**SVEUČILIŠTE U SPLITU**

**FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE**

**ZAVRŠNI RAD**

**DECENTRALIZIRANE APLIKACIJE   
NA ETHEREUM BLOCKCHAIN-U**

**Ivo Kovačević**

**Split, srpanj 2018**

SVEUČILIŠTE U SPLITU

FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

Preddiplomski studij: **Računarstvo**

Oznaka programa: 120

Ime i prezime: **Kovačević Ivo**

Broj indeksa: 185-2015

**PRIJAVA ZAVRŠNOG RADA**

Radni naslov: Decentralizirane aplikacije na Ethereum blockchain-u

Zadatak: Potrebno je osmisliti i izraditi decentraliziranu aplikaciju za glasovanje. Aplikaciju izraditi u programskom jeziku Solidity. Izraditi potrebno korisničko sučelje. Objasniti prednosti i mane decentraliziranih aplikacija te njihovu moguću primjenu.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Mentor: izv. prof. dr. sc. Damir Sedlar

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Predsjednik Odbora: prof. dr. sc. Maja Štula

Split, 02.03.2018.

SADRŽAJ

[1 UVOD 1](#_Toc514610238)

[2 ŠTO SU DECENTRALIZIRANE APLIKACIJE? 2](#_Toc514610239)

[2.1 Mane centraliziranih sustava/aplikacija 2](#_Toc514610240)

[3 „BLOCKCHAIN“ – LANAC BLOKOVA 4](#_Toc514610241)

[3.1 Blok 4](#_Toc514610242)

[3.1.1 Podaci 4](#_Toc514610243)

[3.1.2 Identifikator bloka 5](#_Toc514610244)

[3.1.3 Identifikator prethodnog bloka 6](#_Toc514610245)

[3.2 Karakteristike i mehanizmi lanca blokova 8](#_Toc514610246)

[3.2.1 Dokaz o radu (engl. Proof of work) 8](#_Toc514610247)

[3.2.2 Distribuiranost 9](#_Toc514610248)

[3.2.3 Decentraliziranost 10](#_Toc514610249)

[3.3 Kako ustvari radi lanac blokova? 11](#_Toc514610250)

[4 DECENTRALIZIRANE APLIKACIJE 18](#_Toc514610251)

[4.1 Pametni ugovori 18](#_Toc514610252)

[4.1.1 Općenito o pametnim ugovorima 18](#_Toc514610253)

[4.1.2 Primjena pametnih ugovora u decentraliziranoj aplikaciji 20](#_Toc514610254)

[4.2 Korisničko sučelje 22](#_Toc514610255)

[4.3 Pametni ugovori u praksi – jesu li uistinu toliko sigurni? 24](#_Toc514610256)

[5 ZAKLJUČAK 26](#_Toc514610257)

[6 POPIS OZNAKA I KRATICA 27](#_Toc514610258)

[7 LITERATURA 28](#_Toc514610259)

[8 DODATAK A 29](#_Toc514610260)

# UVOD

Korištenjem centraliziranih aplikacija nailazi se na razne probleme kao što su skalabilnost, sigurnost, preopterećenje centralnog sustava, a ispad centralnog sustava znači i prekid rada aplikacije. Ovi problemi se rješavaju primjenom decentraliziranih aplikacija. Ove aplikacije su pohranjene na velikom broju mrežnih čvorova te ukoliko jedan čvor prestane s radom to neće utjecati na rad aplikacije. Sigurnost je također puno veća upravo zbog toga što jednom napisanu aplikaciju više nitko ne može modificirati pa čak ni sam autor aplikacije. To znači da se mogu programirati vrlo povjerljive aplikacije koje će onemogućiti prevaru jer se jednom napisani i dogovoreni kod više ne može mijenjati.

Decentralizirane aplikacije koje danas poznajemo, temelje se na tehnologiji lanca blokova. Lanac blokova se sastoji od puno povezanih blokova od kojih svaki sadrži podatke. Podaci mogu biti transakcije ili pak aplikacije pisane u programskom jeziku Solidity koje se potom kompajliraju i pohrane u blok.

U drugom poglavlju ovog rada su upravo dane mane centraliziranih i prednosti decentraliziranih aplikacija i sustava. Osnovni pojmovi lanca blokova i njegove sastavnice objašnjavaju se u trećem poglavlju rada. Također, u ovom poglavlju objašnjavanja se i sigurnosni mehanizam koji čini lanac blokova toliko sigurnim. Četvrto poglavlje daje prikaz decentraliziranih aplikacija, kreiranja pametnih ugovora, programskih jezika te se ispituje sigurnost decentraliziranih aplikacija. U petom poglavlju dani su odgovarajući zaključci koji se odnose na primjenu decentraliziranih sustava, dok je kod decentralizirane aplikacije za glasovanje dan u dodatku na kraju rada.

# ŠTO SU DECENTRALIZIRANE APLIKACIJE?

Danas najpoznatije i najrasprostranjenije aplikacije su centralizirane aplikacije. Postoji jedan centralni čvor o kojem ovisi rad kompletnog sustava. Centralni čvor upravlja svim drugim čvorovima te svi ostali čvorovi zavise o centralnom čvoru. Kod distribuiranih sustava više čvorova rade zajedno kako bi ostvarili neku zadaću. Dakle nema centralnog čvora koji obavlja svu zadaću, nego je posao podijeljen na više čvorova. U decentraliziranim sustavima posao se obavlja na svim čvorovima te je svaki čvor neovisan o drugim čvorovima [1]. Prestanak rada jednog čvora ne znači i pad cijelog sustava. Slika 2-1 prikazuje razliku između centraliziranih, decentraliziranih i distribuiranih sustava.

a) centralizirani

b) decentralizirani

c) distribuirani

Slika 2‑1 Razlika između: a) centraliziranih, b) decentraliziranih i c) distribuiranih sustava

## Mane centraliziranih sustava/aplikacija

Glavna mana centraliziranih sustava je što rad sustava ovisi o jednom jedinom čvoru. Prestanak rada jednog centralnog čvora rezultira padom kompletnog sustava. To može biti veoma loše ukoliko se radi o aplikacijama kod kojih i sekunda prestanka rada znači ogromne materijalne gubitke.

Budući da o jednom čvoru ovisi rad cijelog sustava, preko toga čvora idu svi zahtjevi te se sav posao obavlja na tom čvoru. To znači da centralni čvor mora biti veoma moćno računalo koje će moći obaviti tako zahtjevan posao pouzdano, bez zastoja i grešaka. Ovakva organizacija sustava je izvediva kod jednostavnih i srednje složenih sustava. Kod složenih sustava poput Facebook-a, Google-a, Amazona itd. ovakva arhitektura ne bi donijela željene rezultate, stoga se ti sustavi distribuiraju.

Drugi veliki nedostatak ovakvih sustava je ta što jedna korporacija kontrolira ogromnu količinu podataka, primjerice Facebook. Prilikom registracije na Facebook u zamjenu za besplatan servis, dijele se podaci korisnika. Korisnik nema uvid što se sve radi s tim podacima što otvara mogućnosti zlouporabe tih podataka. Uzmimo za primjer Uber. Interni sustav Uber-a omogućuje praćenje svih vožnji u trenutnom vremenu na nekom području. Na temelju tih podataka može se zaključiti npr. tko će prenoćiti negdje samo jednu noć i ostale veoma povjerljive informacije.

Korisnici ukoliko žele podijeliti neke podatke na Internet moraju platiti određenoj „host“ organizaciji. Dakle, korisnici plaćaju dijeljenje vlastitih podataka raznim „host“ kompanijama koje potom imaju kontrolu nad tim podacima što im daje ogromnu moć.

Sve ove spomenute probleme rješavaju decentralizirane aplikacije koje predstavljaju novu generaciju aplikacija. Iako su još u ranim začecima, decentralizirane aplikacije su veoma dobro prihvaćene zbog njihove velike sigurnosti, decentraliziranosti, ali i same ideje i nametanje drugačijeg načina razmišljanja na koji do sada nismo navikli.

# „BLOCKCHAIN“ – LANAC BLOKOVA

„Blockchain“ nije ništa drugo nego lanac blokova koji sadrže informacije. Blokovi su povezani tako da svaki blok sadrži vlastiti identifikator te identifikator prethodnog bloka (Slika 3‑1).

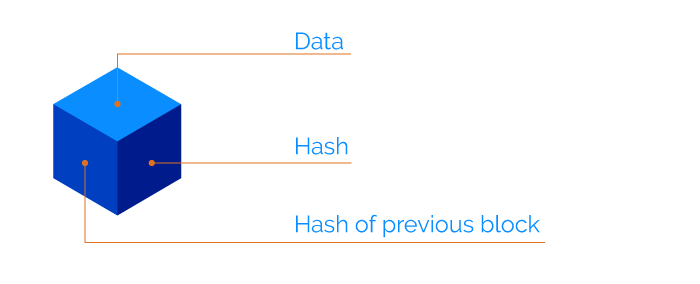


Slika ‑ Primjer lanca blokova

Lanac blokova možemo zamisliti kao već dobro poznate liste. Svaki član liste sadrži pokazivač na sljedećeg člana liste i neke podatke.

## Blok

Svaki blok unutar lanca se sastoji od podataka (engl. Data), vlastitog identifikatora (engl. Hash) te identifikatora prethodnog bloka (engl. Hash of previous block) kao što je prikazano na slici (Slika 3‑2).



Slika ‑ od čega se sve sastoji bok

### Podaci

Polje podaci mogu sadržavati proizvoljne podatke i ovise o vrsti lanca blokova. Npr. kod Bitcoina polje podaci sadrži informacije o transakciji:

* Od koga? (engl. From) – označava pošiljatelja koji šalje određeni iznos.
* Kome? (engl. To) – označava primatelja koji prima taj isti iznos.
* Koliko? (engl. Amount) – odnosi se na iznos koji se šalje (npr. pošiljatelj šalje 2 BTC primatelju).

Kod Ethereum lanca moguće je u polje podaci pohranjivati i nešto što se zove pametni ugovor (engl. Smart contract). Upravo na tim pametnim ugovorima se temelje decentralizirane aplikacije koje su tema ovog rada. Pametni ugovori će detaljnije biti objašnjeni u poglavlju 4.1.

### Identifikator bloka

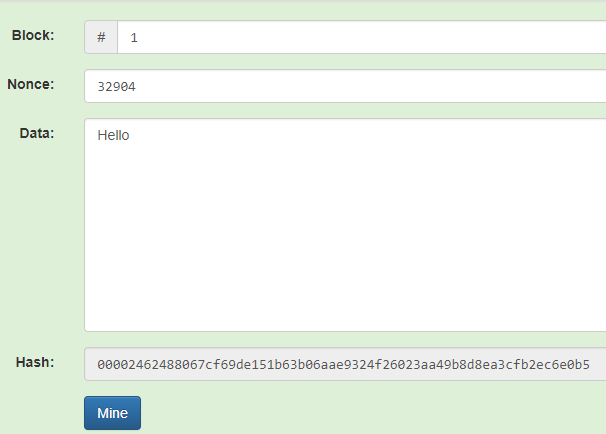
Identifikator bloka (engl. Hash) se može zamisliti kao otisak prsta koji jedinstveno određuje svaki blok. Dakle, ne postoje dva bloka s istim identifikatorom. Identifikator se generira na osnovu funkcije koja uzima određene parametre vezane za taj blok te vraća jedinstveni identifikator koji se potom dodjeljuje tom bloku.

f (razni parametri) 🡪 jedinstveni identifikator.

Parametri ovisno o vrsti lanca mogu biti:

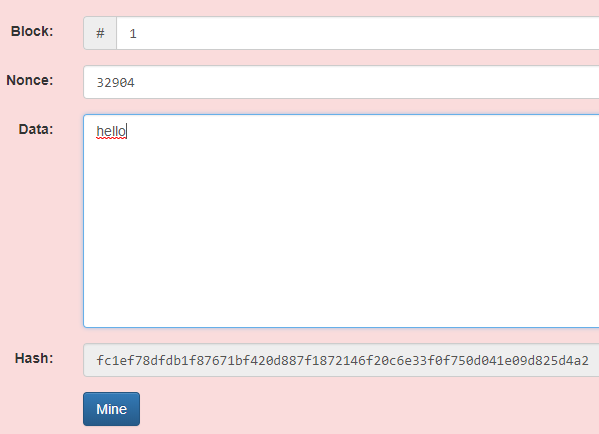
* Podaci bloka
* Identifikator prethodnog bloka
* Trenutno vrijeme u sekundama (engl. Timestamp)
* Broj koji je potrebno pogoditi da identifikator započinje određenim brojem nula (engl. Nonce)

Na slici (Slika 3‑3) vidimo jednostavan primjer određivanja identifikatora bloka na osnovu rednog broja bloka, broja koji je potrebno pogoditi da bi identifikator započinjao s određenim brojem nula (engl. Nonce) i podatka kojeg unosima (engl. Data).



Slika ‑ Određivanje identifikatora bloka na osnovu podataka u bloku

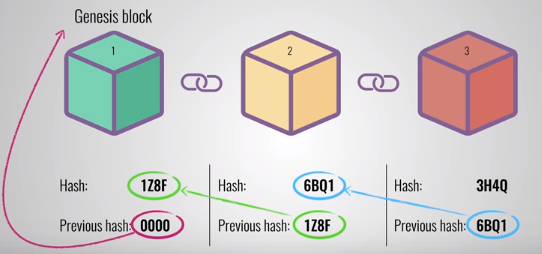
Ukoliko samo malo promijenimo neki od podataka na osnovu kojih se generira identifikator (engl. Hash), identifikator će se skroz promijeniti te on više neće biti ispravan (Slika 3‑4). Sada je potrebno opet pogoditi identifikator da počinje s određenim brojem nula. Ovaj mehanizam se naziva dokaz o radu (engl. Proof of work).



Slika ‑ Identifikator bloka se promijenio prilikom promjene podatka unutar bloka

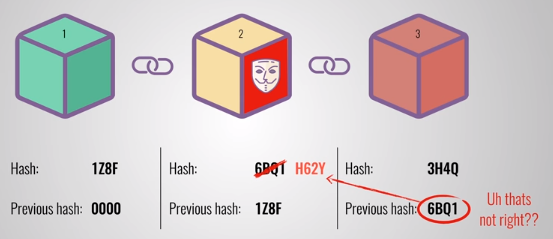
### Identifikator prethodnog bloka

Svaki blok (osim prvog bloka u lancu) sadrži identifikator (engl. Hash) prethodnog bloka (Slika 3‑5). Na taj se način blokovi povezuju u lanac blokova (engl. Blockchain).



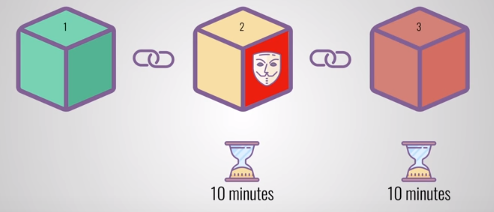
Slika ‑ Primjer povezanosti blokova

Ukoliko se promjeni bilo koji podatak unutar bloka, promijenit će se i identifikator bloka. Posljedica toga bit će puknuće lanca blokova jer slijedeći blok više ne sadrži ispravan identifikator prethodnog bloka (Slika 3‑6). Na slici vidimo da blok 3 više ne sadrži ispravan identifikator bloka 2 jer se bloku 2 promijenio identifikator zbog mijenjanja nekih podataka unutar bloka tj. došlo je do pucanja lanca.



Slika ‑Primjer puknuća lanca blokova

Da bi se riješio ovaj problem i skup blokova opet povezao u neprekidni lanac potrebno je ponovno proračunati identifikator svakog bloka počevši od mijenjanog pa nadalje. Moderna računala mogu proračunati oko 100 000 identifikatora po sekundi (engl. Hash). Da bi se usporilo nastajanje novih blokova, postoji nešto što se zove dokaz o radu (engl. Proof of work) što će biti detaljnije objašnjeno u poglavlju 3.2.1. Kod Bitcoina je potrebno oko 10 minuta za stvoriti novi blok (Slika 3‑7).



Slika ‑Vrijeme potrebno za nastanak novog bloka

## Karakteristike i mehanizmi lanca blokova

### Dokaz o radu (engl. Proof of work)

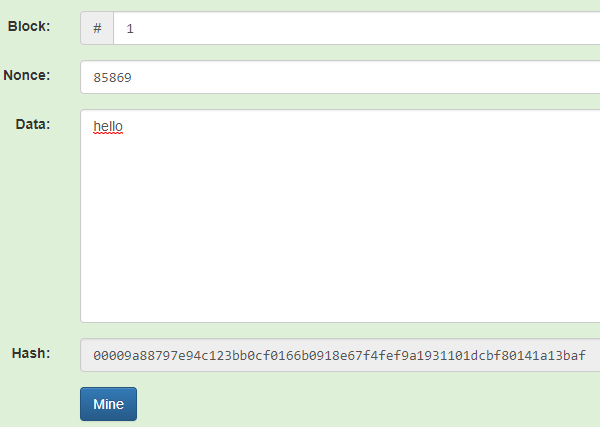
Dokaz o radu je mehanizam koji usporava nastanak novih blokova. Razlog tome je sprječavanje raznih napada na mrežu kao što su spamanje, lažne transakcije, mijenjanje sadržaja pojedinog bloka itd. što se postiže zahtijevanjem određene količine računalnog posla koje uzima određeno vrijeme.

Uzmimo za primjer da imamo blok koji sadrži transakciju gdje osoba X šalje osobi Y 20 kuna. Da ne postoji dokaz o radu, osoba Y bi mogla promijeniti iznos u bloku tako da umjesto 20 kuna napiše 200 kuna jer bi tada vrlo brzo mogla ponovno proračunati identifikator (engl. Hash) toga bloka i svih slijedećih blokova te bi lanac bio konzistentan. Upravo zbog toga što postoji dokaz o radu, ovakva mijenjanja sadržaja bloka su onemogućena jer bi trebala enormna količina računale snage da ponovno proračuna identifikator tolike količine blokova.

Dokaz o radu nam daje određena pravila i ograničenja pri određivanju identifikatora bloka. Identifikator bloka je ispravan jedino i samo ako počinje s određenim brojem nula. Broj nula s kojim identifikator mora započinjati se mijenja s vremenom kako bi se postiglo da za stvaranje svakog bloka treba otprilike 10 minuta (kod Bitcoin-a).

Na slici (Slika 3‑8) prikazan je pronalazak ispravnog identifikatora bloka što rješava problem iz poglavlja 3.1.2 (Slika 3‑4). Vidimo da identifikator započinje s određenim brojem nula (u primjeru radi jednostavnosti uzeto je da je taj broj 4) te se identifikator smatra ispravnim.

Možemo uočiti i da se promijenilo polje „Nonce“. Taj je broj trebalo pogoditi da dobijemo identifikator sa slike koji počinje s 4 nule.

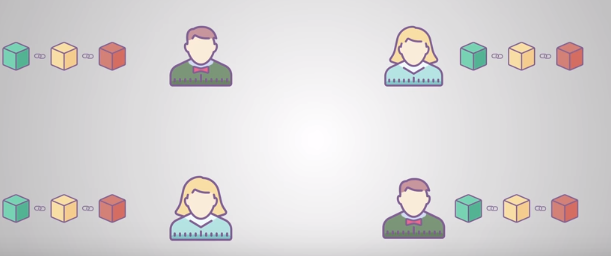


Slika ‑ Pronalazak ispravnog identifikator bloka nakon mijenjanja podataka u bloku

### Distribuiranost

Lanci blokova imaju svojstvo distribuiranosti tj. svima je omogućeno sudjelovanje koristeći P2P mrežu.

Upravo svojstvom distribuiranosti postiže se još jedan nivo sigurnosti ovakvog sustava. Razlog je u tome što svaki sudionik na mreži može nadgledati transakcije. Svi sudionici na mreži imaju vlastitu kopiju lanca blokova (Slika 3‑9).

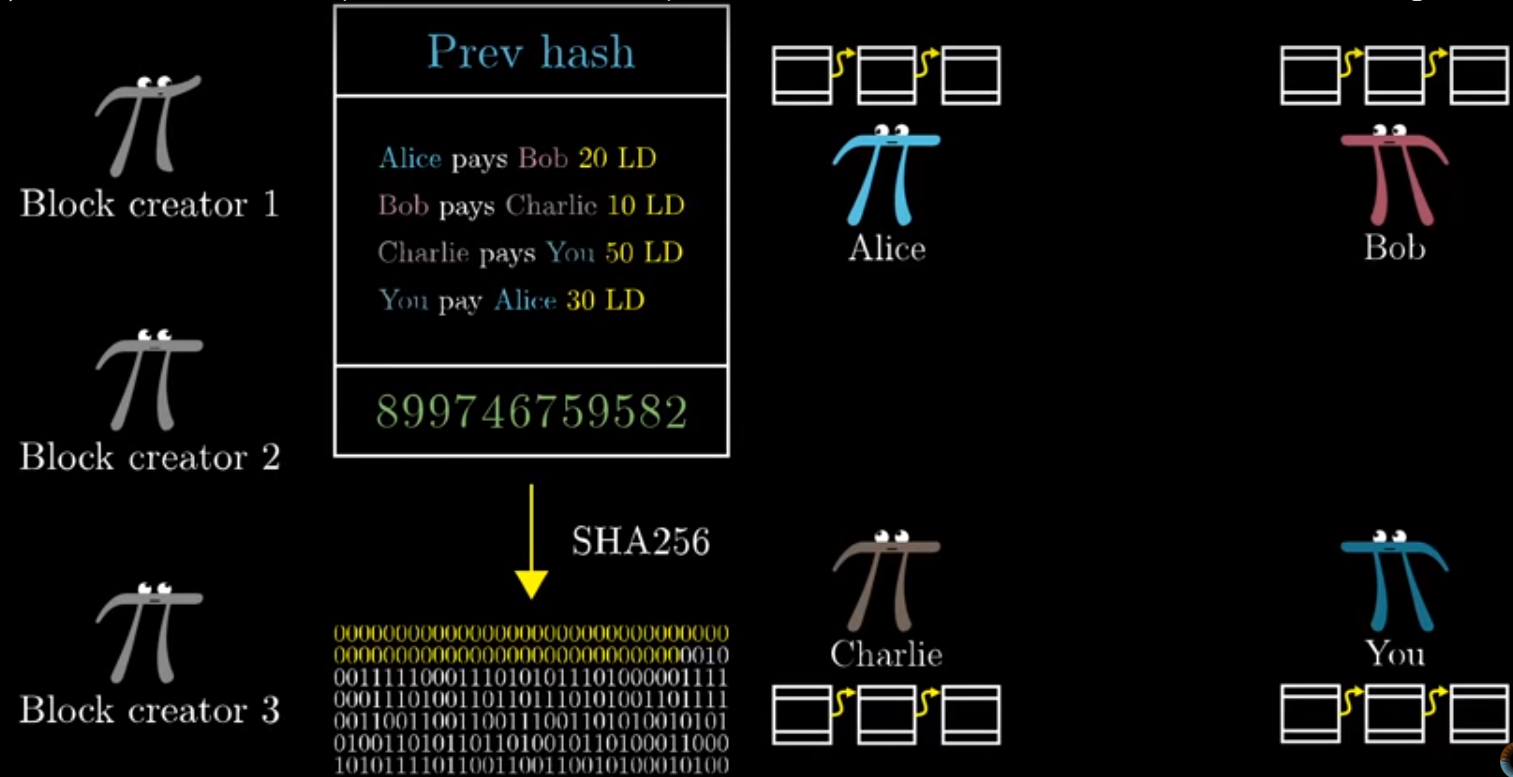


Slika ‑ Svaki sudionik na mreži ima vlastitu kopiju lanca blokova

Ukoliko neki od sudionika pokuša prevariti ostale npr. promjenom neke od transakcija želi dobiti veći iznos, svi ostali sudionici na mreži će vidjeti da se njihov lanac blokova razlikuje od lanca blokova zlonamjernog sudionika te će znati kako je ta njegova namjera zlonamjerna te će biti nevažeća. Da bi se zlonamjernom sudioniku ostvarila njegova namjera on mora imati više od 50% računalne snage kako bi imao većinu u tom distribuiranom svijetu sudionika. Iako vrlo lako moguće zvuči činjenica da neki sudionik ima više od 50% računalne snage (vlade, vojska itd.) u stvarnom svijetu je gotovo nemoguća. Upravo na tome se temelji velika sigurnost ovakvog sustava.

Distribuiranost se može promatrati iz dva pogleda. Prvi je već navedeni da je svima omogućeno sudjelovanje i nadgledanje transakcije i novih blokova, a drugi se odnosi na rudare (engl. Miner).

Na mreži se osluškuju transakcije te kada se nakupi određen broj transakcija one se spreme u jedan blok. Dakle, svaka transakcija se ne sprema u vlastiti blok, nego se više transakcija spremi u jedan blok što značajno ubrzava proces transakcija na mreži. Potom rudari, koji osluškuju te transakcije nastoje pogoditi ispravni identifikator (engl. Hash) bloka. Svi rudari koji sudjeluju u pogađanju ispravnog identifikatora se međusobno natječu te je pobjednik onaj koji prvi pogodi ispravni identifikator tako da on započinje s odgovarajućim brojem nula kao što je već objašnjeno u poglavlju 3.2.1. Rudar koji je pogodio ispravni identifikator potom dobiva određenu nagradu za svoj rad. Taj se uspješno napravljeni blok potom emitira na mrežu. Svi sudionici koji sudjeluju na mreži taj blok dodaju na kraj svog lanca (Slika 3‑10).



Slika ‑ Rudari (engl. Block creator/Miner) te sudionici (Alice, Bob, Charlie, You)

### Decentraliziranost

Decentraliziranost ovakvog sustava proizlazi iz toga što svaki sudionik na mreži posjeduje kopiju lanca blokova. Ukoliko jedan od čvorova prestane s radom to ne predstavlja nikakav problem za normalno funkcioniranje sustava zbog toga što svi ostali sudionici na mreži imaju taj isti lanac blokova.

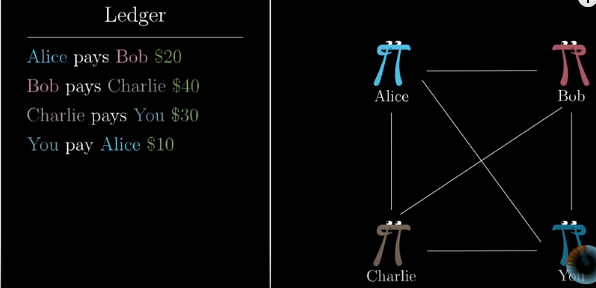
Dakle, u decentraliziranom sustavu svi čvorovi sustava imaju iste podatke i rade isti posao te nestanak jednog čvora nema nikakve posljedice na rad sustava.

## Kako ustvari radi lanac blokova?

U prethodnim poglavljima objašnjeni su razni pojmovi decentralizirane mreže. Svi ti pojmovi mogu izazvati pomutnju budući da se radi o sasvim novoj tehnologiji i načinu razmišljanja.

Stoga će se u ovom poglavlju na konkretnom primjeru Bitcoin-a pokazati kako ustvari radi lanac blokova (engl. Blockchain).

Za početak uzmimo da imamo jednu knjigu u kojoj pišemo sve transakcije koje se događaju na mreži.

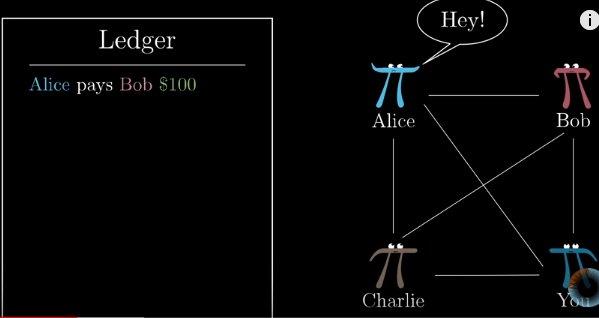


Slika ‑ Primjer javne knjige transakcija

Neka imamo vrlo jednostavan protokol koji se sastoji od pravila:

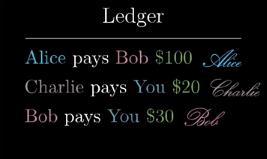
* Svatko može dodati transakciju u javnu knjigu
* Ne smije se uplatiti više nego što se posjeduje

Što sprječava nekoga da doda transakciju koja ide u njegovu korist bez da druga strana to odobri? Npr. Bob stavi da mu Alice treba platiti 100$ bez da Alice to odobri.



Slika ‑ Bob dodaje transakciju bez odobrenja Alice

Problem se riješi uz malo kriptografije tj. digitalnog potpisa.



Slika ‑ Dodavanje potpisa kako bi se odobrile transakcije

Ideja se sastoji od toga da postoji nešto što sudionici mogu dodati pored transakcije, označavajući kako su vidjeli i odobrili transakciju.

Što nas sprječava da kopiramo nečiji potpis?

Ključ je u tome što se potpis (engl. Signature) generira na osnovu privatnog ključa (engl. Private key) i ostalog sadržaja bloka. Dakle samo osoba s privatnim ključem može generirati ispravan potpis.

Svaki sudionik na mreži posjeduje vlastiti par privatnog/javnog ključa. Javni ključ se podijeli sa svim sudionicima na mreži, dok se privatni ključ ne govori nikome i strogo je povjerljiv. Na osnovu javnog ključa (engl. Public key) drugi sudionici znaju od koga dolaze transakcije te na koju adresu uputiti transakciju.

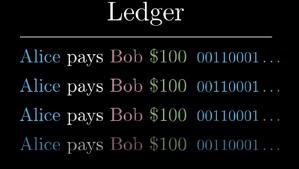
Generiranje ključa možemo zamisliti kao funkciju koja kao parametre prima neke podatke te privatni ključ a vraća jedinstveni potpis. Upravo zbog toga što podatke bloka uzima kao parametar rješava problem da se neki potpis kopira na drugu poruku.

Kako bi se provjerila ispravnost potpisa koristi se funkcija koja za parametre uzima podatke bloka, potpis i javni ključ te kao rezultat vraća točno/netočno (engl. True/false) ovisno o tome je li potpis ispravan ili netočan.

Potpis se sastoji od 256 bitova, što znači da će biti gotovo nemoguće pogoditi nečiji potpis „na sreću“ bez da se posjeduje privatni ključ. S 256 bitova moguće je napraviti 2256 kombinacija, a to je nezamislivo velik broj (od 78 znamenki). Zaključujemo da je gotovo nemoguće pogoditi taj broj.

Zaključak je: samo osoba koja ima privatni ključ može generirati taj potpis.

Što sprječava nekog da nakon prve potvrde samo kopira tu transakciju n puta budući da poznaje potpis koji je odobrio tu transakciju (Slika 3‑14)?



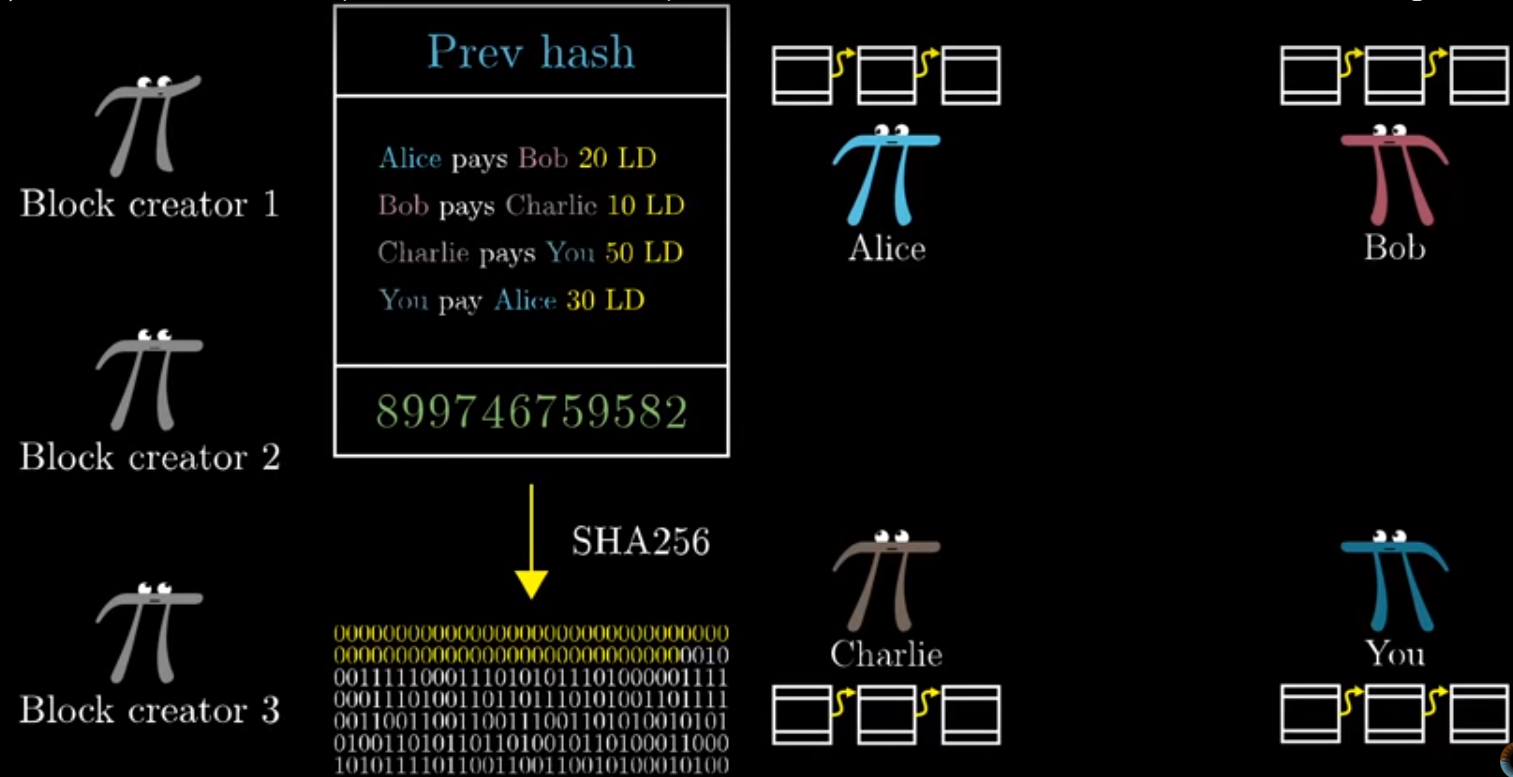
Slika ‑ Kopiranje transakcije n puta nakon što se potpiše kao ispravna

Trik je u tome što svaka transakcija nosi svoj ID što znači da signature nikad neće biti isti.

Slijedeći pojam je decentraliziranost. Svaki sudionik ima svoju kopiju javne knjige. Kada neki sudionik želi napraviti transakciju, transakcija se emitira svim ostalim sudionicima na mreži koji tu čuju i upišu u svoju javnu knjigu. Ukoliko se ne učini nešto više ovo je veoma loš sistem. Kako biti siguran da se svi slažu koja je javna knjiga ispravna? Kako biti siguran da svi ostali sudionici slušaju transakcije koje dolaze? Npr. netko vam uplati 20 kuna i kako biti siguran da su svi ostali sudionici „čuli“ da vam je poslano 20 kuna te da će te tih 20 kuna moći koristiti i njima raspolagati?

Problem je riješen tako da se vjeruje onoj „javnoj knjizi“ na koju je potrošeno najviše računalnog rada (engl. Computation work). Dakle, računalni rad predstavlja mjeru kome vjerovati. Vjeruje se uvijek onom zapisu koji ima najviše računalnog rada – dokaz o radu (engl. Proof of work).

Dakle, slušaju se sve transakcije koje se potom stavljaju u blok te se traži valjani identifikator za taj blok. Identifikator bloka da bi bio prihvatljiv mora započinjati s određenim brojem nula. Nakon što se nađe identifikator, taj se blok šalje svim sudionicima na mreži da ga dodaju na kraj svoje kopije lanca blokova (engl. Blockchain).



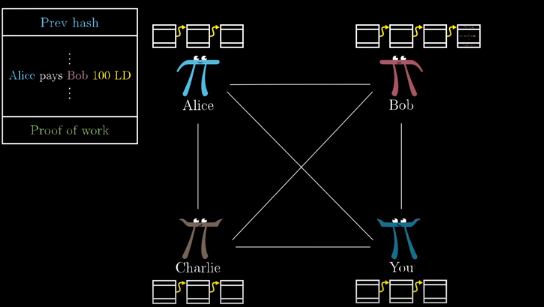
Slika ‑ Rudari (engl. block creator) pogađaju ispravni identifikator za blok koji se potom emitira sudionicima

Rudari (engl. Block creator/Miner) slušaju sve transakcije i probaju otkriti valjani identifikator prije ostalih rudara. Rudar koji prvi otkrije valjani identifikator dobije nagradu. Nagrada je određeni broj „novčića“ koji se prikupe od takse uzete od svake transakcije.

Svi ostali sudionici na mreži (Alice, Bob, Charlie, You) ne slušaju na sve transakcije nego samo na blokove koji se emitiraju od strane rudara.

Sada, kada smo objasnili osnovne pojmove vezane uz lanac blokova, na primjeru ćemo pokazati pokušaj prevare.

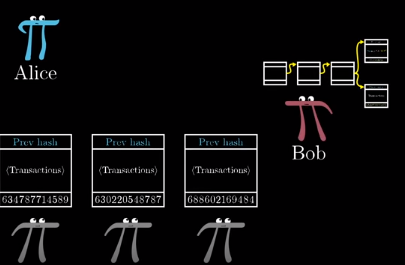
Uzmimo za primjer da Alice pošalje Bobu 100 kuna bez da svoju transakciju emitira mrežom.



Slika ‑ Alice plaća Bobu 100 kuna bez da svoju transakciju emitira svima na mreži

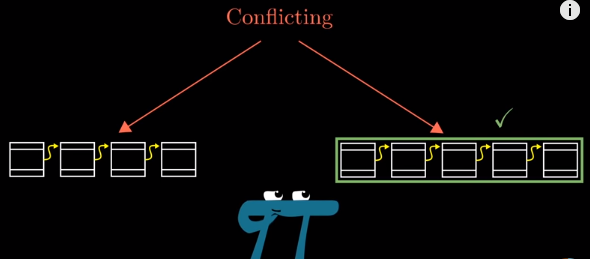
Da bi bilo moguće da blok pošalje samo Bobu, Alice treba pogoditi ispravni identifikator bloka prije svih ostalih rudara. Sasvim je moguće da Alice pogodi identifikator prije svih ostalih te da Bob doda taj blok na kraj svog lanca. Svi ostali sudionici na mreži tada neće znati za tu transakciju i vjerovat će da Alice ima tih 100 kuna.

Problem za zlonamjernog sudionika se skriva u tome što će Bob i dalje slušati emitirane blokove od svih rudara na mreži. U jednom trenutku nastat će konflikt između dva lanca blokova te će se lanac od Boba početi granati (Slika 3‑17).



Slika ‑ Grananje Bobovog lanca uslijed pokušaja prevare

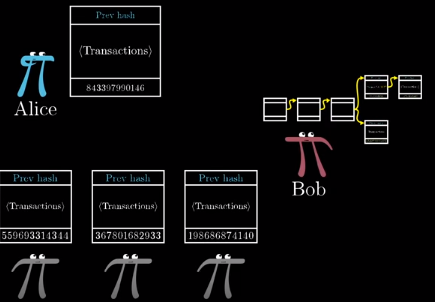
Pravilo glasi da se uvijek vjeruje dužem lancu jer je u njega uloženo više računalnog rada (Slika 3‑18).



Slika ‑ Uvijek se vjeruje lancu u kojeg je uloženo više rada

Trenutno, Bob ne zna koji je dio lanca ispravan jer su obje grane lanca jednake duljine stoga Bob i dalje nastavlja slušat emitirane blokove.

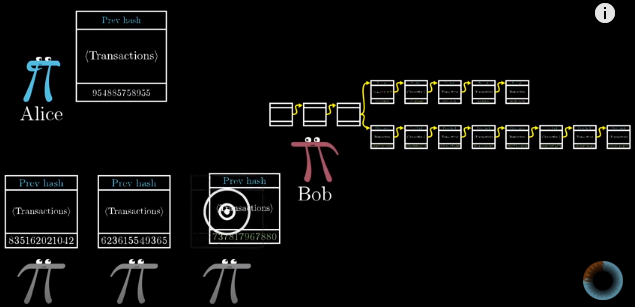
Da bi Alice uspjela u svom naumu mora i dalje pogađati ispravne identifikatore blokova prije svih ostalih rudara.



Slika ‑ Alice mora nastaviti pogađati identifikator prije svih ostalih rudara

Alice će možda pogoditi identifikator nekoliko puta prije ostalih rudara ukoliko ima toliko sreće, ali svakako neće moći održati lanac dužim dugoročno. Sve dok Alice nema najmanje 50% računalne snage u odnosu na sve ostale rudare gotovo je nemoguće da se to dogodi.

Dakle, svi ostali rudar koji rade zajedno će brže tržiti identifikator bloka nego Alice koja radi sama te će njihov lanac biti duži i Bob će zaključiti kako je Alicina grana lanca pokušaj prevare. Potom će odbaciti tu granu lanca i sve se vraća u normalu.



Slika ‑ Bob prati sve blokove dok ne postane očito koji je lanac ispravan

Zaključujemo da ukoliko se dogodi konflikt ne donosimo zaključke o ispravnosti odmah na temelju sljedećeg bloga nego se pričeka još par blokova te se odluči koji je lanac ispravan. S ovim zaključkom, objedinjeni su svi važni pojmovi i sigurnosni aspekti lanca blokova te su pokazani na konkretnom primjeru.

# DECENTRALIZIRANE APLIKACIJE

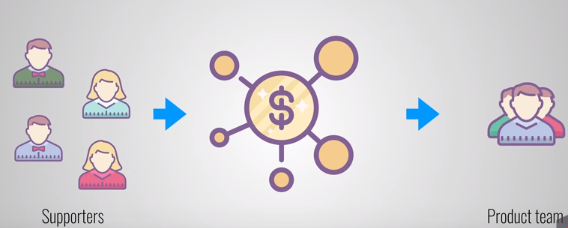
## Pametni ugovori

### Općenito o pametnim ugovorima

Pametni ugovori (engl. Smart contracts) su kao i obični ugovori u stvarnom svijetu. Razlika je u tome što su pametni ugovori potpuno „digitalni“. To su mali računalni programi koji su pohranjeni unutar lanca blokova.

Potrebe za pametnim ugovorima ćemo pokazati na primjeru Kickstarter platforme. Kickstarter je jedna od najpopularnijih platformi za prikupljanje sredstva. Korisnici mogu otići na Kickstarter platformu, pokrenuti projekt, postaviti koliko novca im treba za projekt i početi prikupljati novac od ljudi koji vjeruju u taj projekt.

Kickstarter predstavlja „treću stranu“ koja se nalazi između projekt tima i osoba koje podupiru tj. financiraju projekt (Slika 4‑1). To znači da obje strane moraju vjerovati Kickstarter-u da će raspodijeliti sredstva pošteno, kako je dogovoreno.



Slika ‑ Kickstarter kao posrednik između projektnog tima i sponzora projekta

Aplikaciju kao što je Kickstarter je moguće napraviti i korištenjem pametnih ugovora. Korištenjem pametnih ugovora nije nam potrebna „treća strana“ kao što je Kickstarter.



Slika ‑ Korištenje pametnih ugovora za izradu Kickstarter platforme

Sponzori projekta će prebaciti novac na pametni ugovor (Slika 4‑3). Ukoliko projekt uspije, prikupljeni novac se automatski šalje projektno timu, u suprotnom se novac vraća sponzorima projekta.



Slika ‑ Sponzori novac prebacuju na pametni ugovor

Budući da je pametni ugovor pohranjen unutar lanca blokova, sve je distribuirano što znači da ni jedna strana ne može kontrolirati novac.

Zašto vjerovati pametnom ugovoru?

Upravo zbog toga što je pametni ugovor pohranjen u bloku unutar lanca bokova. Kada se jednom napiše ugovor više ga nitko ne može izmijeniti, pa čak ni osoba koja ga je napisala. Ukoliko netko pokuša promijeniti ugovor, mijenja se identifikator bloka te dolazi to pucanja lanca kao što je već objašnjeno u ranijim poglavljima.

Iz navedenog proizlazi velika moguća primjena pametnih ugovora (bankarski, imovinski, pravni ugovori itd.).

### Primjena pametnih ugovora u decentraliziranoj aplikaciji

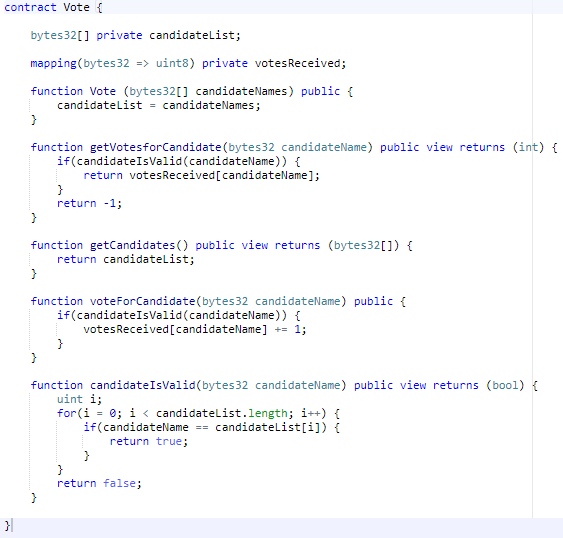
Najveći lanac blokova koji podržava pametne ugovore je Ethereum. Pametni ugovori za Ethereum se programiraju u programskom jeziku Solidity.

Današnje aplikacije se uglavnom sastoje od „backend-a“ koji komunicira s bazom podataka te isporučuje te podatke „frontend-u“ koji sadrži korisničko sučelje. Pametni ugovor (engl. Smart contract) u decentraliziranim aplikacijama predstavlja „backend“. On se pohrani unutar bloka u lancu blokova te se interakcija odvija pomoću korisničkog sučelja.

Solidity je objektno orijentirani programski jezik vrlo sličan ostalim objektnim jezicima.

Pametni ugovor je ustvari klasa koja ima vlastite varijable i metode. U ovom završnom radu, napravit ćemo pametni ugovor za glasanje. Korisnici će moći glasati za željenog kandidata te vidjeti trenutno stanje glasova.

Pametni ugovor pisat ćemo u besplatnom „online“ editoru Remix. Remix je jedan od najboljih besplatnih editora koji osim samog pisanja pametnih ugovora pruža i razne druge mogućnosti kao što su: pohranu pametnog ugovora u blok (na test mreži), jednostavno korisničko sučelje za interakciju sa pametnim ugovorom, kompajler i debugger itd. Kôd pametnog ugovora za glasanje dan je na slici (Slika 4‑4).



Slika ‑ Pametni ugovor za decentraliziranu aplikaciju za glasanje

Pametni ugovor deklarira se ključnom riječi „contract“ što odgovara ključnoj riječi „class“ iz drugih objektno orijentiranih programskih jezika.

Pametni ugovor Vote ima dvije varijable: candidateList i votesReceived. candidateList varijabla služi za pohranu svih kandidata, a votesReceived predstavlja broj glasova za pojedinog kandidata.

Na osnovu naziva metoda u pametnog ugovoru već se može zaključiti što koja metoda radi. Metode:

* getVotesforCandidate kao parametar prima ime kandidata, a vraća broj glasova za tog kandidata.
* getCandidates vraća imena svih kandidiranih kandidata.
* voteForCandidate kao parametar prima ime kandidata te poveća broj glasova tom kandidatu za jedan.
* candidateIsValid provjerava je li uneseni kandidat na listi kandidata, te na osnovu toga vraća istina/laž.

Nakon što je pametni ugovor napisan i uspješno kompajliran može se pohraniti unutar bloka u lancu blokova. Za potrebe ovog rada koristit ćemo test mrežu.

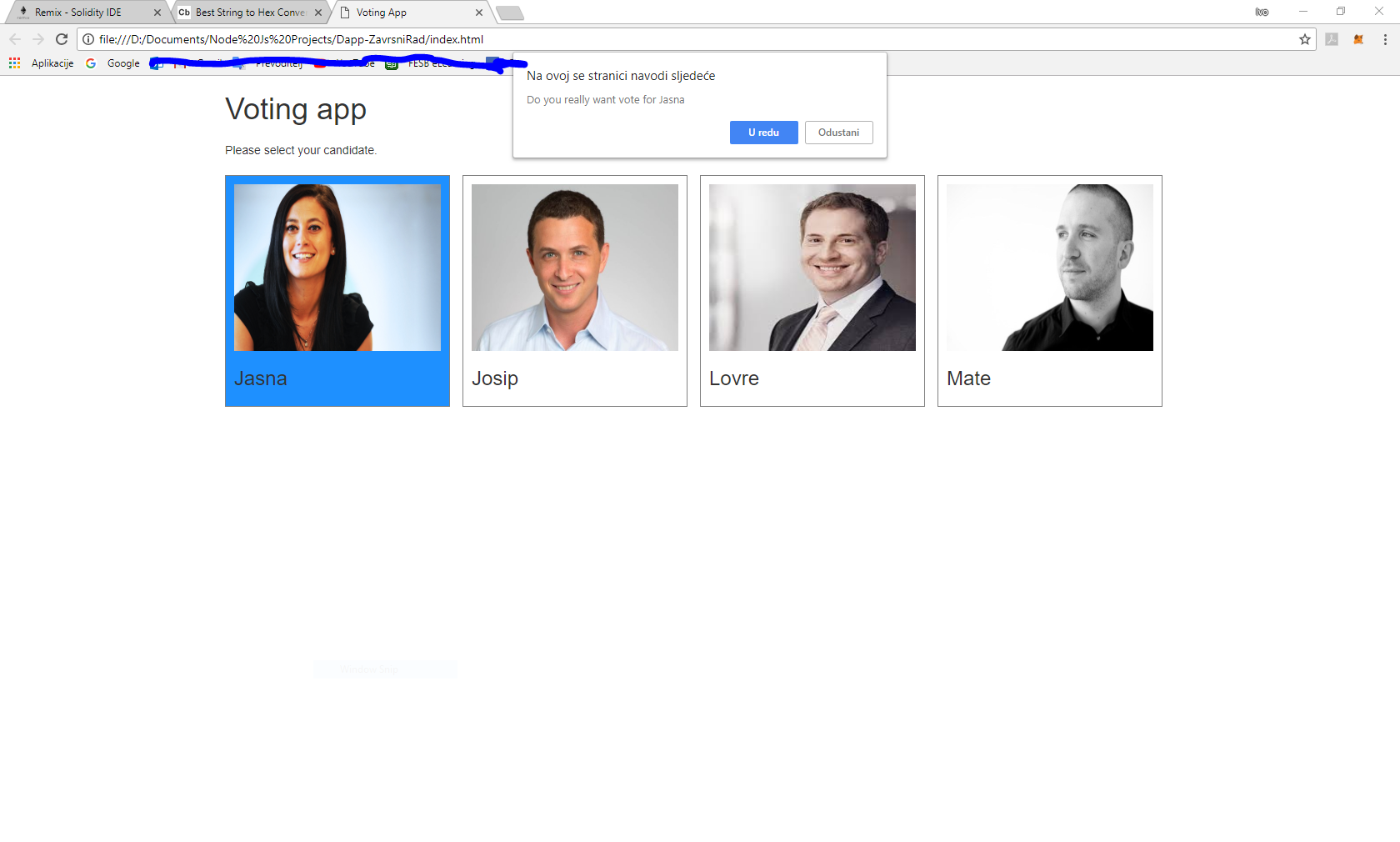
S objavom pametnog ugovora zaključili smo poglavlje o „backend-u“.

## Korisničko sučelje

Korisničko sučelje decentralizirane aplikacije za glasanje izrađeno je od standardnih web jezika kao što su HTML, CSS, javascript. Ono pruža mogućnost interakcije s pametnim ugovorom koji je pohranjen unutar lanca blokova na test mreži. Za interakciju s pametnim ugovorom koristi se web3 modul.

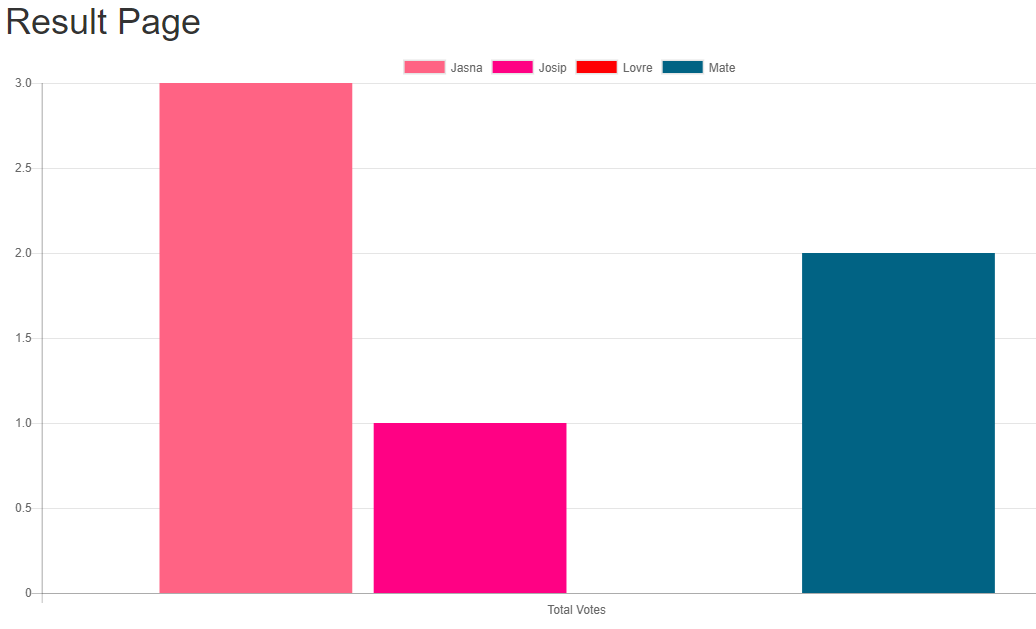
Web3 se instalira pomoću npm alata. Sadrži objekt koji ima razne metode za interakciju s pametnim ugovorom.

Korisničko sučelje je vrlo jednostavno te omogućava odabir željenog kandidata nakon čega se pojavi prozorčić za potvrdu odabira kao što je prikazano na slici (Slika 4‑5).



Slika ‑ Izgled korisničkog sučelja aplikacije za glasanje

Nakon potvrde, otvori se stranica koja prikazuje trenutno stanje glasova za svakog kandidata (Slika 4‑6).



Slika ‑ Stranica na kojoj se prikazuju rezultati glasanja

Grafovi se dobiju korištenjem Chart.js javascript plugin-a.

Ono što je potrebno istaknuti je korištenje asinkronog programiranja za interakciju sa pametnim ugovorom. Kod asinkronog programiranja nema blokiranja koda dok se ne izvrši određena funkcija, nego se nastavlja dalje sa izvođenjem. Nakon što određena funkcija izvrši pozove se callback funkcija.

Istaknuti dio koda:

// returns all candidates

function returnCandidates() {

return new Promise((resolve, reject) => {

vote.getCandidates((err, candidates) => {

if(err) {

reject(err);

} else {

resolve(candidates);

}

});

});

}

// returns no. of votes for particular candidate

function getVotesForCandidate(candidate) {

return new Promise((resolve, reject) => {

vote.getVotesforCandidate(candidate, (err, result) => {

if(err) reject(err);

resolve(Number('' + result));

});

});

}

async function getVotes() {

let candidates;

try {

candidates = await returnCandidates();

} catch (e) {

console.log(e);

}

for(let i = 0; i < candidates.length; i ++) {

let no = await getVotesForCandidate(candidates[i]);

resultArr.push(no);

}

}

//After getting resultArr call makeChart function.

getVotes().then(() => {

makeChart(resultArr);

});

Funkcija koja radi graf trenutnog stanja glasova makeChart() se pozove tek nakon što se dobiju svi glasovi za pojedinog kandidata. Uočavamo da je getVotes asinkrona funkcija (prefiks async). Korištenjem prefiksa async dobivamo mogućnost korištenja await opcije koja nam omogućava da programiramo asinkrono na sinkroni način. Budući da je asinkrono programiranje tema za sebe, neće toliko biti objašnjeno u ovom završnom radu.

## Pametni ugovori u praksi – jesu li uistinu toliko sigurni?

U ovom radu su objašnjeni razni sigurnosni aspekti decentraliziranih aplikacija i sve to zvuči gotovo savršeno, ali jesu li uistinu toliko sigurne?

Iako je sam lanac blokova veoma siguran i pouzdan slaba točka su upravo pametni ugovori. Problem je u kodu koji se koristi za stvaranje pametnih ugovora jer je podložan raznim kvarovima (engl. Bugs). U 2016. godini, haker je zaradio oko 50 milijuna dolara u kripto valutama tako što je ubacio kvar u kod pametnog ugovora.

Ipak, pametni ugovori sigurno predstavljaju novu generaciju tehnologije te je njihova primjena u budućnosti zagarantirana. U samo par mjeseci 2017-te godine, broj pametnih ugovora je skočio s 500 000 na oko 2 000 000.

Zasigurno će se poraditi i na sigurnosti pametnih ugovora te će uistinu biti sigurni u praksi koliko i u teoriji.

# ZAKLJUČAK

Iako su danas najviše rasprostranjene, centralizirane aplikacije imaju veliki broj nedostataka. Svi ti spomenuti nedostaci otvaraju prostor za novu generaciju aplikacija. Novu generaciju aplikacija predstavljaju upravo decentralizirane aplikacije kod kojih nema jednog centralnog čvora na kojem se nalazi cijela aplikacija, nego se taj sustav sastoji od velikog broja čvorova od kojih svaki sadrži svoju kopiju aplikacije. To rezultira tome da pad jednog čvora ne znači pad kompletnog sustava što može biti veliki problem ukoliko se radi o veoma bitnim sustavima.

U ovom radu smo također objasnili sigurnosne aspekte decentraliziranih aplikacija. Upravo zbog velike sigurnosti doživjele su tako veliki BOOM. Iako su još u ranim začecima, glavna ideja sigurnosti je objašnjena u ovom radu koja će sigurno predstavljati temelj za njihovu buduću nadogradnju.

Na kraju ovog rada trebali bi biti sposobni objasniti prednosti decentraliziranih aplikacija, osnovne sastavnice lanca blokova te osnovne sigurnosne mehanizme. Također, trebali bi dobiti i osnovno znanje o načinu stvaranja decentraliziranih aplikacija. U ovom radu na primjeru jednostavne aplikacije za glasanje objedinili smo sve važne pojmove na konkretnom praktičnom primjeru.

Na kraju ovog rada mogu iznijeti zaključak da će decentralizirane aplikacije sigurno predstavljati važnu ulogu u budućnosti. Ukoliko se to i ne dogodi (što je vrlo malo vjerojatno), glavni temelj na koji se oslanjaju ovakve aplikacije će zasigurno naći uporabu u nekim novim idejama upravo zbog niza prednosti koje smo objasnili u ovom radu.

Kako bilo da bilo, decentralizirane aplikacije donijele su jedan novi način razmišljanja što predstavlja veliko osvježenje u odnosu na dosadašnje klasične aplikacije. Moguće je da upravo ova tehnologija dovede do razvoja još boljih i naprednijih ideja te neke nove slijedeće generacije aplikacija.

# POPIS OZNAKA I KRATICA

BTC – Bitcoin

P2P – Peer to peer

ID - identifier

Npm – nodejs package manager

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

# LITERATURA

1. „A guide to BuildingYour First Decentralized Application“, s interneta, <https://www.youtube.com/watch?v=gSQXq2_j-mw&t=436s&list=PLv_dedsxCcn_i1CrI9gBSa8UZVtQmpOb2&index=3>, 20. svibnja 2018.
2. „Ever wonder how Bitcoin (and other cryptocurrencies) actually work?“, s interneta, <https://www.youtube.com/watch?v=bBC-nXj3Ng4&t=71s>, 20. svibnja 2018.

# DODATAK A

Kod pametnog ugovora VotingContract.sol:

pragma solidity ^0.4.18;

contract Vote {

bytes32[] private candidateList;

mapping(bytes32 => uint8)     private votesReceived;

function Vote(bytes32[] candidateNames) public {

candidateList = candidateNames;

}

function getVotesforCandidate(bytes32 candidateName) public view returns (int) {

if(candidateIsValid(candidateName)) {

return votesReceived[candidateName];

}

return -1;

}

function getCandidates() public view returns (bytes32[]) {

return candidateList;

}

function voteForCandidate(bytes32 candidateName) public {

if(candidateIsValid(candidateName)) {

votesReceived[candidateName] += 1;

}

}

function candidateIsValid(bytes32 candidateName) public view returns (bool) {

uint i;

for(i = 0; i < candidateList.length; i++) {

if(candidateName == candidateList[i]) {

return true;

}

}

return false;

}

}

index.html datoteka:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Voting App</title>

<link href='https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css' rel='stylesheet' type='text/css'>

<link href='assets/css/main.css' rel='stylesheet' type='text/css'>

<template id="candidate-template">

<div class="candidate">

<div class="candidate-img">

<img src="" alt="">

</div>

<h3></h3>

</div>

</template>

</head>

<body class="container">

<h1 class="page-title">Voting app</h1>

<p id="label-par">Please select your candidate.</p>

<div id="voting-part" class="voting-part"></div>

<script src="./node\_modules/web3/dist/web3.min.js" type="text/javascript"></script>

<script src="./assets/js/bundle.js"></script>

</body>

</html>

seetheresults.html datoteka:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Results</title>

<link href='https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css' rel='stylesheet' type='text/css'>

</head>

<body class="container">

<h1 class="page-title">Result Page</h1>

<canvas id="myChart"></canvas>

<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.4.0/Chart.min.js"></script>

<script src="./node\_modules/web3/dist/web3.min.js" type="text/javascript"></script>

<script src="./assets/js/bundle2.js"></script>

</body>

</html>

index.js datoteka:

// garbage dream earth sail rate sport crime cliff inherit crop useless burst

// ["0x4a61736e61","0x4a6f736970","0x4c6f767265","0x4d617465"]

const utils = require('web3/lib/utils/utils');

if (typeof web3 !== 'undefined') {

web3 = new Web3(web3.currentProvider);

} else {

web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));

}

web3.eth.defaultAccount = web3.eth.accounts[0];

const voteContract = web3.eth.contract([{"constant":true,"inputs":[],"name":"getCandidates","outputs":[{"name":"","type":"bytes32[]"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":true,"inputs":[{"name":"candidateName","type":"bytes32"}],"name":"getVotesforCandidate","outputs":[{"name":"","type":"int256"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":false,"inputs":[{"name":"candidateName","type":"bytes32"}],"name":"voteForCandidate","outputs":[],"payable":false,"stateMutability":"nonpayable","type":"function"},{"constant":true,"inputs":[{"name":"candidateName","type":"bytes32"}],"name":"candidateIsValid","outputs":[{"name":"","type":"bool"}],"payable":false,"stateMutability":"view","type":"function"},{"inputs":[{"name":"candidateNames","type":"bytes32[]"}],"payable":false,"stateMutability":"nonpayable","type":"constructor"}]);

let vote = voteContract.at("0x306f60f029a284237e081bdcf1972658f32e0a42");

// First step. Validate address and redirect to the voting page.

// @var votingAddress stores valid address for later use.

let votingAddress;

// Get all candidates from a contract and add them to the page.

const votingDiv = document.getElementById('voting-part');

vote.getCandidates((err, candidates) => {

if(err) {

console.log(err);

} else {

candidates.forEach(candidate => {

let candName = utils.toUtf8(candidate);

let temp = document.getElementById('candidate-template');

let candElement = document.importNode(temp.content, true);

let h3 = candElement.querySelector('h3');

h3.innerHTML = candName;

let img = candElement.querySelector('img');

img.src = './assets/img/' + candName.toLowerCase() + '.jpg';

img.alt = candName;

candElement.childNodes[1].onclick = onCandidateClick;

votingDiv.appendChild(candElement);

});

}

});

function onCandidateClick(e) {

let candidate = e.currentTarget;

let name = candidate.querySelector('h3').innerHTML;

//let inpAdr = document.getElementById('voting-address');

//if(web3.eth.accounts.includes(inpAdr.value)) {

//votingAddress = inpAdr.value;

if(confirm("Do you really want vote for " + name)) {

// incrementing votes for that candidate

vote.voteForCandidate(name);

// redirect to result page after voting.

location.href = './seetheresults.html';

}

}

results.js datoteka:

const utils = require('web3/lib/utils/utils');

if (typeof web3 !== 'undefined') {

web3 = new Web3(web3.currentProvider);

} else {

web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));

}

web3.eth.defaultAccount = web3.eth.accounts[0];

const voteContract = web3.eth.contract([{"constant":true,"inputs":[],"name":"getCandidates","outputs":[{"name":"","type":"bytes32[]"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":true,"inputs":[{"name":"candidateName","type":"bytes32"}],"name":"getVotesforCandidate","outputs":[{"name":"","type":"int256"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":false,"inputs":[{"name":"candidateName","type":"bytes32"}],"name":"voteForCandidate","outputs":[],"payable":false,"stateMutability":"nonpayable","type":"function"},{"constant":true,"inputs":[{"name":"candidateName","type":"bytes32"}],"name":"candidateIsValid","outputs":[{"name":"","type":"bool"}],"payable":false,"stateMutability":"view","type":"function"},{"inputs":[{"name":"candidateNames","type":"bytes32[]"}],"payable":false,"stateMutability":"nonpayable","type":"constructor"}]);

let vote = voteContract.at("0x306f60f029a284237e081bdcf1972658f32e0a42");

// array which stores results.

let resultArr = [];

// returns all candidates

function returnCandidates() {

return new Promise((resolve, reject) => {

vote.getCandidates((err, candidates) => {

if(err) {

reject(err);

} else {

resolve(candidates);

}

});

});

}

// returns no. of votes for particular candidate

function getVotesForCandidate(candidate) {

return new Promise((resolve, reject) => {

vote.getVotesforCandidate(candidate, (err, result) => {

if(err) reject(err);

resolve(Number('' + result));

});

});

}

async function getVotes() {

let candidates;

try {

candidates = await returnCandidates();

} catch (e) {

console.log(e);

}

for(let i = 0; i < candidates.length; i ++) {

let no = await getVotesForCandidate(candidates[i]);

resultArr.push(no);

}

}

//After getting resultArr call makeChart function.

getVotes().then(() => {

makeChart(resultArr);

});

// function that prints out chart

function makeChart(arr) {

let ctx = document.getElementById('myChart').getContext('2d');

let chart = new Chart(ctx, {

// The type of chart we want to create

type: 'bar',

// The data for our dataset

data: {

labels: ["Total Votes"],

datasets: [{

label: "Jasna",

backgroundColor: 'rgb(255, 99, 132)',

data: [arr[0]],

},{

label: "Josip",

backgroundColor: 'rgb(255, 1, 132)',

data: [arr[1]],

},{

label: "Lovre",

backgroundColor: 'rgb(255, 0, 2)',

data: [arr[2]],

},{

label: "Mate",

backgroundColor: 'rgb(1, 99, 132)',

data: [arr[3]],

}]

},

// Configuration options go here

options: {

scales: {

yAxes: [{

ticks: {

beginAtZero: true

}

}]

}

}

});

}

main.css datoteka:

.voting-part {

margin-top: 20px;

display: flex;

width: 100%;

}

.candidate {

padding: 10px;

border: 1px solid gray;

flex-basis: 0;

flex-grow: 1;

margin-right: 15px;

}

.candidate:hover {

background-color: dodgerblue;

}

.candidate-img img{

width: 100%;

height: 200px;

}

#voting-address-error {

display: none;

padding: 10px;

}

#voting-part-results {

visibility: hidden;

}

#label-par {

margin-top: 20px;

}

Github repository: <https://github.com/ikovac/Dapp>