

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
RIJEKA

PAMETNI UGOVORI NA ETHEREUM PLATFORMI

profesor: Kristijan Lenac

Autor:

Mislav Tvrđinić

Sadržaj

1. Uvod.....	3
2. Zašto Ethereum?	3
3. Moj udžbenik	4
4. Alati.....	4
4.1 Truffle	4
4.2 Ganache.....	5
4.3 Solidity	5
4.4 NodeJS i NPM	5
5. Pametni ugovor	6
5.1. Funkcionalnost.....	6
5.2. Programski kod	6
6. Testiranje.....	11
6.1. Truffle & JavaScript	11
7. Implementacija i procjena troškova.....	15
7.1. Postavljanje ugovora na javnu mrežu	15
7.2. Procjena troškova.....	16
8. Zaključak.....	17

1. Uvod

Pametni ugovor je nadogradnja na koncept blockchaina, a omogućuje pohranjivanje programskog koda na mreži. U ovom slučaju, govorimo o *Ethereum* platformi koja služi razvijanju decentraliziranih, sigurnih aplikacija. Te aplikacije bazirane su na pametnim ugovorima, odnosno setu naredbi i pravila programiranih u programskom jeziku *Solidity*. Ponašaju se kao i obični ugovori, odnosno omogućuju razmjenu novca, vlasništva nad objektima neke vrijednosti i to potpuno transparentno. Do sada smo se kod takvih razmjena oslanjali na posrednike koji bi garantirali sigurnost te uživali povjerenje obje strane. Pametni ugovor služi upravo kao taj posrednik, tj. garantira uspješnu i sigurnu transakciju.

Primjena pametnih ugovora je velika, a u ovom projektu fokusirati ćemo se na plaćanje, odnosno razmjenu dobara. Kao što je već napomenuto, povjerenje između stranaka je najveći problem tradicionalnog poslovanja. Cilj projekta je kreirati *trustless* način poslovanja koristeći pametne ugovore na *Ethereum* blockchain mreži.

2. Zašto Ethereum?

Postoji više platformi za razvijanje decentraliziranih aplikacija, od kojih su dvije najpoznatije Ethereum i EOS. Kao i ostale blockchain tehnologije, obje se susreću s trilemom skalabilnosti. Tri osnovne karakteristike su: sigurnost, decentralizacija te skalabilnost. Većina tehnologija briljira u dvije od tri navedene karakteristike. Trenutno, Ethereum mreža može procesirati 15 transakcija po sekundi, što je očiti problem za skalabilnost, no postoji vrlo interesantno rješenje i plan koji bi trebali sustav učiniti višestruko bržim. S druge strane, EOS procesira transakcije na 21-om čvoru, što sustav čini relativno centraliziranim.

Programski jezik *Solidity* namijenjen pisanju pametnih ugovora je odlično dokumentiran i ima veliku potporu programera, što olakšava učenje i razvoj, dok se kod za EOS piše uglavnom u C++-u.

Ethereum platforma, razvojni tim i ideja su vrlo obećavajući pa čine savršen izbor za implementaciju pametnog ugovora.

3. Moj udžbenik

Moj udžbenik je web oglasnik polovnih školskih knjiga koji učenicima i njihovim roditeljima pruža jednostavnu i brzu kupovinu / prodaju potrebnih udžbenika. Korisnici mogu kreirati oglase te naručivati željene knjige. Kako bi ta funkcionalnost bila moguća, kupci moraju biti sigurni da će oglašivači uvijek isporučiti knjige. Taj posao mogao bi obavljati posrednik, odnosno neko tijelo unutar organizacije zaduženo za „provođenje pravde“.

Ukoliko želimo ostvariti povjerenje bez korištenja posrednika, moramo se okrenuti pametnim ugovorima. Knjiga je objekt koji ima neku vrijednost te postoji vlasnik koji je posjeduje. Naša transakcija obuhvaća razmjenu vlasništva u zamjenu za novac, što je moguće realizirati korištenjem pametnih ugovora.

4. Alati

Kako bismo razvili rješenje preko pametnih ugovora, moramo koristiti alate namijenjene programiranju istih na blockchainu. Također, klijentski dio naše aplikacije mora komunicirati s Ethereum platformom. Aplikacija je razvijena koristeći standardne web tehnologije: HTML, CSS, JS te PHP za komunikaciju s bazom podataka i NodeJS za neke osnovne upite prema blockchainu (npr. izračun cijene knjiga).

4.1 Truffle

Truffle je alat koji služi za prevođenje, testiranje i razvijanje pametnih ugovora napisanih u programskom jeziku Solidity na Ethereum blockchainu. Jedan je od najkorištenijih IDE-ova u Ethereum zajednici. Neke od značajki *Trufflea* su:

1. Automatsko testiranje ugovora
2. Ugrađeno prevođenje ugovora, povezivanje i razvijanje
3. Ugrađena kontrola binarnih datoteka
4. Generiranje novih ugovora i testova
5. Podrška za JavaScript, CoffeeScript, SASS, ES6 i neke druge
- ...

Alat je namijenjen programerima kojima je potreban razvojni okvir kako bi bolje organizirali decentraliziranu aplikaciju i jednostavno postavili testno okruženje.

4.2 Ganache

Ganache je osobni blockchain za razvijanje na Ethereum platformi, a omogućuje razvijanje pametnih ugovora, aplikacija i testiranje. Dostupan je kao desktop aplikacija te kao alat kroz komandnu liniju, starijeg naziva TestRPC.

4.3 Solidity

Solidity je objektno-orijentiran, viši programski jezik namijenjen implementaciji pametnih ugovora. Dizajniran je s namjenom za *Ethereum Virtual Machine (EVM)* a na razvoj su utjecali programski jezici C++, Python te JavaScript. Solidity je statically typed (potrebno definirati tipove varijabli), podržava nasljeđivanje, knjižnice te kompleksne tipove definirane od strane korisnika uz još brojne mogućnosti.

4.4 NodeJS i NPM

NodeJS je *open-source*, višeplatformsko JavaScript run-time okruženje koje može izvršavati JS kod izvan preglednika, a koristi se za *client-side scripting* gdje se kod ugrađuje u HTML web stranice te se pokreće JS engine-om korisnikova preglednika, te *server-side scripting* za pružanje dinamičnog web sadržaja prije nego što se web stranica pošalje korisniku. Ideja je korištenje JavaScripta kako na klijentskoj, tako i na poslužiteljskoj strani, odnosno *JavaScript everywhere* paradigma.

NPM je *packet manager* (Node upravitelj paketima) za JavaScript programski jezik. Sastoji se od komandne linije te online baze javnih i privatnih paketa, zvan *npm registry*. Registru se pristupa preko klijenta, a pakete je moguće pretraživati koristeći web stranicu npm-a.

5. Pametni ugovor

5.1. Funkcionalnost

Ugovor je namijenjen trgovini između dvije osobe, oglašivača i kupca. Kada osoba želi postaviti oglas za svoju knjigu, ugovor će biti postavljen na Ethereum mrežu. Prilikom postavljanja ugovora, potrebno je proslijediti dvije informacije: ID knjige (svaka knjiga namijenjena školskom obrazovanju ima svoj jedinstveni ID) te njezinu cijenu. Konstruktor ugovora će kreirati ugovor s tom cijenom te će svatko moći obaviti kupnju knjige (osim njega samoga).

Zadanu cijenu moguće je promijeniti pozivom adekvatne funkcije. Ona može biti pozvana samo od strane vlasnika ugovora, tj. oglašivača.

Kupovina knjige započinje procesom narudžbe pri čemu kupac poziva funkciju ugovora odnosno šalje potrebnu količinu *ethera* ugovoru. Ukoliko je transakcija uspješna, ugovor će zadržati primljeni *ether*, a status ugovora će se promijeniti. Ukoliko kupac želi otkazati narudžbu pozvati će funkciju istoimene operacije što će u suštini vratiti ugovor u početno stanje, tj. knjiga će ponovno biti na prodaju.

Ako narudžba nije otkazana knjiga će stići na odredište. U tom trenu kupac poziva funkciju za potvrdu dostave nakon čega se sredstva s ugovora šalju na račun bivšeg vlasnika knjige. Ugovor sa svim podacima ostaje na mreži te mu se može pristupiti potpuno transparentno.

5.2. Programski kod

Ovaj pametni ugovor namijenjen je oglašavanju knjiga, stoga je njegov naziv *BookAd*. Ugovor se odnosi na određenu knjigu – *bookId*, a definiraju ga cijena – *bookPrice* i osoba koja ga je postavila, odnosno *owner*. Jednom kada kupac odluči kupiti knjigu i provede transakciju, postati će *buyerAddr*, odnosno njegova adresa će biti pohranjena u varijablu. U tom trenutku ugovor mijenja svoje stanje jer je narudžba zaprimljena i potrebno je blokirati ostale potencijalne narudžbe. To je ostvareno varijablom *order* koja je pri inicijalizaciji ugovora postavljena na negativnu vrijednost. Varijable *sent* i *shipment* odnose se na slanje paketa.

```
// Important variables
uint public bookID;

uint private bookPrice;

address payable owner;

address payable buyerAddr;

// Is true if book was ordered
bool private order = false;

// Is true if owner sent the book
bool private sent = false;

// Is true if the buyer received the book
bool private shipment = false;
```

Slika 1: Varijable

Radi pojednostavljenja programskog koda koristim modifikatore koji postavljaju uvjete rada funkcija. Primjerice, funkcija *sendOrder* koristi modifikator *isActive* kako bi osigurali da se ista može pozvati jedino dok je ugovor u aktivnom stanju. Ostali modifikatori služe provjeri pozivatelja funkcija.

```
modifier isActive {
    require (!order);
    _;
}

modifier isOwner {
    require (owner == msg.sender);
    _;
}

modifier isBuyer {
    require (buyerAddr == msg.sender);
    _;
}
```

Slika 2: Modifikatori

Ugovor nastaje pozivanjem konstruktora. To je funkcija koja inicijalizira ugovor te u ovom slučaju prima dvije ulazne vrijednosti. Za kreiranje ugovora potrebno je predati ID knjige i njezinu cijenu. Naravno, unesene vrijednosti moraju biti veće od nule. Također, konstruktor postavlja adresu pozivatelja kao adresu vlasnika ugovora, odnosno knjige.

```
// The smart contract's constructor
constructor (uint _bookID, uint _bookPrice) public {

    // No empty values
    require(_bookID > 0);
    require(_bookPrice > 0);

    // The seller is the contract's owner
    owner = msg.sender;

    // Book information
    bookID = _bookID;

    // Price has to be set in Eth
    bookPrice = _bookPrice * 1e18;
}
```

Slika 3: Konstruktor

Kako bismo lakše pratili operacije nad ugovorom i omogućili dinamičko generiranje sučelja potrebno je definirati evente. Tijekom pozivanja funkcija i njihovog izvršavanja želimo zabilježiti neke događaje, npr. promjenu cijene. Stoga je definirano pet evenata. Argumenti (*uint*, *address*) su upravo vrijednosti koje želimo pohraniti u zapise (logs).

```
// Event triggered for new order
event OrderSent(address buyer, uint money);

// Event triggered for price change
event PriceUpdate(uint _price);

// Event triggered for shipment confirmation
event ShipmentConfirmed(uint _now);

// Event triggered when order gets canceled
event OrderCanceled(address _buyer, uint _now);

// Event triggered when book gets sent
event BookSent(address _owner);
```

Slika 4: Eventi

Kada kupac želi naručiti knjigu pozvati će funkciju *sendOrder*. Ona je tipa *payable* što znači da je moguće poslati odnosno primiti *ether*. Postoje dva tipa poziva funkcija, *call* i *transaction*. Transakcije mijenjaju stanje ugovora dok pozivi dohvaćaju podatke. Ovu funkciju zovemo kao transakciju pri čemu kupac šalje količinu *ethera* zadanu cijenom. Provjerava se je li uplaćen dovoljan iznos ili je kupac preplatio, pri čemu mu se vraća

višak sredstava. Adresa kupca sprema se u varijablu *buyerAddr* a stanje ugovora mijenja se u „naručen“ postavljanjem varijable *order* na istinitu vrijednost.

Na kraju se emitira event da je narudžba uspješno obavljena te *ether* čeka na adresi ugovora.

```
// The function to send purchase orders
// requires fee
// Payable functions returns just the transaction object, with no custom field.
// To get field values listen to OrderSent event.
function sendOrder() payable public isActive{

    // Owner not allowed
    require (owner != msg.sender);

    //Value sent to the contract has to match book price
    require (msg.value >= bookPrice, "Insufficient funds.");

    //Return ether if overpaid
    msg.sender.transfer(msg.value - bookPrice);

    // Buyer found
    buyerAddr = msg.sender;
    order = true;

    // Trigger the event
    emit OrderSent(msg.sender, msg.value);

}
```

Slika 5: Funkcija za narudžbu

Nadalje, *cancelOrder* je funkcija kojom kupac može poništiti narudžbu. Jedino je on može pozvati što je definirano korištenjem modifikatora *isBuyer*. Tijekom izvršavanja funkcije provjerava se stanje isporuke, tj. narudžbu knjige koja je u procesu dostave nije moguće poništiti. Ova funkcija se prvenstveno koristi u slučaju kada oglašivač duže vrijeme ne isporučuje knjigu, kako sredstva ne bi ostala zamrznuta. Novac se vraća kupcu, a stanje narudžbe se resetira te je knjigu ponovno moguće naručiti.

```

// The function to cancel ongoing order
// Funds get returned to buyerAddr
function cancelOrder () payable public isBuyer {

    // Order can not be canceled if the book has already been sent
    require(!sent);

    // Returning money to the buyer
    buyerAddr.transfer(bookPrice);

    // Resetting contract
    order = false;

    // Trigger the event
    emit OrderCanceled(msg.sender, now);

}

```

Slika 6: Funkcija za otkazivanje narudžbe

Kada vlasnik knjige obavi isporuku pozvati će *bookSent* funkciju. Jednom kada je uspješno pozvana blokirati će se otkazivanje narudžbe. Kada knjiga stigne na destinaciju kupac poziva funkciju *confirmShipment* pomoću koje potvrđuje uspješnost dostave. Tim pozivom sredstva sjedaju na račun bivšeg vlasnika knjige.

```

// The function to confirm shipping
// Once confirmed, funds get transfered - payable
// Only buyer can confirm it, cannot be called before ordering
function confirmShipment() payable public isBuyer {

    // Update the bool
    shipment = true;

    // Funds get transfered to ex owner of the book
    owner.transfer(bookPrice);

    // Trigger the event
    emit ShipmentConfirmed(now);

}

// Say that book has been shipped
// Only owner can call it, but not before order
function bookSent () public isOwner {

    // Is there an order?
    require(order);

    // Update the bool
    sent = true;

    // Trigger the event
    emit BookSent(msg.sender);

}

```

Slika 7: Funkcije za potvrđivanje

U ugovoru su prisutne još neke funkcije koje pružaju korisne informacije. Definirani su *getteri* (funkcije za dohvaćanje informacija) za cijenu oglasa i status, što je korisno kod filtriranja aktivnih oglasa i njihovo prikazivanje na web stranici. Uz to, postoji opcija za promjenu cijene oglasa koju vlasnik može pozvati i promijeniti cijenu ukoliko smatra da je to potrebno.

Kompletna programski kod moguće je vidjeti na *github* repozitoriju.

6. Testiranje

Za testiranje pametnog ugovora koristim ranije spomenuti *Truffle Framework*. Truffle je odličan alat za testiranje koda napisanog u *Solidity-u* jer omogućava migracije, tj. olakšava postavljanje ugovora na mrežu jer se prilagođava potrebama tijekom razvoja.

Ganache je lokalni blockchain s kojim *Truffle* komunicira. Jednom kada se pokrene, aplikacija sluša određeni port za interakciju s mrežom te generira početne adrese i privatne ključeve kako bi mogli pozivati funkcije ugovora i obavljati transakcije.

6.1. Truffle & JavaScript

Postavljanje radnog okruženja je sastavljeno od par koraka:






Instalacija

```
npm install -g truffle
```

Generiranje datoteka

```
truffle init (ili) truffle unbox MetaCoin
```

MetaCoin je jednostavan primjer pametnog ugovora i adekvatnih testova, a pomaže kod razumijevanja organizacije direktorija i potrebnih datoteka.

 contracts	19.3.2019. 11:46	File folder	
 migrations	18.3.2019. 20:37	File folder	
 test	20.3.2019. 17:27	File folder	
 LICENSE	18.3.2019. 20:37	File	2 KB
 truffle-config.js	19.3.2019. 17:28	JavaScript File	1 KB

Slika 8: Struktura direktorija

Direktorij *contracts* sadrži pametne ugovore, *migrations* sadrži datoteke napisane u JavaScriptu kojima su definirane migracije po koracima. Kako bi koristili mogućnost migracija u mapi *contracts* nalazi se ugovor naziva *Migrations.sol* koji je nastao pozivom naredbe *truffle init*. Potrebno je kodirati migraciju našeg ugovora. Migracije su numerirane kako bi se znao redoslijed njihova izvršavanja. Inicijalnu migraciju već imamo pa je potrebno kodirati postavljanje našeg pametnog ugovora, datoteka *2_deploy_contracts.js*.

```
const BookAd = artifacts.require("BookAd");

module.exports = function(deployer) {
  deployer.deploy(BookAd, 2121, 10);
};
```

Slika 9: Migracija ugovora

Migracije pozivamo naredbom:

```
truffle migrate
```

Nadalje, direktorij *test* sadrži JavaScript datoteke u kojima su napisani koraci za testiranje funkcionalnosti pametnog ugovora. Datoteka *book-ad.js* sadrži neke od tih provjera. Na slici je vidljiv primjer testa u JS-u. U ovom slučaju provjerava se stanje varijable nakon narudžbe knjige. Kod *assert.equal(status, true)* zahtjeva da je stanje varijable uistinu *true* te će u tom slučaju test proći.

```
// Status should've changed
it("should return status as true", async () => {
  let instance = await BookAd.deployed();
  let status = await instance.getStatus.call();

  assert.equal(status, true);
})
```

Slika 10: Primjer testa

Primjerice, datoteka *book-ad-cancel.js* testira otkazivanje narudžbe. U ovom testu provjeravamo stanje na računima prije i nakon izvršavanja pametnog ugovora, stanje varijabli tijekom procesa i ispravnost pozivanja funkcija.

Kod *await instance.sendOrder(...)* zahtjeva uspješan poziv te ukoliko se on ne realizira *Truffle* će nam to pokazati.

```
const BookAd = artifacts.require("BookAd");

contract("Testing BookAd Cancelling - 10eth", async accounts => {

  it("should cancel order correctly", async () => {
    let instance = await BookAd.deployed();

    // Starting balances
    let ownerBalance = await web3.eth.getBalance(accounts[0]);
    let buyerBalance = await web3.eth.getBalance(accounts[1]);

    // Price of the book in ether
    let price = await instance.getPrice.call();
    price = web3.utils.fromWei(price);

    // Status should be false, no order yet
    let status = await instance.getStatus.call();
    assert.equal(status, false);

    // Sending order
    await instance.sendOrder({from: accounts[1], value: 1e19});
  });
});
```

Slika 11: Isječak testa

Uz datoteke *book-ad.js* i *book-ad-cancel.js* u direktoriju *test* nalazi se i datoteka *book-ad-buy.js* kojom se provjerava kompletna kupovina i računi. *Truffle* pruža čisto okruženje kod pokretanja različitih testova, tj. kod interakcije s *Ganache* blockchainom osigurava da datoteke ne dijele informacije međusobno kako ne bi došlo do pogrešaka.

Kako bismo započeli testiranje potrebno je pokrenuti *Ganache*:

```
ganache-cli -p 7545
```

```
Windows PowerShell
Block Time: Fri Mar 22 2019 01:22:47 GMT+0100 (GMT+01:00)

eth_getTransactionReceipt
PS C:\Users\Mislav\Desktop\Smart Contract Payment\eth-books> ganache-cli
-p 7545
Ganache CLI v6.4.1 (ganache-core: 2.5.3)

Available Accounts
=====
(0) 0xc445258f0757186b4053a1f9126ea99b1f3eb712 (~100 ETH)
(1) 0xb75d853dc143d050463141c55297ebbcae3e4b94 (~100 ETH)
(2) 0x61b90085b1071737e82455da5a328cc5aa3f73fc (~100 ETH)
(3) 0xb2f3aaa5e2d53c22c518c42f5ae9ab232030170e (~100 ETH)
(4) 0x5cae6643c66e5023f54bb5d4637bb408fb20118c (~100 ETH)
(5) 0x2c2b8ebc8db8b6f4660442bac022b5e0be402e99 (~100 ETH)
(6) 0xb664217ef9dcd2d2dd472d48ed6796c00163a717 (~100 ETH)
(7) 0x3e26ec8363678a61ebc6234fe0ceb65d013eca67 (~100 ETH)
(8) 0x00e3923f9512e864114af2fa436f548072a17661 (~100 ETH)
(9) 0x6d35112c45618528d1c8a1abfe28ea565a744ec1 (~100 ETH)

Private Keys
=====
(0) 0x11cc0f8b3a71ac9688bc9fb0f4de1be51f5ba96d845fe32de96356bd86b60610
(1) 0x5bd402f7cc0d59e4c4e038d9b6bd056ee266aafb97f71542ebd663f30f6a6eef
(2) 0x1604b634f7bd94b793cbb7a9543fd458e913aa48547b017888c0dc6dd2f4d657
(3) 0x8c24932b85e0212f55bb77503a3c7315a449fd467799ee4e8269ee12566344
(4) 0x2e15e0d361c2a4d10b887ed9643be3ef607f72d471655567049d49a41f5bdcec
(5) 0x02d910d652dce2d093628bbb6d63032ebc67c09713e293295687f86ac6991842
(6) 0xcdf188bc3e38dabdbfbc9d070e9b93c9ebc61ba5cc1c3cfa4f0ba8013c5f08146
(7) 0xd2f92c171726dd5be4bbd04c72e078f028373c1230d37c5226600afef215cbcb
>

HD Wallet
=====
Mnemonic:      advance airport pass silver cook fantasy rice grace recycle domain stairs tria
Base HD Path:  m/44'/60'/0'/0/{account_index}

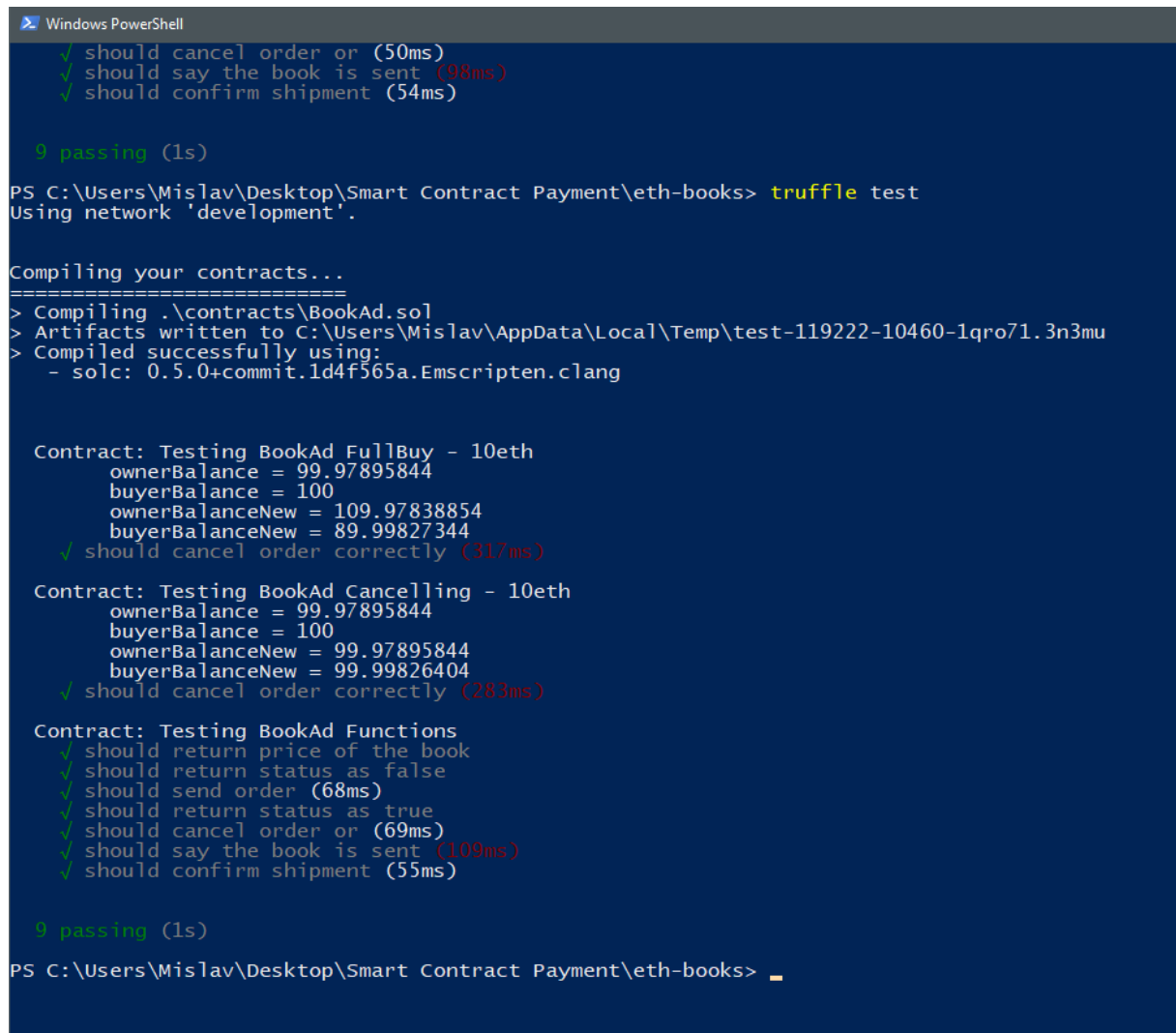
Gas Price
> -
```

Slika 12: Ganache

Test pokrećemo naredbom:

```
truffle test
```

Nakon prevođenja koda ugovora započinje testiranje. Na slici vidimo rezultate sva tri testa. Također, kod prvog i drugog testa vidljivo je stanje na računima što nam govori da se sredstva uspješno prenesu između računa, odnosno vrate kupcu u drugom slučaju.



```
Windows PowerShell
✓ should cancel order or (50ms)
✓ should say the book is sent (98ms)
✓ should confirm shipment (54ms)

9 passing (1s)

PS C:\Users\Mislav\Desktop\Smart Contract Payment\eth-books> truffle test
Using network 'development'.

Compiling your contracts...
=====
> Compiling .\contracts\BookAd.sol
> Artifacts written to C:\Users\Mislav\AppData\Local\Temp\test-119222-10460-1qro71.3n3mu
> Compiled successfully using:
  - solc: 0.5.0+commit.1d4f565a.Emscripten.clang

Contract: Testing BookAd FullBuy - 10eth
  ownerBalance = 99.97895844
  buyerBalance = 100
  ownerBalanceNew = 109.97838854
  buyerBalanceNew = 89.99827344
  ✓ should cancel order correctly (317ms)

Contract: Testing BookAd Cancelling - 10eth
  ownerBalance = 99.97895844
  buyerBalance = 100
  ownerBalanceNew = 99.97895844
  buyerBalanceNew = 99.99826404
  ✓ should cancel order correctly (283ms)

Contract: Testing BookAd Functions
  ✓ should return price of the book
  ✓ should return status as false
  ✓ should send order (68ms)
  ✓ should return status as true
  ✓ should cancel order or (69ms)
  ✓ should say the book is sent (109ms)
  ✓ should confirm shipment (55ms)

9 passing (1s)

PS C:\Users\Mislav\Desktop\Smart Contract Payment\eth-books> _
```

Slika 13: Truffle test output

7. Implementacija i procjena troškova

7.1. Postavljanje ugovora na javnu mrežu

Testiranje smo proveli na lokalnom blockchainu, a sada smo spremni postaviti ugovor na glavnu mrežu kako bi ga korisnici mogli koristiti. To je moguće ostvariti korištenjem Truffle-a. Alat je originalno postavljen da funkcionira u suradnji s lokalnim blockchainom, kao što smo do sada koristili. Kako bi omogućili postavljanje ugovora na glavnu mrežu, potrebno je ažurirati postavke Truffle-a.

Također, ovaj proces zahtjeva konekciju s *live* mrežom, što je moguće korištenjem jednog od više Ethereum klijenata. Dva najpopularnija klijenta su *go-ethereum* i *cpp-ethereum*, nazvani prema programskim jezicima u kojima su napisani. Jednom postavljeni klijent mora biti u potpunosti sinkroniziran s mrežom. Klijent mora imati barem jednu registriranu adresu te autorizaciju za slanje transakcija. Postavljanje ugovora nije besplatno, stoga je potrebna određena količina *ethera* na računu.

Alternativa postavljanju klijenta je korištenje Infura-e. To je gomila Ethereum čvorova koja nudi sigurnu, pouzdanu i skalabilnu interakciju s mrežom. Infura omogućuje korištenje Ethereum API-a, za koje se dobije određeni projektni identifikator, što čini korištenje sustava znatno jednostavnijim. Mnogi drugi alati poput MetaMask-a za interakciju s decentraliziranim aplikacijama direktno iz web preglednika oslanjaju se na Infura-u.

Zadana Truffle konfiguracija izgleda ovako:

```
module.exports = {
  rpc: {
    host: "127.0.0.1",
    port: 8545
  }
};
```

Kako bismo osigurali da Truffle zna na koju mrežu želimo postaviti pametni ugovor, moramo nadodati posebnu konfiguraciju za glavnu mrežu:

```
module.exports = {
  networks: {
    "live": {
      network_id: 1,
      host: "127.0.0.1",
      port: 8546 // Different than the default below
    }
  },
  rpc: {
    host: "127.0.0.1",
    port: 8545
  }
};
```

Postavili smo novu vrijednost za *port* kako bi onemogućili potencijalne greške s postavljanjem na druge mreže. Također, vrijednost *network_id* postavili smo na 1, što je oznaka da se naša konfiguracija odnosi na glavnu mrežu, jer ona jedina ima ID mreže 1. Privatne i testne mreže koriste druge identifikacijske brojeve.

Nakon postavljanja konfiguracije slijedi postavljanje ugovora na mrežu. Prilikom ove operacije potrebno je navesti na koju mrežu želimo postaviti ugovor, inače će se koristiti zadana konfiguracija. Unutar Truffle-a pokrećemo naredbu:

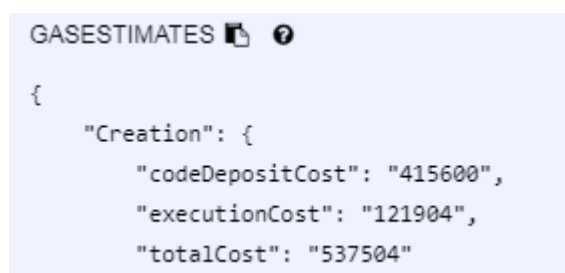
```
$ truffle migrate --network live
```

S završetkom operacije naš ugovor biti će dostupan na određenoj adresi (vidljivo unutar naredbenog retka).

7.2. Procjena troškova

Postavljanje ugovora na mrežu nije besplatno, jer zahtjeva procesorsku snagu rudara čiji se posao mora isplatiti. Transakcije i izvršavanje koda na Ethereum mreži plaćaju se resursom naziva *gas*. Slično automobilskom svijetu, gdje motori zahtijevaju gorivo, ovdje se zahtjeva digitalni oblik goriva. Cijena gasa nije konstantna, već varira o raznim parametrima, poput zakrčenosti mreže. Unatoč tome, postavljanje ugovora na mrežu iziskuje uvijek istu količinu gasa, a računa se prema formuli: minimalan trošak od 32 000 + 200 gasa po bajtu izvornog koda.

Web urednik naziva *Remix* (razvijen od strane *Ethereum Foundation*) koji služi pisanju pametnih ugovora u Solidity-u pruža razne korisne informacije o ugovoru pa tako i troškove. U ovom slučaju, postavljanje ugovora na mrežu koštalo bi ukupno 537 504 gasa.



```
GASESTIMATES ⓘ ⓘ  
  
{  
  "Creation": {  
    "codeDepositCost": "415600",  
    "executionCost": "121904",  
    "totalCost": "537504"
```

Slika 14: Aproksimacija troškova postavljanja ugovora

Ovaj broj moguće je preračunati u kunske troškove. Cijena gasa računa se u jedinici Gwei (Giga-Wei), koja predstavlja milijarditi dio jednog ethera. Prema kalkulacijama web alata za informacije o gasu naziva *ethgasstation*, cijena postavljanja ovog ugovora na mrežu iznosi 0.00473 ethera, što se trenutno ugrubo preslikava u 5 kuna. Također, funkcije ugovora koje mijenjaju stanje blockchaina nisu besplatne, primjerice trošak funkcije kojom se postavlja narudžba iznosi 30816 gasa (30-ak lipa).

Prema tome, vidimo da pametni ugovori nisu besplatni te da njihova kompleksnost i dizajn mogu stvarati ozbiljne troškove. U ovom primjeru, dizajn ugovora mogli bismo izmijeniti na način da ugovor sadrži polje oglasa umjesto samo jednog, kako bi potencijalno uštedjeli novac na postavljanju pojedinačnih ugovora na mrežu. Također, kod programiranja pametnih ugovora treba uzeti u obzir rad s memorijom i složenost pojedinih operacija kako bi minimizirali utrošak gasa.

8. Zaključak

Nakon uspješnog programiranja pametnog ugovora i testiranja moguće je napraviti klijentski dio u obliku web stranice, u čijoj se pozadini vrti ugovor. To je moguće ostvariti interakcijom s blockchainom pomoću *Web3* knjižnice modula koji sadrže specifične funkcionalnosti za komunikaciju s mrežom. Takvu aplikaciju mogli bismo nazvati decentraliziranom jer bi se operacije nad ugovorom i njegove funkcije izvršavale na Ethereum čvorovima, a ne na našem serveru niti klijentskom računalu.

Na primjeru web oglasnika polovnih knjiga riješili smo problem koordinacije kupovine i isporuke te opterećenje izvršavanja procesorskih operacija skrenuli na Ethereum mrežu i njezine čvorove. Također, povijest svih transakcija i operacija nad ugovorom javno je dostupna i nepromjenjiva, što čini sustav sigurnim i povjerljivim. Ovaj projekt pokazuje da decentralizirane aplikacije imaju široku primjenu te su zbog svoje transparentnosti i sigurnosti zanimljive krajnjim korisnicima, a i programerima.

Unatoč tome, analizom troškova utvrdili smo da blockchain nije sasvim besplatna i savršena tehnologija kojom ćemo riješiti sve probleme, već da postoje određene prepreke s kojima se suočava.