УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Дејан Џуња

Blockchain дистрибуирани системи и њихова примена у трговини

ДИПЛОМСКИ РАД
- Основне академске студије -

Нови Сад, 2018.

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:		
Идентификациони број, **ИБР**:		
Тип документације, **ТД**:		Монографска документација
Тип записа, **ТЗ**:		Текстуални штампани материјал
Врста рада, **ВР**:		Дипломски рад
Аутор, **АУ**:		Дејан Џуња
Ментор, **МН**:		др Душан Гајић, доцент
Наслов рада, **НР**:		Blockchain дистрибуирани системи и њихова примена у трговини
Језик публикације, **ЈП**:		Српски/латиница
Језик извода, **ЈИ**:		Српски/енглески
Земља публиковања, **ЗП**:		Република Србија
Уже географско подручје, **УГП**:		АП Војводина
Година, **ГО**:		2018.
Издавач, **ИЗ**:		Ауторски репринт
Место и адреса, **МА**:		Нови Сад, Трг Доситеја Обрадовића 6
Физички опис рада, **ФО**: (поглавља/страна/цитата/табела/слика/графика/прилога)		5/44/0/4/23/0/0
Научна област, **НО**:		Електротехника и рачунарство
Научна дисциплина, **НД**:		Примењене рачунарске науке и информатика
Предметна одредница/Кључне речи, **ПО**:		Базе података, дистрибуирани информациони системи, blockchain
УДК		
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, **ВН**:		
Извод, **ИЗ**:		У овом раду описани су основни концепти *blockchain* технологије и једне од њених познатијих имплементација, *Hyperledger Fabric*-а. Такође, описана је и апликација имплементирана на Hyperledger Fabric платформи, развијена са циљем управљања материјалним добрима.
Датум прихватања теме, **ДП**:		17.08.2018.
Датум одбране, **ДО**:		22.08.2018.
Чланови комисије, **КО**:	Председник:	др Дину Драган, доцент
	Члан:	др Милан Сегединац, доцент
	Члан, ментор:	др Душан Гајић, доцент

Потпис ментора

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:	
Identification number, **INO**:	
Document type, **DT**:	Monographic publication
Type of record, **TR**:	Textual printed material
Contents code, **CC**:	Bachelor Thesis
Author, **AU**:	Dejan Džunja
Mentor, **MN**:	Dušan Gajić, Assistant Professor, Ph.D
Title, **TI**:	Blockchain Distributed Systems and Their Application in Trading
Language of text, **LT**:	Serbian
Language of abstract, **LA**:	Serbian/English
Country of publication, **CP**:	Republic of Serbia
Locality of publication, **LP**:	AP Vojvodina
Publication year, **PY**:	2018.
Publisher, **PB**:	Author's reprint
Publication place, **PP**:	Novi Sad, Dositeja Obradovića sq. 6
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)	5/44/0/4/23/0/0
Scientific field, **SF**:	Electrical and computer engineering
Scientific discipline, **SD**:	Applied computer science and informatics
Subject/Key words, **S**/**KW**:	Databases, distributed information systems, blockchain
UC	
Holding data, **HD**:	The Library of Faculty of Technical Sciences, Novi Sad
Note, **N**:	
Abstract, **AB**:	This thesis presents the fundamentals of the blockchain technology and one of its popular implementations, Hyperledger Fabric. It also describes an application implemented using the Hyperledger Fabric platform, which is developed for asset management in trading.
Accepted by the Scientific Board on, **ASB**:	17.08.2018.
Defended on, **DE**:	22.08.2018.

Defended Board, **DB**:	President:	Dinu Dragan, Assistant Professor, Ph.D	
	Member:	Milan Segedinac, Assistant Professor, Ph.D	Menthor's sign
	Member, Mentor:	Dušan Gajić, Assistant Professor, Ph.D	

	УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА** 21000 НОВИ САД, Трг Доситеја Обрадовића 6 **ЗАДАТАК ЗА ИЗРАДУ ДИПЛОМСКОГ (BACHELOR) РАДА**	Датум: Лист/Листова: 4/44

(Податке уноси предметни наставник - ментор)

Врста студија:	☒ Основне академске студије ☐ Основне струковне студије
Студијски програм:	Рачунарство и аутоматика
Руководилац студијског програма:	др Мирослав Поповић, ред. проф.

Студент:	Дејан Џуња	Број индекса:	РА 52/2014
Област:	Електротехничко и рачунарско инжењерство		
Ментор:	др Душан Гајић, доцент		

НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА
ИЗДАЈЕ СЕ ЗАДАТАК ЗА ДИПЛОМСКИ (Bachelor) РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:

- проблем – тема рада;
- начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;
- литература

НАСЛОВ ДИПЛОМСКОГ (BACHELOR) РАДА:

Blockchain дистрибуирани системи и њихова примена у трговини

ТЕКСТ ЗАДАТКА:

- Проучити основне концепте и механизме blockchain дистрибуираних система.

- Упознати се са Hyperledger Fabric платформом, као примером конкретне имплементације blockchain-а.

- Упоредити Hyperledger Fabric мрежу и релационе базе података у примени

- Упознати се са Hyperledger Composer алатом за развој пословних апликација базираних на Hyperledger Fabric платформи.

- Специфицирати модел blockchain мреже намењене за трговину, који садржи дефиниције ресурса, учесника и операција које учесници могу извршавати над ресурсима.

- На основу специфицираног модела имплементирати одговарајуће софтверско решење засновано на Hyperledger Fabric-у.

Руководилац студијског програма:	Ментор рада:

Примерак за: ☐ - Студента; ☐ - Ментора

SADRŽAJ

1 UVOD

Brz tehnološki napredak i širenje Interneta jesu direktan uzrok razvoju novih i unapređenih načina komunikacije između ljudi. Jedan od važnijih vidova komunikacije između ljudi ili pravnih entiteta jesu transakcije koje služe kao ugovor o razmeni dobara, usluga, novca i dr.

Ako se posmatra još od početaka savremene civilizacije, transakcija bi se na visokom nivou apstrakcije mogla definisati kao zvanični ugovor koji potvrđuje vlasništvo dobara, finansijskih sredstava nekog entiteta. Takav ugovor je dosta važan, jer uverava treće strane da neko dobro bilo ono finansijske ili materijalne prirode zaista pripada pravom vlasniku. Bez takve vrste potvrde, treća strane nemaju informaciju o pravom vlasniku te im to uvodi nelagodu pri stvaranju novih transakcija.

Kako se društvo razvijalo transakcije su zaista poprimale različite oblike; od razmene materijalnih dobara ili finansijskih sredstava preko novčanica do elektronskih zapisa u bazama podataka. Zbog ogromnog porasta broja i kompleksnosti transakcija, mnogi njeni oblici ili načini na koji se izvršavaju su postali izuzetno nepouzdani. Kod razmene dva materijalna dobra je u većini postojala usmena transakcija (nema zabeleženog podatka, te ne znamo da li je došlo do nekog zlonamernog dela kao što je krađa), novčanice više nemaju pokriće i mogu se falsifikovati, a zapisi u bazama podataka se mogu zlonamerno izmeniti.

Zbog gore navedenih razloga dolazimo do gubljenja poverenja u dalje stvaranje transakcija jer se više ne zna šta je istina. Kako se ne bi posumnjalo u validnost transakcije dolazimo do toga da svaka stranka koja ima mogućnost da započne istu mora da pregleda svaku prethodnu kako bi se steklo poverenje. Ovo bi rešilo problem nepoverenja, ali je dosta neefikasno i neprihvatljivo ako uzmemo u obzir današnji standard i kompleksnost. Sledeće rešenje bi bilo da postoji treće lice[1] u koje će sve stranke imati poverenje. Treće lice bi bilo zaduženo za validiranje i zapisivanje svake transakcije (primer sa zapisivanjem u centralnu bazu podataka), ali šta ako zapisnici nestanu ili treće lice postupi zlonamerno i naknadno izmeni neke transakcije?

U prethodnom pasusu smo spomenuli rešenje koje je izuzetno neefikasno, ali ako bi se sprovelo rešili bi probleme posrednika i poverenja. Da li možemo omogućiti da svaka stranka ne mora da proverava svaku prethodnu transakciju i da svaka nova stranka koja uđe u našu grupu automatski sve prethodne izvršene transakcije smatra za validne na osnovu nekih pravila?

Odgovor na ovo pitanje je dao *Satoshi Nakamoto*[2] osmišljavanjem osmišljavanjem koncepta *blockchain*-a i predlaganjem njegove prve primene u vidu kriptovalute Bitcoin opisane u radu (engl. *whitepaper*[3]) Bitcoin: A Peer-to-Peer Electronic Cash System [1]. Koncept

[1] Još jedno lice u transakciji ili drugačije rečeno posrednik (engl. *third party*)[2]

[2] Ne zna se tačan identitet Satoshi Nakomoto-a niti da li je to jedna osoba ili grupa. U ovom radu ćemo je spominjati u muškom rodu, što je i pretpostavka cele internet zajednice

[3] engl. *whitepaper* predstavlja izveštaj koji objašnjava prirodu karakterstičnog problema i predlaže rešenje na isti. Cilj ove vrste izveštaja je da u što kraćim tezama upozna korisnika sa problemom.

blockchain i kriptovaluta star je, dakle već celu deceniju i nastao je uglavnom kao odgovor na finansijsku krizu iz 2008. godine. Njegov rad je rešio probleme poverenja spomenutih u ovom poglavlju, ali naravno ni ono nije savršeno jer nosi sa sobom nedostatke druge prirode koje su otkrivene posle nekog vremena.

Cilj ovog rada jeste upoznavanje sa blockchain tehnologijom i njenom primenom u trgovini. Uz ovaj rad je realizovano i softversko rešenje koje će bolje približiti primenu i osnovi mehanizam ovih tehnologija. Softversko rešenje na nivou dokaza koncepta (engl. *proof of concept*[4]) je realizovano pomoću biblioteke otvorenog koda (engl. *open source*[5]) *Hyperledger Fabric* i alata *Hyperledger Composer*.

Nakon uvoda sledi drugo poglavlje "Osnove *blockchain* sistema" koje objašnjava strukturu podataka *blockchain*-a i njegove osnovne mehanizme rada kao što su konsenzus algoritmi. Treće poglavlje predstavlja projekat *Hyperledger* i njegovu platformu *Hyperledger Fabric* koja omogućava kreiranje *blockchain*-a rešenja radi primene u industriji. Nakon ovoga sledi praktični deo rada koji opisuje primenu i implementaciju softverskog rešenja uz pomoću alata *Hyperledger Composer*. Za kraj ostaje zaključak koji iznosi konačne rezultate rada i kritike.

[4]engl. *proof of concept* - skica projekta. Kreira se radi demonstracije koja ima u cilju potvrđivanja potencijala ideje [3]

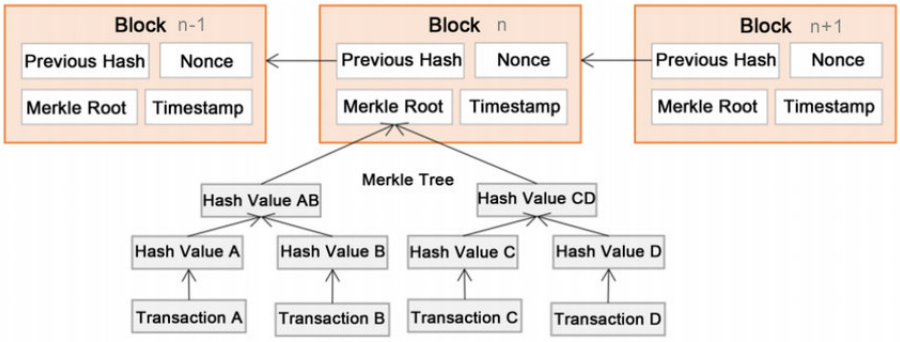
[5]engl. *open source* - kod dostupan svima za korišćenje i modifikaciju

2 OSNOVE BLOCKCHAIN SISTEMA

Blockchain predstavlja rastuću listu podataka otpornu na naknadnu modifikaciju bilo koje vrste (brisanje, izmena i dr). Svaki novi unos u strukturu zavisi od prethodnih unosa, te modifikovanjem jednog podatka se ruši konzistetnost ostalih [4]. Ova struktura se koristi kao dnevnik transakcija koja se konstantno replicira, deli i sinhronizuje među njenim korisnicima (engl. *distributed ledger*) [5]. Jednostavnije rečeno svaki korisnik poseduje kopiju knjige i pri svakom konačnom upisu u knjigu svi korisnici mreže se dogovaraju preko unapred zadatih pravila (konsenzus - engl. *consensus*) da li je upis u knjigu validan i koji član grupe će izvršiti upis.

2.1 Opis strukture podataka

Sa stanovišta računarskih nauka, *blockchain* je jednostruko unazad spregnuta lista[6]. Svaki element liste (u daljem tekstu blok), se identifikuje hešom koji je generisan od strane duple SHA256[7] funkcije. U daljim poglavljima je izučena detaljnija struktura bloka, Merkleovog stabla i sigurnosne odlike ove strukture.



Slika 1: Blockchain struktura (preuzeto sa *researchgate.net*)

2.1.1 Struktura bloka

Blok je kontejner podataka koji sadrži zaglavlje (engl. *header*) i transakcije. Veličina zaglavlja je oko 80 bajta, svaka transakcija zauzima oko 250 bajta dok transakcija u bloku uglavom ima više od 500. *Tabela 1* prikazuje strukturu bloka, dok *tabela 2* prikazuje strukturu zaglavlja bloka. Struktura bloka u navedenim tabelama odgovara Bitcoin-u. Česta zabuna kod strukture blokova je da ono zadrži i svoj heš koji ga jedinstveno identifikuje, što nije slučaj iz sigurnosnih razloga.

[6]engl. *back-linked list* - struktura koja se sastoji iz listova gde svaki sadrži adresu prethodnog

[7]Secure Hash Algorithm (SHA) - familija heš funkcija koje se koristi u kriptografiji. Za svaki definisani string generiše heš veličine 32 bajta. Funkcije SHA familije imaju jedan pravac tj ne postoji inverzna funkcija koja bi na osnovu izlaza iz SHA funkcije dobila ulaz.

Veličina [bajt]	Ime polja	Opis
4	Veličina bloka	Broj koji označava veličinu bloka u jedinici bajt
80	Zaglavlje bloka	Polja u zaglavlju bloka
1-9	Brojač transakcija	Broj transakcija
500*250	Transakcije	Transakcije u bloku

Tabela 1: Struktura bloka [6]

Veličina [bajt]	Ime polja	Opis
4	Verzija	Verzija protokola
32	Adresa prethodnog bloka	Adresa/heš prethodnog bloka tj roditelja
32	Merkle koren	Heš korena Merkle stabla
4	Timestamp	Vreme kreacije bloka
4	Težina	Težina bloka za Proof-of-Work algoritam
4	Nonce	Brojač za Proof-of-Work algoritam

Tabela 2: Struktura zaglavlja bloka [6]

2.1.2 Otpornost strukture na promene

U prethodnom poglavlju spomenuto je da blok ne sadrži zasebnu varijablu za heš koji ga jedinstveno identifikuje. Umesto toga heš se pri svakom pristupu bloka može izračunati pomoću podataka iz zaglavlja bloka u *tabeli 2*. Bilo koji pokušaj modifikacije bloka će uzrokovati promenu njenog jedinstvenog heša, te sledeći blok tj potomak će ostati sa starim hešom svoga roditelja i time dolazi do 'otkačivanja' lanca. Kako bi se otkačivanje lanca sprečilo morali bi da se izmene svi naredni blokovi u lancu tako da njeni heševi budu validni što postaje sve teže i teže kako lanac raste [6]. Na primer modifikacijom bloka k bi došlo do lančane reakcije te bi moralo da se modifikuje narednih *n-k-1* blokova, što je u praksi skoro nemoguće. Težina modifikacije *k-tog* bloka raste dodavanjem novih.

U prethodnom paragrafu je spomenuto da blok ne sadrži svoj sopstevni heš koji ga jedinstveno identifikuje. Razlog je što ukoliko bi došlo do modifikacije nekog podatka u bloku došlo bi do izmene heša ukoliko bi ga ponovo računali, ali stari heš bi i dalje ostao zapisan u bloku te ne bi došlo do otkačivanja lanca jer bi potomak i dalje imao "validan" heš roditelja. Naravno verovatno bi modifikacijom bloka došlo do nekonzistenosti transankcija ali to bi bilo primetljivo posle određenog vremena. Zato blok ne sadrži svoj heš i svaki pokušaj izmene bloka će se lako primetiti i odmah odbaciti iz sinhornizacije.

2.1.3 Blok hešing algoritam (engl. *Block hashing algorithm*)

Za izračunavanje jedinstvenog heša za blok potrebni su sledeći parametri:

- Verzija ("ver")
- Heš roditelja ("prev_block")
- Heš Merkle korena ("mrkl_root")

- Vreme kreacije bloka (engl. *Timestamp*) ("time")
- Težina (engl. *Difficulty bits*) ("bits")
- Nonce ("nonce")

Koraci su sledeći [7]:

- Svaku vrednost transformisati u *little endian hex*[8] format
- Dobijene vrednosti konkatenirati
- Transformisati iz heksadecimalnog u binarni format
- Primeniti dvaput SHA256 funkciju
- Transformisati binarni u heksadecimalni format i obrnuti redosled
- Kraj algoritma. Rezultat je heš veličine 256 bita

```
>>> import hashlib
>>> header_hex = ("01000000" +
 "81cd02ab7e569e8bcd9317e2fe99f2de44d49ab2b8851ba4a308000000000000" +
 "e320b6c2fffc8d750423db8b1eb942ae710e951ed797f7affc8892b0f1fc122b" +
 "c7f5d74d" +
 "f2b9441a" +
 "42a14695")
>>> header_bin = header_hex.decode('hex')
>>> hash = hashlib.sha256(hashlib.sha256(header_bin).digest()).digest()
>>> hash.encode('hex_codec')
'1dbd981fe6985776b644b173a4d0385ddc1aa2a829688d1e0000000000000000'
>>> hash[::-1].encode('hex_codec')
'00000000000000001e8d6829a8a21adc5d38d0a473b144b6765798e61f98bd1d'
```

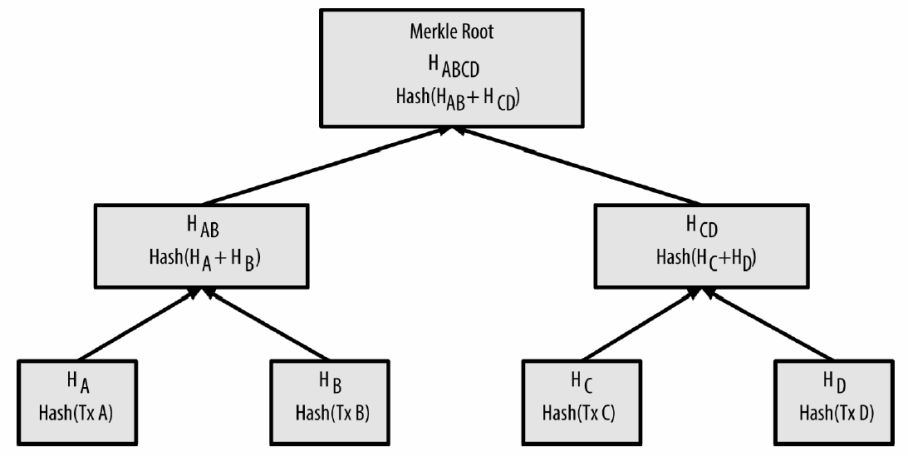
Slika 2: Blok hešing algoritam (engl. *Block hashing algorithm*)
(preuzeto sa *wikipedia.org*)

2.1.4 Merkleovo stablo (engl. *Merkle Tree, hash tree*) u *blockchain* strukturi

Merkleovo stablo (*slika 3*) je vrsta balansiranog binarnog stabla koje se koristi u kriptografiji za efikasnu proveru da li se neki podatak nalazi u listovima stabla, bez otkrivanja informacija o istim [8]. Jednostavnije rečeno ovo je struktura koja samo vraća *true* ili *false* odgovor u zavisnosti da li podatak koji je dat pripada tom stablu ili ne.

Merkle stablo je pomoćna struktura izgrađena nad blokovima konkretnih podataka (heširanim) koji su listovi u odnosu na stablo. Ukoliko postoji N heširanih podataka i treba da se verifikuje jedan podatak, za to će biti potrebno $\log_2 N$ iteracija. Bez ove strukture broj iteracija bi bio N.

[8]Format koji određuje u kom redosledu se bajtovi čuvaju. U slučaju little endian najniži bajt se postavlja na adresu koja je najniža tj ona koja ima najnižu vrednost



Slika 3: Merkleovo stablo (preuzeto sa *[6]*)

Broj podataka	Merkleovo stablo $O(\log_2 N)$	Provera linearnim putem $O(N)$
10	3	10
10^2	6	10^2
10^4	13	10^4
10^6	20	10^6
10^8	26	10^8

Tabela 3: Broj iteracija sa i bez Merkleovog stabla

Merkleovo stablo se koristi u *blockchain*-u za proveru validnosti transakcije ili radi provere postojanja u određenom bloku. Na *slici 3* je primer Merkleovog stabla čiji listovi čine transakcije. Za N transakcija potrebno je $2N$ heširanja kako bi se stvorilo stablo. Broj transakcija mora biti paran, ukoliko to nije slučaj poslednja transakcija se zapisuje još jednom.

Za heširanje se koristi double-SHA256[9] funkcija. Stablo se gradi tako što se heširaju sve transakcije, potom se za svaka dva heša vrši konkatenacija, te se ponovo primenjuje se heširanje i tako se dolazi do prvog nivoa čvorova; potom se isti postupak ponavlja rekurzivno samo sa čvorovima dok ne ostane jedan čvor koji neće imati svog para, taj čvor se naziva Merkleov koren i on je jedan od varijabli zaglavlja bloka (*tabela 2*)[6].

Primer kreiranja stabla (slika 2)

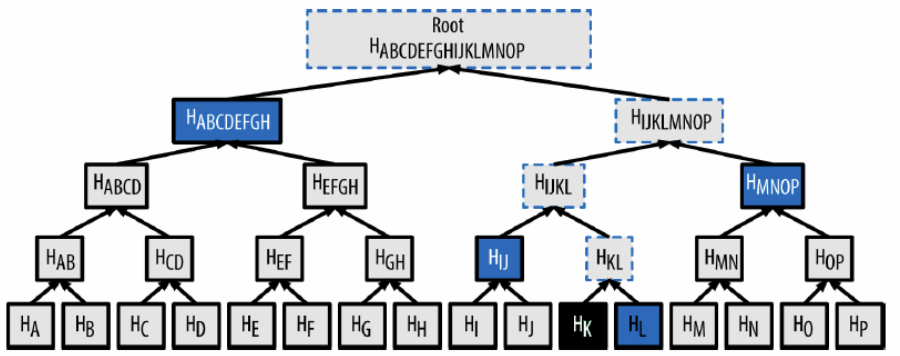
1. Heširati sve transakcije sa double-SHA256 funkcijom
(primer. H_A =double-$SHA256(T_xA)$)

[9]double-SHA256 funkcija je duplo primenjivanje SHA256 funkcije odnosno $SHA256(SHA256(X))$ gde je X ulaz.

2. Izvršiti konkatenaciju svaka dva heša, pritom ih heširati
(primer. H_{AB} =double-$SHA256(H_A + H_B)$)

3. Ponavljati korak *2* rekurzivno dok ne ostane jedan čvor koji će činiti koren stabla.
(primer. H_{ABCD} =double-$SHA256(H_{AB} + H_{CD})$)

Zadataka *full-node* klijenta da je generiše Merkleov put (engl. *merkle-path*) koji služi za validaciju transakcije. Merkleov put u ovom slučaju sastoji iz četiri heša od 32 bajta koja se nalaze u Merkleovom stablu i predstavlja putanju koja vodi do uspešne validacije [6].



Slika 4: Merkleov put (preuzeto sa *[6]*)

Na *slici 4* označeno plavom bojom je prikazan Merkleov put ($H_{ABCDEFG} + H_{IJ} + H_L + H_{MNOP}$), a crnom bojom je označena transakcija kojoj treba da se proveri pripadnost stablu. Sledi par koraka koji objašnjavaju kako se vrši potvrda transakcije pomoću Merkleovog puta sa *slike 4*.

Proveravaju se sledeći uslovi:

1. H_{KL} ==double-$SHA256(H_K + H_L)$
2. H_{IJKL} ==double-$SHA256(H_{IJ} + H_{KL})$
3. $H_{IJKLMNOP}$ ==double-$SHA256(H_{IJKL} + H_{MNOP})$
4. $H_{ABCDEFGHIJKLMNOP}$ ==double-$SHA256(H_{ABCDEFGH} + H_{IJKLMNOP})$

Ukoliko jedan uslov padne, transakcija H_K nije validna. Ovi nabrojani koraci pored validacije pokazuju i koliko ju je brzo verifikovati; za 16 transakcija izvršeno je 4 provera, te $log_2(N)$ efikasnost zaista jeste istina u praksi jer $log_2(16) = 4$.

2.2 Učesnik u *blockchain* mreži

Svaki računar koji se spoji sa *blockchain* mrežom je njen učesnik. Konkretno kod *bitcoin*-a postoje tri vrste učesnika.

***Full-node* učesnici** imaju kopiju celog *blockchain*-a odnosno kopiju svakog bloka i svih transakcija, što zauzima dosta memorije (više od 170GB). Oni konstatno osluškuju mrežu, i njihova uloga je da održavaju mrežu i proveravaju validnost svi transakcija.

***Lightweight* učesnici** su obični korisnici *blockchain* mreže koji nemaju veliku odgovornost kao *Full-node* klijenti, jer oni samo pristupaju mreži radi informacija o transakcijama koje se tiču njih, te nema potrebe da sadrže celu kopiju mreže. *Full-node* klijenti validiraju njihove transakcije.

Rudari (engl. *miners*) jesu *Full-node* učesnici, ali imaju dodatni posao koji pored validacije podrazumeva kreiranje i dodavanje bloka na *blockchain* mrežu, gde se mora poštovati njen konsenzus koji je u ovom slučaju dokaz posla (engl. *Proof-of-work*). Ovo je hardverski zahtven zadatak jer da bi dodali blok moraju rešiti intezivan računarski zadatak koji je *brute-force* [10] prirode. Za svaki dodati blok dobijaju zauzvrat nagradu.

2.3 Konsenzus algoritmi (engl. *Consensus algorithms*)

Blockchain mreža nema centralni autoritet, pa ne postoji posrednik koji će odlučiti o ubacivanju sledećeg podatka u mrežu. O tome odlučuju učesnici na osnovu predefinisanih pravila koje određuje konsenzus. Učesnici mreže se moraju na osnovu pravila konsenzusa dogovoriti da li će sledeći podatak biti ubačen u mrežu. Glavni problem je što postoje zlonamerni učesnici koji će pokušati dodati nevalidne podatke u mrežu radi njihove koristi. Pomoću konsenzus algoritma odlučuje se o tome ko će sledeći dodati blok, te nakom njenog ubacivanja svi učesnici moraju poštovati konzistentost mreže i primiti kopiju novog bloka (u slučaju da su *Full-node*). U ovom poglavlju su obrađena tri konzensus algoritma a to su:

1. Dokaz posla (engl. *Proof of Work*)
2. Dokaz posedovanja (engl. *Proof of Stake*)
3. Praktična vizantijska tolerancija greške (engl. *Practical Byzantine Fault Tolerance*)

2.3.1 Dokaz posla (engl. *Proof of Work*)

Prvi konsenzus algoritam primenjen na *blockchain* mrežu konkretno *bitcoin*. Rudari motivisani nagradom se takmiče u tome kome će prvom ostali učensici u mreži prihvatiti blok na osnovu potrošene energije tj najvećem dokazu posla. Uloga rudara je validacija novih transakcija i ubacivanje istih u blok, ali ovaj deo posla uopšte nije zahtevan i veoma

[10] Hardverski zahtevna radnja koja podrazumeva nasumično ili sistematično generisanje šifara, heševa u nadi pogađanja [9]

se brzo završava. Izuzetno procesorski zahtevan posao je rešavanje računarskog zadatka koji se svodi na pogađanje odgovarajućeg heša koji odgovara zadatom cilju (engl *target hash*).

Kako bi rudari ubacili svoj blok u glavni lanac oni moraju manipulisati podacima u zaglavlju bloka i primeniti Blok hešing algoritam (engl. *Block hashing algorithm*). Ukoliko je dobijeni heš manji od zadatog cilja (engl. *target hash*), smatra se da se rudar dokazao u radu i njegov blok će biti ubačen u glavni lanac i dobiće nagradu. Ukoliko se desi da rudar pogodi cilj, a zlonamerno je manipulisao transakcijama, njegov blok će vrlo brzo biti proveren od strane drugih učesnika mreže i biće proglašen nevalidnim, te će rudar izgubiti svoju nagradu i vreme koje je utrošio na rudarenje. Posle odbačenog bloka, prihvatiće se ubrzo blok drugog rudara ukoliko je pogodio heš i nije zlonamerno menjao transakcije [6].

Česta je zabuna kod rudarenja da posle kreiranog bloka, rudari pokušavaju da pogode heš menjajući samo pomoćnu varijablu nonce koja ima 2^{32} kombinacija. Današnji hardver je toliko jak da se kroz sve kombinacije varijable *nonce* prođe za par sekundi. Podsećanjem na Blok hešing algoritam (engl. *Block hashing algorithm*) primetićemo da se u obzir uzima celo zaglavlje bloka. Sagledajući da se *timestamp* u bloku menja u milisekundi i da ima isto 2^{32} kombinacija dolazimo do zaključka da postoji 2^{64} kombinacija heša koje treba isprobati, što je već ozbiljniji broj. Zbog ogromne popularnosti *bitcoin* težina se povećava iz dana u dan, pa su rudari počeli menjati *coinbase* transakciju kako bi uticali na merkleov koren i stigli do cilja (engl. *extranonce*), što povećava 2^{64} mogućih kombinacija više hiljada puta. *Coinbase* transakcija je prva u svakom bloku i ona isplaćuje rudarima nagradu na određenu adresu [10].

Ovaj algoritam je baziran na lutriji zato i spada pod posebnu katergoriju zvanom *lottery based*. Može se desiti da rudar iz prvog pokušaja pogodi heš i dobije nagradu, ali to je izuzetno retko i bilo je moguće na počecima bitcoina. Iako izuzetno efikasan, algoritam troši ogromne količine energije.

```
I am Satoshi Nakamoto0 => a80a81401765c8eddee25df36728d732...
I am Satoshi Nakamoto1 => f7bc9a6304a4647bb41241a677b5345f...
I am Satoshi Nakamoto2 => ea758a8134b115298a1583ffb80ae629...
I am Satoshi Nakamoto3 => bfa9779618ff072c903d773de30c99bd...
I am Satoshi Nakamoto4 => bce8564de9a83c18c31944a66bde992f...
I am Satoshi Nakamoto5 => eb362c3cf3479be0a97a20163589038e...
I am Satoshi Nakamoto6 => 4a2fd48e3be420d0d28e202360cfbaba...
I am Satoshi Nakamoto7 => 790b5a1349a5f2b909bf74d0d166b17a...
I am Satoshi Nakamoto8 => 702c45e5b15aa54b625d68dd947f1597...
I am Satoshi Nakamoto9 => 7007cf7dd40f5e933cd89fff5b791ff0...
```

Slika 5: SHA256 - Male promene na ulazu, daju velike na izlazu (preuzeto sa *[6]*)

Jedan od zlonamernih način kršenja konsenzusa je 51% napad (engl. *51% attack*), koji podrazumeva da određena grupa rudara poseduje 51% heš moći na celoj mreži. Na taj način će u većini slučajeva ta grupa uvek pobeđivati u zadatku pa ubacivati svoje blokove sa zlonamernim transakcijama koje mogu uzrokovati duplo trošenje (engl. *double spend*). Ovo je izuzetno težak i skup poduhvat, čak i sa 51% heš moći postoji mogućnost da napad bude neuspešan.

2.3.2 Dokaz posedovanja (engl. *Proof of Stake)*

Za razliku od dokaza posla koji daje nagradu onom učesniku koji reši računarski zadatak, ovaj algoritam bira učesnika na osnovu njegovog procenta posedovanja novčića u mreži. Na primer učesnik koji poseduje 5% novčića teoretski će imati 5% šanse da dobije priliku validiranja i dodavanja bloka. Algoritam u potpunosti izbacuje hardversku komponentu i rešava problem ogromnog trošenja energija.

Algoritam je pokupio dosta kritika jer po ovoj logici učesnici koji imaju najviše novčića dobijaju najveću šansu da prikupe nagradu. Ponovo kao kod dokaza posla postoji šansa za 51% napad gde ovog puta učesnik mora posedovati 51% novčića kako bi kreirao i validirao zlonamerne transakcije koje uzrokuju duplo trošenje (engl. *double-spend*). Pretpostavlja se da učesnik neće imati motivaciju za ovakav napad jer time rizikuje da drugi primete ovakve transakcije što može uzrokovati naglu prodaju novčića i strmoglav pad vrednosti [11].

U dokazu posedovanja ne postoji mogućnost stvaranja novih novčića, svi novčići su u mreži konačni i učesnici zarađuju samo na provizijama transakcija bez ikakvog rizika. Iz ovog razloga algoritam se susreće sa problemom koji se naziva "Ništa na ulogu" (engl. *Nothing at Stake*). Zbog toga što korisnici mogu samo dobiti nagradu za dodavanja bloka bez da išta ulože i izgube, postoji mogućnost da se ne ponašaju pošteno [12]. *Ethereum* grupa kao odgovor na ovaj problem je predložila protokol *Casper*.

Casper implementira takva pravila da će se bilo kakvo zlonamerno ponašanje kažnjavati. Ovako bi radila kombinacija dokaza posla i protokola *Casper*[13]:

- Svi učesnici odnosno validatori ulažu svoj deo kriptovalute
- Posle ulaganja, validatori kreću da validiraju blokove. U trenutku kada nađu potencijalni blok za lanac, stavljaju uložen deo na njega. Verovatnoća da će biti izabrani da dodaju blok zavisi od količine uloženog.
- Ukoliko validator bude izabran da doda blok, on dobija nagradu i ulog nazad.
- Bilo kakav pokušaj zlonamernog ponašanja kažnjava se odbijanjem bloka i oduzimanjem uloženog

2.3.3 Praktična vizantijska tolerancija greške (engl. *Practical Byzantine Fault Tolerance*

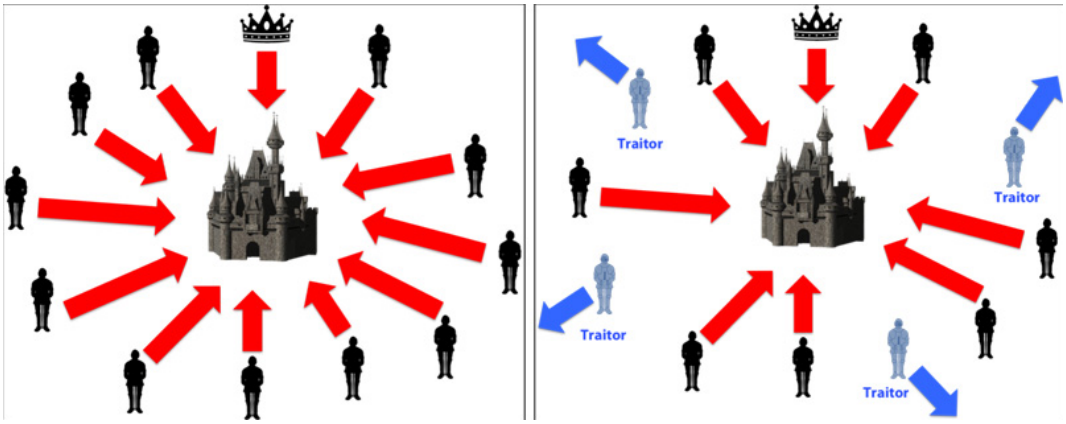
Praktična vizantijska tolerancija greške je algoritam koji rešava probleme karakteristične za distribuirane sisteme. Takvi problemu mogu nastati gašenjem ili zlonamernim ponašanjem nekih od čvorova koji čine sistem.

Problem je predstavljen kroz priču o grupi generala i Vizantijskoj armiji koja je okružila grada. Grad treba da se napadne, uloga generala je da smisle plan kada napasti, a kada se povući. Svaki general ima svoju diviziju i mogu komunicirati o napadu preko glasnika. Međutim ono što komplikuje ovaj problem je što postoji mogućnost da su neki generali izdajnici te mogu otežati donošenje konačne odluke. Bitno je da dođu do jedinstvene odluke jer greška u komunikaciji može dovesti do nekoordinisanog napada, što može uzrokovati poraz i gubitka ljudi u vojsci [14].

Generali ovde predstavljaju čvorove u *blockchain*-u. Algoritam glasi da za $3f+1$ čvorova, maksimalno f se mogu ponašati zlonamerno. Nakon što svaki čvor primi poruku, oni moraju

na osnovu unapred zadatih pravila validirati tu poruku koja može predstavlja blok ili skup transakcija. Nakon ovoga svaki čvor izvršava glasanje na osnovu validnosti poruke. Glasovi se prebrojavaju i ukoliko ih ima barem $f + 1$, poruka odnosno blok se konačno zapisuje u lanac [15].

Postoji mogućnost za postojanje više od $3f + 1$ čvorova što može pomoći sistemu da ima veću toleranicju na neuspeh (engl. *Fault tolerance*), ali to u praksi više usporava komunikaciju [15].



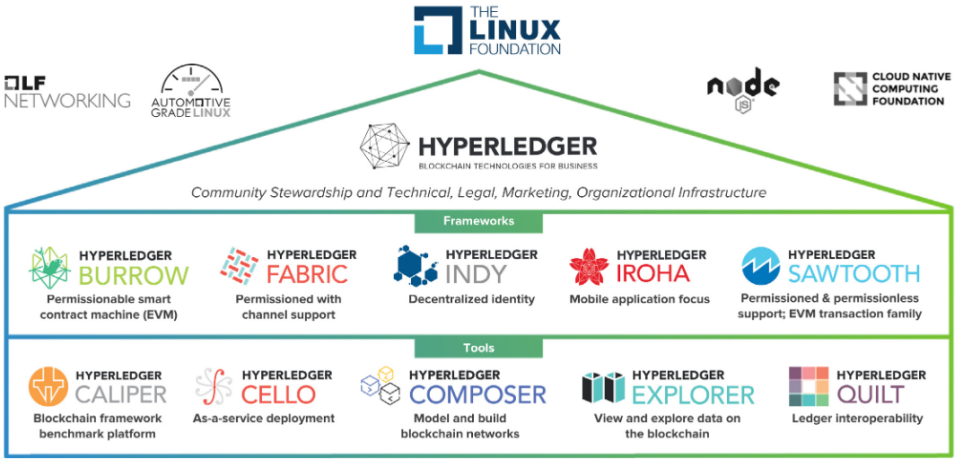
Slika 6: Prakitčna vizantijska tolerancija greške
(preuzeto sa *blockonomi.com*)

3 UVOD U HYPERLEDGER

Prva primena tehnologije *blockchain* je došla sa *Bitcoin*-om, i trenutno postoji velika zabuna da svaki *blockchain* ima i svoju kriptovalutu. Međutim to nije tačno, jer *Bitcoin* je grubo rečeno samo formiran skup podataka sačuvan *blockchain*-u. *Blockchain* tehnologija predstavlja novu vrstu distribuiranih baza podataka koja nema centralni autoritet, te se industrija zainteresovala za primenu istih u poslovne svrhe. Tako je osnovan *Hyperledger* projekat.

3.1 O projektu *Hyperledger*

Hyperledger je projekat otvorenog koda (engl. *open-source*) koji je započela Linux fondacija (engl. *Linux Foundation*) u decembru 2015 godine. Cilj ovog projekta je udruženje industrijskih giganata i Linux zajednice radi razvoja i primene *blockchain* tehnologija [16]. Jedni od glavnih igrača u industriji koji doprinose ovom projektu su: *Airbus*, *American Express*, *Baidu*, *Cisco*, *IBM*, *Intel*, *SAP* i drugi [17].



Slika 7: *Hyperledger* familija (preuzeto sa *hyperledger.org*)

Ukratko o projektima započetih od strane *Hyperledger*-a:

- *Hyperledger Fabric* je platforma za kreiranje *blockchain* rešenja. Odlikuje ga modularna arhitektura koja omogućava izmenu konsenzus algoritma, servisa članstva i drugih komponenti. Omogućava pisanje pametnih ugovora koji sadrže pravila ponašanja sistema. Ovaj rad posvećuje posebno poglavlje *Fabric*-u [18].
- *Hyperledger Indy* je platforma za kreiranje *blockchain* rešenja koja imaju svoju primenu u čuvanju digitalnih identiteta. *Indy* pruža potrebne alate, biblioteke i komponente za stvaranje decetralizovanog sistema koji omogućavaju privatnost, verifikovanje digitalnih identiteta i dr [18].
- *Hyperledger Iroha* je platforma koja omogućava uključivanje infrastrukturnih projekata u *blockchain* [18].

• *Hyperledger Sawooth* je modularna platforma za kreiranje *blockchain* rešenja koja omogućavaju čuvanju digitalnih zapisa (podaci o vlasništvu). Koristi konsenzus algoritam zvani Dokaz po isteklom vremenu (engl. *Proof of elapsed time*) [18].

• *Hyperledger Composer* je alat koji omogućava jednostavno i brzo kreiranje *blockchain* aplikacija. Softversko rešenje ovog rada je napisano pomoću ovog alata. Više o njemu u narednim poglavljima [18].

• *Hyperledger Explorer* je alat koji omogućava parsiranje i jednostavan prikaz informacija sa *blockchain*-a [18].

• Postoje i drugi projekti kao što su *Caliper*, *Cello*, *Quilt* i tako dalje [18].

3.2 Razlike između javnih i poslovnih *blockchain* rešenja

Javna (engl. *permissionless*) *blockchain* rešenja nemaju svog vlasnika i njeno stanje održava javnost preko konsenzusa kao što su Dokaz posla (engl. *Proof of work*), Dokaz posedovanja (engl. *Proof of stake*) i dr. Svaki korisnik može pristupiti mreži i imati svoju ulogu bilo to *Full-node*, *Lightweight* učesnik i drugi. Iz razloga što održavanje ovakve mreže zahteva određene računarske resurse mora postojati neki oblik vrednosti kao što je kriptovaluta koja bi motivisala korisnike da ne odu. Podaci sa javnog *blockchain*-a su dostupni za javnost, sve transakcije su transparentne i bez problema sa mogu videti podaci o adresama i koliko kriptovaluta ima u istim. Tok transakcija se lako može pratiti preko određenog softvera ili čak ručno. Ono što je privatno na ovoj mreži su identiteti osoba koje stoje iza transakcija [19].

Poslovna *blockchain* rešenja se koriste u industriji. S obzirom da se ovakve mreže uglavnom nalaze na privatnim serverima, ne može im svako pristupiti. Svaki učesnik da bi pristupio mora imati definisan identitet i pravo pristupa tj ulogu koja mu definiše dozovljene načine doprinošenja mreži. Takođe omogućava takvu zaštitu da se u svaku transakciju upiše i njen autor, te svaki pokušaj zlonamernog ponašanja može biti lako otkriven. Ovakva vrsta mreže se zove i *blockchain* sa pravom pristupa (engl. *permissioned*). Čvorovi u poslovnim *blockchain* mrežama se ne takmiče u rešavanju računarskih zadataka niti u prisvojavanju kriptovaluta. Ovi tipovi mreža nemaju kriptovalutu niti neku vrstu nagrade koja se dobija pri kreiranju novog bloka. Poenta njihovog konzenusa je usaglašavanje oko sledećeg upisa bloka u mrežu i tolerancija na grešku (engl. *Byzantine Fault Tolerance*[11]), ukoliko neki čvor padne ili počne da šalje zlonamerne transakcije, sistem će moći da održi svoj rad zapisivajući samo one transakcije koje su validne. Na tabeli 4 se nalazi poređenje različitih mreža.

[11]Za sistem kažemo da je *BFT (engl. Byzantine Fault Tolerant)* ukoliko može nastaviti da funkcioniše bez normalnog rada određenog broja čvorova. Ne treba mešati *BFT* i *PBFT* (engl. *Practical Byzantine Fault Tolerance*), iz razloga što je *BFT* svojstvo sistema tj njegovog konzensus algoritma, dok je *PBFT* konkretan konsenzus algoritam.

	Bitcoin	*Ethereum*	*Hyperledger platform*
Kriptovaluta	Da	Da	Ne
Kontrola pristupa	Ne	Ne	Da
Pseudo-anonimnost	Da	Da	Ne
Otpornost na izmenu	Da	Da	Da
Modularnost	Ne	Ne	Da
Konsenzus algoritam	PoW	PoW (prelazak na PoS)	po izboru (Kafka, Solo itd)

Tabela 4: Poređenje različitih mreža

3.3 *Hyperledger Fabric*

Javna *blockchain* rešenja kao što su *bitcoin* i *ethereum* svojim nedostatkom funkcionalnosti i fleksibilnost nisu mogli da zadovolje potrebe industrije, ali su bile podstrek da se napravi platforma *Fabric* koja je popunila te nedostatke. *Hyperledger Fabric* je platforma otvorenog koda koja omogućava kreiranje odgovarajućeg *blockchain* rešenja radi primene u industriji. Mogućnosti koje čine ovu platformu dobrim rešenjem od javnih *blockchain*-a je poverljivost, skalabilnost i modularna arhitektura [19].

3.3.1 Uvod u funkcionalnosti platforme

Fabric se posebno izdvaja kao platforma jer pruža servis članstva (engl. *Membership service provider*). Ono omogućava proveru identiteta učesnika na mreži pre pristupa istoj. Takođe postoji mogućnost kontrole prava pristupa gde se svakom identitetu može dodeliti određena uloga koja određuje prava čitanja i pisanja na mreži. Na primer određenom članu se može zabraniti kreiranje specifičnih transakcija ili čitanje istih.

Fabric dozvoljava privatnu komunikaciju između čvorova (engl. *node*), a to se radi preko kanala (engl.*channels*). Kanal predstavlja grupu u koju su upisani određeni čvorovi. Ti čvorovi imaju mogućnost privatnog komuniciranja odnosno kreiranja takvih transakcija koje će biti vidljive isključivo članovima grupe, odnosno kanala. Ova funkcionalnost ima veliku primenu. Na primer, firme u mreži mogu kreirati kanal, kako bi razmenjivali materijalna dobra po boljem dogovoru između sebe, te bi želeli da dogovori takve prirode ne budu vidljivi firmimama, odnosno njihovim čvorovima van kanala. To znači da će samo čvorovi koji pripadaju kanalu imati posebnu kopiju *ledger*-a njihovih transakcija.

U trenutnom poglavlju navedeno je da članovi mogu biti ograničeni na izvršavanje više tipova transakcija. S obzirom da poslovni sistemi mogu biti izuzetno kompleksni, pri projektovanju takvog sistema *Fabric* daje mogućnost kreiranja više tipova transakcija. To se radi pomoću Koda lanca (engl. *chaincode*). Kod lanca čini poslovnu logiku transakcije, i ono može naložiti različite uslove pre konačnog izvršenja transakcije odnosno zapisa u lanac. Na primer da bi jedna strana poslala drugoj strani novac, pre toga mora nastupiti provera količine novčanih sredstava na računu prve strane.

Arhitektura poslovnih sistema može dostići veliki nivo kompleksnosti, te platforma na kojoj se gradi poslovni sistem mora biti modularne arhitekture. *Fabric* upravo ispunjava te uslove jer njena arhitektura dozvoljava zamenu različitih komponenti mreže kao što su enkripcija, konsenzus, servis sa kontrolu prava pristupa.

3.3.2 Arhitektura *Fabric* mreže

Za formiranje *Ledger*-a odnosno *Fabric* mreže najpre je potreban dogovor između entiteta koji žele da doprinesu istoj. Entiteti mogu biti različite organizacije, firme, privatna lica i drugo, ali suština je da imaju zajedničku delatnost ili motivaciju. U daljem tekstu entiteti su organizacije. Organizacije najpre formiraju udruženje (engl. *consortium*) u okviru kog ugovaraju pravila (engl. *policy*) radi vođenja mreže [19]. Pravila se po zajedničkom dogovoru mogu naknadno menjati [21].

Sledi detaljniji opis elemenata arhitekture u *Fabric*-u prema [20] i [21]:

- Digitalno dobro (engl. *asset*) čini vrednost u transakcijama između organizacija. *Fabric* dozvoljava definisanje više tipova digitalnih dobara u okviru mreže, bila to materijalna ili novčana sredstva. Ono je prezentovano na mreži u obliku JSON ili binarnog formata.

- Kod lanca (engl. *chaincode*) je odgovoran za biznis logiku mreže, te ga možemo smatrati i pametnim ugovorom (engl. *smart contract*). Korisnici kroz klijentsku aplikaciju biraju odgovarajuću komandu koja će se izvšiti nad *ledger*-om. Komanda korisnika pokreće određeni kod lanca koji može izvršavati čitanje ili pisanje u *ledger*. Kod lanca se piše u jeziku *Go* ili *NodeJS*. Pri njegovom pisanju ono mora implementirati interfejs koji se sastoji iz funkcija *Init*, *Query*, *Invoke*. Svaka ima svoje uloge:

  - *Init* postavlja početna podešavanja prilikom prvog instanciranja odnosno, postavljanja njenog koda lanca u mrežu.
  - *Invoke* šalje transakciju koja se posle upisuje u blok.
  - *Query* čita podatke sa *ledger*-a. Ne vrši izmene te se takve transakcije ne zapisuju.

  Kod lanca je sličan jednoj *backend* funkciji veb aplikacije koja vrši izmenu, čitanje ili zapis u bazu podataka.

- Raspoređivački servis (engl. *ordering service*) je vrsta čvora (engl. *node*) koja raspoređuje nadolazeće transakcije u blok. Transakcije se uglavnom raspoređuju po redosledu kojem su došli (engl. *first-come first-serve*.Njihova funkcije je slična onoj koju imaju rudari u *Bitcoin*.

- *Peer* su čvorovi mreže koji pripadaju jednoj organizaciji. Svaki čvor čuva kopiju lanca. Čvor može imati više uloga u mreži. Te uloge su:

  - Odobravajuću čvor (engl. *Endorsing peer*) ima ulogu u izvršavanju koda lanca (engl. *chaincode*) odnosno transakcije i vraćanja klijentu odgovor o uspešnosti iste.
  - Validirajući čvor (engl. *Committing peer*) ima ulogu validiranja dobijenog bloka transakcija i dodavanje istog na kopiju svog *ledger*-a.

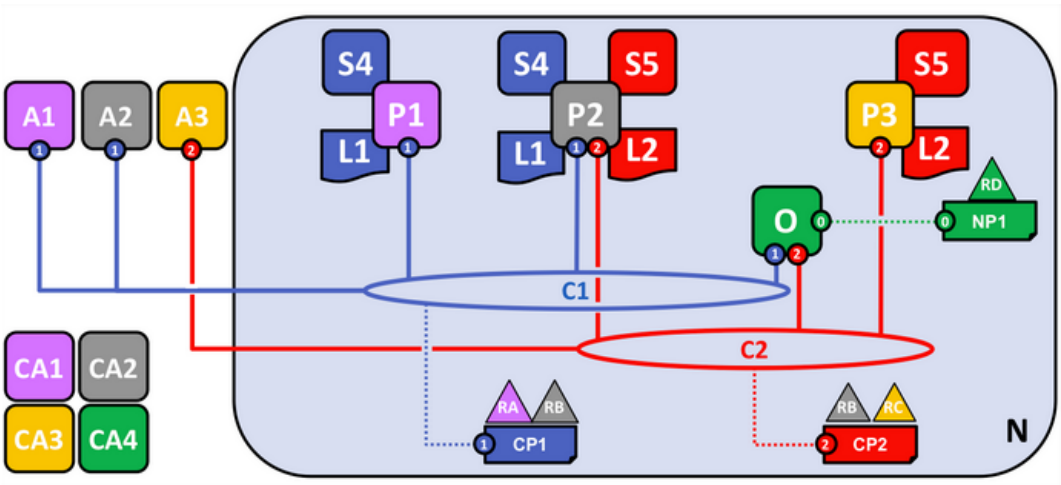
- *Ledger* se sastoji iz dva dela. Jedan deo oslikava trenutno stanje u sistemu (engl. *World state*) koje predstavlja bazu podataka nad kojom se vrše operacije modifikacija

na osnovu informacija u dolazećim transakcijama. U drugi deo oslikava *blockchain* odnosno dnevnik transakcija (engl. *Transaction log*) koji sadrži istoriju svi promena koje su dovele sistem do trenutno stanja. Za čuvanje konkretnih podataka *Fabric* podržava dve *NoSQL* baze, *LevelDB* i *CouchDB*.

- Kanali (engl. *channels*) su privatni načini komunikacije između više čvorova. Sve transakcije izvršene u okviru kanala su validirane od strane čvora u istom. Čvorovi imaju posebnu kopiju *ledger*-a koji sadrži samo one transakcije izvršene u okviru njihovog kanala. Za kanale je karakteristična dinamičnost jer čvorovi se mogu pridružiti neograničenom broj kanala, a mogu i iste napustiti. Pošto su kanali privatne prirode, novi čvorovi im se mogu pridružiti samo ukoliko to dozvoljava servis članstva (engl. *Membership service provider*). Svaki kanal može imati svoje kodove lanca (engl. *chaincode*).

Odluka organizacija da naprave *Fabric* mrežu zahteva početno ulaganje od strane svake. U zavisnosti od međusobnog dogovora svaka je dužna obezbediti određenu količinu čvorova bilo to *peer*-ovi ili raspoređivački servisi. Ti čvorovi će biti njihovi predstavnici u mreži. Sledi primer prema [21] i *slici 8*:

Organizacije *RA*, *RB*, *RC* i *RD* su odlučile napraviti mrežu. Boje na slici slikovito označavaju vlasništvo organizacije nad čvorovima, klijentskim aplikacijama i sertifikata za autorizaciju. Najpre svaki kanal ima svoj sertifikat koji mu daje pristup mreži ili kanalu. Organizacijama *RA*, *RB*, *RC* i *RD* pripadaju sertifikati *CA1*, *CA2*, *CA3* i *CA4* redom. Organizacija *RD* poseduje na mreži Naređivački servis *O* i polisu mreže koja sadrži uslove na osnovu kojih određuje preduslove koje mora ispuniti transakcija pre zapisa u blok. *RA* i *RB* su se dogovorili da naprave kanal *C1* i polisu kanala u kojem je zapisano udruženje između njih (na primer samo oni smeju zapisivati transakcije u kanalu). *RA* je doprinela kanalu sa *peer*-om *P1*, a *RB* sa *P2*. *Peer*-ovi kanala *C1* sadrže kopiju privatnog *ledger*-a *L1* i koda lanca *S4*. *RA* i *RB* su dali redom klijentskim aplikacijama *A1* i *A2* da pristupe kanalu *C1*. Ista pravila važe za kanal *C2*.



Slika 8: Primer organizacije *Fabric* mreže (preuzeto sa *hyperledger.org*)

3.3.3 Konsenzus u *Fabric*-u

Sve validne transakcije u bloku koji je tek upisan u lokalni *ledger* čvora, se mora uspešno sinhronizovati kroz celu mrežu ili kanal. Odnosno svi čvorovi kroz međusobnu komunikaciju prolaze proces odlučivanja o tome da li će upisati novi blok u svoje lokalne *ledger*-e. Ovakav proces koji omogućava red i sinhornizaciju podataka u mreži je konsenzus. *Fabric*-ova modularna arhitektura dozvoljava različite konsenzus algoritme, u zavisnosti od potrebe sistema. Generalno rečeno, postizanje konsenzusa u *Fabric*-u se postiže kroz tri koraka:

1. Odobravanje transakcija (engl. *endorsement*) od strane čvorova preko polise mreže ili kanala i predefinisanih uslova u kodu lanca (engl. *chaincode*). Na primer polisa mreže može biti da za svaku odobrenu transakciju mora postojati m od n odobrenja od strane čvorova.

2. Uređivačka faza (engl. *ordering phase*) koja prima sve odobrene transakcije i postavlja ih u blok po određenom redosledu prema polisi mreže.

3. Validacija (engl. *validation*) vrši validaciju celog bloka, i ukoliko je sve prema pravilima odobrava dodavanja bloka u *ledger*.

Fabric podržava dva konsenzus algoritma: *SOLO* i *Kafka*. Planira se podrška *PBFT* algoritma.

3.3.4 Tok transakcije

U ovom potpotpodglavlju je izučen tok transakcije od slanja do njenog smeštanja u lanac. Neophodno je podesiti sve čvorove, polise, kanale pre mogućnosti izvršenja transakcije u *Fabric* mreži [23].

(1) Klijent šalje zahtev za transakciju, to može biti na primer zahtev za kupovinu materija-
lnog dobra. Zahtev se šalje određenim odobravajućim (engl. *endorsing*) čvorovima u zavisnosti od polise mreže ili kanala. Polisa može nalagati da zahtev moraju potpisati čvorovi organizacija na čije internno stanje utiče isti. Čvorovi potpisuju zahtev, te ga šalju nazad klijentskoj aplikaciji čiji je sledeći korak da potpiše i kreira preporuku transakcije (engl. *transaction proposal*) za zapis u *ledger*.

(2) Odobravajući čvorovi verifikuju preporuku transakcije koju su dobili od strane klijenta. Proverava se validnost sledećih stavki prema [23]:

- Struktura preporuke transakcije.
- Provera jedinstvenosti (na osnovu heša) preporuke transakcije, tj da li je ista preporuka poslata pre.
- Verifikacija potpisa pomoću servisa članstva.
- Provera da li klijent ima prava da na operaciju koju zahteva nad *ledger*-om.

Nakon uspešne provere, odobravajući čvorovi uzimaju argumente preporuke transakcije i koristi iste za pokretanje odgovarajućeg koda lanca. Kod lanca na osnovu trenutnog

stanja *ledger*-a kreira odgovor o uspešnosti transakcije i strukturu *read-write set.Read-write set* struktura sadrži dva skupa zvana *read set* i *write set*. Odgovor se šalje klijentskoj aplikaciji. *Read set* predstavlja skup jedinstvenih ključeva i njihovih verzija odnosno podatka o poslednjoj izmeni. *Write set* predstavlja skup jedinstvenih ključeva i njihovih novih vrednosti koje se trebaju zapisati u *ledger*. Ukoliko postoje zahtevi za brisanje, oni se nalaze u *write set* skupu, samo što umesto nove vrednosti u paru ključ-vrednost sadrže oznaku (engl. *isDelete*) za brisanje [24]. Ovom operacijom se ne vrši izmena *ledger*-a, nego se simulira, odnosno analizira kakvo bi stanje bilo da je ista operacija izvršena.

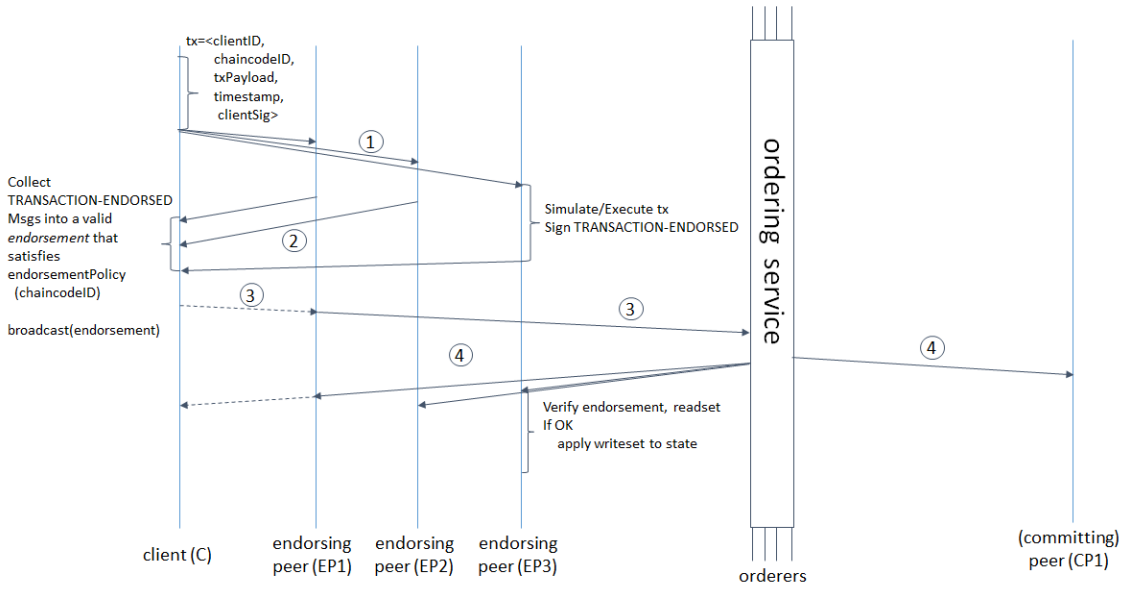
(3) Klijentska aplikacija od odobravajućih čvorova dobija odgovore[12] (engl. *response*). Odgovori od odobravajućih čvorova se porede, odnosno proverava se da li su *read-write* skupovi identični i validiraju se potpisi čvorova. Sledeći korak zavisi od tipa zahteva koji je klijent poslao u koraku (1). Ukoliko se zahtev vrši samo radi očitavanja podataka, onda se taj tip transakcije obično ne zapisuje u *ledger*. Klijent može na osnovu odgovora od odobravajućih čvorova pročitati potrebne podatke. Međutim ukoliko je zahtev poslat radi zapisa u *ledger*, onda klijent odgovore koje je dobio spakuje u konkretnu transakciju i pošalje istu uređivačkom servisu (engl. *ordering service, ordering peers*).

(4) Uređivački servis prima pristigle transakcije i tipično ih raspoređuje u blok po kanalu i hronološki.

(5) Uređivački servis šalje svim čvorovima u mreži kreiran blok. Čvorovi primaju blok i proveravaju njen sadržaj radi provere da transakcijama nije naknadno manipulisano pre smeštanja u blok.

(6) Nakon prethodne provere, čvorovi dodavaju blok u svoj lokalni *ledger*. Time se transakcija završava. Ukoliko je u toku ovih koraka ustanovljeno da neka transakcija nije validna, ona se takođe zapisuje u *ledger*, ali ne menja njeno stanje. Na *slici 9* je prikazan UML diagram događaja toka transakcije.

[12] Sadrži podatak o uspešnosti transakcije i *read-write* skup



Slika 9: UML diagram događaja toka transakcije
(preuzeto sa *stackoverflow.com)*

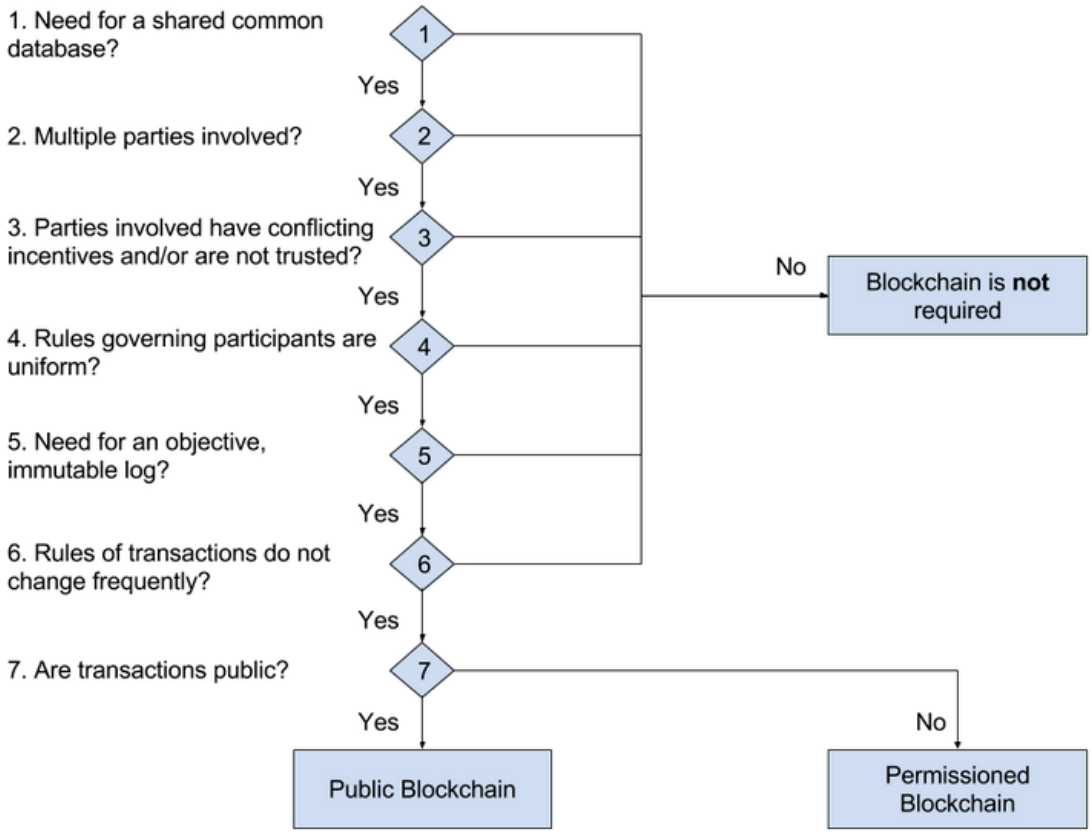
3.4 Poređenje *Fabric* mreže i relacione baze podataka u primeni

Koncept relacione baza podataka je star pet decenija. Kako je vreme prolazilo od njenog nastanka industrija ga je razvijala i ispostavilo se kao najefikasnije rešenje za skladištenje podataka. Dok je *blockchain* tehnologija relativno mlada i tek nalazi primenu u industriji. Međutim *Hyperledger* zajednica je napravila *Fabric* platformu i ona se aktivno razvija kako bi *blockchain* polako našao svoje mesto u industriji. *Fabric* mreža i relaciona baza podataka (u dalje tekstu RBP) skladište podatke. Da li će *Fabric* preuzeti mesto RBP?

Neće, iz razloga što oba koncepta imaju svoje prednosti i mane koje treba analizirati, te u zavisnosti od toga koje su odlike potrebne sistemu, doneti odluku. RBP ima ogromnu prednost u performansama kada je u pitanju čitanje ili modifikacija, dok *Fabric* teško da će stići do takvih performansi iz razloga što je decentralizovan i bilo kakva izmena nad *ledger*-om zahteva komunikaciju između čvorova, autentikaciju sa istim i mnošto koraka već opisanih u poglavlju 3.3.4 Tok transakcije. Međutim ukoliko postoji potreba da više organizacija želi da napravi zajedničku bazu u koju će upisivati podatke o zajedničkim poslovanjima, onda *Fabric* pobeđuje. To je zato što pravljenje zajedničkog RBP bi zadalo dosta problema, iz razloga što je centralizovan i mnogo je lakše zlonamerno izmeniti podatake o poslovanju. U *Fabric* mreži svaka organizacija bi imala svoje čvorove koji bi međusobno razmenjivali transakcije po konsenzus algoritmu i upisivali ih u svoje *ledger*-e. Još jedna prednost je što *legder* ima dosta robustnu strukturu podataka otpornu na izmene i pruža istoriju svi uspešnih i neuspešnih transakcija. Vrlo je komplikovano za RBP oponaša strukturu *ledger*-a, te da skladišti sve transakcije i pravi kopije svojih pređašnjih stanja koje će organizovano čuvati.

Idealno bi bilo koristiti RBP ukoliko postoji jedna organizacija ili više njih sa velikim

poverenjem i potrebno im je rešenje koje će pružati velike performanse i masovno skladištiti podatke. *Fabric* je odlično rešenje ukoliko više organizacija nema preveliko poverenje jedno u drugo, ali žele da sarađuju i potrebna im zajednička baza podataka sa istorijom svojih transakcija. Na taj način neće imati veliku brigu o mogućim zlonamernim izmenama ili kvara jednog servera odnosno čvora koji je učesnik u mreži. Idealno bi bilo da u *ledger*-u transakcije sadrže samo ključne informacije vezane za poslovanje, iz razloga što velike transakcije opterećuju *ledger* i njegove čvorove, te dolazi do smanjenja performansi. Na *slici 10* je prikazan jednostavan diagram koji pokazuje kako napraviti izbor između RBP i *Fabric*.



Slika 10: *Fabric* ili RBP (preuzeto sa *edx.org*)

4 OPIS SOFTVERSKOG REŠENJA

U ovom poglavlju opisani su korišćeni alati i softversko rešenje namenjeno za primenu *blockchain*-a u trgovini. Softversko rešenje je realizovano na *blockchain* platformi *Hyperledger Fabric* uz pomoć alata *Hyperledger Composer* radi bržeg razvijanja.

4.1 Opis alata

Hyperledger Composer je alat koji omogućava lako i brzo kreiranje *blockchain* rešenja na platformi *Hyperledger Fabric*. Ovaj alat čini skup više alata koji omogućavaju brzo kreiranje poslovne logike za *Fabric* mrežu bez mnogo tehničkog znanja o samom *blockchain*-u. *Composer* omogućava da kroz samo modelovanje učesnika, digitalnih dobara i definisanje transakcija vezanih za njih napravi *blockchain* mrežu na *Fabric* platformi. Cela arhitektura *Composer*-a se bazira na četiri fajla:

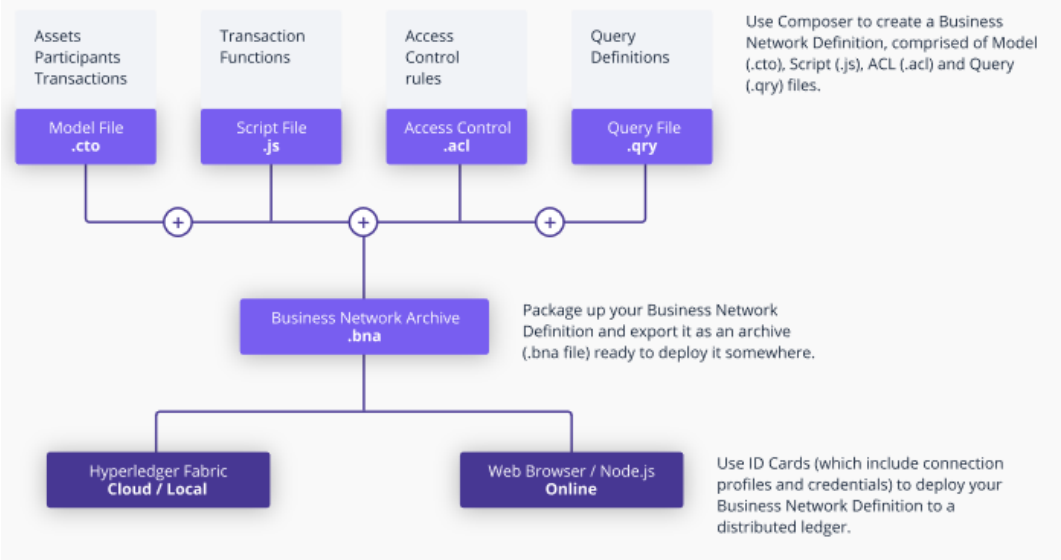
- Model fajl (engl. *Model File (.cto)*) definiše digitalna dobra, učesnike u mreži i transakcije. Kao jezik modelovanja se koristi *Hyperledger Composer modeling language* koji podseća na objektno orijentisane jezike iz razloga što dozvoljava rukovanje primitivnim tipovima kao što su *String*, *Integer*, *Double*, *Long*, *DateTime*, *Boolean* i kreiranje složenijih kao što su enumeracije, nizovi, klase i koncepti. Koncepti su apstraktne klase. Podržano je nasleđivanje između više klasa i apstraktnih klasa. Svaku klasu koja čini participanta ili digitalno dobro potrebno je obeležiti jedno od njihovih polja koje će biti jedinstveni identifikator u mreži. Postoji mogućnost i definisanja regularnih izraza za primitivne tipove.

- Skript fajl (engl. *Script File (.js)*) definiše poslovnu logiku već deklarisanih transakcija u model fajlu. Za pisanje logike se koristi *JavaScript* programski jezik.

- Upitni fajl (engl. *Query File (.qry)*) definiše upite pomoću kojih se na osnovu predefinisanih upita čitaju podaci iz *blockchain*.

- Fajl kontrole pristupa (*Access Control File (permissions.acl)*) definiše kontrolu pristupa nad participantima u mreži. Postoji mogućnost ograničavanja na nivou kontrole toka (engl. *if conditions*), uloga, ili konkretnog participanta. Na primer participant ne može vršiti transakciju materijalnih dobara kojih ne poseduje.

Composer gore navedene fajlove može zapakovati u biznis fajl (engl. *Business Network Archive (.bna)*) koji će činiti celu arhitekturu mreže. Taj fajl se može koristiti za brzo pokretanje *Fabric* mreže na klaudu ili lokalu. Na *slici 11* se vide komponente potrebne za *Composer*.



Slika 11: Fajlovi potrebni za *Composer* (preuzeto sa *hyperledger.org*)

Composer nudi i veb rešenje zvano *Playground* koje omogućava testiranje mreže u veb pretraživaču. Omogućava definisanje modela mreže preko četiri navedena fajla, izvršavanje upita, izvršavanje transakcija i pregled istorije istih.

Postoje alati kao što su *Swagger* i *Yeoman*. *Swagger* koji osnovu biznis fajla generiše *REST API* metode kojim klijentske aplikacije mogu lako pristupiti i time na pregledan način vršiti čitanje i izmene sa *ledger*-a. Dok *Yeoman* generiše skelet *Angular* aplikaciju na osnovu postojećih *REST API*.

4.2 Opis implementacije

U ovom potpoglavlju je detaljno opisana implementacija softverskog paketa realizovanog pomoću alata *Composer* i *Playground*. Implementacija ovog paketa spada u oblast trgovine, konkretno sa materijalnim dobrima, u daljem tekstu proizvodima. Jedna od glavnih problematika u trgovini različitih proizvoda je originalnost i način ophođenja prethodnih vlasnika prema istom. Zbog ovih problema kompanije gube milijarde godišnje, a potencijalni kupci završe sa falš proizvodom ili ukoliko je proizvod polovan bivaju lagani o pređašnjem broju vlasnika. Kao što je već bilo spomenuto u poglavlju 3.4, relaciona baza nije toliko efikasna u decentralizovanom čuvanju velikih transakcionih logova. *Fabric* ovde ulazi kao spasioc jer bi omogućio udruživanje kompanija koje imaju ovaj zajednički problem. Kompanije bi mogle da naprave svoju *Fabric* mrežu koja bi omogućila kupcima da se uvere u originalnost proizvoda bio on nov ili polovan. U daljem tekstu sledi opis implementacije.

4.2.1 Participanti i pomoćni tipovi

Na *slici 12* se mogu videti definicije participanta (engl. *participants*) i pomoćnih tipova (engl. *enumerations*).

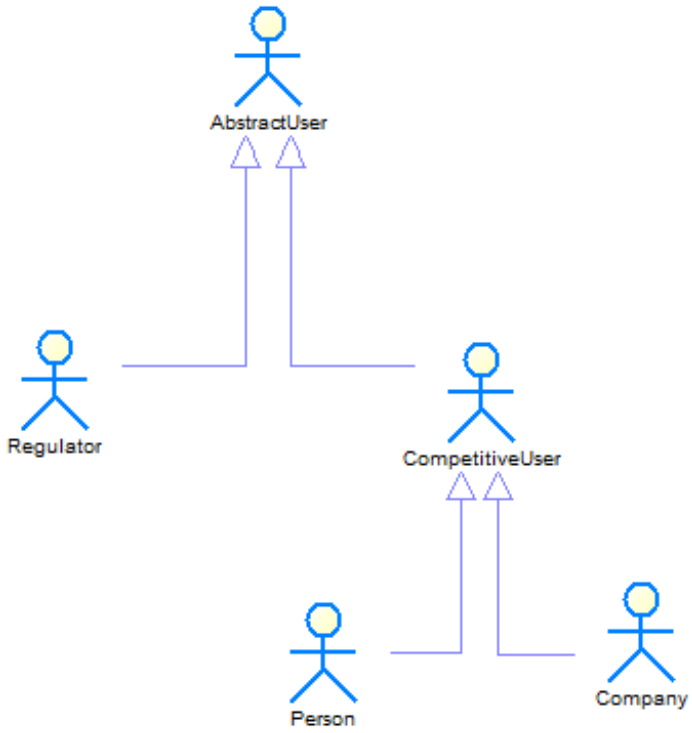
```
enum UserStatus {
  o NOT_VERIFIED
  o VERIFIED
  o PENALTY
}

enum UserType{
  o Company
  o Person
}

enum ItemListStatus{
  o WAIT
  o SOLD
}

abstract participant AbstractUser identified by username{
  o String username
}

participant Regulator extends AbstractUser{
  o String name
}

abstract participant CompetitiveUser extends AbstractUser {
    o UserStatus userStatus default = "NOT_VERIFIED"
    o Long balance default = 0
    o UserType type
}

participant Person extends CompetitiveUser{
  o String firstname
  o String lastname
}

participant Company extends CompetitiveUser{
  o String companyName
}
```

Slika 12: Model participanata i pomoćnih tipovi u mreži

Tip *UserStatus* ima tri stanja: nije verifikovan (*NOT_ VERIFIED*), verifikovan (*VERIFIED*) i penal (PENALTY). U daljem tekstu radi boljeg razumevanja, ova stanja su detaljno objašnjena.

Tip *UserType* ima dva stanja: kompanija (*Company*) i osoba (*Person*), ovaj tip je zapravo pomoćni tip radi koji je kreiran radi lakše implementacije biznis logike iza transakcija, te nije od velike važnosti detaljnije objašnjavanje.

Tip *ItemListStatus* ima dva stanja: čekanje (*WAIT*) i prodato (*SOLD*), gde prvo stanje označava da izlistani *Item* nije prodat, dok drugo označava upravo suprotno.



Slika 13: *UML use case* dijagram klase participanata u mreži

Na *slici 13* je UML use case dijagram nasleđivanja koji prikazuje hijerarhiju između participanata u mreži.

AbstractUser sadrži polje *username* koji ga jedinstveno identifikuje. Svrha ovog participanta je apstrakcija koja dovodi do lakše implementacije transakcija (skript fajl) i kontrole pristupa (fajl kontrole pristupa).

Regulator sadrži polje *name* koje može označavati ime legalne institucije. Uloga regulatora je verifikovanje novih članova u mreži. Bez verifikovanja članovi imaju samo mogućnost čitanja. Ukoliko dođe do zlonamernog ponašanja, on ima prava da postavi penal učesniku. Penal zamrzava sve račune učesnika, dok ne dođe do skidanja istog. U rešenju za produkciju, verifkacija bi se mogla vršiti popunjavanjem formi sa poverljivim podacima učesnika (engl. *KYC - Know you customer/client*[13]).

[13]Proces verifikovanja privatnog ili poslovnog lica radi sticanja poverenja u daljem poslovanju.

CompetitiveUser spada u učesnika mreže koji može biti privatno ili poslovno lice. Sastoji se iz polja: *balance*, *userStatus* i *type*. *Balance* označava koliko ima novčanih sredstava na računu. *Type* vrednost nije toliko važna za ovaj opis. *UserStatus* je vrednost od koje zavisi da li učesnik može da: čita ili piše po mreži (verifikovan), čita po mreži (nije verifikovan) ili ima zamrznute račune (penal). U sledećem pasusu su opisane uloge konkretnih instanci *CompetitiveUser*: *Person* i *Company*.

Person predstavlja privatno lice, čija je uloga kupovina i prodaja prozivoda. Ima mogućnost kupovine novih proizvoda od *Company* i prodaju istih drugim *Person* učesnicima. Sadrži polja *firstname* i *lastname* koja označavaju redom njegovo ime i prezime.

4.2.2 Digitalna dobra

Na *slici 14* se mogu videti definicije digitalnih dobara (engl. *assets*).

```
abstract asset AbstractList identified by listid{
  o String listid
}

asset Item identified by itemid{
    o String itemid
    o String itemName
    o String description
  --> CompetitiveUser owner
}

asset ListItem extends AbstractList{
    --> Item item
    o Long price
    o ItemListStatus status
}
```

Slika 14: Model digitalnih dobara u mreži

AbstractList je pomoćni tip koji služi za lakše podešavanje kontrole pristupa, te detaljnije objašnjavanje nije potrebno.

Item predstavlja model materijalnog dobra. Jedinstveno ga identifikuje polje *itemid*. Kao opisni atributi se korisite polja *itemName* i *description*, dok polje *owner* predstavlja vlasnika odnosno referencu na učesnika konkretne klase *CompanyReference*.

ListItem predstavlja klasu omotač (engl. *wrapper*) koja označava da je *item* koji on sadrži za prodaju ili je prodat. Tu informaciju sadrži polje *status*.

4.2.3 Transakcije

Na *slici 15* se mogu videti definicije transakcija (engl. *transactions*).

```
abstract transaction AbstractTransaction {
}

abstract transaction RegulatorTransaction extends AbstractTransaction{
}

transaction Verify extends RegulatorTransaction {
  --> CompetitiveUser user
}

transaction IssuePenaltyDeletePenalty extends RegulatorTransaction {
  --> CompetitiveUser user
    o Boolean setStatus
}

transaction Deposit extends AbstractTransaction {
  --> CompetitiveUser user
    o Long deposit
}

transaction BuyItem extends AbstractTransaction {
  --> Person buyer
  --> ListItem listItem
    o String comment
    o Integer rating default=5
}
```

Slika 15: Model transakcija u mreži

Transakcije predstavljaju akciju kreiranja, čitanja, izmene na *ledger*-u. U *Composer*-u se definišu slično kao participanti i digitalna dobra, s tim što imaju iza sebe i biznis logiku. U alatu *Composer* biznis logika transakcije se piše u jeziku *JavaScript*.

Implementacija softvera za trgovinu sadrži šest transackija. Dve transackije *AbstractTransaction*, *RegulatorTransaction* su apstraktne i one pomažu pri implementaciji prava pristupa.

Uloga *Verify* transakcije je verifikovanje korisnika od strane *Regulator*-a. Ona sadrži polje *user* koje referencira participanta koji treba da se verifikuje.

Pokretanjem transakcije *Deposit* uvećava se stanje na računu učesnika koji je referenciran. Polje *balance* označava iznos za koji će se račun uvećati. Pošto postoji mogućnost greške i uvećanja tuđeg računa, u kontroli pristupa je podešeno da ulogovani korisnik može samo sebi uvećati račun.

Biznis logika *BuyItem* obezbeđuje prenos vlastništva u zamenu za finansijska sredstva

naznačena u *listItem* polju. Pri pokretanju transakcije postoji mogućnost upisa komentara i ocene o dobijenom *item*-u. Komentar i ocenu ostavlja kupac. Postoje dve varijante razmene: *Company*-i prodaje *Person*-u i *Person* prodaje *Person*-u. To je iz razloga što privatno lice ne može ubacivati materijalno dobro u mrežu, to sme samo kompanija, a posle kompanija može prodavati materijalno dobro privatnom licu. Na taj način se stvara lanac prodaje, te svako privatno lice koje bude kupilo materijalno dobro će moći da vidi tačne podatke o: broju lica koja su posedovala materijalno dobro i komentare prilikom svakog prenosa vlasništva.

4.2.4 Kontrola pristupa

Kako bi mreža normalno funkcionisala nije dovoljna definicija modela i biznis logike transakcija. Bez kontrole pristupa ukoliko neko poseduje informacije o *REST API* mreže, on ima mogućnost da zlonamerno vrši izmene nad istoj i čita poverljive podatke. Alat *Composer* nudi svoj jezik za kontrolu pristupa zvani *ACL* (*Access Control Language*) [25]. Jezik je dosta jednostavan i pomoću njega se pišu pravila koja definišu kontrolu pristupa. Radi pravilne implementacije softverskog rešenja ovog rada napisano je 14 pravila. Na *slici 16* su prikazana dva implementirana pravila. *RegulatorCanVerify* je primer jednostavnog pravila, dok je *BuyItemRule* primer kompleksnog.

```
rule RegulatorCanVerify{
  description: "Samo regulator moze vrsiti transakcije za regulatore"
    participant: "org.tradingapp.Regulator"
    operation: ALL
    resource: "org.tradingapp.CompetitiveUser"
    transaction: "org.tradingapp.RegulatorTransaction"
    action: ALLOW
}

rule BuyItemRule{
  description: "Obicni korisnici kupuju item"
    participant(m): "org.tradingapp.Person"
    operation: ALL
    resource: "org.tradingapp.*"
    transaction(tx): "org.tradingapp.BuyItem"
    condition: (m.getIdentifier() == tx.buyer.getIdentifier() && m.userStatus == "VERIFIED")
    action: ALLOW
}
```

Slika 16: Primer dva *ACL* pravila

Pravilo *RegulatorCanVerify* glasi u nastavku: Svaki ulogovani participant koji je klase *Regulator* (*participant*), može izvršiti (*action*) sve operacije modifikacije (*operation*) nad resursom (*resource*) klase *CompetitiveUser* samo preko izvršenja transakcije *RegulatorTransaction* (*transaction*). S obzirom da je *RegulatorTransaction* apstraktna transakcija, pod ovo pravilo upadaju sve transakcije koje nasleđuju spomenutu.

Pravilo *BuyItemRule* glasi u nastavku: Svaki ulogovani participant koji je klase *Person*, može izvršiti sve operacije modifikacije nad svakim resursom pod *namespace* lokacijom *org.tradingapp.**, samo pod zadatim uslovom preko transakcije *BuyItem*. Drugim rečima ovo pravilo omogućava da svaki participant klase *Person* kupuje *item* samo za sebe.

Pravilu *BuyItemRule* je data dozvola za modifikovanje bilo kog resursa unutar *namespace*, iz razloga što je potrebno promeniti balans participanata *CompetetiveUser* i *owner* polje koje označava vlasnika u *item* klasi. To čini biznis logiku transakcije za razmenu materijalnih dobara.

Ukoliko je softverska implementacija jednostavnije prirode i pravila pristupa nisu u cilju, mora se naglasiti da je obavezno ostaviti ona koja su automatski generisana. *Composer* zahteva da se za svaku transakciju ka mreži aktivira tačno jedno pravilo koje će odrediti prava pristupa. Ukoliko u konfiguraciji nema ni jednog pravila ili planirana transakcija ka mreži se ne poklapa ni sa uslovima ijednog pravila, onda se ona odbija. Drugim rečima, *ACL* pravila su veoma striktna, i zahteva eksplicitno definisanje pravo pristupa za bilo koji pokušaj pristupa *ledger*-a, ukoliko to nije urađeno pristup se odbija. Takođe je bitno naglasiti da pri svakoj transakciji, *Composer* proverava poklapanje sa pravilom krećući se od gore ka dole. Iz tog razloga je preporuka da se na prvim mestima definišu specifičnija pravila, a na poslednjim ona pravila koja uzimaju u obzir veći broj jednostavnijih transakcija. Radi lakšeg razumevanja naveden je primer česte greške pri definisanju pravila na *slici* uz objašnjenje.

```
rule EverybodyCanDoAnything {
    description: "Allow all participants to submit transactions"
    participant: "org.tradingapp.AbstractUser"
    operation: ALL
    resource: "org.tradingapp.*"
    transaction: "org.tradingapp.AbstractTransaction"
    action: ALLOW
}

rule BlockedReadsNothing {
  description: "Blokirana firma ili korisnik ne citaju"
    participant(m): "org.tradingapp.CompetitiveUser"
    operation: ALL
    resource: "org.tradingapp.*"
    condition: (m.userStatus == "PENALTY")
    action: DENY
}
```

Slika 17: Primer greške pri definisanju *ACL* pravila

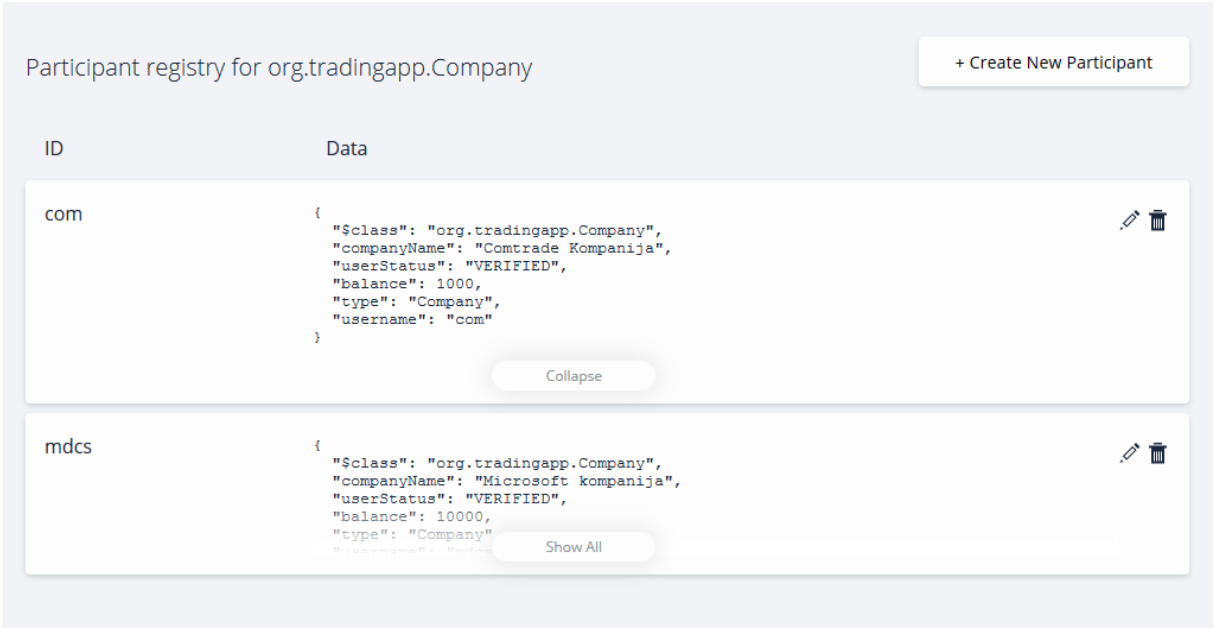
U prethodnim poglavljima je već spomenuto značenje *PENALTY* oznake. Kako bi se realizovao efekat postavljanja te oznake, definisano je pravo *BlockedReadsNothing* koje blokira apsolutno sve moguće operacije nad *ledger*-om, onog participanta koji je označen sa *PENALTY*. Iako su pravila na *slici 17* tačno definisana, greška je u redosledu.

BlockedReadsNothing neće imati efekat iz razloga što je na drugom mestu, te će se prvo pravilo *EverybodyCanDoAnything* aktivirati. Participant koji je penalizovan se savršeno

poklapa sa prvim pravilom, te zbog toga će ga *Composer* aktivirati i dozvoliti sve operacije nad *ledgerom*. Onog trenutka kada *Composer* nađe poklapanje, ne nastavlja se traženje daljih pravila. Rešenje na ovaj problem bi bila zamena redosleda ova dva pravila, kako bi se *BlockedReadsNothing* prvi aktivirao za participanta koji je penalizovan.

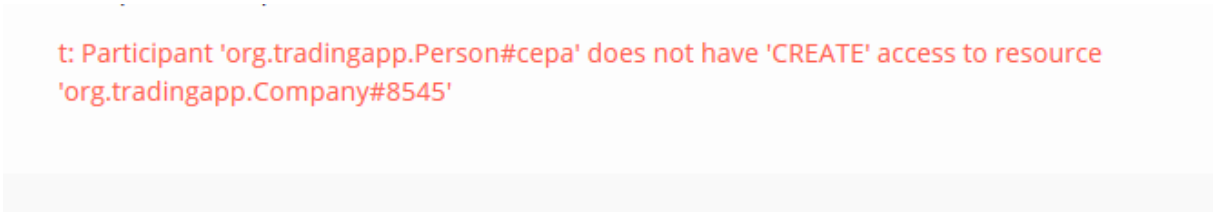
4.2.5 Opis aplikacije razvijene u *Hyperledger Playground*-u

Hyperledger Playground je dodatni alat *Composer*-u, koji omogućava osnovni korisnički interfejs za pokretanje jednostavnih transakcija koje uzrokuju promene na *Fabric*-u kao što su brisanje, modifikacija, kreiranje i dr. Na *slici 18* se vidi lista participanata na *ledger*-u. Postoji mogućnost kreiranja novog participanta, ali to zavisi isključivo od uloge ulogovanog korisnika. Pravila pristupa ovde važe!



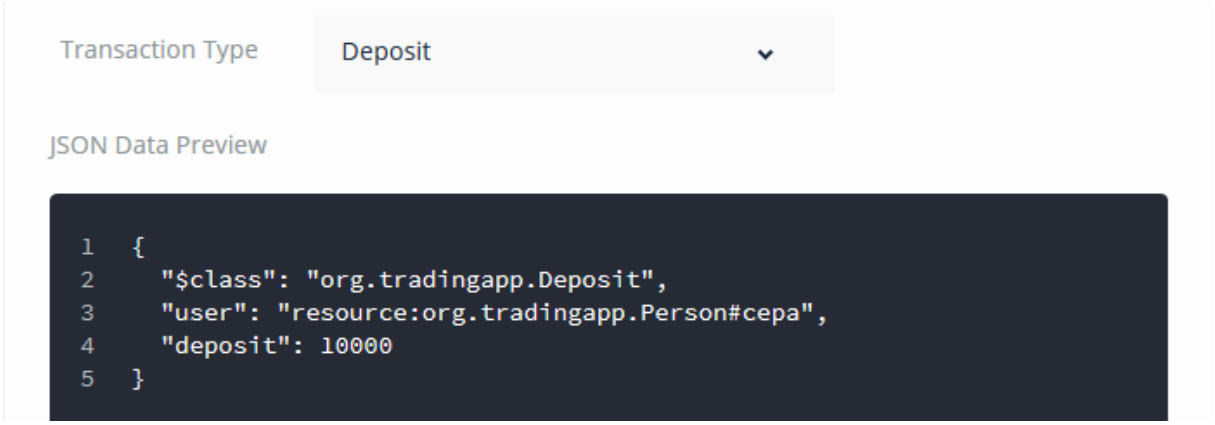
Slika 18: Lista participanata

Na *slici 19* je prikazana poruka koja ističe da je trenutno ulogovanom korisniku zabranjeno kreiranje nove kompanije.



Slika 19: Poruka o zabrani izvršenja transakcije

Sledi primera kreiranja nova transakcije za uplatu na račun. Na *slici 20* se nalazi modalni dijalog za kreiranje nove transakcije. *Transaction Type* je padajući meni koji dozvoljava izbor transakcija, te na osnovu toga se generiše odgovarajući zapis transakcije u *JSON* formatu. Na *slici 20* je prikazan primer validnog zapisa transakcije *Deposit*. Atribut *$class* označava tip transakcije, *user* označava participanta kojem se uplaćuje, a *deposit* označava koliko novca se uplaćuje. Treba napomenuti da je važna kontrola pristupa, te bez obzira na pravilan zapis *Deposit* transakcije, potrebno je da bude ulogovan korisnik klase *Person* sa identifikatorom *cepa*. Na *slici 21* je prikazano odgovarajuće pravilo kontrole pristupa za ovakav vid transakcije.

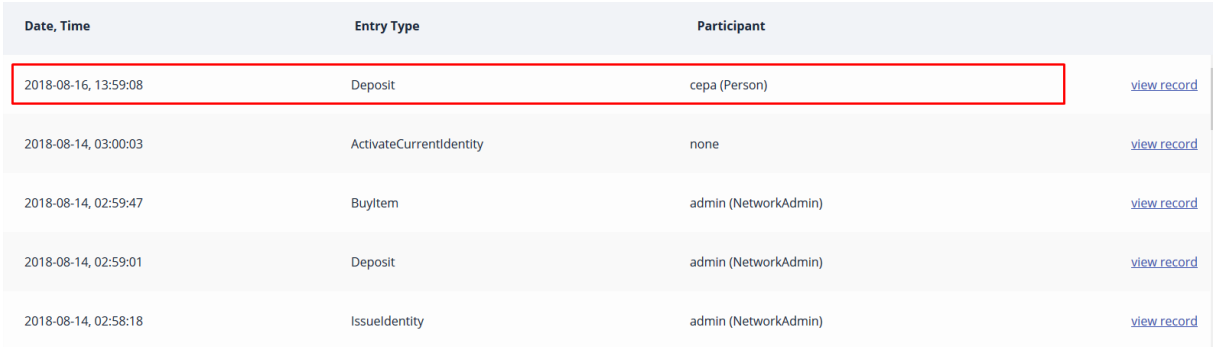


Slika 20: Modalni dijalog za kreiranje nove transakcije

```
rule CompUserDeposit{
  description: "Korisnik moze vrsiti depozit samo na svoj nalog"
    participant(m): "org.tradingapp.CompetitiveUser"
    operation: UPDATE
    resource(v): "org.tradingapp.CompetitiveUser"
    transaction(tx): "org.tradingapp.Deposit"
    condition: (m.getIdentifier() == tx.user.getIdentifier() && m.userStatus == "VERIFIED")
    action: ALLOW

}
```

Slika 21: Odgovarajuće pravo pristupa za transakciju *Deposit*

Ukoliko je model transakcije pravilno zapisan i pravo pristupa ispoštovano, transakcija će proći, te će biti uspešno zabeležena na *ledger*-u. *Slika 22* prikazuje istoriju svih transakcija gde je crvenim okvirom označena novonastala *Deposit*.



Date, Time	Entry Type	Participant	
2018-08-16, 13:59:08	Deposit	cepa (Person)	view record
2018-08-14, 03:00:03	ActivateCurrentIdentity	none	view record
2018-08-14, 02:59:47	BuyItem	admin (NetworkAdmin)	view record
2018-08-14, 02:59:01	Deposit	admin (NetworkAdmin)	view record
2018-08-14, 02:58:18	IssueIdentity	admin (NetworkAdmin)	view record

Slika 22: Istorija transakcija na *ledger*-u

```
/**
 * Sample transaction processor function.
 * @param {org.tradingapp.Deposit} tx The sample transaction instance.
 * @transaction
 */
async function Deposit(tx) {

    let user = tx.user

    user.balance += tx.deposit

    return getParticipantRegistry('org.tradingapp.' + user.type)
        .then(function (competetiveUserReg) {
            return competetiveUserReg.update(user)
        });

}
```

Slika 23: Poslovna logika transakcije *Deposit* zapisana u *JavaScript* jeziku

5 ZAKLJUČAK

Blockchain je mlada tehnologija, te još uvek nema široku primenu u industriji. Potrebno je vremena, rada, ulaganja i mnogo testiranja pre nego što zameni aktuelne metode čuvanja poverljivih podataka o identitetu, imovini, računima i dr. Najpre je bitno i obrazovati ljude o ovoj tehnologiji i njenoj moći, kako bi se izbacio treći posrednik kao što je banka. Za sada je Bitcoin bio uspešan, ali je uspeo i da napravi problem, iz razloga što većina ljudi zainteresovana za ovu valutu nema ideju kako funkcioniše. Ne naravno u nekom ozbiljnijem tehničkom poznavanju, nego u najosnovnijem. Sistem na kom je baziran Bitcoin je decentralizovan i demokratizovan, te nema svog vlasnika. Umesto da se sagleda sistem na kom funkcioniše i uvidi da postoji mogućnost oduzimanja velike moći od države, većina ljudi vidi ovo kao mašinu za brzu zaradu novca i ništa više.

Blockchain definitivno ima svetlu budućnost u poslovnom domenu, i kako godine prolaze njegova uloga će postajati sve veća. Ogromni se problemi mogu rešiti, a jedan od tih je praćenje materijalnih dobara od trenutka ubacivanja u *blockchain*. Naravno postoji mogućnost zloupotrebe, na primer naknadno ubacivanje materijalnog dobra koji nije kvaliteta kojeg bi trebao da bude. Ovakav problem bi se rešio ukoliko bi se išao korak dalje, i čuvali podaci o materijalu i delovima od koje je materijalno dobro proizvedeno. To bi definitivno bilo rešenje problema, jer materijalno dobro ne bi moglo da se ubaci u mrežu sve dok se ne navede odakle potiču materijali od kojih je isto proizvedeno.

Za širu praktičnu primenu će trebati vremena ne zbog nekog problema u *blockchain*-a, nego zbog politike. Da bi se rešenje na problem opisan u prethodnom pasus realizovalo, potrebno je udružiti ogroman broj kompanija i dogovoriti da svaka sve svoje poverljive podatke drži na jednoj mreži, to je dosta komplikovan proces. Dobro je spomenuti da je taj problem već rešila *Hyperldger Fabric* platforma odličnim rukovanjem privatnosti svojim sistemom kanala. Ova mogućnost dozvoljava da svi participanti unutar istog kanala imaju zasebnu kopiju *ledger*-a na kom se nalaze sve transakcije izvršene među njima. Dobra ideja bi bila da firmin ulazak u *Fabric*-ov kanala, zakonski potvđuje partnerstvo sa svim članovima istog, te bi se razmena izvršavala bez papirologije samo u okviru mreže i to bi drastično smanjilo troškove.

Hyperledger Composer je odličan alat koji uklanja posao naporne konfiguracije *Fabric*-a. Iz tog razloga je i korišćen u ovom radu jer je omogućio laku implementaciju i brzu demonstraciju. Nažalost nije demonstrirao glavnu moć *blockchain* tehnologija, a to je decentralizacija. Konkretno u ovoj implementaciji je korišćen samo jedan računar i konsenzus algoritam zvani *SOLO* koji se koristi samo u fazi testiranja, te se zasniva samo na odlukama jednog čvora koji ima prava da upravlja svim sadržajem zapisanim na *ledger*-u.

Predlog za dalji rad bi bila potpuna realizacija iste aplikacije na *Fabric*-ovoj mreži. Ne bi se koristio *Composer*, nego isključivo *Fabric*. Pisali bi se pametni ugovori u jeziku *GO*, i radila bi se ozbiljna kontrola pristupa. Bilo bi dobro, ukoliko bude mogućnosti i resursa da se izbegne korišćenje *SOLO* konsenzus algoritma, te umesto toga da se stavi *KAFKA* ili Praktična vizantijska tolerancija greške. Takvu vrstu rešenja bi zahtevalo pravu mrežu sa bar dva čvora, te bi se na taj način prikazala decentralizovana osobina *blockchain*-a.

Nakon uspešne realizacije *backend* dela, napravila bi se konkretna *Angular frontend* aplikacija koja bi komunicirala sa *Fabric*-ovim *REST API* servisima generisanim od strane

Swagger-a. Ovom implementacijom bi dobili konkretnu *blockchain* aplikaciju spremnu za korišćenje u poslovnom domenu.

LITERATURA

[1] S. Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System, October, 2008. https://bitcoin.org/bitcoin.pdf, poslednji pristup 08.08.2018.

[2] Investopedia, Third party transaction
https://www.investopedia.com/terms/t/third-party-transaction.asp, poslednji pristup 08.08.2018.

[3] Wikipedia, Proof of concept
https://en.wikipedia.org/wiki/Proof_of_concept, poslednji pristup 08.08.2018.

[4] Wikipedia, Blockchain
https://en.wikipedia.org/wiki/Blockchain, poslednji pristup 08.08.2018.

[5] Wikipedia, Distributed ledger
https://en.wikipedia.org/wiki/Distributed_ledger, poslednji pristup 08.08.2018.

[6] Andreas Antonopoulos. *Mastering Bitcoin (Second Edition, Second Print): Programming the Open Blockchain*, poslednji pristup 08.08.2018.

[7] Wikipedia, Block hashing algorithm
https://en.bitcoin.it/wiki/Block_hashing_algorithm,
poslednji pristup 09.08.2018.

[8] Wikipedia, Merkle tree or hash tree
https://en.wikipedia.org/wiki/Merkle_tree, poslednji pristup 09.08.2018.

[9] Wikipedia, Brute-force attack
https://en.wikipedia.org/wiki/Brute-force_attack, poslednji pristup 09.08.2018.

[10] Stackoverflow, Nonce size, will it always be big enough?
https://bitcoin.stackexchange.com/questions/1781/
nonce-size-will-it-always-be-big-enough, poslednji pristup 10.08.2018.

[11] Investopedia, Proof of Stake
https://www.investopedia.com/terms/p/proof-stake-pos.asp,
poslednji pristup 11.08.2018.

[12] Github, Nothing at Stake
https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQs,
poslednji pristup 11.08.2018.

[13] Blockgeeks, Casper protocol
https://blockgeeks.com/guides/ethereum-casper/, poslednji pristup 11.08.2018.

[14] Wikipedia, Byzantine fault tolerance
https://en.wikipedia.org/wiki/Byzantine_fault_tolerance,
poslednji pristup 11.08.2018.

[15] Miguel Castro, Barbara Liskov. Practical Byzantine Fault Tolerance
http://pmg.csail.mit.edu/papers/osdi99.pdf, poslednji pristup 11.08.2018.

[16] Wikipedia, Hyperlegder
https://en.wikipedia.org/wiki/Hyperledger, poslednji pristup 11.08.2018.

[17] Hyperledger, Members
https://hyperledger.org/members, poslednji pristup 11.08.2018.

[18] Hyperledger Whitepaper
https://www.hyperledger.org/wp-content/uploads/2018/08/
HL_Whitepaper_IntroductiontoHyperledger.pdf, poslednji pristup 11.08.2018.

[19] Hyperledger Fabric docs, Introduction
https://hyperledger-fabric.readthedocs.io/
en/release-1.2/whatis.html, poslednji pristup 12.08.2018.

[20] Hyperledger Fabric docs, Fabric Model
https://hyperledger-fabric.readthedocs.io/
en/release-1.2/fabric_model.html, poslednji pristup 12.08.2018.

[21] Hyperledger Fabric docs, Fabric Network
https://hyperledger-fabric.readthedocs.io/
en/release-1.2/network/network.html, poslednji pristup 12.08.2018.

[22] Skcript, Consensus in Hyperledger Fabric
https://www.skcript.com/svr/consensus-hyperledger-fabric/,
poslednji pristup 12.08.2018.

[23] Hyperledger Fabric docs, TX flow
https://hyperledger-fabric.readthedocs.io/en/release-1.2/txflow.html,
poslednji pristup 12.08.2018.

[24] Hyperledger Fabric docs, Read Write
https://hyperledger-fabric.readthedocs.io/en/release-1.2/readwrite.html,
poslednji pristup 12.08.2018.

[25] Hyperledger Composer docs
https://hyperledger.github.io/composer/latest/reference/cto_language,
poslednji pristup 15.08.2018.

SPISAK SKRAĆENICA

Skraćenica	Značenje
SHA	***Secure Hash Alghorithm*** - familija kripotografskih heš funkcija.
BFT	***Byzantine fault tolerance*** - svojstvo sistema i algoritma.
PBFT	***Practical Byzantine Fault Tolerance*** - konsenzus algoritam.
UML	***Unified Modeling Language*** - standardni jezik za vizuelno prikazivanje objektnog modela.
REST	***Representational State Transfer*** - definiše mehanizme komunikacije između veb servisa.
API	***Application programming interface*** - definiše interfejse radi korišćenja mehanizama određene biblioteke.
RBP	***Relaciona baza podataka*** - poseban tip baze podataka kod kojeg se organizacija podataka zasniva na relacionom modelu.
JSON	***JavaScript Object Notation*** - je tekstualno baziran otvoreni standard dizajniran za razumljivu ljudima razmenu podataka. Izveden je iz *JavaScript* jezika.

SPISAK SLIKA

SPISAK TABELA

BIOGRAFIJA

Džunja Dejan je rođen 17. septembra 1995. godine u Novom Sadu. Završio je Gimnaziju Jovan Jovanović Zmaj, smer obdareni učenici u matematičkoj gimnaziji. Odmah nakon završetka srednje škole, školske 2014/2015. godine upisuje Fakultet tehničkih nauka u Novom Sadu, smer Računarstvo u automatika, usmerenje Primenjene računarske nauke i informatika. Osnovne akademske studije je završio u roku, školske 2017/2018 godine. Položio je sve ispite propisane planom i programom. Master studije nastavlja školske 2018/2019 u Moskvi na fakultetu *Skolkovo Institute of Science and Technology*, usmerenje nauka o podacima (engl. *Data Science*).