c8675dec19ebbf5051ad98d7ca4a93d9d2429
Gas price: 1.000000009 gwei Gas limit: 56992 Nonce: 241
LinkToken.transfer confirmed Block: 9354871 Gas used: 51811 (90.91%)

LinkToken.transfer confirmed Block: 9354871 Gas used: 51811 (90.91%)

Isplacen kontrakt
LinkToken.transfer confirmed Block: 9354871 Gas used: 51811 (90.91%)

Transaction sent: 0x13775053bec9339fc42f002ee2ebec3164ca2812659a2d51693735b72537c4b7
Gas price: 1.000000009 gwei Gas limit: 174062 Nonce: 242
Lottery.endLottery confirmed Block: 9354874 Gas used: 153439 (88.15%)

Lottery.endLottery confirmed Block: 9354874 Gas used: 153439 (88.15%)

Pobednik je izracunat!
PS C:\Users\Uros\Desktop\eth-projekat-lottery\lottery> |
```

Slika 1 – prikaz deploy-ovanja pametnog ugovora na Rinkeby testnet

Pre komande koja će deploy-ovati ugovor na Rinkeby obavezno je kompajlirati ugovor komandom: **brownie compile**

Komandom **brownie run scripts/deploy.py --network rinkeby** ugovor se deploy-uje na Rinkeby. Kada bismo izostavili iz ove komande deo **--network rinkeby**, ugovor bi se deploy-ovao na naš personalni blockchain, koji je obezbedio Ganache.

Kada se pokrene deploy-ovanje vidimo adresu na kojoj je deploy-ovan ugovor, kao i hash svake transakcije ponaosob, kao i to koliko je cena gasa u Gwei-ju, koji je limit gasa, kao i nonce bloka. Nonce bloka predstavlja random broj koji je prikladan rudarima, može se iskoristiti samo jednom i dodat je na heširan blok.

Slika 2 prikazuje kako izgleda prikaz ugovora čiju adresu pretražujemo na Rinkeby Testnetu.

Etherscan
Rinkeby Testnet Network

All Filters Search by Address / Txn Hash / Block / Token / ENS

Home Blockchain Tokens Misc Rinkeby

Contract 0x592043C8fa8c364e353Bb57028111090A4D2260

Contract Overview

Balance: 0 Ether

More Info

My Name Tag: Not Available

Contract Creator: 0x421741c46b6acfd320f at txn 0x7eb23f78cf3e05102a7

Transactions Internal Txns Erc20 Token Txns Contract Events

Latest 4 from a total of 4 transactions

Txn Hash	Method	Block	Age	From	To	Value	Txn Fee
0x13775053bec9339fc4	End Lottery	9354874	4 hrs 31 mins ago	0x421741c46b6acfd320f	0x592043c8fa8c364e35	0 Ether	0.000133438001
0x72c8d2b689e6ffed1b	Enter	9354870	4 hrs 32 mins ago	0x421741c46b6acfd320f	0x592043c8fa8c364e35	0.01711490705626 Ether	0.000001028
0x54547332a5a21db30e	Start Lottery	9354869	4 hrs 33 mins ago	0x421741c46b6acfd320f	0x592043c8fa8c364e35	0 Ether	0.000035156
0x7eb23f78cf3e05102a7	0x60c9040	9354865	4 hrs 34 mins ago	0x421741c46b6acfd320f	Create: Lottery	0 Ether	0.00113777001

[Download CSV Export]

Powered by Ethereum

Preferences

Slika 2 – Prikaz ugovora na Rinkeby testnetu

Na slici 2 možemo primetiti nekoliko značajnih polja. Polje **Balance** označava kolika je vrednost koja se nalazi na adresi ugovora u Ethereum-u. Polje **Contract Creator** nam daje adresu korisnika koji je deploy-ovao ovaj ugovor. Mogu se primetiti razne transakcije koje su bile izvršene, kao i njihov hash, naziv metoda, vreme kada su poslednji put izvršene, kao i vrednost koja se šalje tim transakcijama.

Prikaz detalja transakcija:

The screenshot displays the 'Transaction Details' page for a transaction on the Rinkby Testnet. The interface includes tabs for 'Overview', 'Internal Txns', and 'State', with 'Overview' being the active tab. A warning message at the top states: '[This is a Rinkby Testnet transaction only]'. The transaction details are as follows:

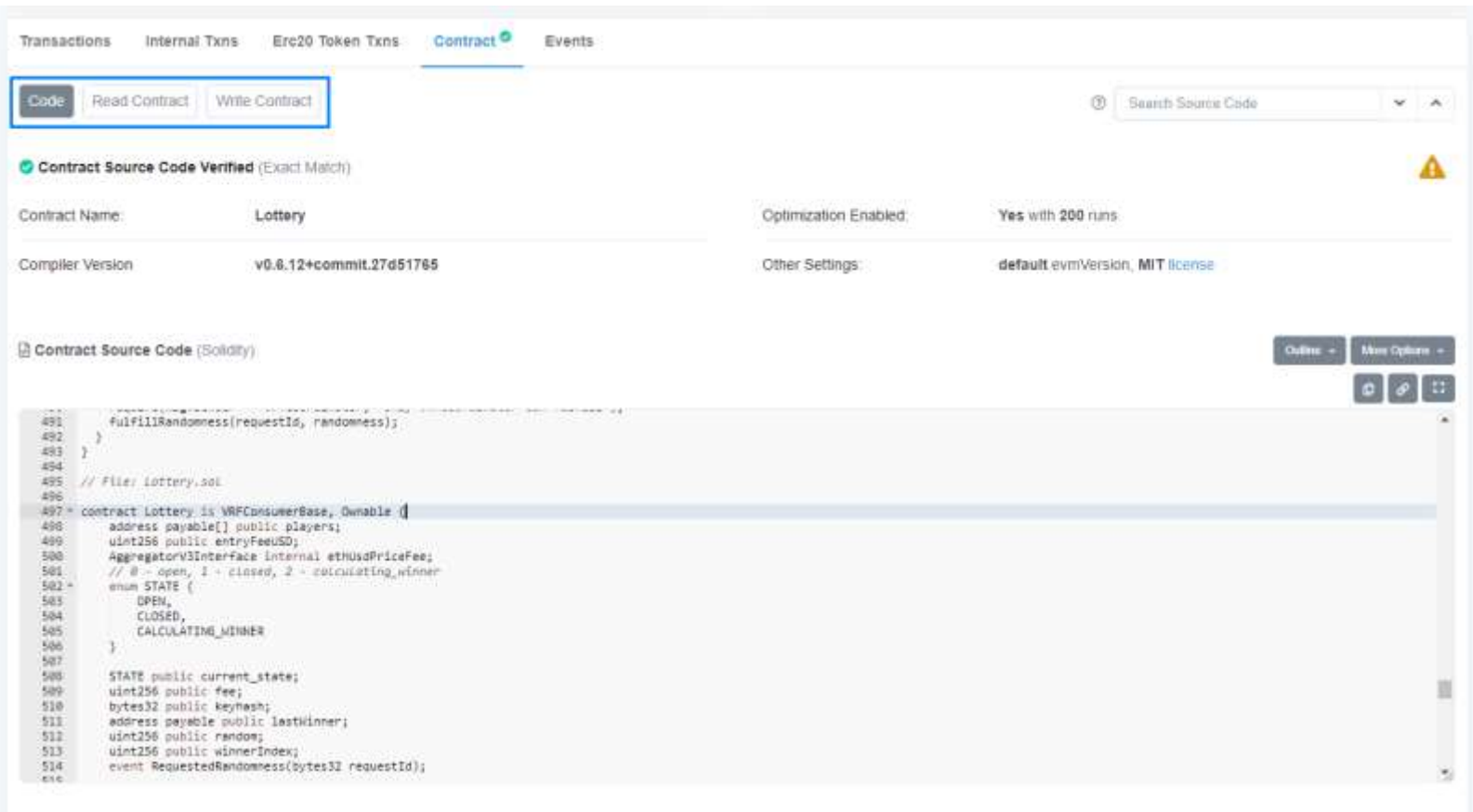
Field	Value
Transaction Hash	0x72c8d2b689e6ffed1b41465c4a9f8427e26085b86199e55d3a38dea328ab040
Status	Success
Block	9354870 (64 Block Confirmations)
Timestamp	16 mins ago (Sep-25-2021 02:23:44 PM +UTC)
From	0x421741c48b6achd3202046bb3d558af51305bbf
To	Contract 0x592043c8a80c354e353bb57028111090a4d2260
Value	0.017114807056280657 Ether (\$0.00)
Transaction Fee	0.000091828000826452 Ether (\$0.00)
Gas Price	0.000000001000000009 Ether (1.000000009 Gwei)
Txn Type	0 (Legacy)
Gas Limit	101,010
Gas Used by Transaction	91,828 (90.91%)
Base Fee Per Gas	9 wei (0.000000009 Gwei)
Burnt Fees	0.000000000000826452 Ether
Nonce	240
Input Data	Function: enter() MethodID: 8xe97dcb62

Slika 3 – detalji transakcije

Prikazivanje detalja transakcije vidimo još neke bitne informacije, neke od njih su ko je izvršio

transakciju i za koga je namenjena, u ovom slučaju namenjena je za adresu ugovora, a pozvao je korisnik sa adresom koja je naznačena u polju **From**. Moguće je primetiti da iako je vrednost 0.017.. Ethereum-a, da piše da je ta vrednost u stvari 0\$, iz razloga što se koriste faucet-i.

Možemo i pogledati sam kod ugovora, njegova polja, funkcije i drugo, prikaz na slici 4.



The screenshot displays the Etherscan interface for a contract named "Lottery". At the top, there are tabs for "Transactions", "Internal Txns", "ERC20 Token Txns", "Contract" (which is selected), and "Events". Below the tabs, there are three buttons: "Code", "Read Contract", and "Write Contract". A search bar for "Search Source Code" is also present. The contract's status is "Contract Source Code Verified (Exact Match)". The contract name is "Lottery", and the compiler version is "v0.6.12+commit.27d51765". The optimization is enabled, set to "Yes with 200 runs". The other settings are "default evmVersion, MIT license". The contract source code is displayed in a text area, showing Solidity code for a lottery. The code includes a contract named "Lottery" that inherits from "VRFCConsumerBase" and "Ownable". It defines a "payable" array for players, a "fee" in uint256, a "keyhash" in bytes32, and a "lastWinner" address. It also defines a "random" uint256 and a "winnerIndex" uint256. The code includes a "RequestedRandomness" event and a "fulfillRandomness" function. The contract is currently in the "OPEN" state.

Slika 4 – prikaz samog ugovora

Vidimo polje **Contract Source Code** gde se nalaze kodovi svih ugovora koji su korišćeni. Od drugih značajnih polja primećujemo **Read Contract** i **Write Contract**. Polje Read Contract služi samo za čitanje raznih vrednosti, dakle tu ne postoje funkcije koje mogu izvršiti promenu stanja, dok važi obrnuto za Write Contract. Tu se nalaze samo funkcije koje menjaju stanja ugovora.

O tim funkcijama će biti malo opširnije u daljem tekstu. Polja Read Contract su prikazana na slici 5, odnosno Write Contract na slici 6.

Code
Read Contract
Write Contract

1. current_state	↓
1 uint8	
2. entryFeeUSD	→
3. fee	→
4. getEntranceFee	↓
17311978982257770 uint256	
5. keyhash	→
6. lastWinner	↓
0x421741c46b6acfd320f2045bb3d558af51305bbf address	
7. owner	↓
0x421741c46b6acfd320f2045bb3d558af51305bbf address	
8. players	→
9. random	↓
66080283960968112775388366412182734352403221523127187456822826861221238061882 uint256	
10. votedNotToRaise	→
11. votedToRaise	→
12. winnerIndex	→

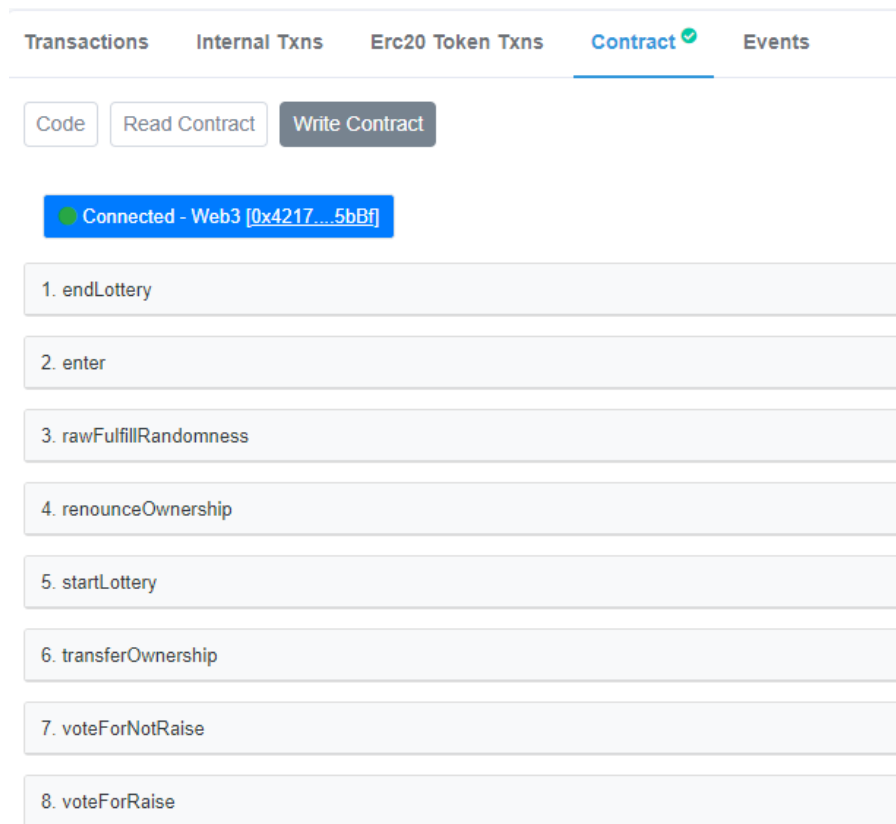
Slika 5 – Read Contract

Polje `current_state` označava trenutno stanje lutrije, iz koda se može razaznati koji index je za koje stanje. 0 – predstavlja Otvoreno stanje, 1 – Zatvoreno, 2 – Izračunavanje pobjednika.

`entryFeeUSD` predstavlja koliki je iznos potreban za ulaz na lutriju izražen u dolarima.

`getEntranceFee` predstavlja isti taj iznos u Ethereum-u. `lastWinner` predstavlja adresu posljednjeg pobjednika lutrije.

`Players` polje predstavlja niz adresa svih trenutnih igrača. `VotedNotToRaise` i `votedToRaise` predstavljaju nizove adresa igrača koji su glasali da ne žele da se podigne ulog, odnosno onih koji to žele.



Slika 6 – Write Contract

Ovde možemo primetiti sve funkcije ugovora koje mogu izmeniti stanja kada se izvrše.

Vidimo da je potrebno biti konektovan na Metamask (digitalni novčanik), iz razloga što je za sve ove transakcije potreban određeni gas, kao i adresa korisnika koji izvršava te funkcije.

Svi korisnici mogu izvršiti sve funkcije osim startLottery i endLottery. Te dve funkcije može izvršiti samo vlasnik ugovora.

Primećujemo da je dozvoljeno i prebacivanje vlasništva ovog ugovora na neku drugu adresu.

Funkcija rawFulfillRandomness se koristi za dobijanje potpuno nasumičnog pobednika, funkcija je napravljena tako da je gotovo nemoguće pretpostaviti pobednika, jer se svaki put drugačije hešira. Takođe je moglo biti izvršeno neko drugo heširanje, recimo keccak256. Problem je što se ovaj algoritam uvek hešira na isti način.

U daljem tekstu biće detaljnije pojašćeno o raznim funkcionalnostima pametnih ugovora.

Pametni ugovori

Solidity je najrasprostraniji programski jezik koji se koristi za pisanje pametnih ugovora. Predstavlja objektno-orijentisani jezik i koristi se na raznim blockchain platformama.

Na slici 7 će biti prikazana polja koja su korišćena u ugovoru, naziv Solidity fajla je **Lottery.sol**.

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.6.6;

import "@chainlink/contracts/src/v0.6/interfaces/AggregatorV3Interface.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@chainlink/contracts/src/v0.6/VRFConsumerBase.sol";

contract Lottery is VRFConsumerBase, Ownable {
    address payable[] public players;
    uint256 public entryFeeUSD;
    AggregatorV3Interface internal ethUsdPriceFee;
    enum STATE {
        OPEN,
        CLOSED,
        CALCULATING_WINNER
    }

    STATE public current_state;
    uint256 public fee;
    bytes32 public keyhash;
    address payable public lastWinner;
    uint256 public random;
    uint256 public winnerIndex;
    event RequestedRandomness(bytes32 requestId);

    address[] public votedToRaise;
    address[] public votedNotToRaise;
```

Slika 7 – prikaz polja u Solidity-ju

Potrebno je koristiti licence za pametne ugovore, postoji više licenca ja sam u ovom projektu koristio MIT licencu. Primećuje se da je korišćena verzija Solidity-ja 0.6.6, oznaka ^ označava da se koriste sve verzije od 0.6 do 0.7.

Importovana su tri ugovora, o svakom će biti napisano ukratko čemu služe.

Ugovor **Ownable.sol** je već ranije napisan ugovor koji je u ovom projektu korišćen da bi određene funkcije mogao da pokreće vlasnik ugovora, kao i da daje mogućnost trenutnom vlasniku da prepiše ugovor na drugog korisnika, tj da promeni vlasnika, kao i još neke funkcije koje za ovaj projekat nisu bile potrebne.

Ugovor **AggregatorV3Interface.sol** je korišćen da bi se dobile trenutne vrednosti valute Ethereum koja se stalno menja, pa nije praktično hard-kodovati tu vrednost.

Ugovor **VRFConsumerBase.sol** je implementiran iz razloga da bi se dobio pravi nasumični redni broj pobjednika, kasnije ćemo malo o LINK tokenima koji se koriste da bi se fundirao ovaj ugovor, kao i o Request-Receive metodi.

Ovi ugovori implementiraju još neke ugovore, interfejsa i biblioteke kao što su Context, AggregatorV3Interface, LinkTokenInterface, SafeMathChainLink, VRFRequestIDBase.

Vidimo da klasa Lottery koja je predstavljena kao contract Lottery nasleđuje dve klase, tj. ugovora, a to su Ownable.sol i VRFConsumerBase.sol i njih je moguće override-ovati.

Polje players predstavlja niz adresa svih igrača koji trenutno prisustvuju lutriji i sve te adrese su plative, to znači da je moguće prebaciti vrednost na te adrese, kao i prebaciti vrednost sa tih adresa na bilo koju drugu.

Polje entryFeeUSD predstavlja cenu u dolarima koja je potrebna da bi korisnik učestvovao na lutriji i ona je hardkodovana i njena vrednost kada se deploy-uje ugovor iznosi 50\$.

Polje ethUsdPriceFee predstavlja iznos koji je potreban za učestvovanje u Ethereum-u.

Vidimo polje current_state koje predstavlja neku vrednost iz enum-a STATE.

Fee i keyhash polja se koriste za dobijanje nasumičnog broja, tačnije njega koristi ugovor VRFConsumerBase, biće opširnije o tome kod funkcija ugovora.

Polje lastWinner prikazuje adresu poslednjeg pobjednika. Random prikazuje koji je nasumični broj generisan.

Polje winnerIndex označava redni broj poslednjeg pobjednika.

Polje event RequestedRandomness – događaj u Solidity-ju je nasledni član ugovora i on se emituje i čuva argumente u evidenciji transakcija.

Polja votedToRaise i votedNotToRaise predstavljaju nizove adresa korisnika koji su glasali da se ulog duplira, odnosno onih koji su glasali da se ne duplira.

Sada ćemo preći na neke od funkcija.

```
function getEntranceFee() public view returns (uint256) {
    (, int256 price, , , ) = ethUsdPriceFee.latestRoundData();
    uint256 adjustedPrice = uint256(price) * 10**10; //da bi imalo 18 decimala
    uint256 costToEnter = (entryFeeUSD * 10**18) / adjustedPrice;
    return costToEnter;
}
```

Slika 8 - funkcija getEntranceFee

Funkcija getEntranceFee je javna i ima povratnu vrednost čiji je tip uint256. View reč označava da je ova funkcija read-only, tačnije ne izvršava nikakvu promenu stanja u ugovoru. Za dobijanje potrebne vrednosti je korišćena funkcija latestRoundData() koja ima povratnu vrednost poslednje vrednosti Ethereum na marketu i imaće ukupno 8 decimala i iz tog razloga se množi sa još 10 decimala da bi se dobila vrednost u Wei valuti, jedan ether je jednak 10^{18} Wei-ja.

```
function voteForRaise() public {
    uint256 alreadyVoted;
    for (uint256 i = 0; i < votedToRaise.length; i++) {
        if (msg.sender == votedToRaise[i]) {
            alreadyVoted = 1;
        }
    }
    for (uint256 i = 0; i < votedNotToRaise.length; i++) {
        if (msg.sender == votedNotToRaise[i]) {
            alreadyVoted = 1;
        }
    }
    require(alreadyVoted == 0, "Vec ste glasali");
    require(
        current_state == STATE.CLOSED,
        "Igra je vec zapoceta, sacekajte da se zavrsi"
    );
    votedToRaise.push(msg.sender);
}
```

Slika 9 – funkcija voteForRaise

Funkcija voteForRaise je javna funkcija, koju može bilo koji korisnik da pozove. Ona će vršiti promenu stanja u ugovoru, iz razloga što menja niz tako što dodaje adrese korisnika koji su

glasali da se duplira ulog. **Require** u Solidity-ju predstavlja neki uslov da bi se određena funkcija izvršila, ako taj uslov nije zadovoljen, u ovom slučaju će štampati dato obaveštenje.

Funkcija `voteForNotRaise` radi po istom principu kao i `voteForRaise`.

```
function startLottery() public onlyOwner {
    require(
        current_state == STATE.CLOSED,
        "Jos uvek nije dozvoljeno pokretanje igre"
    );
    if (votedToRaise.length > votedNotToRaise.length) {
        entryFeeUSD = entryFeeUSD * 2;
    }
    current_state = STATE.OPEN;
    random = 0;
}
```

Slika 10 – funkcija `startLottery`

Funkcija `startLottery` se koristi da bi se sama lutrija pokrenula i nju može da pokrene isključivo vlasnik ugovora. Tokom pozivanja proverava se da li je lutrija u Zatvorenom stanju, tj. da nije pokrenuta ili da se ne bira pobednik. Pošto samo vlasnik ugovora može da je pokrene, ova funkcija nije decentralizovana, ali to nije toliko bitno jer ne utiče na odabir pobednika. If uslov proverava da li ima više korisnika koji su glasali da se duplira ulog, ako je taj uslov ispunjen `entryFeeUSD` se duplira. Jasno je da ova funkcija menja stanje ugovora.

```
function enter() public payable {
    require(current_state == STATE.OPEN);
    require(msg.value >= getEntranceFee(), "Nedovoljno ETH!");
    players.push(payable(msg.sender));
}
```

Slika 11 – funkcija `enter`

Pozivanjem ove funkcije bilo koji korisnik može da pristupi lutriji ako je ona započeta. Proverava se da li je vrednost transakcije koju šalje korisnik dovoljna da on pristupi lutriji, ako jeste u niz `players` se dodaje još jedna adresa igrača koja je plativa.

```
function endLottery() public onlyOwner {
    current_state = STATE.CALCULATING_WINNER;

    //princip request-receive
    bytes32 requestId = requestRandomness(keyhash, fee);
    emit RequestedRandomness(requestId);
}
```

Slika 12 – funkcija endLottery

Funkcija koju takođe može pozvati samo vlasnik ugovora i koja poziva funkcija requestRandomness koja vraća vrednost čiji je tip bytes32, koja će se dalje koristiti u izračunavanju nasumičnog porednika. Ova funkcija se nalazi u ugovoru VRFConsumerBase, a sada će biti malo više o tome.

Korišćenjem pametnog ugovora Chainlink VRF se dobija nasumični broj, ovaj ugovor koristi [Request & Receive Data](#) cycle. Funkcija requestRandomness inicira zahtev za nasumični broj. Funkcija fulfillRandomness je funkcija koja prima i radi **nešto** sa nasumičnim brojem.

Da bi se ugovor koristio potrebno je da korisnik ima dovoljan broj LINK tokena. Link tokeni se koriste uglavnom za testiranje ugovora. Od podataka nam je potrebno sledeće:

- adresa LINK tokena za odgovarajuću mrežu, u slučaju kod ovog projekta za Rinkeby
- adresa VRF Coordinator
- keyhash – javni ključ na osnovu kojeg se formira slučajnost
- fee – naknada koja je potrebna za izvršavanje VRF zahteva.

Dalje je potrebna API referenca za VRFConsumerBase.

Konstruktor izgleda ovako:

```
constructor(address _vrfCoordinator, address _link) public
```

_vrfCoordinator predstavlja adresu do Chainlink VRF Coordinator-a, dok _link predstavlja adresu do LINK tokena.

Kada VRFCoordinatora dobije obaveštenje da je valid VRF zahtev poziva se funkcija fulfillRandomness, koja od parametara ima bytes32 requestId i uint256 randomness. RequestId predstavlja povratno vrednost funkcije requestRandomness, dok randomness predstavlja nasumični broj.

Po tom principu funkcioniše Request-Receive, i na slici 13 će biti prikazana funkcija fulfillRandomness.

```

function fulfillRandomness(bytes32 _requestId, uint256 _random)
    internal
    override
    {
        require(
            current_state == STATE.CALCULATING_WINNER,
            "Nemoguće je pokrenuti ovu funkciju još uvek"
        );
        require(_random > 0, "Nije izračunat pobjednik");
        winnerIndex = _random % players.length;
        lastWinner = players[winnerIndex];
        lastWinner.transfer(address(this).balance);

        // nova lutrija
        players = new address payable[](0);
        votedToRaise = new address[](0);
        votedNotToRaise = new address[](0);
        current_state = STATE.CLOSED;
        random = _random;
    }

```

Slika 14 – funkcija fulfillRandomness

U ovoj funkciji će biti određen pobjednik tako što će se pronaći redni broj pobjednika u nizu players i taj pobjednik će biti isplaćen celokupnom vrednošću koja se nalazi na adresi ugovora Lottery.

U daljem tekstu će biti prikazano kako izgleda sama interakcija između ugovora i Ethereum-ove virtualne mašine.

Deploy-ovanje ugovora i pozivanje raznih funkcija korišćenjem Brownie framework-a

Brownie je framework koji se koristi za manipulaciju pametnog ugovora, i koji cilja Ethereum virtualnu mašinu. Brownie se pokreće iz terminala, on se oslanja na Web3.py. Web3.py je Python biblioteka koja se koristi za interakciju sa Ethereumom.

Omogućava slanje transakcija, interakciju sa pametnim ugovorom, čitanje vrednosti blokova, i još mnogo drugih stvari.

Originalan API je izveden iz Web3.js JavaScript API-ja, ali je napredovao i napravljen da radi i uz pomoć Python programskog jezika.

Slede neke od bitnih pomoćnih funkcija koje se koriste u glavnim funkcijama za interakciju sa pametnim ugovorom.

```
FORKED_LOCAL_ENVIRONMENTS = ["mainnet-fork", "mainnet-fork-dev"]
LOCAL_BLOCKCHAIN_ENVIRONMENTS = ["development", "ganache-local"]

def get_account(index=None, id=None):
    if index:
        return accounts[index]
    if id:
        return accounts.load(id)
    if (
        network.show_active() in LOCAL_BLOCKCHAIN_ENVIRONMENTS
        or network.show_active() in FORKED_LOCAL_ENVIRONMENTS
    ):
        return accounts[0]

    return accounts.add(config["wallets"]["from_key"])
```

Slika 15 – pomoćna funkcija get_account

Nizovi FORKED_LOCAL_ENVIRONMENTS i LOCAL_BLOCKCHAIN_ENVIRONMENTS predstavljaju nizove na koje se može ugovor deploy-ovati ili testirati.

Moguće je dodati novu mrežu pomoću brownie komande: brownie networks add [environment] [id] host=[host] [KEY=VALUE, ...]

Vidimo da postoje dva parametara, index i id koji nisu obavezni. Ako se koristi bilo koji od navedenih mreža iz gornja dva niza, funkcija vraća nalog sa rednim brojem 0. Ako to nije slučaj izvršiće se dodavanje novog naloga sa privatnim ključem koji je definisan u .env fajlu.

Vidimo u poslednjoj liniji da se poziva config, on predstavlja vrednosti iz fajla **brownie-config.yaml**. U tom fajlu se nalaze korisnicima dostupne vrednosti – slika 16.

```
dependencies:
  - smartcontractkit/chainlink-brownie-contracts@1.1.1
  - OpenZeppelin/openzeppelin-contracts@3.4.0

dotenv: .env

compiler:
  solc:
    remappings:
      - "@chainlink=smartcontractkit/chainlink-brownie-contracts@1.1.1"
      - "@openzeppelin=OpenZeppelin/openzeppelin-contracts@3.4.0"

networks:
  default: development
  development:
    keyhash: "0x2ed0feb3e7fd2022120aa84fab1945545a9f2ffc9076fd6156fa96eaff4c1311"
    fee: 1000000000000000000
  rinkeby:
    vrf_coordinator: "0xb3dCcb4Cf7a26f6cf6B120Cf5A73875B7BBc655B"
    eth_usd_price_feed: "0x8A753747A1Fa494EC906cE90E9f37563A8AF630e"
    link_token: "0x01BE23585060835E02B77ef475b0Cc51aA1e0709"
    keyhash: "0x2ed0feb3e7fd2022120aa84fab1945545a9f2ffc9076fd6156fa96eaff4c1311"
    fee: 1000000000000000000
    verify: True
  mainnet-fork:
    eth_usd_price_feed: "0x5f4eC3Df9cbd43714FE2740f5E3616155c5b8419"

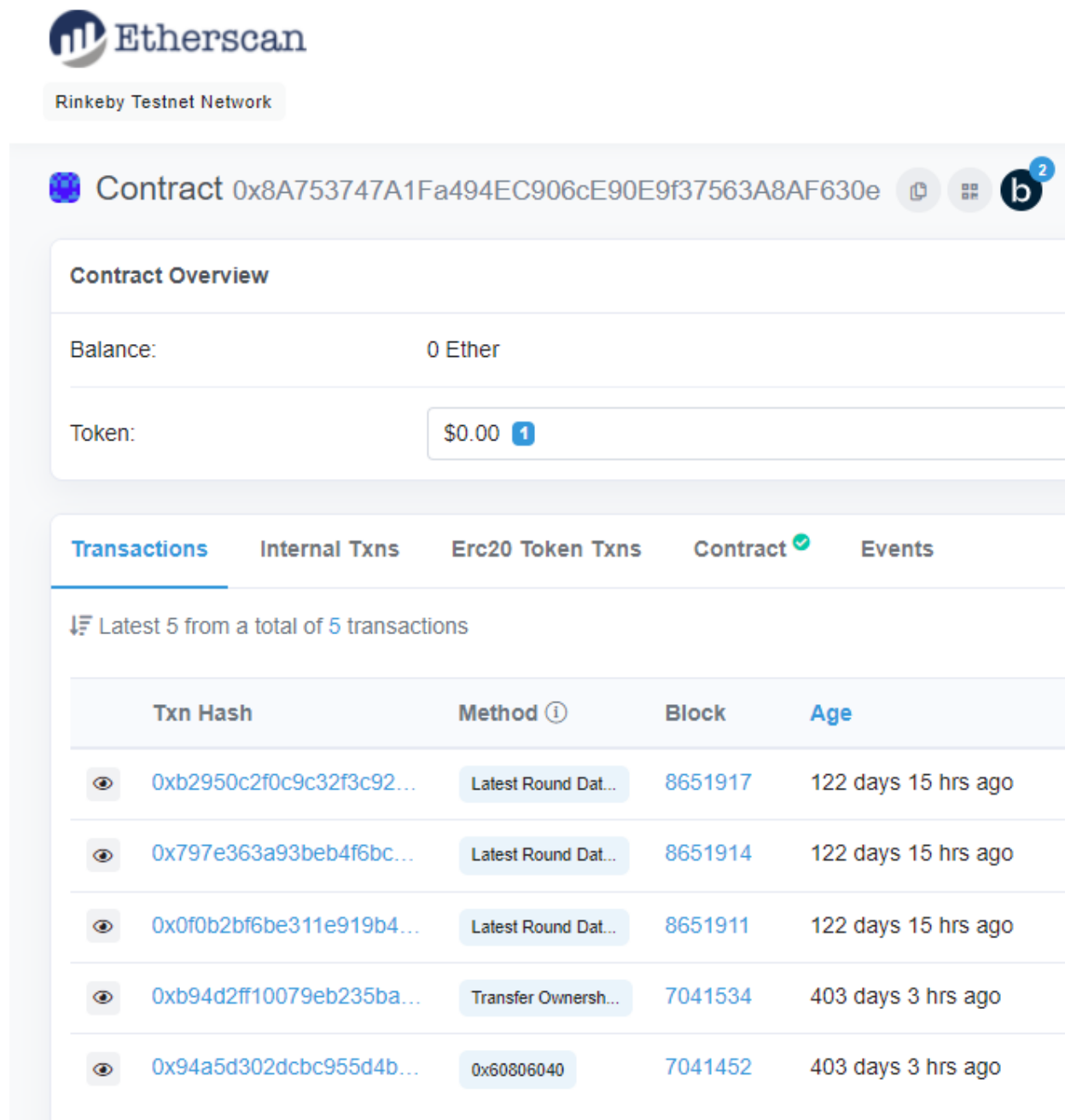
wallets:
  from_key: ${PRIVATE_KEY}
```

Slika 16 – fajl brownie-config.yaml

Primećuje se da su u ovom fajlu sve vrednosti hardkodovane. U dependencies se nalaze ugovori koje smo importovali, jer brownie ne može da ih registruje i kompajlira, pa se zato vrši remapping.

Tu je i polje dotenv koje povezuje .env fajl, gde se nalaze bitne privatne informacije, kao što je na primer privatni ključ naloga.

Networks daje informacije u zavisnosti da li vršimo interakciju na lokalnom blockchain-u ili na raznim test mrežama, kao što su Kovan, Rinkeby i ostale. U ovom fajlu se nalaze razne adrese koje su potrebne za određene ugovore. Uzmimo, recimo, adresu za **eth_usd_price_feed** na Rinkeby mreži – slika 17.



The screenshot shows the Etherscan interface for the Rinkeby Testnet Network. At the top, the Etherscan logo and 'Rinkeby Testnet Network' are visible. Below this, the contract address **0x8A753747A1Fa494EC906cE90E9f37563A8AF630e** is displayed. The 'Contract Overview' section shows a balance of 0 Ether and a token value of \$0.00. The 'Transactions' tab is selected, showing a list of the latest 5 transactions. The table includes columns for Txn Hash, Method, Block, and Age.

Txn Hash	Method	Block	Age
0xb2950c2f0c9c32f3c92...	Latest Round Dat...	8651917	122 days 15 hrs ago
0x797e363a93beb4f6bc...	Latest Round Dat...	8651914	122 days 15 hrs ago
0x0f0b2bf6be311e919b4...	Latest Round Dat...	8651911	122 days 15 hrs ago
0xb94d2ff10079eb235ba...	Transfer Ownersh...	7041534	403 days 3 hrs ago
0x94a5d302dc955d4b...	0x60806040	7041452	403 days 3 hrs ago

Slika 17 – adresa za **eth_usd_price_feed** na Rinkeby mreži

Dakle ova adresa predstavlja adresu već postojećeg ugovora, koju ćemo mi iskoristiti za različite funkcije u našim ugovorima.

Nakon pojašnjenja samog brownie-config.yaml fajla, biće dosta jasnije sledeće funkcije.

```
def get_contract(contract_name):
    contract_type = contract_to_mock[contract_name]
    if network.show_active() in LOCAL_BLOCKCHAIN_ENVIRONMENTS:
        if len(contract_type) <= 0:
            deploy_mock()
            contract = contract_type[-1]
        else:
            contract_address = config["networks"][network.show_active()][contract_name]
            contract = Contract.from_abi(
                contract_type._name, contract_address, contract_type.abi
            )
    return contract
```

Slika 18 – pomoćna funkcija get_contract

Funkcija pomoću koje se dobija koji je ugovor trenutno u pitanju, od parametara imamo njegov naziv. Ako se izvršavanje ove funkcije vrši na lokalnoj mreži, prvo se proverava da li je već deploy-ovan taj ugovor, ako nije onda će se izvršiti funkcija deploy_mock, a ako jeste uzima se tip ugovora poslednjeg deploy-ovanog.

U suprotnom ako se ova funkcija vrši na mrežama koje nisu lokalne onda se uzimaju vrednosti adresa iz config fajla i pomoću ugrađene funkcije **Contract.from_abi** dobija pravi ugovor.

Funkcija deploy_mock je funkcija iz interfejsa **AggregatorV2V3Interface.sol** koja dinamički kreira nove ugovore po potrebi. Za pozivanje te funkcije je potrebno donirati LINK tokenene.

Doniranje Link tokena se vrši pomoću sledeće funkcije:

```
def fund_with_link_token(
    contract_address, account=None, link_token=None, amount=1000000000000000000
):
    account = account if account else get_account()
    link_token = link_token if link_token else get_contract("link_token")
    tx = link_token.transfer(contract_address, amount, {"from": account})
    tx.wait(1)
    print("Isplacen kontrakt")
    return tx
```

Slika 19 – funkcija fund_with_link_token

Podrazumevana vrednost je 0.1 LINK token. Funkcija koja se poziva je link_token.transfer, koja od parametara prima adresu ugovora koji se treba fundirati, koliko LINK tokena želimo da doniramo i koja adresa vrši transakciju.

U daljem tekstu će biti malo više o samom deploy.py fajlu koji je zaslužan za postavljanje ugovora na Rinkeby testnet kao i za razne transakcije.

```
def deploy_lottery():
    account = get_account()
    lottery = Lottery.deploy(
        get_contract("eth_usd_price_feed").address,
        get_contract("vrf_coordinator").address,
        get_contract("link_token").address,
        config["networks"][network.show_active()][ "fee"],
        config["networks"][network.show_active()][ "keyhash"],
        {"from": account},
        publish_source=config["networks"][network.show_active()].get("verify", False),
    )
    print("Kontrakt je deploy-ovan")
    return lottery
```

Slika 20 – funkcija deploy_lottery

Pomoću ove funkcije se vrši deploy-ovanje ugovora na određenu mrežu. Get_account funkcija, kao i get_contract funkcija su već detaljnije objašnjene u gornjem tekstu. Sama funkcija deploy koja se poziva iz ugovora Lottery, služi za postavljanje ugovora na mrežu.

Od parametara prima adresu ugovora za eth_usd_price feed, kao i adrese ugovora za vrf_coordinatora i link_token. Prima adrese za fee i keyhash, koje će biti korišćene da bi se odredio nasumični broj, i o tome je bilo reči u gornjem tekstu.

Za svaku transakciju koja menja stanje ugovora je bitan parametar {"from": account}, koji daje adresu korisnika koji poziva ovu funkciju.

Publis_source polje obezbeđuje da ugovor bude verifikovan kada se postavi na mrežu. Daje informacije adresu vlasnika, potvrđuje licencu i drugo.

```
def start_lottery():
    account = get_account()
    lottery = Lottery[-1]
    starting_tx = lottery.startLottery({"from": account})
    starting_tx.wait(1)
    print("Pokrenuta lutrija")
```

Slika 21 – funkcija start_lottery

Struktura ove funkcije je malo drugačija nego prethodne. Vidimo da se adresa korisnika koji poziva funkciju dobija na isti način, ali sledeća linija nije kao prethodna.

Lottery promenljiva uzima za svoju vrednost objekat poslednjeg ugovora koji je deploy-ovan. Vidimo da se vrši transakcija koja se poziva iz ugovora Lottery i od parametara prima adresu korisnika koji je pozvao ovu funkciju.

Starting_tx.wait(1) se koristi da bi program sačekao da se izvrši transakcija, pa tek onda nastavlja dalje kroz izvršavanje programa.

Funkcije vote_to_raise i vote_not_to_raise se izvršavaju na potpuno isti način kao i funkcija start_lottery, zato što ne šalju nikakvu vrednost, za razliku od funkcije enter_lottery:

```
def enter_lottery():
    account = get_account()
    lottery = Lottery[-1]
    value = lottery.getEntranceFee() + 1000000000
    tx = lottery.enter({"from": account, "value": value})
    tx.wait(1)
    print("Uspesno ste se platili ucesce za lutriju")
```

Slika 22 – funkcija enter_lottery

Kada se šalje ova transakcija, zajedno sa njom se šalje određena vrednost koja je potrebna za učešće na lutriji ka oi dodatnih 1000000000 zbog raznih taksi tokom izvršavanja transakcije.

Ostali delovi funkcije su identični kao kod funkcije start_lottery.

```
def end_lottery():
    account = get_account()
    lottery = Lottery[-1]
    tx = fund_with_link_token(lottery)
    tx.wait(1)
    ending_transaction = lottery.endLottery({"from": account})
    ending_transaction.wait(1)
    # sleep jer treba da se izvrse ove funkcije za random broj
    time.sleep(60)
    print("Pobednik je izracunat!")
```

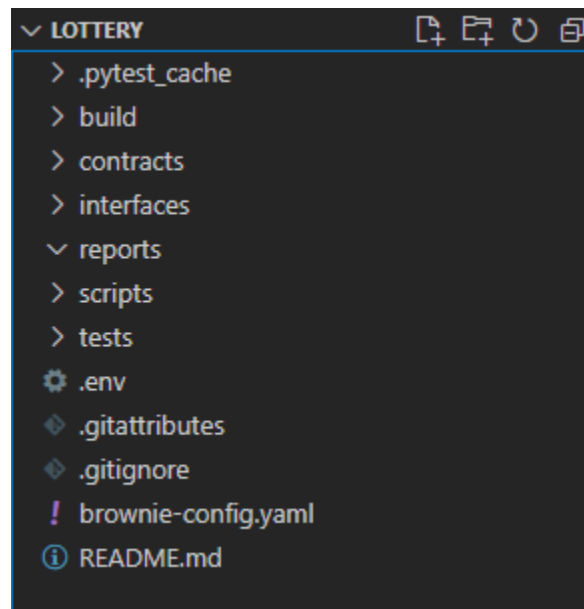
Slika 23 – funkcija end_lottery

U funkciji end_lottery se izvršavaju dve transakcije, jedna koja vrši doniranje LINK tokenima, druga koja vrši samu funkciju endLottery iz ugovora Lottery.

time.sleep(60) se koristi da bi sve ove transakcije uspele da se izvrše na Ethereum-u, jer je Ethereum dosta spor. Ovo vreme može da varira zavisi od situacije do situacije.

Testiranje pomoću Brownie framework-a

Kada se pristupi folderu preko terminala u kom želimo napraviti projekat, potrebno je koristiti komandu **brownie init**, tada dobijamo sledeću strukturu.



Slika 24 – struktura projekta

Nama su bitni sledeći folderi: build, contracts, interfaces, scripts i tests.

Folder **build** predstavlja folder u kome se nalaze svi fajlovi JSON strukture koji predstavljaju ugovore, interfejse, testove, dok postoji folder deployments gde možemo pronaći sve verzije ugovora koje smo deploy-ovali do sada.

U folderu **contracts** se nalaze svi ugovori koji su pisani u Solidity-ju koji se koriste u projektu.

Folder **interfaces** predstavlja sve interfejse koji su uključeni u projekat.

Folder **scripts** sadrži skripte pisane u Python-u za interakciju sa Ethereum-om, u slučaju kod ovog projekta, to su fajlovi deploy.py i help.py.

Folder **tests** služi za testiranje ugovora i raznih funkcija tog ugovora. U daljem tekstu će biti malo više o testiranju, kao i prikaz nekih testova za ovaj projekat.

Testiranje se može vršiti lokalno, kao i na određenim Testnet-ovima. Najbolje je prvo izvršiti lokalno, jer je poznato da retko koji program može da se napiše bez ikakvih bagova iz prvog pokušaja. Zato se koristi lokalno testiranje, jer je dosta brže nego vršiti testiranje na testnet-u.

Lokalno testiranje se vrši komandom u terminalu: **brownie test**, jer je za podrazumevanu mrežu definisan ganache, tj. da se testira na lokalnu.

U folderu tests se nalaze dva fajla – test_lottery_unit.py koji služi za testiranje na lokalnu i test_lottery_integration.py koji služi za testiranje na testnetu. Sledi prikaz fajla test_lottery_unit.py.

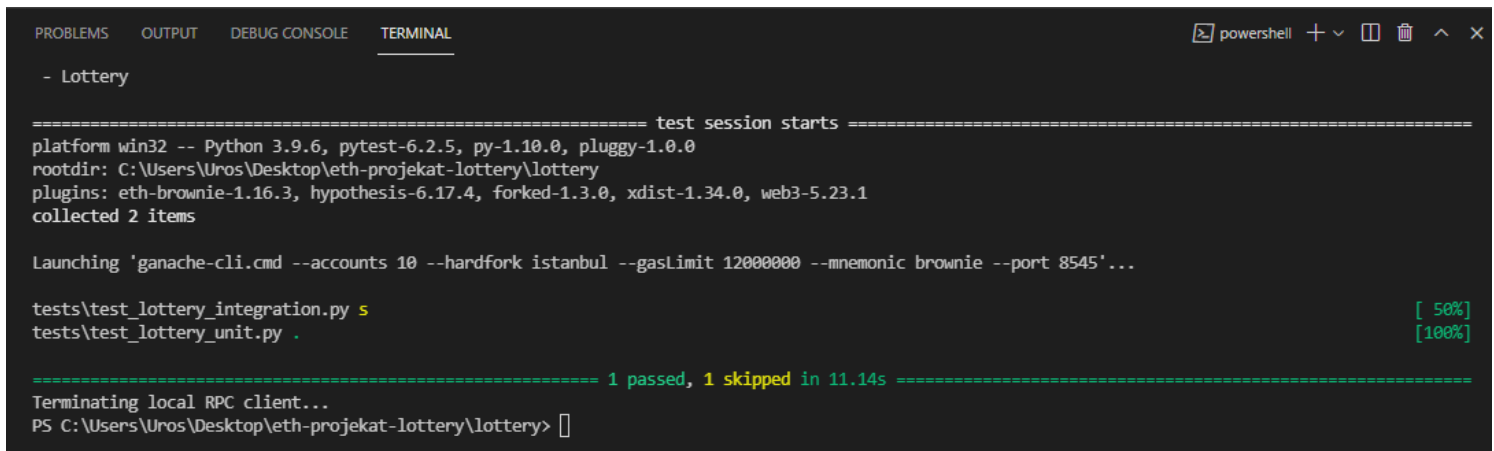
```
def test_can_pick_winner():
    if network.show_active() not in LOCAL_BLOCKCHAIN_ENVIRONMENTS:
        pytest.skip()
    lottery = deploy_lottery()
    account = get_account()
    lottery.startLottery({"from": account})
    lottery.enter({"from": account, "value": lottery.getEntranceFee()})
    lottery.enter({"from": get_account(index=1), "value": lottery.getEntranceFee()})
    lottery.enter({"from": get_account(index=2), "value": lottery.getEntranceFee()})
    fund_with_link_token(lottery)
    transaction = lottery.endLottery({"from": account})
    request_id = transaction.events["RequestedRandomness"]["requestId"]
    RNG = 1000
    get_contract("vrf_coordinator").callbackWithRandomness(
        request_id, RNG, lottery.address, {"from": account}
    )
    starting_balance_of_account = account.balance()
    balance_of_lottery = lottery.balance()
    assert lottery.lastWinner() == get_account(index=1)
    assert lottery.balance() == 0
    assert account.balance() == starting_balance_of_account + balance_of_lottery
```

Slika 25 – prikaz fajla test_lottery_unit.py

Primećujemo dosta poznatih funkcija, samo deploy-ovanje ugovora, razne transakcije za pokretanje ugovora, pristupanje lutriji, doniranje LINK tokenima, prikaz kako funkcioniše funkcija za dobijanje nasumičnog igrača, koja je u ovom slučaju hard-kodovana tako da je nasumični broj koji se dobija jednak 1000, dakle pobjednik će biti igrač sa rednim brojem 1 u nizu players. Nasumični broj je hardkodovan u ovom slučaju zbog testiranja.

Koristi se funkcija callbackWithRandomness koja će pozvati druge funkcije iz drugih ugovora i na kraju smestiti pobjednika u polje lastWinner. Takođe primećuje se da će brownie ovu funkciju testiranja da zanemari ako se testiranje vrši na mrežama koje nisu lokalne.

Sledi dalji prikaz konzole nakon izvršavanja komande za testiranje.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
- Lottery

===== test session starts =====
platform win32 -- Python 3.9.6, pytest-6.2.5, py-1.10.0, pluggy-1.0.0
rootdir: C:\Users\Uros\Desktop\eth-projekat-lottery\lottery
plugins: eth-brownie-1.16.3, hypothesis-6.17.4, forked-1.3.0, xdist-1.34.0, web3-5.23.1
collected 2 items

Launching 'ganache-cli.cmd --accounts 10 --hardfork istanbul --gasLimit 12000000 --mnemonic brownie --port 8545'...

tests\test_lottery_integration.py s [ 50%]
tests\test_lottery_unit.py . [100%]

===== 1 passed, 1 skipped in 11.14s =====
Terminating local RPC client...
PS C:\Users\Uros\Desktop\eth-projekat-lottery\lottery>
```

Slika 26 – prikaz konzole nakon lokalnog testiranja

Prvo što se primećuje u konzoli jeste da je pokrenut ganache na kome će se izvršiti testiranje. Vidimo da se fajl `test_lottery_integration.py` zanemaruje, iz razloga što u njemu ne postoje funkcije koje se testiraju lokalno.

Ovaj test je prošao dakle svi uslovi (assert-ovi) koji su zadati su ispunjeni, kako je sam programer zamislio.

Ista funkcija `test_can_pick_winner` je implementirana u fajlu `test_lottery_integration.py` koja će biti testirana na mrežama koje nisu lokalne, recimo Rinkeby.

Testiranje će se vršiti pomoću komande **`brownie test --network rinkeby`**, sledi prikaz fajla.

```
def test_can_pick_winner():
    if network.show_active() in LOCAL_BLOCKCHAIN_ENVIRONMENTS:
        pytest.skip()
    lottery = deploy_lottery()
    account = get_account()
    lottery.startLottery({"from": account})
    lottery.enter({"from": account, "value": lottery.getEntranceFee()})
    lottery.enter({"from": account, "value": lottery.getEntranceFee()})
    fund_with_link_token(lottery)
    lottery.endLottery({"from": account})
    time.sleep(60)
    assert lottery.lastWinner() == account.address
```

Slika 27 – prikaz fajla `test_lottery_integration.py`

Ovde nije potrebno izvršiti razne funkcije za dobijanje nasumičnog korisnika, jer će to za nas uraditi funkcija `endLottery` iz ugovora `Lottery`.

Zaključak

Glavno pitanje je kako je projekat povezan sa Ethereum-om?

Povezivanje sa Ethereum-ovim blockchain-om je izvršeno preko third party software-a **INFURA** koji predstavlja Ethereum-ov API, ka oi IPFS API, i još mnogo toga, uglavnom olakšava ceo taj proces. IPFS je peer-to-peer mreža koja se koristi za skladištenje fajlova.

Da bi se povezali sa blockchain-om potrebno je napraviti nalog na <https://infura.io/> i napraviti novi projekat, gde će se kreirati ID tog projekta koji treba smestiti u .env fajl.

Bitno je napomenuti da sve tehnologije imaju svoje prednosti i mane. Glavne mane Ethereum-ovog blockchain-a su to što ne služi samo kao kriptovaluta, već na njegovom blockchain-u mogu da se podižu razni softveri, to ga čini dosta ranjivim zbog prevelikog broja podataka, pa dolazi često do obaranja mreže, čak i do hakovanja, već se desilo da je Ethereum-u ukradeno preko 600 miliona dolara.

Prednosti Ethereum-a su te što ne postoji nikakva centralizacija, niti tajnost. Svi source code-ovi su dostupni svakom i svako ima pristup njima.

Ceo projekat je rađen sa Test kriptovalutama, dakle nema pojavljivanja nikakvog stvarnog novca u mom slučaju. Uglavnom test kriptovalute koriste svi programeri koji rade na blockchain-u da ne bi došlo do gubitka novca zbog nekog problema. Takođe se koriste i zbog raznih transakcija, svaka transakcija ima određenu taksu, dok programer pri razvoju softvera pokrene razne transakcije po nekoliko stotina puta, zavisi od programa do programa.

Ethereum blockchain ima mesta za napredak. Iz godine u godinu se sve više koristi, jer su sami korisnici prepoznali prednosti ove tehnologije, a to je da mogu imati potpuno poverenje u tehnologiju.

Literatura

- [1] Solidity documentation – 29.08.2021 <https://docs.soliditylang.org/en/v0.6.6/>
- [2] Solidity tutorial – 29.08.2021. <https://www.youtube.com/watch?v=ipwxYa-F1uY>
- [3] Brownie documentation – 2.09.2021. <https://eth-brownie.readthedocs.io/en/stable/>
- [4] Web3.py documentation – 2.09.2021. <https://web3py.readthedocs.io/en/stable/index.html>
- [5] Chainlink documentation – 13.09.2021. <https://docs.chain.link/>
- [6] Brownie github repository – 13.09.2021. <https://github.com/eth-brownie/brownie>
- [7] Chainlink github repository – 17.09.2021 <https://github.com/smartcontractkit/chainlink>
- [8] Stackoverflow – 20.09.2021. <https://stackoverflow.com/>
- [9] Ethereum Stackexchange – 22.09.2021. <https://ethereum.stackexchange.com/>
- [10] Ethereum development documentation – 23.09.2021.
<https://ethereum.org/en/developers/docs/>
- [11] Github repositories on profile PatrickAlphaC – 24.09.2021.
<https://github.com/PatrickAlphaC>