Raspodijeljene glavne knjige i kriptovalute

Druga laboratorijska vježba: Pametni ugovori

Prosinac 2020.

Uvod

Cilj vježbe je upoznati se sa Solidity programskim jezikom za programiranje pametnih ugovora na EVM-u (engl. *Ethereum Virtual Machine*), te sigurnosnim detaljima na koje morate obratiti pažnju prilikom pisanja pametnih ugovora.

Truffle

Za potrebe laboratorijske vježbe pripremljen je početni kod pisan u Solidity jeziku. Za potrebe razvoja potrebno je koristiti Truffle okruženje (trufflesuite.com/). Truffle omogućava jednostavan razvoj novih ugovora, postavljanje ugovora na blockchain te njihovo jednostavno testiranje. Struktura Truffle projekta sastoji se od 3 direktorija:

- `contracts` - direktorij u kojem se nalaze svi pametni ugovori,
- `migrations` - direktorij koji sadrži tzv. migracije pametnih ugovora. Migracije trenutno nisu bitne za potrebe rješavanja ove laboratorijske vježbe. Više o migracijama možete saznati ovdje: truffleframework.com/docs/truffle/getting-started/running-migrations,
- `test` - direktorij koji sadrži sve testove pametnih ugovora.

Truffle podržava testove koji se mogu pisati u jezicima Solidity ili Javascript. U početnom kodu su već implementirani testovi, pisani u jeziku Solidity. Primjetite da su testovi u Solidityu zapravo novi pametni ugovori čije metode implementiraju logiku testova. Truffle svaki test pokreće u čistom okruženju (engl. *clean room*) kako bi se svaki test izvršio neovisno o drugim testovima.

Instalacija

Za instalaciju Truffle okruženja potrebni su `Node.js` okruženje i `npm` (engl. *Node package manager*), dostupni na: nodejs.org, te Python 2. Nakon što su istalirani `Node.js` i `npm`, ovisnosti projekta možete instalirati pokretanjem naredbe `npm install`. Naredba će instalirati sve ovisnosti potrebne za normalno korištenje sustava ($\sim$ 150MB) u direktorij `node_modules`.

Kako bi Truffle omogućio brzo i lagano testiranje ugovora potreban mu je testni blockchain (po mogućnosti prisutan na lokalnom uređaju). Naredbom `npm run start` pokrenut će Ganache - lokalni Ethereum čvor (engl. *full node*) koji je dio Truffle okruženja (truffleframework.com/ganache). Nakon što je pokrenut čvor, pozivanjem naredbe `npm run test` pokrenut će se svi testovi. Pojedine testove unutar jedne datoteke moguće je pokrenuti predajom relativne putanje do te datoteke skripti za pokretanje testova (npr. `npm run test test/AuctionTest.sol`)
Preporuča se korištenje IDE-a koji ima podršku za Solidity, poput editora Atom (atom.io) ili Intellij IDEA (jetbrains.com/idea).

Zadaci

1. **Crowdfunding**
   Prvi zadatak je napisati pametni ugovor koji će omogućiti vlasniku ugovora grupno prikupljanje sredstava (engl. *crowdfunding*), poput usluge koju pružaju Kickstarter, Indiegogo i sl. Svako grupno prikupljanje sredstava je ograničeno vremenom trajanja i novčanim ciljem. Pametni ugovor mora omogućiti bilo kome da uplati sredstva u projekt dok je grupno prikupljanje aktivno. Ako je do kraja grupnog prikupljanja dosegnut cilj, onda vlasnik ugovora ima pravo preuzeti investirana sredstva, u suprotnom investitori imaju pravo na povrat novca. Za vrijeme trajanja grupnog investiranja vlasnik ugovora nema pravo preuzeti trenutno prikupljena sredstva, niti investitori imaju pravo na povrat investicije. Investitorima trebaju biti omogućeno investirati više puta. U početnm kodu je već pripremljen kostur ugovora pod nazivom `Crowdfunding.sol`. U ugovoru morate dopuniti tri metode, `invest()`, `refund()` i `claimFunds()` koje moraju implementirati gore navedene funkcionalnosti. Testovi za ovaj zadatak se nalaze u `CrowdfundingTest.sol` datoteci.

2. **Aukcija**
   U ovom zadatku potrebno je napisati ugovor koji će služiti kao osnova za ostale vrste aukcija (zadatci 3 i 4). Općenito, aukcija je proces trgovanja u kojem se više potencijalnih kupaca nadmeće oko cijene za određeni predmet aukcije. Aukciju možemo gledati kao automat s konačnim brojem stanja:

   - NOT_FINISHED - Aukcija još traje.
   - SUCCESSFUL - Aukcija je uspješno završila, tj. netko je ponudio dovoljno veliku cijenu za predmet aukcije prije nego je aukcija završila.
   - NOT_SUCCESSFUL - Aukcija nije uspješno završila - to se može dogditi iz dva razloga: (i) Nitko nije ponudio cijenu za predmet aukcije do završetka aukcije, (ii) netko je ponudio najveću cijenu za predmet aukcije, ali iz nekog razloga ta osoba nema pravo kupiti predmet aukcije (npr. ponuđena cijena je manja od minimalne cijene).

   Vaš zadatak je napisati ugovor koji implementira dvije metode: `settle()`, koja implementira završetak aukcije i šalje sredstva prodavaču, te `refund()`, koja implementira povrat sredstva investitoru ako prodavač nije u mogućnosti isporučiti predmet aukcije ili ako neki uvjet auckije nije zadovoljen.

   U slučaju da je definiran sudac (kao arbiter aukcije), metodu `settle()` za završetak aukcije mogu pozvati samo sudac ili pobjednik aukcije. Ako sudac ne postoji onda bilo tko može zatražiti da se sredstva pobjednika aukcije prebace na prodavača predmeta aukcije. Metodu za završetak aukcije se smije pozvati samo kada je aukcija uspješno završena (SUCCESSFUL).

   U slučaju da je definiran sudac, metodu `refund()` za povrat sredstva kupcu mogu pozvati samo prodavač ili sudac. Ako sudac ne postoji onda bilo tko može zatražiti da se sredstva vrate kupcu. Metodu za povrat sredstva se smije pozvati samo onda kada aukcija nije uspješno završena (NOT_SUCCESSFUL).

   Za ovaj zadatak je pripremljen početni kod u datoteci `Auction.sol`. Morate dopuniti metode `settle()` i `refund()`. Testovi za ovaj zadatak nalaze se u datoteci `AuctionTest.sol`.

3. **Nizozemska aukcija**
   Nizozemska aukcija kreće s visokom cijenom koja se postepeno snižava do nekog predodređenog iznosa nakon kojeg se cijena predmeta aukcije više ne snižava. Čim neki kupac ponudi trenutnu cijenu za predmet aukcije, aukcija (uspješno) završava te taj kupac postaje pobjednik aukcije. Ukoliko cijena dostigne minimalni predoređeni iznos bez ijedne ponude, smatra da je aukcija završila neuspješno.

   Vaš zadatak je napisati pametni ugovor koji odgovara nizozemskoj aukciji. Ugovor mora naslijediti svojstva pametnog ugovora iz 2. zadatka (`Auction.sol`). Ugovor mora imati svojstva

da bilo tko može kupiti predmet aukcije za vrijeme dok je aukcija aktivna (cijena predmeta nije pala ispod minimalne cijene ili već netko prije nije kupio predmet). Da bi netko kupio predmet aukcije mora pozvati metodu `bid()` s vrijednosti (`msg.value`) većom ili jednakom trenutnom vrijednosti predmeta aukcije. Ako je kupac slučajno preplatio predmet, morate mu vratiti razliku. Ako je netko pozvao metodu `bid()` s manje novaca nego je trenutna vrijednost predmeta aukcije, morate odbiti takvu ponudu i vratiti kupcu novce koje je poslao. Cijena predmeta aukcije linearno opada s vremenom, za predefiniranu vrijednost, od početka aukcije. Trenutno vrijeme možete dobiti upotrebom `time()` metode definirane u `Auction.sol`. U početnom kodu je pripremljen kostur pametnog ugovora nizozemske aukcije u datoteci `DutchAuction.sol` u kojem morate popuniti metodu `bid()`. Testovi za ovaj zadatak nalaze se u datoteci `DutchAuctionTest.sol`.

4. **Engleska aukcija**
   Engleska aukcija je "klasična" aukcija gdje cijena predmeta počinje od niske početne cijene i raste kada kupac predlaže cijenu koju je spreman platiti. Aukcija završava uspješno ukoliko postoji najviša ponuda, nakon što određen period nitko ne predloži novu najveću cijenu. Ukoliko ne pristigne niti jedna ponuda viša od početne cijene, akcija završava neuspješno.

   Vaš zadatak je omogućiti bilo kome da predloži novu višu cijenu za predmet aukcije. Nova vrijednost cijene mora biti veća od trenutne za neki predodređeni iznos (bilo bi besmisleno povisiti cijenu s 500.000$ na 500.001$). Ako kupac ne predloži dovoljno veliku cijenu, takva ponuda se ne smije prihvatiti i sredstva moraju biti vraćena kupcu. Ako netko predloži novu cijenu koja je dovoljno velika onda se ta cijena uzima kao nova referentna cijena. Da bi netko predložio novu cijenu mora uplatiti taj iznos na račun ugovora (to se treba dogoditi prilikom poziva metode `bid()`), a kupcu koji je imao prethodno najveću ponudu se vraća njegov iznos. Vaš zadatak je implementirati metodu `bid()` s gore navedenim svojstvima te implementirati metodu `getHighestBidder()`. Metoda `getHighestBidder()` treba vratiti adresu kupca koji je uspješno ponudio najveću cijenu i ostvario pravo na kupnju predmeta aukcije po toj cijeni. Ako takav ne postoji zato što još nitko nije prihvatio početnu cijenu ili aukcija nije završila, tada ta metoda mora vratiti adresu `0x0`. U početnom kodu je pripremljen kostur pametnog ugovora engleske aukcije u kojem se nalaze metode `bet()` i `getHighestBidder()` koje trebate implementirati. Testovi za ovaj zadatak nalaze se u datoteci `EnglishAuctionTest.sol`.

5. ***Dodatni zadatak - Lov na blago**
   Ovo je dodatni zadatak koji nije obavezno riješiti, te ne nosi bodove.

   U početnom kodu su pripremljeni pametni ugovori `TreasureHunt` i `TreasureHuntSolution` u datoteci `TreasureHuntSolution.sol`. Cilj zadatka je prebaciti sredstva s ugovora `TreasureHunt` na svoju adresu. Vaš zadatak je proučiti kod ugovora `TreasureHunt` i otkriti koju vrijednost trebate poslati u metodu `openTreasure(uint256)` kako bi si prebacili Ether pohranjen na tom pametnom ugovoru. Vrijednost s kojom to možete postići zapišite u metodi `getSolution()` ugovora `TreasureHuntSolution`. Ako se odlučite rješavati ovaj zadatak odkomentirajte test za provjeru zadatka u datoteci `TreasureHuntTest.sol`.

Predaja

Laboratorijsku vježbu možete rješavati sami ili u grupi od najviše 2 ljudi. Ako se odlučite raditi laboratorijsku vježbu u grupi, pošaljite zahtjev za grupni rad s imenima i JMBAG-om studenata koji će raditi u grupi na adresu rgkk@fer.hr **do 11.12.2020. u 23:59.**.

Laboratorijsku vježbu predajete putem MS Teamsa tako da učitate, pod "Assignment", *.zip* datoteku koja mora sadržavati cijeli projekt laboratorijske vježbe osim `node_modules` direktorija te ostalih datoteka koje vaš IDE stvori. **Prilikom predaje vježbe obratite pažnju na to da morate pritisnuti gumb "Turn In" kako bi ispravno predali rješenje. Ukoliko ne pritisnete "Turn In" riješenje laboratorijske vježbe Vam neće biti priznato.** Naziv predane datoteke mora biti u obliku *ime-prezime-jmbag.zip*. Ukoliko radite u grupi, neka naziv datoteke bude ime, prezime

i JMBAG jednog od studenata iz grupe. **Rok za predaju laboratorijske vježbe je 16.12.2020. u 23:59. Sve predaje nakon roka neće biti priznate (osim u slučaju opravdane više sile), što povlači pad laboratorijske vježbe te predmeta.**

Ukoliko imate pitanja vezana za laboratorijsku vježbu, biti će održanje konzultacije preko MS Teams platforme uz prethodni dogovor putem elektroničke pošte `rgkk@fer.hr`.
Uvjet za priznavanje rješenja laboratorijske vježbe je ispravno riješen prvi zadatak. Napominjemo da je ovo također i uvjet za prolazak predmeta.

Naputci

- Laboratorijska vježba je pisana u Solidity jeziku verzije `v0.4.24`. Ovo je stara verzija te je trenutno aktivna verzija `v0.7.5`. Prilikom čitanja dokumentacije pripazite koju verziju dokumentacije čitate. Također, prilikom pisanja ugovora jako pripazite na sigurnosne propuste, pogotovo zato što compiler verzije `v0.4.24` Vas ne upozorava na većinu loših praksi i sigurnosnih propusta koji su eliminirani u višim verzijama jezika ili ih je moguće detektirati u compile time-u.

- Preporuka je da pročitate cijelu dokumentaciju Solidity jezika, kako Solidity upravlja memorijom pametnih ugovora i sigurnosne oblikovne obrasce (poput "*withdrawal*" obrasca) koje bi trebali koristiti kako bi vaši ugovori bili sigurni. Znanje o oblikovnim obrascima nije potrebno za uspješno rješavanje laboratorijske vježbe, ali bi vam moglo biti izuzetno korisno ako se odlučite baviti programiranjem pametnih ugovora. Projekt OpenZeppelin (dostupan na: github.com/OpenZeppelin/openzeppelin-solidity) nudi već gotove implementacije najčešćih pametnih ugovora (poput ugovora ERC20 Token) i najboljih praksi koje se trebaju koristiti pri pisanju pametnih ugovora.

- Truffle okruženje nudi mogućnost jednostavnog debugiranja pametnih ugovora. Proces debugiranja pametnih ugovora je znatno drugačiji od debugiranja u nekom "običnom" programskom jeziku. Da bi mogli debagirati neku metodu pametnog ugovora prvo morate taj pametni ugovor postaviti na blockchain te pozvati metodu sa željenim argumentima. Tek nakon što je transakcija koja je izvršila poziv metode sa željenim argumentima zapisana u blockchainu, možete debugirati korake izvršenja metode. Više o debugiranju možete saznati ovdje: www.sitepoint.com/debugging-with-truffle-cli. Budući da se svaki Solidity test uporabom Truffle okruženja izvršava u svom okruženju (engl. *clean-room*), nije lagano debugirati testove. Preporuka je da koristite Solidity sustav logiranja događaja (engl. *event*) koji Solidity omogućava uporabom `emit` naredbe. Emitirani događaji će biti ispisani nakon što Truffle pokrene testove, samo u slučaju da se nije dogodila greška. U slučaju da se greška dogodila, tj. pozvana je `revert()` funkcija, onda će biti ispisana samo poruka koja je predana funkciji `revert()`. Zato se preporuča da pri svakom pozivu funkcije koja baca grešku (`revert`, `require`, ...) predate smislenu poruku uz koju je lako doći do uzroka greške.

- Ethereum ne dopušta postavljanje jako velikih pametnih ugovora na blockchain. Da bi mogli pokrenuti testove (konkretno CrowdfundingTest) morate predati `-allowUnlimitedContractSize` zastavicu pri pokretanju lokalnog blockchaina. To je već pripremljeno u `start-local-node` skripti.

- Ako pri testiranju rješenja dobijete grešku oblika: " Error: sender doesn't have enough funds to send tx. The upfront cost is: 100000000000 and the sender's account only has: 98406187804". Vjerojatno je račun kojeg sustav za testiranje koristi prazan te nije moguće konstruirati novi testni ugovor. Problem možete riješiti tako da resetirate lokalni bloackchain čvor.

- Primjetite da u kodu postoji sučelje `Timer` (u datoteci `Timer.sol`) koje omogućuje dohvat trenutnog vremenskog trenutka (engl. *timestamp*). Komponenta koja pruža vrijeme je namjerno izdvojena kako bi omogućila testiranje neovisno o stvarnom vremenu. U slučaju da se u implementacijama pametnih ugovora koristi ključna riječ `now` za dohvat vremena, ne bi postojala

mogućnost kontroliranja vremena prilikom testiranja te bi pisanje istih bilo puno zahtjevnije. Za više informacija o testiranju s vremenom (ali primjenjivo na bilo koji drugi efekt na koji ne možemo utjecati prilikom testiranja) možete pročitati ovdje: softwareengineering.stackexchange.com/questions/235145/real-time-unit-testing-or-how-to-mock-now.