

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1731

**Primjena raspodijeljene knjige zapisa za kontrolu
ulaza**

Krešimir Ostović

Voditelj: *Krešimir Pripužić*

Zagreb, Lipanj 2018.

Sadržaj

Uvod.....	7
1. Platforma raspodijeljene knjige Hyperledger.....	8
1.1. Općenito.....	8
1.2. Hyperledger u odnosu naspram drugih platformi.....	9
1.3. Hyperledger Fabric programski okvir.....	9
1.4. Posebnost Hyperledger Fabric programskog okvira.....	10
1.4.1. Modularnost.....	10
1.4.2. Sustav s dozvolom za ulaz u komunikacijski kanal.....	11
1.4.3. Pametni ugovori.....	11
1.4.4. Brzina te sigurnost i algoritam kreiranja transakcija.....	13
1.5. Mreža Hyperledger Fabric programskog okvira.....	14
1.5.1. Sastavnice mreže.....	14
1.5.2. Vrste čvorova u mreži i njihove uloge.....	15
1.5.3. Konfiguracija.....	16
1.5.4. Tok transakcija.....	16
2. Platforma raspodijeljene knjige Ethereum.....	18
2.1. Općenito.....	18
2.2. Ethereum u odnosu naspram drugih platformi.....	19
2.3. Karakteristike Ethereum platforme.....	19
2.3.1. Sustav bez dozvole za ulaz na mrežu Ethereum.....	20
2.3.2. Brzina.....	20
2.3.3. Sigurnost i algoritmi za kreiranje transakcija.....	20
2.3.4. Pametni ugovori.....	22
2.4. Mreža Ethereum platforme.....	23
2.4.1. Sastavnice mreže.....	24
2.4.2. Tok transakcija.....	24

3. Aplikacija za kontrolu ulaza.....	26
3.1. Hyperledger platforma.....	26
3.1.1. Hyperledger Composer programski alat.....	27
3.1.2. Konfiguracija Hyperledger Fabric mreže i čvorova za aplikaciju kontrole ulaza.....	30
3.1.3. Logika Composer blockchain aplikacije za kontrolu ulaza.....	31
3.1.4. Frontend Hyperledger aplikacije za kontrolu ulaza.....	38
3.1.5. Simulacija čvora za kontrolu ulaza na Hyperledgeru.....	41
3.2. Aplikacija za kontrolu ulaza na platformi Ethereum.....	44
3.2.1. Truffle programski alat.....	44
3.2.2. Konfiguracija mreže i čvorova.....	45
3.2.3. Logika aplikacije za kontrolu ulaza.....	46
3.2.4. Frontend Ethereum aplikacije za kontrolu ulaza.....	53
3.2.5. Ethereum aplikacija kontrole ulaza na uređajima za kontrolu ulaza.....	54
4. Usporedba aplikacije na Ethereumu i Hyperledgeru.....	57
4.1. Usporedba na temelju performansi.....	57
4.2. Usporedba na temelju zahtijevanih razina povjerenja.....	58
4.3. Usporedba na temelju sigurnosti podataka i privatnosti.....	59
4.4. Usporedba na temelju složenosti aplikacije.....	60
4.5. Zaključak usporedbi.....	61
Zaključak.....	63
Literatura.....	64
Sažetak.....	66
Summary.....	67
Privitak.....	68

Popis kratica

RFID - Radio-frekvencijska identifikacija (eng. Radio-frequency identification)

DLT - Distributed ledger technology – tehnologija raspodijeljene knjige zapisa

BFT – Byzantine fault tolerance – Otpornost na Bizantske pogreške

CFT - Crash fault tolerant – Otpornost na pad sustava

DBMS - Database management system – Sustav za upravljanjem baza podataka

PoW – Proof of Work – Dokaz rada

PoS – Proof of Stake – Dokaz posjedovanja

PoA – Proof of Authority – Dokaz autoriteta

ZKP - Zero knowledge proof – Dokaz znanja

CA – Certificate authority – Upravljanje potvrdama

PKI – Public Key Interface – Infrastruktura javnih ključeva

ASIC - Application-specific integrated circuit

EVM – Ethereum Virtual Machine – Ethereumov virtualni stroj

ABI - Application binary interface

GUI - Graphical user interface

OOP – Object Oriented Programming - Objekto orijentiran jezik

SQL - Structured Query Language

JSON - JavaScript Object Notation

REST - Representational state transfer

CRUD - Create, read, update and delete

IDE - Integrated development environment

Popis oznaka i mjernih jedinica

B – Byte

MB – Megabyte = 10^6 Byte

TPS – Transakcija po sekundi

Popis tablica

Tablica 4.2: Usporedba vremena čekanja 5 faza Hyperledger Fabrica u ms (milisekundama)

Popis slika

Slika 1.1: Hyperledger logo

Slika 1.2: Hyperledger Fabric logo

Slika 1.3: Opis rada sa chaincodom te pristup chaincode ledgeru

Slika 1.4: Dijagram toka transakcije u Hyperledger Fabricu

Slika 2.1: Ethereum logo

Slika 2.2: Način rada pametnih ugovora na Ethereum platformi

Slika 2.3: Čvorovi s posljednjih n blokova šalju transakcije čvorovima rudarima

Slika 3.1: Dijagram povezivanja sastavnica Composer aplikacije sa Hyperledger Fabric blockchainom

Slika 3.3: Razni resursi kojima se može pristupiti preko REST sučelja

Slika 3.4: Primjer slanja POST operacije na REST sučelju

Slika 3.6: Izgled WEB sučelja za pregled korisnika. Moguća je i modifikacija u slučaju administratorskih privilegija

Slika 3.5: Izgled WEB sučelja za unos podataka i slanje transakcija na chaincode

Slika 3.7: Prikaz simulacije triju uređaja za kontrolu ulaza na različitim fakultetima na platformi Hyperledger Fabric

Slika 3.9: Logo Truffle alata

Slika 3.10: Sučelje Ganache simulatora s prikazom na trenutne aktivne račune te brojem blokova i transakcija koje su napravili tijekom slanja transakcija za kontrolu ulaza

Slika 3.12: Remix IDE s otvorenim prozorima koda pametnog ugovora piiSZG, donji prozor u kojem Remix sluša transakcije na mreži te desni prozor postavki i ručnog testiranja funkcija pametnog ugovora

Slika 3.13: Izgled WEB sučelja Ethereum aplikacije za kontrolu ulaza

Slika 3.14: Prikaz simulacije triju uređaja za kontrolu ulaza na različitim fakultetima na platformi Ethereum

Slika 4.2: Merkle tree struktura podataka koju koristi blockchain

Uvod

U zadnje vrijeme se sve češće govori o tehnologiji raspodijeljene knjige zapisa (eng. *distributed ledger*) te se cijela tehnologija pokušava implementirati u što više sektora. Sama tehnologija *distributed ledger* nije nova, kao niti podvarijanta Lanac blokova (eng. *Blockchain*) međutim dobila je na popularnosti početkom 2010tih godina kada je platforma Bitcoin koja je implementirala istoimenu digitalnu valutu postala veoma popularna. Naime digitalnom valutom Bitcoin se može trgovati te pruža dozu anonimnosti i ne zahtijeva nikakvu razinu povjerenja u druge osobe uključene na platformu. Problem s platformom Bitcoin je što su se samo transakcije Bitcoin digitalne valute mogle odvijati, odnosno nije postojala potpora za programskim kodom koji bi se mogao izvršavati na takav način. Programeri su zaključili da bi bilo dobro napraviti novu platformu na kojoj je moguće raspodijeljeno izvršavati kod. Ethereum platforma nastala 2014. godine, namijenjena izvršavanju programskog koda preko javnog *blockchaina* te je zbog toga postala jednom od najpopularnijih platformi koji se baziraju na tehnologiji *distributed ledger*.

Nasuprot platformi Ethereum je platforma Hyperledger osnovana od strane Linux fondacije koja je imala za svrhu napraviti sustav koji će biti djelomično privatniji odnosno često se spominje pridjev korporativni. Projekt je populariziran sredinom 2017. kada se u njega uključio značajan broj korporacija poput Intela i IBM-a. Kontrola ulaza je jedan od problema koji muči ustanove. Naime u svakom trenutačnom sustavu postoji ljudska ruka, sistemski administrator, koji proizvoljno može manipulirati podacima u sustavu te tako potencijalno učiniti zlonamjerne radnje. Cilj ovoga rada je usporediti dvije platforme na tehnologiji *distributed ledger* te performanse i potrebne razine povjerenja za aplikaciju kontrole ulaza bazirane na RFID karticama.

1. Platforma raspodijeljene knjige Hyperledger

1.1. Općenito

Hyperledger platforma osnovana je od strane Linux Fondacije 2015. godine te je svrha bila napraviti sustav temeljen na distributed ledgeru koji će djelomično biti privatniji kako bi ga međusobno koristile korporacije. Projekt je populariziran sredinom 2017. kada se u njega uključio značajan broj većih korporacija poput Intela i IBMa sa svojim programskim okvirima (eng. *framework*) Sawtooth te Fabric, te se sve češće spominje pridjev korporativni *blockchain*.



HYPERLEDGER

Slika 1.1: Hyperledger logo

Rastom popularnosti kriptovaluta (eng. *cryptocurrency*) i tehnologije u pozadini, poduzeća su počela istraživati mogućnost privatnog korištenja ove tehnologije. Unatoč samoj ideji privatnog blockchain sustava koji se kosi s pravilom da blockchain sustavi ne smiju biti privatni zbog moguće manipulacije, mnoge, pretežito automobilske tvrtke su počele implementirati Hyperledger kao rješenje svojih problema. [1]

Ideja Hyperledger platforme se generalno ne razlikuje previše od ostalih platformi. Podatci su zapisani u *blockchainu*, zapisuju ih čvorovi koji izvršavaju i validiraju transakcije. Svaki čvor posjeduje lokalnu kopiju knjige zapisa te kada primi novo stanje knjige zapisa, prvo provjeri jeli novo stanje validno sa stanjem njegove lokalne kopije. Ako nije, novo stanje se odbacuje. Najpoznatije platforme su otvorene mreže bez dozvole upada, odnosno svatko može biti u mreži. Ubrzo su tvrtke shvatile da za njihove potrebe, korištenje mreže bez dozvole nije pametno te su pokušale napraviti nešto drugačije, a to je rezultiralo mrežom sa dozvolom za upad Hyperledger.

1.2. Hyperledger u odnosu naspram drugih platformi

Hyperledger je predvodnik kontroverzne skupine mreža koje su zatvorene te se u mrežu, odnosno kanal u Hyperledgeru, ulazi s dozvolom. U Europskoj Uniji je u svibnju 2018. godine stupio na snagu zakon o zaštiti privatnih podataka te može doći do narušavanja podataka u slučaju nepravilnog korištenja javnih platformi. Osim toga, u mnogim slučajevima, pogotovo bankovnim transakcijama, anonimnost, koliko god to korisnici željeli, ne dolazi u obzir zbog rigoroznih zakona.

Korporativno korištenje uključuje sljedeće zahtjeve koji se moraju razmotriti [1]:

- Sudionici - čvorovi u mreži se moraju moći identificirati
- Mreža mora imati dozvolu za ulaz
- Mogućnost velikog broja transakcija
- Brza potvrda transakcija
- Privatnost i povjerljivost transakcija i podataka koji se odnose na transakciju

Većina platformi se nadograđuje te će dio njih u nekom trenutku podržavati sve navedene zahtjeve, međutim Hyperledger je ciljano izgrađena platforma za te zahtjeve.

1.3. Hyperledger Fabric programski okvir

Hyperledger Fabric je programski okvir otvorenog koda nastao u sklopu Hyperledger platforme. Zamišljen je kao raspodijeljena knjiga zapisa koja bi se koristila u korporativne svrhe. Donosi mogućnosti koje zahtijevaju slučajevi korištenja u tvrtkama. Osnovan je od strane Linux Fondacije te je primjer projekta otvorenog koda iako je većinu osmislio IBM, ali implementaciju su programirale mnoge organizacije te mnogi samostalni programeri.

Fabric je veoma modularan te se mogu ukloniti dijelovi arhitekture koji trenutno ne trebaju, pa se s time postiže prilagodljivost i optimizacija raznih slučajeva korištenja koje



Slika 1.2: Hyperledger Fabric logo

tvrtke mogu zatražiti. Sektori poput bankarstva, financija, osiguranja, ljudskih resursa, opskrbnog lanca su primarni slučajevi korištenja, međutim stalno se otkrivaju nove mogućnosti. Fabric podržava poznate programske jezike opće namjene kao što su Java, Go i Node.JS umjesto specifičnog programskog jezika kao što je Solidity na Ethereum platformi. Ujedno to znači da manje vremena odlazi na učenje novih stvari te ne treba dodatnog vremena uložiti u učenje novog jezika što je posebno bitno tvrtkama tako da ne moraju rasipati resurse. [1]

Hyperledger u osnovi podržava implementacije ulaska u mrežu bez dopuštenja, ali su tek pojedini programski okviri implementirali takav način. Fabric je implementirao samo ulazak u mrežu s izričitim dopuštanjem te je ujedno i sigurniji za tvrtke koje ne žele vanjske čvorove u mreži. Čvorovi su poznati jedni drugima, ali si ne moraju vjerovati odnosno model upravljanja mrežom najčešće ovisi o tome koliko si čvorovi vjeruju jedno drugome. Fabric je zamišljen da ima modularnu arhitekturu. Postoje razni moduli, a u cilju im je podržati što više zahtjeva od strane tvrtki koji bi mogli tražiti razne prilagodbe za razne potrebe. Dodaju li se mogućnosti pisanja pametnih ugovora te bolje zaštite razina povjerenja, može se reći da je Fabric programski okvir trenutno bolji od tradicionalnih raspodijeljenih sustava u određenim primjenama.

1.4. Posebnost Hyperledger Fabric programskog okvira

1.4.1. Modularnost

Modularnost Fabrica je velika značajka koja je doprinijela njegovoj popularnosti. Koristi li se unutar jedne tvrtke koja sama sebi sigurno vjeruje, u modelu će se isključiti algoritam usuglašavanja pomoću bizantskih generala (BFT) koji je potreban za okruženje s više tvrtki te uključiti modul poput usuglašavanja sa zaštitom od pada (CFT) . Takvi algoritmi ne traže skupo računanje kao što traži algoritam dokaz rada (Proof of work algorithm) te u načelu svakodnevno funkcioniranje sustava ima slične operacijske troškove kao i drugi raspodijeljeni sustavi.

- **Modul za slaganje transakcija (eng. *ordering service*)**– Pomoću dogovorenog konsenzusa slaganja transakcija, recimo po dospijeću vremena, dospijeću iz pojedinih organizacija itd...slaže u blokove i vraća blokove valjanim čvorovima.

- **Modul za članstvo u mreži (eng. *membership service provider*)** – Asocira čvorove u mreži s odgovarajućim kriptografskim identitetima kao što su LDAP ili OpenID Connect
- **Modul za peer-to-peer širenje poruka (eng. *peer-to-peer gossip service*)** – prosljeđuje blokove čvorovima za koje drugi čvorovi požele da imaju blokove lokalno
- **Modul za pametne ugovore (eng. *chaincode*)** – Podržava kreiranje logike transakcija na čvoru te se pišu u standardno podržanim programskim jezicima
- **Modul za podršku u integraciji s raznim sustavima za upravljanje bazom podataka (eng. *pluggable support of a variety of DBMS*)**
- **Modul za validiranje (eng. *endorsement and validation policy enforcement*)** – Prati da se svaki čvor pridržava pravila koja su nametnuta [1]

1.4.2. Sustav s dozvolom za ulaz u komunikacijski kanal

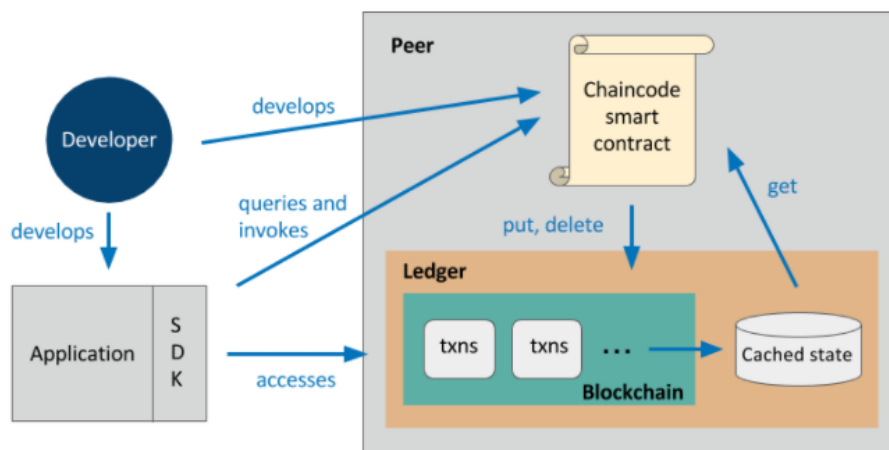
Sustav s dozvolom za ulaz u komunikacijski kanal djeluje u *blockchainu* čiji je princip da su svi čvorovi poznati i identificirani te takvi čvorovi mogu sudjelovati samo ako se pridržavaju pravila koji je konsenzusom izglasani. Takav model donosi povjerenje među čvorovima te stoga nije potrebno obavljati skupe izračune bazirane na derivacijama PoW algoritma. Sustav bez dozvole za ulaz u komunikacijski kanal mora imati algoritam prema kojemu će si čvorovi međusobno vjerovati iako se ne znaju, pa je višestruko skuplji za korištenje. Čvorovi koriste svoje resurse te proračunavaju određenu funkciju kako bi svima potvrdili da im se u tom trenutku može vjerovati. Sustav s dozvolom mora paziti da većina čvorova daje prave informacije te se oslanja na CFT odnosno BFT algoritam. Svi postupci promjene stanja *blockchaina* su zabilježeni, pa se prema tome i svi nelegalni postupci mogu lako otkriti i svi unutar sustava znaju tko je pokušao prevariti sustav.

1.4.3. Pametni ugovori

Programski okvir Fabric koristi pametne ugovore, odnosno chaincode, kao raspodijeljene aplikacije koje se izvršavaju na čvorovima blockchaina nakon što su stvorili konsenzus da će koristiti te aplikacije. Pametni ugovori se mogu nazvati logikom

transakcija te ih svaki čvor izvršava lokalno i lokalno provjeri rezultat transakcije. Pametni ugovori se mogu paralelno izvršavati na mreži i može ih svaki čvor staviti na mrežu te zbog toga ih se mora tretirati kao nepovjerljivim. Zbog toga su se druge platforme odlučile na pristup koji im jamči determinističko izvođenje pametnih ugovora čak i na potencijalno nesigurnim čvorovima.

Pristup koji koriste zove se složi-izvrši (eng. *order-execute*) gdje se prvo potvrđuju transakcije i kada se potvrdi da nisu štetne, šalju se svim čvorovima na izvršavanje. Pristup na koji se Fabric odlučio naziva se izvrši-složi-potvrdi (eng. *execute-order-validate*). Prvo se transakcija izvršava, a ako se sustav nađe u nedopuštenom stanju, transakcija se poništava. Ako je transakcija dobronamjerna, slaže se u blokove pomoću modula za slaganje transakcija te se zatim potvrđuje u okviru aplikacije koja ju koristi, a tek onda šalje na knjigu zapisa.



Slika 1.3: Opis rada sa *chaincodom* te pristup *chaincode* ledgeru

Svaka aplikacija na mreži može imati svoju politiku u kojoj traži minimalan broj sudionika – čvorova u mreži koji moraju potvrditi svaku transakciju za svaki pametni ugovor. S time se dobila modularnost pošto se može odrediti da svaku transakciju mora izvršiti i prihvatiti samo određen broj čvorova u mreži te se za zadaće koje zahtijevaju najviše sigurnosti može staviti da ih svi čvorovi sudionici u mreži moraju izvršiti i potvrditi. Također postoji mogućnost i paralelnog izvršavanja pametnih ugovora ako samo dio čvorova mora izvršiti svaku transakciju, a nedeterminizam se uklanja tako da svi od n čvorova moraju dobiti istu rezultat. Rezultat toga je da se mogu koristiti standardni programski jezici za pisanje pametnih ugovora te su dosad podržani Go, Java te Node.js

Postoje dvije vrste pametnih ugovora, sistemski pametni ugovori (eng. *system chaincode*) te pametni ugovori vezani uz aplikaciju. Sistemski pametni ugovori definiraju parametre kanala te osiguravaju ulaz samo određenih čvorova u kanal te izlaz. Druga namjena im je oblikovati pravila prihvaćanja transakcije u odnosu na broj čvorova u kanalu. Obični pametni ugovori definiraju logiku transakcija te promjene među sudionicima (eng. *participants*) i imovinom (eng. *asset*) zahvaćenom transakcijama.

1.4.4. Brzina te sigurnost i algoritam kreiranja transakcija

U javnim *blockchain* platformama, programski kod pametnih ugovora te podatci koji pristižu na pametni ugovor su javni i dostupni svima zbog značajke da se transakcija mora izvršiti na svakom čvoru te da se na svakom čvoru mora dobiti isti rezultat. Tvrtke obično imaju povjerljive podatke koje žele sakriti od javnosti odnosno ostalih sudionika u mreži, najčešće zbog poslovne tajne, a čak niti kriptiranje nije u nekim slučajevima dovoljno dobra opcija pošto će drugi sudionik znati da se nešto događa ispod stola. Još je veći problem što su podatci pohranjeni na svakom čvoru te se u konačnici mogu probiti, a tvrtke ne žele riskirati ni najmanju mogućnost da konkurencija ima uvid u poslovanje u koje ne bi smjeli imati uvid. Jedna od ideja koju koriste i javne, ali i privatne blockchain platforme poput Hyperledgera je ZKP međutim problem je što računanje ZKP traje te se smanjuju performanse platforme.

Fabric je zbog toga implementirao modul kanala u kojem određeni čvorovi zajedno stvore kanal te samo oni imaju uvid u određene transakcije i podatke o transakcijama između njih čuvajući povjerljivost između strana u kanalu odnosno ne dopuštajući da strana koja je u istom *blockchainu*, ali u drugom kanalu vidi podatke. Može se reći da su kanali zapravo Lanci sa strane (eng. *sidechains*) te će o korisnosti odnosno štetnosti istih biti riječi u poglavlju Usporedba platformi i aplikacija.

Performanse platforme ovise o mnogo varijabli poput veličina transakcije, veličina bloka, broj čvorova, jačina pojedinog čvora. Eksperimenti obavljeni na IBM cloud servisu od strane Fabric programera pokazali su odlične performanse naspram ostalih blockchain platformi. Naime najbolje performanse su postignute za blok od dva MB te se svaka transakcija izvodila od početka do kraja oko pola sekunde. Međutim kako se transakcije izvode u paraleli, postoji nekoliko faza transakcije te su namještene opcije da samo pola čvorova mora potvrditi svaku transakciju, programeri su došli do zapanjujuće brojke od

prosječnih 3000 transakcija po sekundi što je za dva reda veličine više od prosječne *blockchain* platforme. Svi parametri se mogu podešavati, pa je tako za veći broj čvorova za slaganje transakcija rastao broj transakcija po sekundi sve do točke u kojoj je maksimalna iskorištenost protoka komunikacije između čvorovima, nakon toga povećavanjem broja čvorova za slaganje transakcija u blokove zapravo gubimo performanse. Slično je i kod povećanja broja običnih čvorova koji ne potvrđuju transakcije već im se samo dostavljaju podatci. Bržim procesorima se smanjuje vrijeme svake faze na čvorovima, a spremanjem podataka o knjizi zapisa (eng. *ledger*) na RAM umjesto na SSD disk se dobije samo malo povećanje performansi zato što se memorija koristi jedino u fazi validacije podataka sa *ledgerom*. Svi eksperimenti su provedeni od strane Fabric programera na IBM Cloud servisima diljem svijeta te su opisani u [3].

1.5. Mreža Hyperledger Fabric programskog okvira

Mreža Hyperledger Fabric programskog okvira je zamišljena kao sustav u kojemu članovi traže dozvolu prije nego što uđu u *blockchain*. Nekoliko tvrtki može napraviti zajedničku mrežu te se moraju ustvrditi odredbe po kojima novi čvorovi smiju tražiti ulaz u mrežu. Mrežnim odredbama upravljaju svi te se mijenjaju samo ako postoji konsenzus svih strana uključenih u mrežu.

1.5.1. Sastavnice mreže

Mreža se sastoji od nekoliko sastavnih dijelova:

1. **Knjiga zapisa (eng. *ledger*)** - Sastoji se od dva dijela: *blockchaina* koji je ne promjenjiv odnosno radi na principu Merkle Tree algoritma, te trenutnog stanja eng. (world state). Baza trenutnog stanja sastoji se od svih trenutnih ključ-vrijednost parova (eng. key-value pair) koji su bili dodani/obrisani/modificirani valjanim transakcijama u *blockchainu*. Baza trenutnih podataka postoji kako bi se brže izračunala valjanost transakcije od prolaska preko cijelog *blockchaina*. Svaki čvor u mreži sadrži svoju lokalnu kopiju knjige zapisa, te ih logički razdvaja za svaki kanal u kojem sudjeluje. Ako 2 čvora sudjeluju u istim kanalima, oni bi morali imati identičnu kopiju knjige zapisa. Takav pristup se još naziva raspodijeljena knjiga zapisa.

2. **Pametni ugovori (eng. *chaincode*)** – Kod pozvan od strane klijentske aplikacije kako bi se izvršila transakcija odnosno promijenila vrijednost ključ-vrijednost para. Ostatak pogledati 1.4.3.
3. **Čvorovi (eng. *nodes*)** – Sudionik u mreži koji sadrži kopiju knjige zapisa te ima prava na izvršavanje *chaincode*, čitanje/pisanje lokalne knjige zapisa
4. **Usluga za slaganje transakcija (eng. *ordering service*)** - Čvorovi zaduženi za slaganje transakcija u blokove. Zbog modularne arhitekture podržavaju dodatne module koji im definiraju drugačija ponašanja. Sadrže popis svih organizacija udruženih u kanal kojemu slažu transakcije
5. **Kanali (eng. *channels*)** – *Sidechain*ovi čiji pripadnici dijele jedinstvenu knjigu zapisa koju samo oni koriste. Ulazak u kanal mora biti dopušten od strane ostalih članova kanala
6. **Fabric CA** – Čvorovi zaduženi za upravljanje potvrdama preko PKI certifikata. Izdaju certifikate svakom sudioniku koji je autoriziran.

1.5.2. Vrste čvorova u mreži i njihove uloge

Mrežu kreira konzorcij, skup organizacija (eng. *consortium*) koji uključuje sve gore nabrojane dijelove. Čvorovi za slaganje transakcija su administrativni dio mreže zbog toga što sadrže konfiguracije za svaki pojedini kanal u mreži kao što je odrednica tko smije ući u kanal te trenutni čvorovi u kanalu.

Čvorovi mogu biti:

- **Početni čvor (eng. *Anchor peers*)** – Čvor definiran u konfiguraciji čvorova za administraciju kanala kao prvi čvor na koji će se novi čvorovi spojiti
- **Glavni čvor (eng. *Leading peer*)** – Čvor koji preuzima ulogu komunikacije nekoliko čvorova iz iste organizacije u svrhu smanjenja mrežnog opterećenja
- **Potvrđujući čvorovi (eng. *Endorsing peers*)** – Čvorovi koji izvode pametne ugovore odnosno fazu 1 izvršenja transakcije te predlažu odgovor na fazu

- **Izvršni čvorovi (eng. *Committing peers*)** – Pišu novi blok na knjigu zapisa nakon što su im čvorovi za slaganje transakcija dostavili novi blok

U slučajevima manjeg broja čvorova i želje za uštedom resursa, čvorovi mogu preuzeti više uloga, pa se tako može dogoditi da postoji samo jedan čvor za slaganje transakcija te jedan čvor koji objedinjava 4 gore navedene uloge. Trenutno ne postoji limit na broj čvorova, međutim maksimalni broj čvorova za neku mrežu je teoretski limitiran brzinom veze između tih čvorova. Pređe li se granica, može se očekivati značajan pad performansi.

1.5.3. Konfiguracija

Dodavanje nove organizacije odnosno čvora u kanal se radi tako što se poziva sistemski pametni ugovor koji obnavlja konfiguraciju kanala dodajući novi čvor, a takav pametni ugovor može pozvati samo *admin* kanala. Ako je odgovor potvrđan, novi čvor kontaktira početni čvor za ostale čvorove u mreži. Dodavanje kanala se odvija na čvoru za slaganje transakcija te taj čvor prikuplja sve čvorove koji trebaju ući u novi kanal te konfigurira postavke kanala.

1.5.4. Tok transakcija

Nakon što su čvorovi registrirani kod CA čvora i dobili su svoje ključeve, te je konfiguriran kanal s bar dva čvora, može početi proces transakcije. Pametni ugovori koji sadrže logiku transakcije moraju biti poslani čvorovima kako bi čvorovi mogli potvrditi transakcije.

Klijent šalje zahtjev za izvršenjem funkcija pametnog ugovora u kojemu se nalazi logika transakcije te potvrđujući čvorovi simuliraju transakciju.

Potvrđujući čvorovi potvrđuju da je:

1. Transakcija je valjana u ovisnosti o svom trenutnom stanju knjige zapisa
2. Transakcija je jedinstvena odnosno nije kopija neke prijašnje
3. Potpis klijenta koji je inicirao funkciju pametnog ugovora je kriptografski valjan
4. Klijent smije inicirati tu funkciju

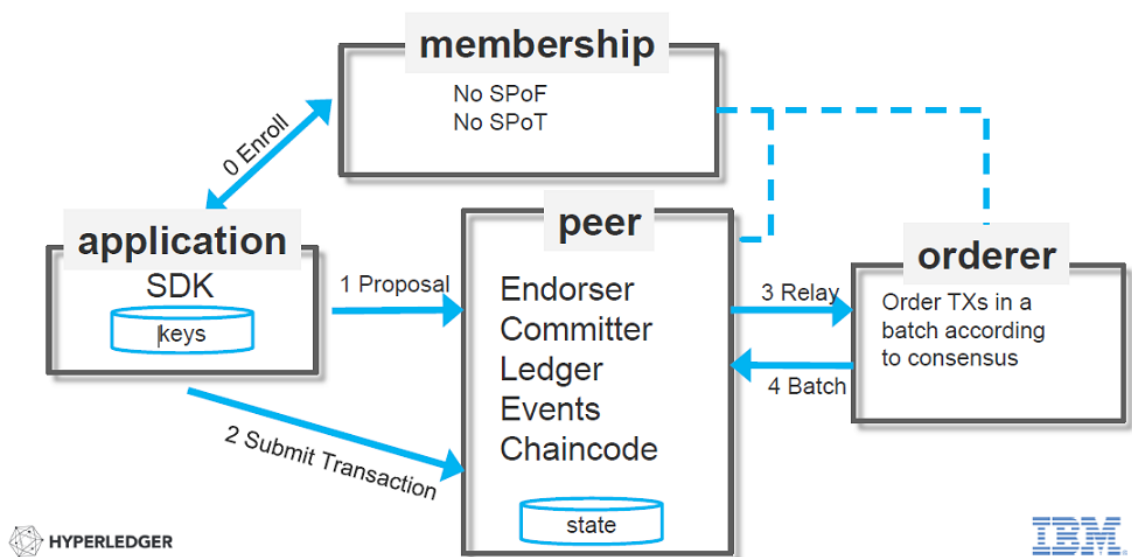
Nakon što je dovoljan broj, tj. postotak od svih čvorova, potvrđujućih čvorova potvrdio izvršavanje transakcija u odnosu na sveukupan broj čvorova, klijentska aplikacija šalje transakcijsku poruku čvorovima za slaganje transakcija. Transakcijska poruka obuhvaća

zahtjev klijentske aplikacije i odgovore čvorova te njihove potpise tako da klijentska aplikacija ne može krivotvoriti odgovore potvrđujućih čvorova.

Čvor za slaganje transakcija slaže transakcije u nove blokove za svaki kanal kojeg opslužuje te ih dostavlja na sve čvorove unutar pojedinog kanala.

Čvorovi još jednom provjeravaju transakciju te je pišu u svoj *blockchain* i izvršavaju na svojoj bazi ako je valjana, odnosno samo pišu na *blockchain* ako nije valjana i označuju kao ne valjalu.

Fabric v1.0 Architecture



Slika 1.4: Dijagram toka transakcije u Hyperledger Fabricu

2. Platforma raspodijeljene knjige Ethereum

2.1. Općenito

Ethereum platforma je nastala 2014. godine kada su njen osnivač Vitalik Buterin te nekolicina ljudi uz njega, koji su do tada radili na Bitcoin projektu, predložili novu blockchain platformu koja bi imala mnogo više mogućnosti u odnosu na Bitcoin platformu. Uvidjevši mogućnosti koje bi nastale implementacijom takve platforme, na inicijativu Vitalika, priključili su se mnogi programeri diljem svijeta te je projekt postao projektom otvorenog koda za razliku od većine tadašnjih kripto platformi (eng. *crypto platforms*).

Inovacije naspram ostalih tadašnjih platformi vezanih uz kriptovalute (eng. *cryptocurrency*) su uključivale poseban *Turing-complete* programski jezik Solidity pomoću kojeg je moguće pisati pametne ugovore odnosno aplikacije koje bi se decentralizirano izvršavale u slučaju kada ih pozove neka vanjska aplikacija, a rezultat bi ostao zabilježen u blockchainu. [5]

Namjena Bitcoin platforme bila je zaobići bankarski sustav, te transakcije odrađivati bez posredstva treće strane. Tim uvjetom su se vodili programeri u Ethereumu te su pomoću pametnih ugovora uspješno



Slika 2.1: Ethereum logo

zaobišli treću, posredničku stranu pri ugovaranju poslova u mnogim slučajevima. Također unutar Ethereum platforme moguće je i osnivati vlastite valute koje će se razmjenjivati u odnosu na Ether valutu platforme Ethereum ili biti samostalne valute.

2.2. Ethereum u odnosu naspram drugih platformi

Ethereum je prva platforma koja je razvila mogućnost spremanja i izvršavanja *Turing-complete* programskih kodova na blockchain platformu. Kako je Ethereum platforma javna odnosno to je sustav bez dozvole za ulaz na mrežu, svi mogu vidjeti sve podatke te se podatci koje korisnik ne želi otkriti moraju kriptirati. Zbog takvog načina rada sustava, platforma je cijenjena u zajednici ljudi koji smatraju da sve mora biti transparentno i anonimno, ali teško dolazi do primjene u bankarskom i korporativnom sustavu u kojem se podatci moraju biti sigurni te u pojedinim slučajevima ne smiju biti anonimni.

Vođeni ovim zahtjevima izgrađen je trenutni sustav:[12]

- Jednostavnost – Jednostavniji protokol je bolji, bez obzira na skupoću ili sporije izvođenje
- Univerzalnost – Uz pomoć programskog jezika Solidity se mora moći napraviti proizvoljna aplikacija
- Modularnost – Pomoću raznih dodatnih modula kreiranih pomoću pametnih ugovora ili biblioteka vezanih uz Ethereum se mora moći dodati proizvoljne usluge
- Agilnost – Poželjna je modifikacija Ethereum protokola i modula kako bi funkcionirali bolje i brže
- Nediskriminirajući i necenzurirajući protokol – Ako netko poželi napraviti loše, ali validne transakcije, ne pokušavati ga blokirati

2.3. Karakteristike Ethereum platforme

Ethereum platforma je predvodnik programirljivih platformi za blockchain sustave te kao takva je imala značajan utjecaj na razvoj kasnijih sličnih platformi. Ipak jedina platforma s kojom se Ethereum mogla uspoređivati u trenutku nastanka je Bitcoin platforma. Mogućnost već opisanih pametnih ugovora koji bi eliminirali posredničku ulogu, potakla je mnoge korisnike i entuzijaste novih tehnologija da ulažu vrijeme i novac u Ethereum platformu.

2.3.1. Sustav bez dozvole za ulaz na mrežu Ethereum

Ethereum platforma slijedi princip interneta za princip ulaska novih čvorova i sudionika u mrežu. Princip je potpuna otvorenost prema svima dakle svi koji posjeduju želju za spojiti se na Ethereum mrežu mogu to učiniti. [13] Također svi sudionici u mreži su pseudoanonimni. Jedina poveznica sa svijetom je adresa njihovog novčanika tj. pametnog ugovora za aplikacije, te ako sudionik ne otkrije svoj identitet u stvarnom svijetu, nitko ne može znati tko se krije iza te adrese. Problem takvog pristupa su regulacije banaka i tvrtki koje posluju s bankama te oni zahtijevaju otkrivanje identiteta u slučaju želje sudionika mreže za poslovanjem s njima. Kako bi spriječili zlonamjerne transakcije i varanje, te ujedno zadovoljili stroge bankarske regulacije, u duhu necenzurirajućeg protokola ostavili su mogućnosti izbora pravnim i regularnim osobama da dokažu svoje postojanje u pravom svijetu. Informacija se može objaviti na internetskoj stranici za pretraživanje Ethereum glavnog lanca.

2.3.2. Brzina

Brzina naspram krypto platformi nastalih kada i Ethereum igrala je značajnu ulogu u populariziranju Ethereum platforme. Prosjek kreiranja novog bloka u trenutku pisanja rada je 15tak sekundi što je značajno brže od platformi nastalih kada i Ethereum, ali i sporije naspram novih platformi koji imaju drugačiji pristup kreiranju transakcija i blokova.

Broj transakcija po sekundi na javnom odnosno glavnom lancu (eng. *main chain*) Ethereum iznosi oko 20 transakcija po sekundi. *Main chain* koristi PoW algoritam Ethash, a ako se na privatnim *chainovima* koristi PoA algoritam u sklopu drugih protokola rudarenja novih blokova, može se postići i do 44 transakcije po sekundi. [7]

2.3.3. Sigurnost i algoritmi za kreiranje transakcija

Za kreiranje odnosno potvrđivanje novih blokova trenutno se koristi BFT PoW Ethash algoritam baziran na SHA3-256 algoritmu te dotični algoritam smanjuje prednost ASIC računala uz pomoć kojih bi određene strane mogle kontrolirati veći dio mreže i time narušiti svojstvo decentraliziranosti kao što se dogodilo na pojedinim platformama. ASIC je skup procesora koji su napravljeni za samo jednu namjenu, recimo računanje hash-a, te ništa drugo ne mogu obaviti. Međutim Ethash algoritam je specifičan tako da se često

moraju dohvaćati vrijednosti iz RAM memorije i to vrijednosti koje u većini slučajeva nisu unutar cache memorije RAMa što usporava procesore u ASICu. Ethereum je zbog inovativnog načina sprječavanja računalnih tvrtki da proizvedu ASICe, približio rudarenje novih blokova običnim korisnicima s grafičkim karticama. Zbog toga je puno teže izvesti 51% napad, situacija u kojoj određena grupa ima 51% cjelokupne računalne snage te kontrolira kreiranje blokova i ništa ih ne sprječava da odobre ne valjale transakcije što je veliki problem kod najpopularnije Bitcoin platforme. Korisnici koji izrudare novi blok trenutno osim transakcijske naknade koje za izvođenje transakcija, dobiju i određenu količinu novog Ethera, pa je stoga Ether trenutno inflacijska valuta bez limita u broju izdanog novca.

Algoritam koji koriste PoA lokalni *chainovi* je puno brži, ali i nesigurniji. Naime kod PoA algoritama, samo određeni čvorovi mogu rudariti nove blokove te ako je neki od čvorova zlonamjerman, on ne mora odobriti valjanu transakciju i tu nastupa cenzura. Zbog toga se PoA algoritmi koriste većinom samo za testiranje mreže i pametnih ugovora.

U budućnosti je moguće da Ethereum platforma pređe na PoS algoritam CASPER koji je trenutno u fazi razvoja. PoS algoritmi ne zahtijevaju gotovo nikakvu računalnu snagu već novi blok kreira slučajna osoba proporcionalno količini valute te platforme. Kako bi spriječili da osobe ulažu svoj novac na više potencijalnih novih blokova te time povećaju mogućnost svog dobitka, predviđene su kazne oduzimanjem novca, kao i kazne za 51% napad. Korisnici koji kreiraju novi blok bi kao nagradu dobili transakcijske naknade od *Gasa* te bi se uvođenjem ovog algoritma zaustavila inflacija novog Ethera. Međutim kako i taj algoritam ima svojih nedostataka kao što su da se bogatiji sve više lakše bogate, očekuje se da će algoritam biti modificirana verzija trenutnih algoritama. [8] [9]

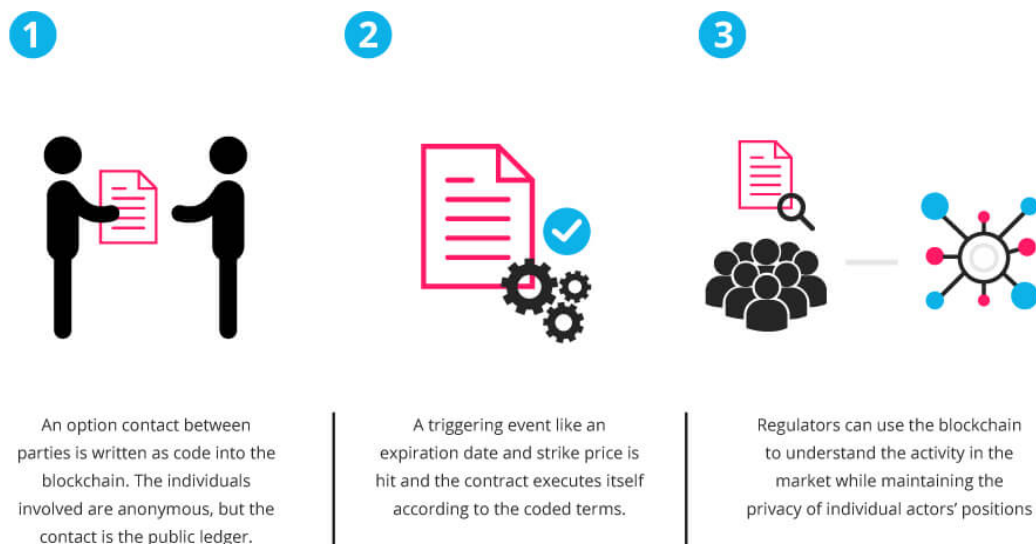
Cijena transakcije ovisi o kompleksnosti pametnog ugovora, te korištenja memorije u istom, te trenutnoj cijeni goriva (eng. *gas*) kojim se pokriva izvršenje transakcije. Prosječna cijena *Gasa* ovisi o količini trenutnih rudara te cijeni glavne valute Ether. Poželi li netko osigurati da se njegova transakcija prije izvrši, tada će staviti veću cijenu za *gasPrice*, te će mu rudari, zbog veće količine novca kojeg će dobiti ako izvrše baš njegovu transakciju, vjerojatnije izvršiti transakciju. Količina *Gasa* koju netko može staviti definirana je parametrom *gasLimit* te je to maksimalna količina koja će se potrošiti, odnosno koju će rudar novog bloka dobiti kao nagradu za izvršenje transakcije, a ako se ne potroši sav *Gas*, vraća se originalnom pošiljatelju transakcije. Stoga je bolje postaviti veći *gasLimit* tako da se transakcija sigurno izvrši. [6] Cijena transakcije u kojoj se samo šalje

valuta s jednog računa na drugi je iznimno niska te je platforma i što se tiče brzine prihvatanja transakcije i cijene transakcije, u značajnoj prednosti nad većinom popularnijih platformi.

2.3.4. Pametni ugovori

Pametni ugovori na Ethereum platformi napisani su u programskom jeziku Solidity koji je posebno smišljen *Turing-complete* programski jezik za dobivanje nedeterminističkih rezultata transakcija u raspodijeljenom okruženju. Objektno je orijentiran jezik te ga se često uspoređuje s JavaScriptom osim zbog sličnosti u sintaksi, i u prenosivosti. Pametni ugovori se izvršavaju u sklopu transakcije na svim čvorovima u mreži, a izvršava ih Ethereumov virtualni stroj (EVM). EVM interpretira pametni ugovor tako da kod pretvori u *bytecode* ugovora te onda pretvori u set instrukcija u ovisnosti o platformi na kojoj se nalazi.[9] EVM postoji za sve platforme na kojima je podržan Ethereumov novčanik (eng. *Ethereum Wallet*) te bi se u slučaju prelaska na PoS algoritam, rudarenje blokova odnosno izvršavanje transakcija moglo odvijati i na uređajima koji troše zanemarivo malo energije.

Jedan od bitnih događaja vezan uz rani razvoj Ethereumu i korištenja pametnih ugovora su virtualne organizacije koje bi se stvarale unutar platforme, te bi djelovale kao organizacije u pravom svijetu uz mogućnost upravljanja resursima od strane svakog člana koje bi te organizacije imale, naravno sve decentralizirano. Međutim kako su pametni ugovori zapisani na *blockchainu*, te ih nakon što su dodane u *blockchain*, naknadno ne može mijenjati, postali su metom *hackera*. Dolazi se do problema sigurnosti koji se spominje unutar blockchain platformi, a to je da ako postoji pogreška u pametnom ugovoru, jedino rješenje je konsenzus svih korisnika za odbacivanjem problematičnog ugovora te kreiranjem novog. Međutim konsenzus služi i kako zlonamjerna strana ne bi u nekom trenutku promijenila pametni ugovor za svoj dobitak.



Slika 2.2: Način rada pametnih ugovora na Ethereum platformi

Pametnim ugovorima se najčešće žele stvoriti decentralizirane aplikacije koje funkcioniraju kao eliminacija posredničkih aplikacija te koriste ili Ether kao valutu ili kreiranu valutu putem pametnih ugovora. [11] Broj decentraliziranih aplikacija u sklopu Ethereum platforme je poveći te se može smatrati da je Ethereum, unatoč svim nedostacima, trenutno najuspješnija decentralizirana javna platforma pomoću koje je moguće izvršavanje pametnih ugovora.

2.4. Mreža Ethereum platforme

Ethereum mreža je otvorenog tipa, pa svi mogu sudjelovati u mreži. Jedino što sudionici trebaju napraviti jest generirati javni i privatni ključ, a adresa je zadnjih 20B javnog ključa. Čvorovi koji žele potvrđivati transakcije odnosno žele rudariti, po trenutnom PoW algoritmu moraju preuzeti cijeli *blockchain* do zadnjeg bloka lanca na koji se spajaju te pokrenuti program koji računa Ethash algoritam i spaja se na glavni lanac.

2.4.1. Sastavnice mreže

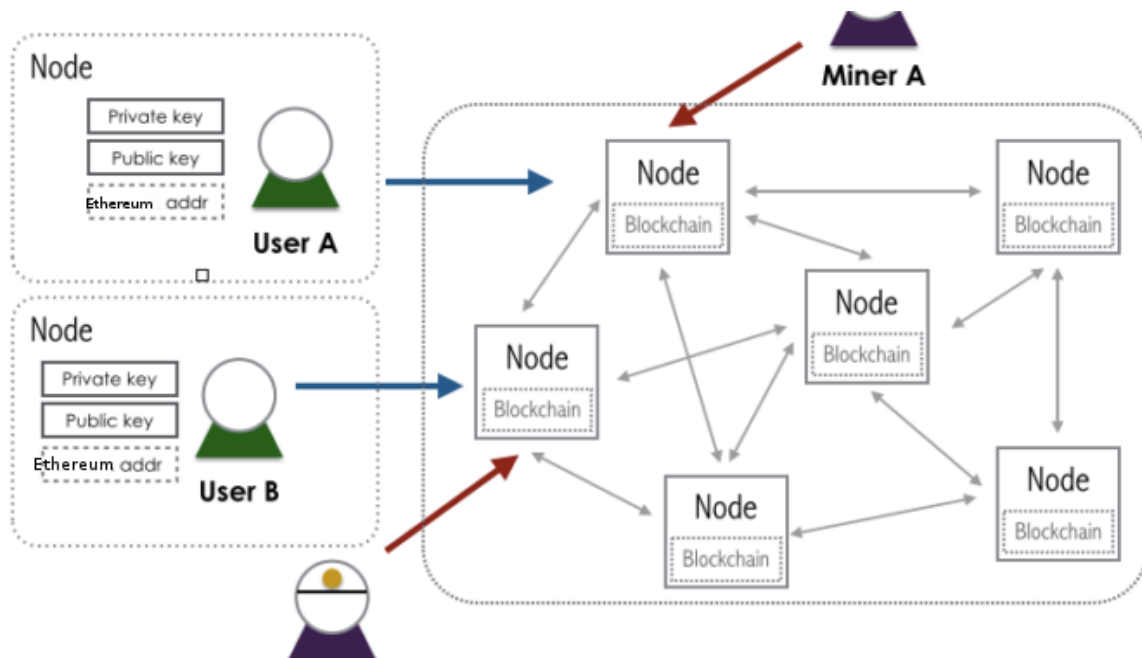
Mreža se sastoji od:

1. **Knjige zapisa (eng. *ledger*)** – Sastoji se od *blockchain* strukture koja je složena po Merkle Tree algoritmu te sadrži sve transakcije i *hash*ove blokova do zadnjeg trenutka te *World state* koja je također složena po Merkle Tree strukturi, ali sadrži mapiranje (eng. *mapping*) na adrese koje su se koristile za izračune valjanosti transakcija te stanja na tim računima
2. **Čvorovi (eng. *nodes*)** – Sudionik u mreži koji ima kopiju *blockchaina* te potvrđuje nove blokove. Postoje čvorovi koji rudare nove blokove te takvi moraju imati cijeli *blockchain*, a čvorovi koji se žele samo spojiti na mrežu mogu imati samo najnovijih n blokova gdje je n određen posebno za svaki protokol za lagane čvorove
3. **Pametni ugovori (eng. *smart contracts*)** – Programski kod izvršavan decentralizirano na svim čvorovima u mreži

2.4.2. Tok transakcija

Potencijalni novi čvorovi moraju pokrenuti klijent za spajanje na željeni protokol. Klijent koristi DevP2P protokol za traženje i spajanje na druge čvorove te počne sinkronizaciju na željeni lanac.

Nove transakcije potpisuju na lokalnom čvoru ako su valjane u odnosu na stanje na lokalnom čvoru te se šalju preko lokalnog čvora na ostale čvorove koji ih mogu provjeriti te širiti dalje.



Slika 2.3: Čvorovi s posljednjih n blokova šalju transakcije čvorovima rudarima

Zadužen li je čvor za kreiranje novih blokova odnosno rudarenje, nove transakcije također provjeri, kompajlira te slaže u blokove. Kada čvor dosegne limit *gasa* po bloku, zapisan je u postavkama Ethereum klijenta i ako je veći, blok se ne priznaje kao validan te isto vrijedi i za limit na veličinu bloka, prekine sa slaganjem novih transakcija u blokove.

Tada pokuša izračunati Ethash algoritam s odgovarajućim parametrima koji uključuju potencijalni novi blok te prijašnje blokove. Pronađeno li je odgovarajuće rješenje, novi blok i pošiljatelj se odašilju svim trenutnim povezanim čvorovima sve dok svi ne dobiju novi blok i potvrde da je valjan. Nakon što potvrde da je valjan, čvorovi izvrše sve transakcije i pametne ugovore u novom bloku te krenu kreirati novi blok. Zbog brzine kreiranja novih blokova, a sporije brzine rasprostranjenja rješenja, čeka se minimalno desetak novih blokova, a preporučuje se i puno više kako bi se sa sigurnošću moglo tvrditi da je transakcija uvrštena u blok. [15]

3. Aplikacija za kontrolu ulaza

Aplikacija na uređajima za kontrolu ulaza funkcionira na način da nakon što korisnik prođe kroz uređaj, uređaj očita RFID na kojemu su TID i JMBAG. TID je jedinstvena oznaka koja se ne može mijenjati jednom kada je zapisana na RFID karticu u tvornici. Ako netko traži ulaz s istim JMBAG-om, a različitim TID-om odnosno različitom karticom, može se ustvrditi da je kartica krivotvorena. JMBAG je također jedinstvena oznaka, ali ona je programabilna te se uz posebne uređaje može modificirati.

Aplikacije simulacije uređaja za kontrolu ulaza koriste tehnologiju Node.JS. Node.JS je platforma koja koristi JavaScript jezik kako bi pokretala serverske aplikacije, te zbog toga što Node.JS pruža jednostavno dodavanje potrebnih biblioteka koje su potrebne aplikaciji za kontrolu ulaza logičan izbor.

Hyperledger Composer pruža biblioteke za rad s JavaScriptom te trenutno Composer biblioteke za JavaScript imaju najviše funkcionalnosti od ostalih podržanih programskih jezika. Ethereum platforma pruža biblioteku Web3.js koja se sastoji od nekoliko modula koji omogućavaju vrlo laku interakciju s Ethereum platformom.

3.1. Hyperledger platforma

Aplikacija na Hyperledger platformi sadrži zapisane korisnike i sveučilišne komponente u svom blockchainu. Pošto se na Hyperledgeru vjeruje čvorovima u kanalu, nema potrebe skrivati podatke. Kada aplikacija na uređajima očita vrijednosti iz RFID-a, kreira transakciju i pošalje na *chaincode*. Aplikacija odnosno pametni ugovor na Hyperledger Fabric platformi vraća rezultat i ovisno o rezultatu aplikacija propušta korisnika ili ne.

3.1.1. Hyperledger Composer programski alat

Alat koji mnogo pomaže u razvoju Hyperledger Fabric aplikacija je Composer. Composer je programski okvir otvorenog koda namijenjen lakšoj integraciji aplikacija Hyperledger Fabric *blockchainu*. [16] Hyperledger Composer pruža biblioteke za rad s

JavaScriptom te trenutno Composer biblioteke za JavaScript imaju najviše funkcionalnosti od ostalih podržanih programskih jezika. Pristup koji Composer zastupa jest preslikavanje postojećih aplikacija i modela baza podataka u aplikaciju i model koji bi odgovarao *blockchainu* uz minimalne nužne modifikacije i poznate programske jezike kako bi se tvrtke što lakše odlučile prijeći na Hyperledger Fabric *blockchain* platformu.

Implementiran je model dozvola koji na temelju tko je osoba i kojoj preddefiniranoj skupini pripada, omogućava određene akcije. Takav model nadopuna je modelu obaveznog ulaska u komunikacijski kanal na kojoj se temelji Fabric.

Logika transakcija se piše u trenutno iznimno popularnom JavaScriptu te programeri vrlo lako mogu modelirati točno ono što im treba kako bi vratili podatke svojoj aplikaciji. Modeliranje baze je u jeziku bliskom OOP jezicima, a postoji i mogućnost upita koji također vrlo slično SQL jezicima mogu vratiti određene strukture podataka.

1) Modeliranje baze chaincodea

Postoje tri vrste modela u sustavu modeliranja baze koju koristi Fabric Composer:

- Imovina (eng. *Asset*) - Sve što je neživo, prenosivo, može biti materijalno i nematerijalno, razna dobra i usluge
- Korisnici (eng. *Participants*) – Sve što se može smatrati živim, nečim što treba potvrditi svoju postojanost identitetom koji se može i ne mora dodati, rade transakcije da bi promijenili bazu/chaincode. Mogu imati svoje identifikacijske kartice
- Transakcije (eng. *Transactions*) – Prenosjenje, kreiranje, brisanje, dodavanje određenih svojstava korisnicima/imovini određeno dozvolama koje korisnici imaju za promjenu stanja
- Događaji (eng. *Events*) – Složena struktura podataka koja služi za vraćanje podataka nakon izvršene transakcije

Modeliranje baze se radi u datoteci `models/*.cto`. Najsličnije je modeliranju baze pomoću klasa u objektno orijentiranim jezicima te se programeri mogu brzo naviknuti što je velika boljka naspram drugim blockchain platformi. Specifičnost modeliranja naspram objektno orijentiranih jezika jest postojanje pokazivača (eng. *pointers*) na strukture, korisnike i imovinu te se često koristi u transakcijama kako bi izmijenili točno to što

želimo i umanjimo mogućnosti pogreške ili kreiranja novog objekta s identičnim identifikatorom. Novi podatci označeni su malim slovom "o", a pokazivači strelicom "-->". Strukture mogu biti indeksirane s obzirom na određeni podatak unutar strukture, obično je to Id podatak.

Moguće je postaviti regularne izraze (eng. *regex*) na nizove znakova, ograničenja u brojevima za cjelobrojne vrijednosti te ograničenja za datume u *DateTime* tipu podataka. Postoje enumeracije koje kada ih stavimo, dotično može poprimiti vrijednost samo podatka iz enumeracije. Koncepti su zapravo strukture, složeni tipovi podataka sastavljeni od nekoliko jednostavnih ugrađenih u jezik za modeliranje, međutim koncepti trenutno ne funkcioniraju na željen način u svim slučajevima te s toga treba biti oprezan s njima. [18]

2) Upiti nad blockchainom

Postavljanje upita slično je kao u SQL jezicima, a datoteka u kojem su zapisani je `queries.qry`. Moguće je vratiti cijele klase ili pojedinačne instance, te postaviti uvijete nad izrazima i podacima u strukturi kojoj pristupamo. Podacima se brže pristupa ako su prethodno indeksirani unutar modeliranja baze. [19] Trenutno postoji ograničenje pod kojim se ne mogu vraćati točno određeni podatci unutar strukture koja nas zanima već moramo vratiti cijelu strukturu. Postoji i ograničenje u kojemu ne možemo pridružiti nekoliko relacija čak i ako su u istoj klasi, te nad njima raditi upite. Postoje i filtri kojima možemo dodatno na krajnjem sučelju filtrirati podatke koje dobijemo u JSON formatu.

3) Kreiranje uvjeta nad pristupom blockchainu

Modeliranje pristupa transakcijama i njihovo izvršavanje omogućeno je s datotekom `permissions.acl`. Moguće je postaviti uvjet na mijenjanje, kreiranje i brisanje određenih korisnika, imovine te pogled na njih. Također je moguće postaviti uvjet i na izvršavanje transakcija te će transakcija izvršiti sve što je u njenoj logici, ali korisnik neće moći pojedinačno izvršiti modifikacije koje se nalaze u transakciji ako mu to nije dopušteno. Trenutno nije moguće napraviti dozvole za mijenjanje samo određenog podatka u nekoj strukturi ili cijeloj klasi.

U modeliranju pristupa se računa da sve što nije eksplicitno dopušteno, je zabranjeno. Moguće je obuhvatiti cijele klase, pa to pojedinačnih instanci. Zbog toga se datoteka gleda

od prvog reda gdje bi trebali biti najspecifičniji uvjeti, te kako se približava kraju datoteke trebaju biti općenitiji uvjeti. Evaluacija uvjeta prestaje čim se prvi uvjet zadovolji. [20]

4) Poslovna logika Composer aplikacije

Poslovna logika transakcija zapisuje se u datoteci `lib/logic.js` te je definirana JavaScriptom. Programerima stoje na raspolaganju sve inicijalne funkcije JavaScripta verzije ES2017 i ranije. Funkcije primaju jedan parametar koji je zapravo JSON niz znakova u kojemu se mogu nalaziti različiti potrebni podatci za obavljanje transakcije. Konvencija je da se proslijedi pokazivač na resurse potrebne za izvršavanje transakcije te da se što manje prosljeđuju dodatni podatci ili podatci koje čvor samostalno može izgenerirati. Svi pozivi funkcija su asinkroni te na to treba posebno pripaziti kako se ne bi dogodilo da u sljedećem koraku ne uvrštavamo nikakvu vrijednost. Na kraju funkcije za transakciju, emitira se događaj koji sadrži podatke o transakciji te služi kako bi aplikacija koja se spaja na blockchain dobila povratne informacije o uspješnosti transakcije.

5) Mreža Composer aplikacije i spajanje na Hyperledger Fabric blockchain

Identifikacijske kartice (eng. *identification card*) služe kako bi se korisnici dodatno identificirali na mreži ako je to potrebno, a osnova po kojemu se korisnici identificiraju jest indeksirani identifikator unutar *Participants* strukture. Pomoću identifikacijske kartice, ako postoje dovoljne ovlasti, se mogu obavljati transakcije, kreirati, brisati te modificirati razne strukture i dodavati nove identifikacijske kartice. Uz administratorske ovlasti mogu se instalirati nove aplikacije na čvorove i kanale.

Nakon što se napišu svi dijelovi Composer aplikacije, aplikacija se pakira u `.bna` format vrlo sličan `.zip` formatu. U takvom formatu, aplikaciju je moguće instalirati na kanal odnosno na svaki čvor posebno u mreži. Aplikacija se instalira pomoću kartica identifikacije, a da bi se aplikacija instalirala, kartica mora imati admin prava na čvor ili kanal. [17]

Mrežu je moguće imati lokalno na jednom računalu, lokalno na mreži računala te raspodijeljeno u oblaku kojeg IBM nudi za potrebe Hyperledger Fabric *blockchaina*. Frontend aplikacije se spajaju preko REST sučelja na blockchain mrežu, a Node.JS aplikacije preko interne Hyperledger Fabric biblioteke koja se uvozi u Node.JS aplikaciju.

3.1.2. Konfiguracija Hyperledger Fabric mreže i čvorova za aplikaciju kontrole ulaza

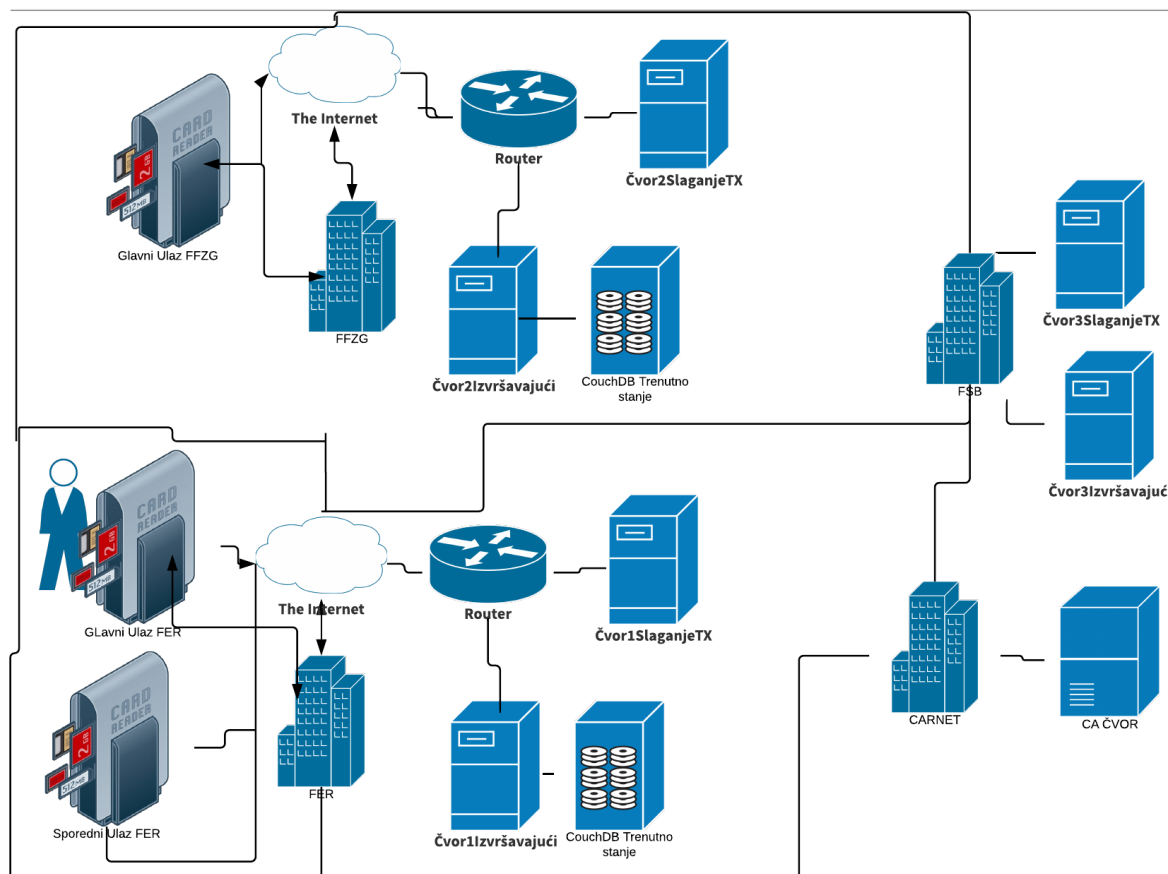
Hyperledger Fabric čvorovi se najjednostavnije pokreću preko Docker slika. Docker slike su izvrstan način za pokretanje mnogo aplikacija u pozadini koji trebaju imati točno određene parametre i biti stalno dostupni, a da ne troše puno resursa na računalima domaćinima. Naime Docker koristi princip virtualnih mašina, ali koriste operativni sustav računala domaćina za izvršavanje sistemskih (eng. *kernel*) zadaća. Također jednom napisane postavke za Docker sliku mogu se izvršiti nebrojeno puta i same pokrenuti po potrebi te time olakšavajući administraciju sustava i pokretanje više instanci.

Prije nego što se kreiraju čvorovi, moraju se izgenerirati privatni i javni ključevi te certifikati za čvorove. Za slučaj da kreiramo CA čvor, samo za njega treba izgenerirati podatke. Certifikati služe kako bi čvorovi potvrdili drugim čvorovima da su to oni. CA čvor inače kreira certifikate za druge čvorove kad ulaze u mrežu, pa u slučaju da imamo CA čvor u mreži, samo za njega treba izgenerirati certifikate. Problem s CA čvorom je što njemu moramo vjerovati, pa u slučajevima jedne tvrtke to treba biti nadležna uprava, a u slučaju više tvrtki ili države bi trebala biti regulativna agencija kojoj vjerujemo. U nekim slučajevima ne možemo vjerovati ni takvim agencijama, pa u mreži ne bi trebao postojati CA čvor već se unaprijed treba dogovoriti oko čvorova koji sudjeluju u mreži.

Certifikati se mogu generirati Fabric CryptoGen alatom koji za parametre dobiva broj organizacija i čvorova unutar organizacije kojima mora izgenerirati certifikate. Početni blok (eng. *Genesis block*) se može izgenerirati Fabric ConfigTxGen alatom koji za parametar prima `configtx` datoteku u koji piše konfiguracija svih čvorova i organizacija u mreži.

U ovom slučaju pokrenu se redom Docker slike:

1. Docker slika za Fabric CA na vratima (eng. *port*) 7054 s certifikatom i privatnim ključevima za prvi čvor unutar željene organizacije
2. Čvor za slaganje transakcija na portu 7050 te izgeneriranim *genesis* blokom
3. Čvor željene organizacije s portovima 7051 i 7053
4. CouchDB baza podataka za čuvanje trenutnog stanja blockchaina na portu 5984



Slika 3.2: Mogući dijagram mreže Hyperledger Fabric Composer aplikacije za kontrolu ulaza. Pošto se samo simulira sustav kontrole ulaza, a računalo na kojem se izvodi je preslabo za više čvorova, ovakav postav je dovoljno dobar. Za slučaj produkcije u kojem bi bilo više čvorova i organizacija će biti nešto više napisano u poglavlju "Usporedba platformi".

Administratorske ovlasti na kanalu i u čvoru se dobivaju putem identifikacijske kartice s administratorskim ovlastima. Identifikacijska kartica se kreira putem Composer naredbe u kojoj će za administratorske potrebe prvog čvora u mreži stajati privatni ključevi prvog čvora i njegov certifikat te je potrebno predati i JSON u kojem se nalazi struktura mreže koju taj čvor može vidjeti. Mreža se nakon toga inicijalizira naredbom `composer network start` s parametrima Administratorskih ovlasti za taj čvor na kojem se pokreće.

3.1.3. Logika Composer blockchain aplikacije za kontrolu ulaza

Kao što je već navedeno u poglavlju 2.1, logika Composer aplikacije se sastoji od četiri stavke. U ovom poglavlju će se objasniti sve stavke na primjeru aplikacije za

kontrolu ulaza. Sav kod je napisan pomoću Visual Studio Code alata sa dodatcima za JavaScript i Composer sintaksu.

1) Modeliranje baze blockchaina aplikacije za kontrolu ulaza

Prostor imena koji se koristi u aplikaciji je **org.szg** koji je odabran zato što označava organizaciju pod imenom szg tj. Sveučilište Zagreb. U modelu se sveukupno nalaze dvije enumeracije, dvije vrste korisnika, tri transakcije, jedna imovina te jedan događaj.

Enumeracije `UniversityComponentName` te `MemberType` služe kako bi standardizirali vrste fakulteta i osoba koje postoje te si olakšali kontrolu upisa novih fakulteta i osoba. Naime u ako je nešto pod enumeracijom, onda može poprimiti vrijednost

Hyperledger Composer REST server		
CheckAccessFSB : A transaction named CheckAccessFSB		
Show/Hide	List Operations	Expand Operations
GET	/CheckAccessFSB	Find all instances of the model matched by filter from the data source.
POST	/CheckAccessFSB	Create a new instance of the model and persist it into the data source.
GET	/CheckAccessFSB/{id}	Find a model instance by {{id}} from the data source.
Member : A participant named Member		
Show/Hide	List Operations	Expand Operations
GET	/Member	Find all instances of the model matched by filter from the data source.
POST	/Member	Create a new instance of the model and persist it into the data source.
GET	/Member/{id}	Find a model instance by {{id}} from the data source.
HEAD	/Member/{id}	Check whether a model instance exists in the data source.
PUT	/Member/{id}	Replace attributes for a model instance and persist it into the data source.
DELETE	/Member/{id}	Delete a model instance by {{id}} from the data source.
Query : Named queries		
Show/Hide	List Operations	Expand Operations
GET	/queries/selectIsFEROpened	Query is particular collage FER in working time
GET	/queries/selectIsFFZGOpened	Query is particular collage FFZG in working time
GET	/queries/selectIsFSBOpened	Query is particular collage FSB in working time
GET	/queries/selectIsOpened	Query is particular collage in working time
GET	/queries/selectMember	Select members with particular jmbag
GET	/queries/selectMembers	Select all members
GET	/queries/selectMembersOfUniversityComponent	Select all members of particular UniversityComponent

Slika 3.3: Razni resursi kojima se može pristupiti preko REST sučelja samo iz enumeracije te se s time sprječava unošenje lošeg unosa. Primjer enumeracije koja se koristi u aplikaciji za kontrolu ulaza:


```
enum MemberType {
    o Student
    o Profesor
    o Staff
}
```

Participant SystemAdministrator klasa se koristi za osobe koje bi bile ovlaštene za provedbu kontrole ulaza na fakultet odnosno slanje transakcija na čvorove u Hyperledger Fabric kanalu. Njima se izdaju i posebne identifikacijske kartice preko kojih šalju transakcije. Participant Member klasa se razlikuje od SystemAdministrator klase po tome što se instancama Members klase ne izdaju identifikacijske kartice. Obje klase se identificiraju po JMBAG podatku te u strukturi primaju jednake podatke. Osim niza znakova JMBAG koji mora imati točno 10 brojeva, struktura sadrži niz znakova za ime, prezime, TipOsobe i TID koji mora imati točno 24 znaka. Pokazivač na komponentu sveučilišta kojoj pripada te polja za niz znakova za svaku transakciju u kojoj je ta osoba tražila pristup. Primjer Participant Members klase:

```
participant Member identified by jmbag {
    o String jmbag regex=/[0-9]{10}/
    o String firstName
    o String lastName
    --> UniversityComponent universityComponent
    o String[] transactionAuthorized optional
    o String[] transactionRevoke optional
    o MemberType memberType
    o String tid regex=/^[A-Z-0-9]{24}/
}
```

Klasa Asset postoji samo kao UniversityComponent te sadrži sve bitne podatke za konkretni fakultet odnosno komponentu sveučilišta. Instance se identificiraju pomoću niza znakova universityKey koji ima točno četiri broja. Dodatno struktura sadrži niz znakova za enumeraciju i ime sveučilišne komponente, brojeve u sekundama u danu kad se

komponenta sveučilišta otvara i zatvara te polja za niz znakova za svaku transakciju u kojoj je toj sveučilišnoj komponenti netko tražio pristup. Primjer klase Asset:

```
asset UniversityComponent identified by universityKey {
  o String universityKey regex=/[0-9]{4}/
  o UniversityComponentName universityName
  o String ComponentName optional
  o Integer opening range=[0,86400]
  o Integer closing range=[0,86400]
  o String[] transactionAuthorized optional
  o String[] transactionRevoke optional
}
```

Tri vrste transakcije su identične u modelu, pa ih se može grupirati, to je učinjeno pomoću abstract modifikatora u kojoj potklasa nasljeđuje svojstva abstract klase MemberAccess. Struktura transakcije za kontrolu ulaza sadrži pokazivač na korisnika koji traži ulaz te pokazivač na komponentu sveučilišta koja prima zahtjev za ulazom. Također šalje se i TID korisnikove kartice koja traži ulaz kako bi se provjerilo je li kartica ta koja je zapisana u blockchainu, a ne krivotvorina. Transakcija CheckAccessFER koja proširuje MemberAccess nasljeđuje sva svojstva MemberAccess klase. Primjer klase transakcije:

```
abstract transaction MemberAccess {
  --> Member member
  --> UniversityComponent universityComponent
  o String tid regex=/^[A-Z-0-9]{24}/
}
transaction CheckAccessFER extends MemberAccess {
}
```

Događaji se emitiraju na kraju izvršavanja transakcije te daju do znanja rezultat pozivatelju transakcije. Aplikacija za kontrolu ulaza u događaju vraća potvrđan odnosno negativan odgovor u prvom podatku u ovisnosti kako je transakcija izvršena te vraća sve podatke transakcije. Primjer događaja MemberEvent iz aplikacije za kontrolu ulaza:

```
event MemberEvent {
  o Boolean memberAccessBool
  o MemberAccess memberAccess
}
```

2) Upiti nad blockchainom aplikacije za kontrolu ulaza

Upiti kreirani za aplikaciju za kontrolu ulaza su raznovrsni s raznovrsnim mogućnostima. Postoje upiti koji vraćaju sve korisnike ili imovinu, upiti koji vraćaju samo instancu tih klasa u ovisnosti o proslijeđenim parametrima. Parametri kao što su JMBAG vraćaju točno jednu instancu neke klase, a parametri kao što su ime fakulteta za pojedinog korisnika vraćaju sve korisnike kojima je zapisano da im je matični upravo taj fakultet. Ipak te upite se gotovo i ne koristi u radu aplikacije za kontrolu ulaza te služe kao pokazni primjer upita i vježba. Upiti koji se koriste su za provjeru rada točno određenog fakulteta te kao rezultat upita dobivamo taj fakultet ili prazno polje ako je fakultet zatvoren. Parametri koji se predaju upitima označeni su znakom \$, te se provjeravaju u WHERE dijelu upita. Primjer upita koji se koristi u aplikaciji za kontrolu ulaza:

```
query selectIsFEROpened {
  description: "Query is particular collage FER in working
time"
  statement:
    SELECT org.szg.UniversityComponent

    WHERE (universityName == "FER" AND _$timeparam >=
opening AND _$timeparam <= closing)
}
```

3) Uvjeti nad pristupom modifikacije blockchaina aplikacije za kontrolu ulaza

Modeliranje uvjeta nad pristupom za administratore može biti kontroverzno na *blockchainu*. Pristup koji je odlučan za aplikaciju za kontrolu ulaza jest da postoje sistemski administratori za svaku komponentu sveučilišta te oni dobiju identifikacijsku karticu s kojom mogu kreirati transakcije koje se odnose na njihovo sveučilište. Naknadno je dodano pravilo po kojemu imaju pravo mijenjati podatke komponente sveučilišta. Tu treba napomenuti da ako sistemski administratori odluče biti zlonamjerni i obrišu određene podatke, ti podatci se i dalje vide u *Historian* dijelu tj. nikad se podatci iz blockchaina ne mogu obrisati. Više o ovoj napomeni i kontroverzama oko administratora u poglavlju "Usporedba platformi".

Također naknadno je dodana i mogućnost da sistemski administratori mogu vidjeti sve podatke od drugih sveučilišta. Korisnike mogu dodavati i mijenjati samo administratori čvorova i kanala, a to bi u pravilu bili administratori sveučilišta. Dodano je pravilo po kojemu administratori čvorova ne mogu mijenjati podatke sveučilišta i korisnika jednom

kad su kreirani. Takvo pravilo je napisano zbog kontroverze oko pojma administrator koji je svemogući te zbog trenutne ne mogućnosti zabranjivanja modifikacije samo određenih podataka unutar resursa kojim netko pristupa.

Primjer pravila po kojemu administrator FER sveučilišne komponente može pozivati transakciju `CheckAccessFER` te izvršavati modifikacije unutar `CheckAccessFER` transakcije :

```
rule FerSysAdminCanFERStuff{
    description: "FerSysAdminCanControlFERAccess"
    participant(p): "org.szg.SystemAdministrator#0036000000"
    operation: CREATE, READ, UPDATE
    resource(r): "org.szg.*"
    transaction: "org.szg.CheckAccessFER"
    condition: (r.getIdentifier() == "0036")
    action: ALLOW
}
```

Identifikacijske kartice koje se dodaju su administratorska identifikacijska kartica za administriranje kanala i svakog pojedinog čvora u mreži, te identifikacijske kartice za svaku komponentu sveučilišta kako bi mogli se postigla interakcija između *chaincode* i uređaja za kontrolu ulaza.

4) Poslovna logika Composer aplikacije za kontrolu ulaza

Svaka funkcija u poslovnoj logici prima jedan parametar koji je ustvari JSON niz znakova. Unutar JSON strukture aplikacije za kontrolu ulaza nalazi se pokazivač na korisnika koji je tražio pristup, pokazivač na komponentu sveučilišta kojoj je tražio pristup te TID kartice kojom korisnik traži pristup.

Poslovna logika aplikacije prvo provjerava je li predani TID jednak TID-u zapisanom unutar pokazivača na strukturu u blockchainu, ako je, kartica kojom je korisnik tražio pristup je valjana. Poslovna logika zatim provjerava kojoj vrsti osobe korisnik pripada te ako je korisnik tip osobe `Staff` ili `Profesor` na tom fakultetu, odobren mu je ulaz iako je fakultet zatvoren, a ako je `Student`, onda se provjerava je li za trenutno vrijeme tražena komponenta sveučilišta otvorena. Ako korisnik nije na tom fakultetu, onda mora

biti osoba tipa Profesor ili Student te fakultet mora biti otvoren kako bi imao pravo pristupa. Osobe tipa Staff na drugim fakultetima nikad nemaju pravo pristupa.

Primjer provjere je li korisnik s tog fakulteta te provjera tipa osobe:

```
if(authorize.member.universityComponent.universityName == "FER"){  
    if(authorize.member.memberType == "Profesor" ||  
authorize.member.memberType == "Staff")  
  
        control = true;  
  
    else{  
  
        let d = new Date();  
  
        let ttime = d.getHours()*hour+d.getMinutes()*minute;  
  
        let queryisOpened = await query('selectIsFEROpened',  
{timeparam : ttime});  
  
        if(queryisOpened[0].universityName == "FER") {  
  
            control = true;  
  
        }  
  
    }  
  
}
```

Primjer modificiranja polja niza znakova transactionAuthorized za sveučilišnu komponentu ako je osoba dobila dozvolu pristupa:

```
if(!authorize.universityComponent.transactionAuthorized) {  
  
    authorize.universityComponent.transactionAuthorized= [];  
  
    authorize.universityComponent.transactionAuthorized.push(authorize.member.jmbag+" "+ Date().toLocaleString());  
  
}  
  
else {  
  
    authorize.universityComponent.transactionAuthorized.push(authorize.member.jmbag+" "+ Date().toLocaleString());  
  
}
```

Primjer dodavanja novih informacija na *blockchain* za modificirane instance Member te UniversityComponent i emitiranje događaja da se transakcija odvila

```
const memberRegistry = await getParticipantRegistry('org.szg.Member');

    await memberRegistry.update(authorize.member);

const assetRegistry = await
getAssetRegistry('org.szg.UniversityComponent');

    await assetRegistry.update(authorize.universityComponent);

    // emit an event

const event = getFactory().newEvent('org.szg', 'MemberEvent');

event.memberAccessBool = control;

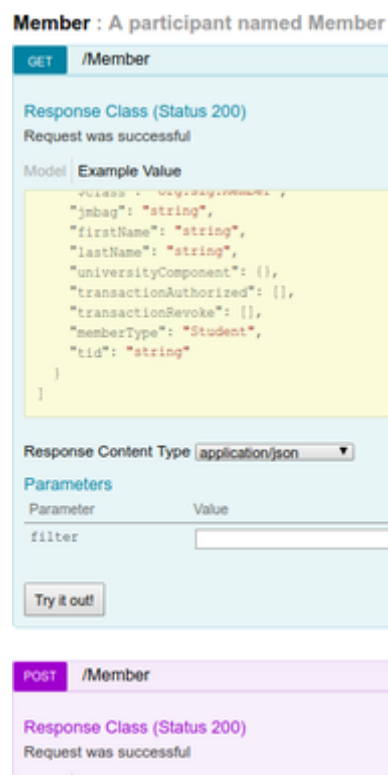
event.memberAccess = authorize;

emit(event);
```

3.1.4. Frontend Hyperledger aplikacije za kontrolu ulaza

- **REST sučelje**

Frontend aplikacija se oslanja na izgenerirano REST sučelje te svi podaci koji se šalju na *chaincode* ili se dobivaju su u JSON formatu pogodnom za REST sučelje. Problem kod trenutne verzije Hyperledger Composera je što ne postoji modul za autentifikaciju i autorizaciju nego se koristi identifikacijska kartica odabrana na početku rada REST servera. Većinom se koristi kartica administratora kanala te služi za sve operacije sa REST-om. U slučaju da se želi promijeniti autentifikacijske mogućnosti, mora se pokrenuti REST server na drugom portu odnosno za testiranje svih mogućnosti mora se imati paralelno uključeno nekoliko REST sučelja. Postoji vanjski modul za autentifikaciju Google OAUTH2.0 koji može pridonijeti poboljšanju autentifikacije međutim pošto se cijela autentifikacija odvija preko Google API-a, u određenim slučajevima je



Slika 3.4: Primjer slanja POST operacije na REST sučelju

takva praksa nepoželjna i štetna.

REST sučelje aplikacije za kontrolu ulaza pokreće se s naredbom

```
composer-rest-server -c admin@pii-szg-network -n never -w
```

true ako se želi testirati administratorska identifikacijska kartica odnosno

```
composer-rest-server -c fer@pii-szg-network -n never -w true
```

-p 2999 ako se želi testirati drugačije autentifikacijske postavke. CRUD operacije su

dozvoljene u skladu s autorizacijskim pravilima, pa je tako sa FER-ovom identifikacijskom

karticom je moguće čitati sve podatke sa svih resursa, ali ne i obavljati druge operacije

osim kreiranja `CheckAccessFER` transakcije. Ako se koristi administratorska

identifikacijska kartica, mogu se obavljati sve operacije nad svim resursima uz napomenu

da operacija brisanja (eng. *Delete*) briše samo iz "Trenutnog stanja" pošto se iz

blockchaina nikada ništa ne može obrisati. Podatci koji su obrisani se i dalje vide u resursu

`/historian` odnosno vidi se kada su kreirani i kada su obrisani ili modificirani.

- **WEB sučelje aplikacije za kontrolu ulaza**

Frontend aplikacija koja se

spaja na sučelje generira se u

Angularu preko alata Yeoman.

Generator koda generira pogled i

kreiranje novih `Participants` te

`Asset`. Logiku za transakcije se treba

dodatno isprogramirati. Aplikacija se

generira na temelju REST sučelja te se

mogu vidjeti svi trenutni resursi i

kreirati novi odnosno modificirati stari

ako je u REST sučelju korištena

administratorska identifikacijska

kartica.

pii-szg-network@0.3.4.bna
angular-app

Assets Participants Transactions
angular-app-wons - HOME

angular-app-wons

jmbag:
0036111222
universityKey:
0036
tid:
E200341201301700026A
Submit

U daljnjem primjeru je pokazana

dodatna logika za slanje transakcija

Slika 3.5: Izgled WEB sučelja za unos podataka i slanje transakcija na *chaincode*

preko REST sučelja. Asinkrone GET i POST funkcije se koriste prema REST sučelju, GET za traženje resursa pomoću upita, a POST za slanje transakcija. Glavni izbornik home se koristi za primanje ulaznih podataka te sadrži logiku za slanje transakcija. Primjer koda dohvaćanja te pozivanja lokalne funkcije koja šalje transakciju na REST sučelje:

```
let jmbag = document.getElementById('jmbag').value;
let universityKey = document.getElementById('universityKey').value;
let tid = document.getElementById('tid').value;
...
let resultMemberArray = await fetchAsync(conn+querySelectMember+jmbag);
let resultMember = resultMemberArray[0];
...
let resultUniversityComponentArray = await
fetchAsync(conn+querySelectUniversityComponent+universityKey);
let resultUniversityComponent = resultUniversityComponentArray[0];
...
let methodCall =
"CheckAccess"+resultUniversityComponent.universityName;
await
callTrasaction(methodCall, resultMember.jmbag, resultUniversityComponent.u
niversityKey, tid);
```

Primjer lokalne funkcije koja sprema niza znakova u JSON format te šalje transakciju preko asinkrone POST operacije:

```
async function callTrasaction(_method, _jmbag, _universityKey, _tid) {
    try {
        let trans = new Object();
        trans.$class = prefixApp+"."+_method;
        trans.member = relationshipMember+_jmbag;
        ...
        let stringTrans = JSON.stringify(trans);
        resultPostArray = await postAsync(conn+api+_method, stringTrans);
        console.log(resultPostArray);
    }
}
```



```
    } catch (error) {  
    ...  
}
```

3.1.5. Simulacija čvora za kontrolu ulaza na Hyperledgeru

Potrebne biblioteke za interakciju Node.JS aplikacije sa Hyperledger Fabric Composer mrežom su `composer-admin`, `composer-client` te `composer-common`



















- **Kreiranje resursa potrebnih za simulaciju**

Kako bi se simulacija aplikacije za kontrolu ulaza pokrenula potrebno je kreirati početne korisnike i imovinu te identifikacijske kartice za autentifikaciju sustava. Skripta `creationScript.js` koristi administratorsku identifikacijsku karticu te s njom kreira sve potrebno za rad sustava. Kreiraju se tri komponente sveučilišta FER, FFZG i FSB svaka sa svojim postavkama, vremenom rada i identifikacijskom oznakom. Nakon toga kreiraju se tri systemska administratora za svako sveučilište sa specifičnim unaprijed dodijeljenim JMBAG-ovima i TID-ovima te im se kreiraju i identifikacijske kartice za interakciju s Hyperledger Fabric *blockchainom*. Naposljetku se kreira 40 korisnika Sveučilišta u Zagrebu s proizvoljnim JMBAG-ovima i TID-ovima koji ovisno o korisniku pripadaju jednom o tri fakulteta te jednom od vrste korisnika. Na temelju tih specifikacija će se u poslovnoj logici *chaincode* provjeravati smiju li osobe ući u fakultet ili ne.

Nakon završetka rada kreacijske skripte moguće je provjeriti njen rad preko REST sučelja ili preko Composer-Playground IDE-a koji se spaja na Hyperledger Fabric mrežu.

- **Node.JS aplikacija simulacije čvora za kontrolu ulaza**

Simulaciju aplikacije za kontrolu ulaza na uređajima se pokreće naredbom `node index.js serverStart FER/FFZG/FSB` Parametar nakon naredbe `node` pokreće skriptu `index.js`, `serverStart` parametar pokreće izvezenu (eng. *export*) funkciju `serverStart`. Treći parametar predajemo ovisno o tome koji uređaj za kontrolu ulaza želimo simulirati te su trenutno podržane simulacije FER-a, FFZG-a i FSB-a. Moguće je pokrenuti sve tri simulacije od jednom te simulirati cijeli sustav. Primjer definiranih parametara ako želimo simulirati FSB uređaj za kontrolu ulaza:

angular-app-wons - HOME								
Member								
jmbag	firstName	lastName	universityComponent	transactionAuthorized	transactionRevoke	memberType	tid	Actions
0035011199	sts	cbc	resource.org.szg.UniversityComponent#0...	0036 FER Tue Jun 05 2018 14:38:54 GM...		Staff	E200341201301700026A6B67	 
0035111188	gsg	ttmd	resource.org.szg.UniversityComponent#0...	0036 FER Tue Jun 05 2018 14:41:54 GM...		Profesor	E200341201301700026A6B49	 
0035111189	meml	mafkmc	resource.org.szg.UniversityComponent#0...	0036 FER Tue Jun 05 2018 14:40:24 GM...		Staff	E200341201301700026A6B47	 
0035111191	mekl	afkic	resource.org.szg.UniversityComponent#0...	0036 FER Tue Jun 05 2018 14:35:09 GM...		Student	E200341201301700026A6B38	 
0035111192	ana	nic	resource.org.szg.UniversityComponent#0...			Student	E200341201301700026A6B41	 
0035111193	gskg	tttk	resource.org.szg.UniversityComponent#0...	0036 FER Tue Jun 05 2018 14:40:39 GM...		Student	E200341201301700026A6B43	 
0035111194	mekl	mafikic	resource.org.szg.UniversityComponent#0...	0036 FER Tue Jun 05 2018 14:39:39 GM...		Student	E200341201301700026A6B44	 
0035111195	sts	ckc	resource.org.szg.UniversityComponent#0...			Student	E200341201301700026A6B42	 
0035111196	anha	aninc	resource.org.szg.UniversityComponent#0...	0036 FER Tue Jun 05 2018 14:41:39 GM...		Student	E200341201301700026A6B45	 

Slika 3.6: Izgled WEB sučelja za pregled korisnika. Moguća je i modifikacija u slučaju administratorskih privilegija

```
case "FSB": port = 3002 //FER 3001, FSB 3002, FFZG 3003
    _universityKey = "0035";
    method = "CheckAccessFSB";
    cardName = "fsb@pii-szg-network";
    businessNetworkDefinition = await
bizNetworkConnection.connect(cardName);

    break;
```

Skripta kreira poslužitelj na *portu* odabranom ovisno o konfiguraciji te se pokreće funkcija `setInterval()` koja određenu funkciju ponavlja svakih n sekundi. Sve funkcije koje se pozivaju od tog trenutka su asinkrone te se čeka na njihovo izvršavanje prije nego što se krene izvršavati sljedeći red koda. Kako bi se simulirao sustav, a ujedno simulacija bila nepredvidljiva, aplikacija uzima slučajnu liniju datoteke `inputs.csv` u kojoj su zapisani svi trenutni korisnici u *blockchainu*.

Nakon što su potrebni podatci izvučeni iz datoteke te je JMBAG-ov niz byte znakova iz datoteke pretvoren u JMBAG, transakcija se kreira i šalje na kanal. Primjer koda koji kreira transakcije te ih šalje transakcije na kanal na izvršavanje:

```
//kreiranje nove transakcije preko klase Factory

let factory = await businessNetworkDefinition.getFactory();

let transaction = factory.newTransaction('org.szg', method);

transaction.universityComponent = factory.newRelationship('org.szg',
'UniversityComponent', _universityKey);

transaction.member = factory.newRelationship('org.szg', 'Member',
_jmbag);

transaction.tid = _tid;

//slanje transakcije na chaincode

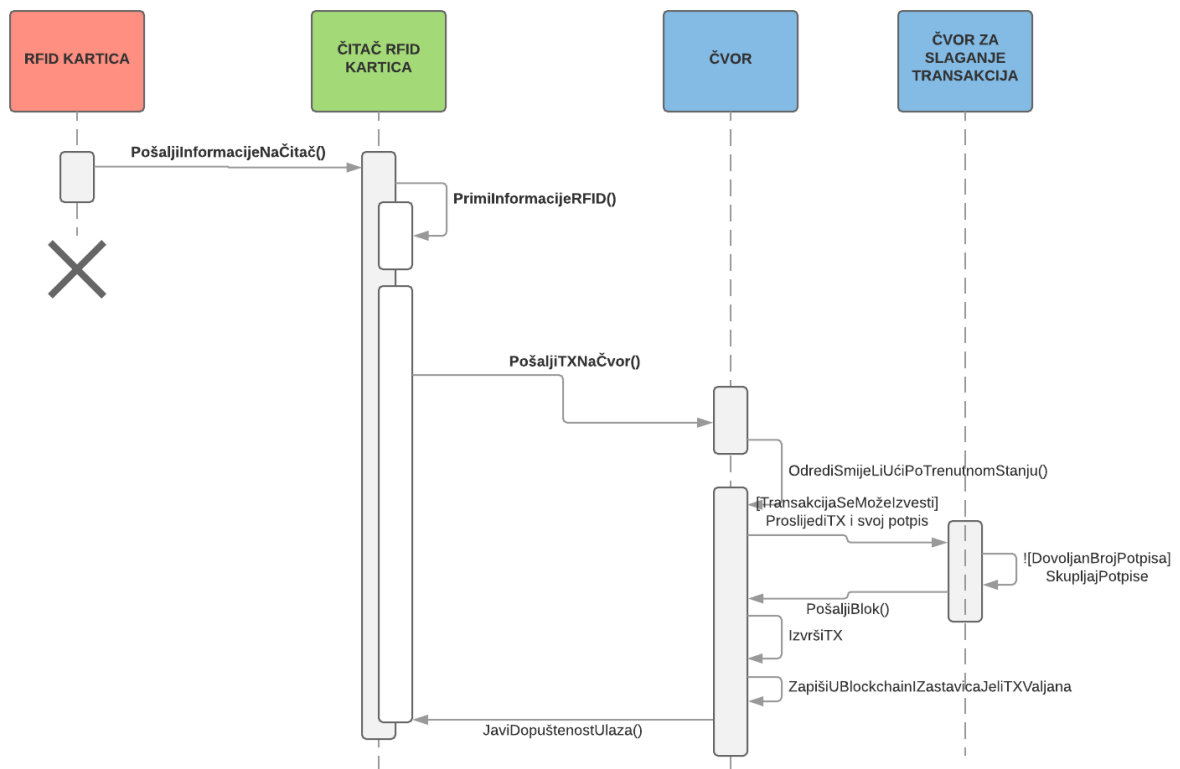
await bizNetworkConnection.submitTransaction(transaction);
```

The screenshot shows a terminal window with three tabs, each representing a different node in the Hyperledger Fabric network. The terminal output shows the following sequence of events:

- Node 1 (Left):** The node is listening on port 3001. It receives a transaction from inputs.csv to chaincode E200341201301700026A6B33,303033363131313133330000. The access is true for 0036111153 at 0036 in Tue May 22 2018 13:01:12 GMT+0200 (CEST).
- Node 2 (Middle):** The node is listening on port 3002. It receives a transaction from inputs.csv to chaincode E200341201301700026A6B31,303033363131313133330000. The access is true for 0036111133 at 0036 in Tue May 22 2018 13:01:27 GMT+0200 (CEST).
- Node 3 (Right):** The node is listening on port 3003. It receives a transaction from inputs.csv to chaincode E200341201301700026A6B69,303033363131313139380000. The access is true for 0035111198 at 0036 in Tue May 22 2018 13:01:42 GMT+0200 (CEST).

The terminal also shows the submission of the transaction to the chaincode and the resulting access status for each node.

Slika 3.7: Prikaz simulacije triju uređaja za kontrolu ulaza na različitim fakultetima na platformi Hyperledger Fabric



Slika 3.8: Sekvencijski dijagram aplikacije za kontrolu ulaza na Fabric Composeru

3.2. Aplikacija za kontrolu ulaza na platformi Ethereum

Aplikacija na Ethereum platformi sadrži *hashirane* JMBAG i *UniversityKey* oznake zbog toga što svi vide sve što je zapisano unutar Ethereum *blockchaina*, a princip na Ethereumu jest da se nikome ne vjeruje. Zbog toga nakon što aplikacija na uređajima očita vrijednosti iz RFID-a *hashira* JMBAG podatak. Koristi se SHA1 algoritam koji daje idealni 20B heksadecimalni niz znakova koji se može spremiti uz mali utrošak *Gasa* kao adresa. Aplikacija odnosno pametni ugovor na Ethereum platformi vraća rezultat i ovisno o rezultatu aplikacija propušta korisnika ili ne.

3.2.1. Truffle programski alat

Alat koji pomaže u razvoju Ethereum je Truffle. Truffle je programski okvir otvorenog koda namijenjen lakšoj integraciji aplikacija Ethereum *blockchainu*. Sadrži kompajler koda, poveziavač biblioteka i postavljanje na *blockchain* na bilo kakvu vrstu mreže. Pomoću menadžera biblioteka vrlo je lako uključiti potrebne biblioteke koje bi trebale za rad. Konzola za interakciju s pametnim ugovorima i *blockchainom* je prisutna, pa se može brzo i interaktivno testirati određene funkcionalnosti, a testiranje pametnih ugovora postoji i na razini cijelog sustava. Ugrađeni simulator *blockchaina* Ganache omogućava lako postavljanje *blockchaina* te brzu interakciju i na slabim računalima. [21]



Slika 3.9: Logo Truffle alata

Truffle podržava generiranje *webpack* stranice koja je povezana s *blockchain* primjerom kako bi ubrzali izradu WEB aplikacija temeljenih na *blockchainu*. U mapi *app* se nalaze datoteke web stranice. Migracije su svojstvo u Trufflu koje postavlja određeni račun kao vlasnika pametnog ugovora te samo taj račun može nadograditi pametni ugovor na novu verziju. Nadograđivanje se obavlja tako da se kreira nova instanca pametnog ugovora pošto u *blockchainu* jednom zapisani podatci se ne mogu modificirati. Migracije služe više kao sigurnosno svojstvo.

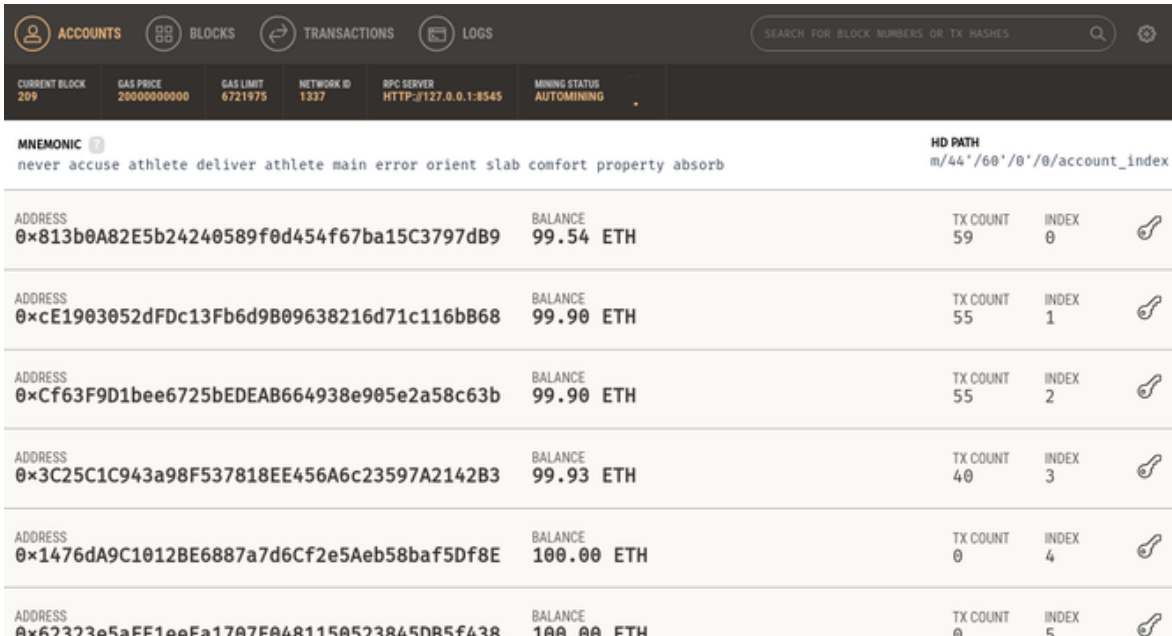
Truffle.js datoteka sadrži podatke o mreži na koju se spaja odnosno adresu i *port* početnog (eng. *bootnode*) čvora te identifikacijsku oznaku mreže. Testiranje pametnog

ugovora se odvija preko `assert` naredbi te su svi slučajevi nezavisni i instanca pametnog ugovora se ponovno kreira.

Poslovna logika aplikacije su pametni ugovori koji se nalaze u mapi `contracts`. Nakon kompajliranja i povezivanja s bibliotekama i drugim pametnim ugovorima, u mapu `builds` se dodaju artefakti (eng. *artefacts*) za svaki pametni ugovor u JSON formatu. *Artefacts* pojedinih pametnih ugovora sadrži ABI ugovora te poveznicu na *blockchain* mrežu na kojoj se treba koristiti. ABI pametnog ugovora služi EVMu za čitanje podataka i prevođenje u strojni jezik računala na kojem se pokreće.

3.2.2. Konfiguracija mreže i čvorova

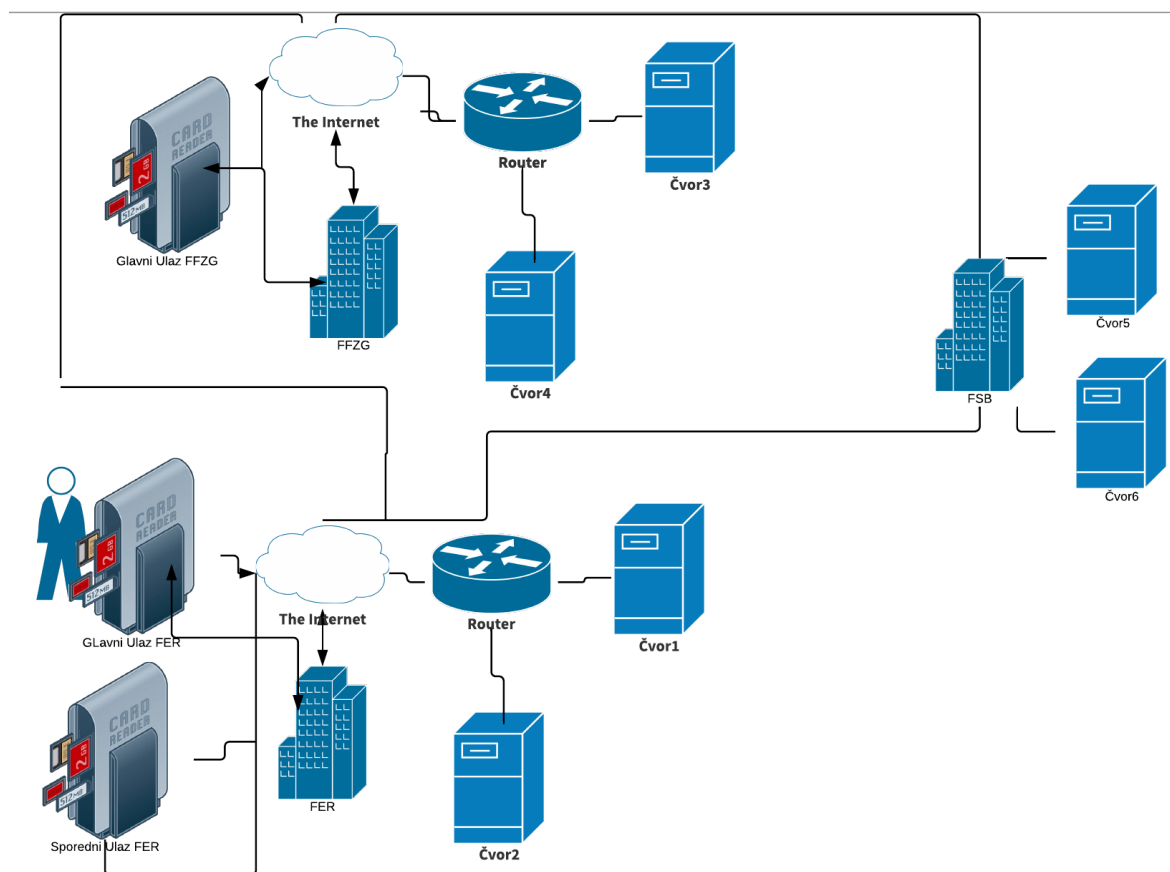
Pokreće se simulator Ethash protokola za kreiranje blokova Ganache. Ganache je nekadašnji simulator testRPC kojeg je Truffle preuzeo kako bi dodao dodane opcije i prilagodio ga Truffle platformi. Simulator radi identično kao i protokol Geth korišten na pravoj mreži osim što ne računa komplicirane *hasheve* s točno određenim brojem nula već simulira računanje i kreira odmah novi blok s novom transakcijom. Moguće je jednostavno kreirati nove račune s određenim brojem *tokena* i postaviti željene opcije za *Gas* tako da simulira pravu mrežu kao i postaviti nerealne opcije za *Gas* zbog testiranja. Ganache podržava CLI te GUI, a u GUI-u je jednostavnije izmijeniti određene postavke kada je simulator već pokrenut. Ganache ima opciju spremanja blockchaina kao i pravi protokoli rudarenja, međutim to usporava simuliranje.



The screenshot shows the Ganache GUI interface. At the top, there are tabs for ACCOUNTS, BLOCKS, TRANSACTIONS, and LOGS. Below the tabs, a search bar is present. A status bar displays various metrics: CURRENT BLOCK (209), GAS PRICE (20000000000), GAS LIMIT (6721975), NETWORK ID (1337), RPC SERVER (HTTP://127.0.0.1:8545), and MINING STATUS (AUTOMINING). Below this, the MNEMONIC phrase is shown: "never accuse athlete deliver athlete main error orient slab comfort property absorb". The HD PATH is also visible: "m/44'/60'/0'/0/account_index". The main section displays a list of accounts with their addresses, balances, transaction counts, and indices.

ADDRESS	BALANCE	TX COUNT	INDEX
0x813b0A82E5b24240589f0d454f67ba15C3797dB9	99.54 ETH	59	0
0xcE1903052dFDc13Fb6d9B09638216d71c116bB68	99.90 ETH	55	1
0xCf63F9D1bee6725bEDEAB664938e905e2a58c63b	99.90 ETH	55	2
0x3C25C1C943a98F537818EE456A6c23597A2142B3	99.93 ETH	40	3
0x1476dA9C1012BE6887a7d6Cf2e5Aeb58baf5Df8E	100.00 ETH	0	4
0x62323e5aFE1eeEa1707E0481150523845DB5f438	100.00 ETH	0	5

Slika 3.10: Sučelje Ganache simulatora s prikazom na trenutne aktivne račune te brojem blokova i transakcija koje su napravili tijekom slanja transakcija za kontrolu ulaza



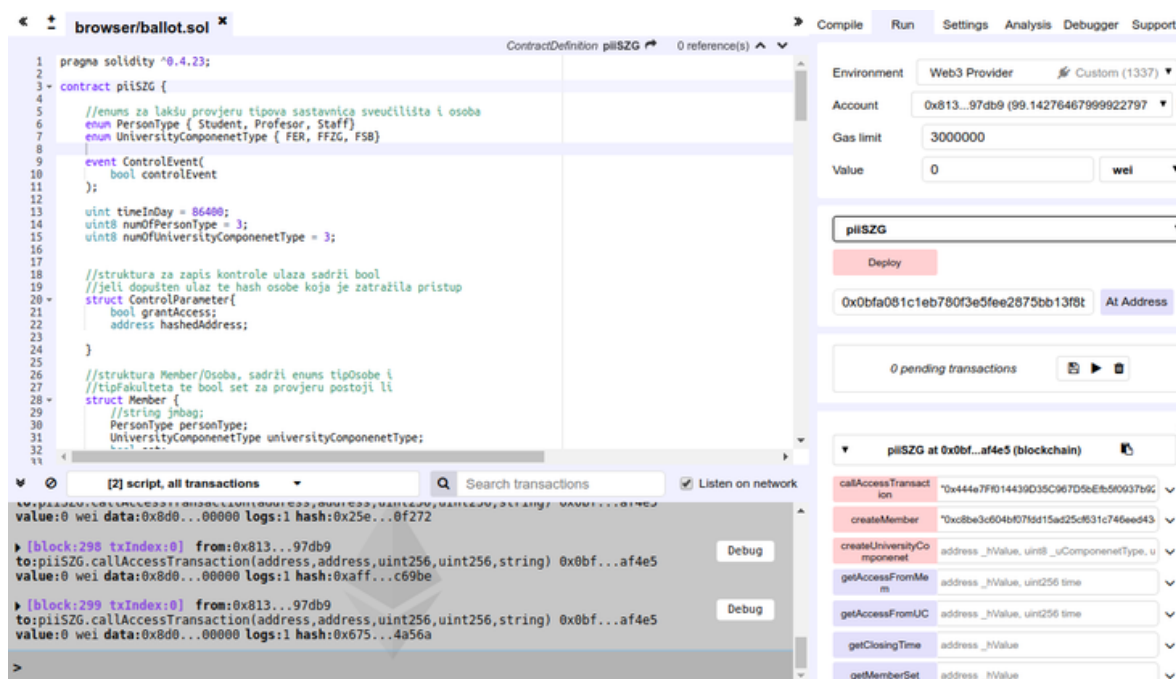
Slika 3.11: Mogući dijagram mreže aplikacije za kontrolu ulaza na Ethereumu

Ako se želi testirati protokol koji se koristi na glavnoj mreži, mora se pokrenuti Geth. Geth protokolom se prikazuje stvarna opterećenost mreže i brzina transakcija na privatnim čvorovima. Geth protokolom se može pokrenuti više instanci čvorova za rudarenje te tako isprobati sustav s više čvorova koji se koristi u pravoj mreži. Kako bi se postavio sustav s više čvorova, jedan od čvorova mora biti početni čvor zadužen za taj dio mreže ujedno se takav čvor zove *Bootnode* čvor. Svim ostalim čvorovima na tom dijelu mreže se u postavke dodaje *bootnode* ključ preko kojeg se ostali čvorovi spajaju *enode* protokolom. Svaki čvor mora imati zasebni prostor na disku kako bi spremao svoj *blockchain*. Čvorovi se sinkroniziraju i šalju transakcije jedno drugome preko *enode* protokola, ali mogu se postaviti tako da ne sadrže cijeli *blockchain* već dio *blockchaina* te se može postaviti da čvorovi ne potvrđuju ispravnost blokova. Obično se čvorovi koji sadrže samo novčanik na javnoj mreži postavljaju na taj način. Blokovi koji rudare moraju sadržavati cijeli *blockchain*.

3.2.3. Logika aplikacije za kontrolu ulaza

- **Korišteni alati za pisanje pametnog ugovora**

Poslovna logika aplikacija na Ethereum platformi nalazi se u pametnom ugovoru. Najzastupljeniji IDE za Programski jezik Solidity u kojem se programiraju pametni ugovori je Remix IDE kreiran od strane programera Ethereum platforme. Remix IDE se može spojiti na *blockchain* čvor te slušati mrežu odnosno vidjeti transakcije koje se obavljaju na mreži, a može koristiti svoj JavaScript simulator. Moguće je koristiti alat za otklanjanje neispravnosti (eng. *debug*). Remix podržava izračun potrošnje *Gasa* za cijeli ugovor te za svaku funkciju posebno olakšavajući pregled nad performansama pametnog ugovora, također nudi pomoć pri pisanju funkcija pomoću statičke analize koda tako da troše manje *Gasa* ako je to moguće.



Slika 3.12: Remix IDE s otvorenim prozorima koda pametnog ugovora piiSZG, donji prozor u kojem Remix sluša transakcije na mreži te desni prozor postavki i ručnog testiranja funkcija pametnog ugovora

- **PiiSZG pametni ugovor**

Pametni ugovor piiSZG se može podijeliti u nekoliko cjelina. Modeliranje podataka, interne funkcije i *getter* funkcije te logika transakcija.

- **Modeliranje podataka**

Struktura podataka u Solidity programskoj jeziku je vrlo slična strukturi podataka u programskom jeziku C i sličnima. U strukturi se mogu zapisati više vrsta jednostavnih tipova podataka ili nekih drugih struktura te sve objediniti u cjelinu. `piiSZG` koristi dvije važne strukture prva kojom se definiraju korisnici te drugom strukturom kojom se definiraju sveučilišne komponente. Implementirana je pomoćna struktura `ControlParameter` u koju se zapisuje *hash* korisnika odnosno komponente sveučilišta te logička vrijednost je li zatraženi ulaz dopušten ili ne. Struktura `Member` u koju se spremaju korisnici pojedinog fakulteta sadrži enumeracije tipa osobe i tipa fakulteta kojem pripada, TID te mapiranje (eng. *mapping*) pojedinog trenutka traženja ulaza na spomenutu pomoćnu strukturu `ControlParameter`.

Enumeracija označava da podatci smiju biti samo iz skupa određenih elemenata, za `tipFakulteta` tj. tip komponente sveučilišta trenutno su implementirani FER, FSB i FFZG, a za `tipOsobe` `Student`, `Profesor` i `Staff`.

Događaji su slični događajima u Hyperledgeru i JavaScript jeziku te izgledaju kao strukture podataka, ako se emitiraju, emitiraju transakciju koja ih poziva te vrijednost koje zabilježimo u strukturu događaja kojeg zovemo. Primjer događaja i pozivanje unutar neke funkcije:

```
event ControlEvent(  
    bool controlEvent  
);  
...  
emit ControlEvent(true);  
...
```

Mapiranje je slično rječnicima u objektnim jezicima, za svaki ključ postoji neka vrijednost. U ovom slučaju za svaki ključ: vrijeme traženja ulaska na fakultet, postoji vrijednost strukture `ControlParameter` u kojoj piše *hash* fakulteta na se tražio ulaz te je li ulaz bio dopušten.

Struktura `UniversityComponenet` sadrži tip komponente sveučilišta, Ethereum adresu, vrijeme otvaranja i zatvaranja dotične komponente sveučilišta u sekundama u danu `[0,86400]` te mapiranje s ključem vremena traženja ulaska, a vrijednošću

ControlParameter u kojemu piše *hash* korisnika koja je tražila ulaz te je li ulaz bio dopušten.

Na posljétku postoje mapiranja s ključem *hash* korisnika te vrijednošću strukturom Member i mapiranje s ključem *hash* sveučilišne komponente te vrijednošću struktura UnivesityComponent. U tim mapiranjima su sadržani svi korisnici odnosno sveučilišne komponente.

Primjer strukture Member te mapiranja koji koristi tu strukturu:

```
struct Member {
    PersonType personType;
    UniversityComponenetType universityComponenetType;
    bool set;
    string tid;
    mapping(uint => ControlParameter) accessM;
}
mapping(address => Member) public members;
```

- **Getter funkcije**

Solidity nije implementirao vraćanje cijele strukture podataka već se podatci moraju dohvaćati posebno, zbog toga se kreiraju funkcije zvane *Getteri*. *Getter* funkcije u piisZG pametnom ugovoru dohvaćaju vrijednosti iz mapiranih struktura. Svim *getter* funkcijama prvi parametar je jednak te označava *hashiranu* vrijednost JMBAG-a odnosno IdFakuteta. U slučaju da se traži struktura ControlParametar, kao parametar predaje se dodatno i trenutak vremena u sekundama u kojem se traži struktura ControlParametar.

Primjer *getter* funkcije za sveučilišnu komponentu kojom se vraća vrijeme zatvaranja sveučilišne komponente u sekundama te *getter* funkcija za vraćanje strukture ControlParametar za određeni trenutak:

```
function getClosingTime(address _hValue) public view returns (uint32){
    require(universityComponenets[_hValue].set == true);
    return universityComponenets[_hValue].closingTime;
}
function getAccessFromUC(address _hValue, uint time) public view returns
(bool, address){
```

```

        require(universityComponenets[_hValue].set == true);
        return
        (universityComponenets[_hValue].access[time].grantAccess,
        universityComponenets[_hValue].access[time].hashedAddress);
    }

```

- **Interne funkcije**

Interne funkcije su pomoćne funkcije koje služe za dodanu provjeru uvjeta koje ne možemo provjeriti s `require()` funkcijom. Interne funkcije se označavaju modifikatorom *internal pure* što znači da ne koštaju ništa *Gasa* za njihovo izvršavanje pošto ne mijenjaju stanje u *blockchainu*. Primjeri takvih funkcija u `piiSZG` pametnom ugovoru su provjera je li dostavljeni niz znakova za `TID` u odgovarajućem formatu:

```

function testStr24(string str) internal pure returns (bool){
    bytes memory b = bytes(str);
    if(b.length != 24) return false;
    for(uint i; i<24; i++){
        if(!(b[i] >= 48 && b[i] <= 57) || (b[i] >= 65 && b[i]
        <= 90) || (b[i] >= 97 && b[i] <= 122)))
            return false;
    }
    return true;
}

```

Drugi primjer u kojem se koriste interne funkcije provjerava jesu li ispunjeni određeni uvjeti za dopuštanje ulaska osobi koja to traži. Ova funkcija provjerava je li osoba s istog fakulteta na kojem traži pristup te ako je, je li fakultet otvoren odnosno ako nije otvoren ima li ta osoba prava svejedno ući zbog privilegiranog statusa za ulaz :

```

function checkSameUniversityOpenedOrPrivileged(address
addressHashMember, address addressHashUniversityComponenet, uint ttime)
internal view returns(bool){
    UniversityComponenet memory _universityComponenet =
    universityComponenets[addressHashUniversityComponenet];
}

```

```

        Member memory _member = members[addressHashMember];

        if(_universityCompenenet.universityCompenenetType ==
        _member.universityCompenenetType){
            if(ttime >= _universityCompenenet.openingTime && ttime
            <= _universityCompenenet.closingTime)
                return true;
            else if(uint(_member.personType) == 1 ||
            uint(_member.personType) == 2)
                return true;
        }
        return false;
    }
}

```

- **Logika transakcija**

Transakcije u *piiSZG* smiju pozivati samo sveučilišne komponente koje su zadužene za to sveučilište na kojem se traži ulaz. U slučaju da sveučilišna komponenta traži odobravanje ulaza za fakultet kojem nije nadležna, *require()* funkcija na početku transakcije će otkriti te potrošiti sav poslani *gas* i proglasiti transakciju ne važećom. Iznimka je transakcija za kreiranje sveučilišne komponente koja kao ključ mapiranja stavlja *hashirani* *IdSveučilišneKomponente* te unutar vrijednosti strukture stavlja adresu Ethereum računa s kojeg je poslana transakcija tako da se svaka sljedeća transakcija mora odviti preko tog Ethereum računa.

Primjer funkcije koja kreira novu sveučilišnu komponentu preko mapiranja ključa *hash* *IdSveučilišta* te uvrštavanja vrijednosti:

```

function createUniversityCompenenet(address _hValue,
UniversityCompenenetType _uCompenenetType,uint32 _openingTime, uint32
_closingTime) public returns(bool){
    require(uint(_uCompenenetType) <
    numOfUniversityCompenenetType && _openingTime <= timeInDay
    && _closingTime <= timeInDay);
    require(universityCompenenets[_hValue].set != true);
}

```

```

        universityComponenets[_hValue] =
        UniversityComponenet(_uComponenetType, _openingTime,
        _closingTime,msg.sender,true);
        emit ControlEvent(true);
        return true;
    }

```

Primjer provjere je li dolazna transakcija od iste adrese koja je zapisana unutar strukture sveučilišne komponente kao vlasnika sveučilišne komponente.

```

require(msg.sender ==
universityComponenets[addressHashUniversityComponenet]
.universityComponenetAccount);

```

Transakcija provjere ulaska sadrži dodatne provjere među kojima je najvažnija provjera je li TID koji je poslan transakciji jednak zapisanom TID-u u strukturi Member koji traži ulaz.

Provjera valjanosti TID-a: preko *hash* funkcije kako bi se ubrzalo izvršavanje u EVM-u

```

if(keccak256(tid) != keccak256(members[addressHashMember].tid)){
    emit ControlEvent(false);
    return false;
}

```

Transakcija provjere ulaska nakon što je provjerila valjanost svih ulaznih vrijednosti, zove interne funkcije koje provjeravaju je li korisnik smije ući ili ne. Nakon što funkcija transakcije provjere dobije povratnu vrijednost da smije ući, izlazi iz provjere te zapisuje vrijednosti trenutnog vremena i potvrdnog stanja o ulazu, odnosno negativnog ako su sve interne funkcije provjere vratile negativnu vrijednost, u ControlParametar tog sveučilišta i korisnika koji je tražio ulaz. Preostaje emitirati povratnu vrijednost tako da aplikacija koja se pretplatila na događaje dobije rezultat transakcije.

```

_access = checkSameUniversityOpenedOrPrivileged
(addressHashMember,addressHashUniversityComponenet,ttime);
if(_access == true){
    universityComponenets[addressHashUniversityComponenet]
.access[transactionDateTime] =
ControlParameter(_access,addressHashMember);
}

```

```

        members[addressHashMember].accessM[transactionDateTime] =
ControlParameter(_access, addressHashUniversityComponenet);
        emit ControlEvent(true);
        return true;
    }...

```

3.2.4. Frontend Ethereum aplikacije za kontrolu ulaza

- **Web3 biblioteka za rad s Ethereum platformom**

Kako bi se Web3 biblioteka uključila u aplikaciju te povezala na čvor dodaju se linije:

```

var Web3 = require('web3');
var web3 = new Web3('http://localhost:8545');

```

Nakon toga se pametni ugovor mora instancirati. Pametni ugovor se instancira tako što se učitaju njegovi kompajlirani artefakti te tada instancirani ugovor može pozivati funkcije pametnog ugovora.

```

import piiszg_artifacts from
'../../build/contracts/piiSZG.json'
var piisZG =
contract(piiszg_artifacts);

```

- **WEB sučelje aplikacije za kontrolu ulaza**

Frontend aplikacija koristi Web3 biblioteku za interakciju s blockchainom. Aplikacija nakon spajanja na odabrani čvor, uzme prvi Ethereum račun na mreži te kako je aplikacija u testnom okruženju, može otključati račun na neograničeno vrijeme. Web sučelje nudi kreiranje nove sveučilišne komponente, kreiranje novog korisnika mreže te slanje transakcije provjere ulaza. U svim slučajevima se preko instanciranog piisZG ugovora zove funkcija unutar piisZG pametnog ugovora.

pii-szg-network Access control for University

piiSZGNetwork@0.2.0 Ethereum Truffle
Loading

Create University Component

universityKey:

 universityComponenetType:
 FER ▼
 openingTime:

 closingTime:

Create Member

jmbag:

 tid:

 personType:
 Student ▼
 universityComponenetType:
 FER ▼

Check Access

jmbag:

 universityKey:

 tid:

Slika 3.13: Izgled WEB sučelja Ethereum aplikacije za kontrolu ulaza

Primjer koda koji uzima vrijednosti unosa, *hashira* JMBAG i *UniversityKey* te zove funkciju pametnog ugovora *callAccessTransaction* s pripadajućim parametrima. Funkcija instance pametnog ugovora se poziva kao lanac obećanja (eng. *Promise chain*) u kojem se transakcije koje su unutar nekog koda ili u *.then()* dijelu izvršavaju tek kada se vrati ispunjeni *Promise* s nekim vrijednostima.

```
var jmbag = document.getElementById('jmbagCheck').value;
var universityKey = document.getElementById('uKeyCheck').value;
var tidCheck = document.getElementById('tidCheck').value;
...
var jmbagHash = Sha1(jmbag);
var jmbagHashed = "0x"+jmbagHash;
var universityKeyHash = Sha1(universityKey);
var universityKeyHashed = "0x"+universityKeyHash;
...
piiSZG.deployed().then(function(instance) {
    instance.callAccessTransaction(jmbagHashed,
    universityKeyHashed, ttime, n, tidCheck, {from: account})
    //on fullfiled promise refresh status about transaction
    .then(function(access) {
        self.setStatus("Transaction complete!");
    });
});
...
```

3.2.5. Ethereum aplikacija kontrole ulaza na uređajima za kontrolu ulaza

Dodatne biblioteke potrebne za Node.JS aplikaciju za Ethereum su Web3 i Sha1

- **Kreiranje resursa potrebnih za simulaciju**

Simulacija aplikacije za kontrolu ulaza treba imati početne korisnike i komponente sveučilišta kako bi se pravilno pokrenula. Skripta *creationScript.js* nakon instanciranja *piiSZG* ugovora, uzme račune s Ethereum mreže. Kreiraju se tri komponente sveučilišta FER, FFZG i FSB svaka sa svojim postavkama, vremenom rada i identifikacijskom oznakom. Nakon toga kreće kreiranje 40 korisnika Sveučilišta u Zagrebu s proizvoljnim JMBAG-ovima i TID-ovima koji ovisno o korisniku pripadaju jednom od

tri fakulteta te jednom od vrste korisnika. Ovisno o komponenti sveučilišta kojem pripadaju, račun te komponente sveučilišta zove `createMember` transakciju.

Nakon završetka rada kreacijske skripte moguće je provjeriti njen uspjeh ili preko konzole u kojemu se ispišu rezultati svake transakcije ili preko Ganache terminala/GUI-a u kojemu se mogu vidjeti svi blokovi te njihove vrijednosti. Također je moguće transakcije pratiti preko Remix IDE-a ako se povežemo na lokalnu mrežu te ju osluškujemo.

- **Aplikacija simulacije uređaja za kontrolu ulaza**

Simulaciju aplikacije na Ethereum platformi za kontrolu ulaza na uređajima se pokreće naredbom `node index.js serverStart FER/FFZG/FSB`. Parametar nakon naredbe `node` pokreće skriptu `index.js`, `serverStart` parametar pokreće izvezenu (eng. *export*) funkciju `serverStart`. Treći parametar predajemo ovisno o tome koji uređaj za kontrolu ulaza želimo simulirati te su trenutno podržane simulacije FER-a, FFZG-a i FSB-a. Moguće je pokrenuti sve tri simulacije od jednom te simulirati cijeli sustav. Primjer definiranih parametara ako želimo simulirati FFZG uređaj za kontrolu ulaza:

```
case "FFZG":
    port = 3006 //FER 3005, FFZG 3006, FSB 3007
    _universityKey = "1111";
    universityKeyHash = Sha1(_universityKey);
    universityKeyHashed = "0x" + universityKeyHash;
    account = accounts[1];
    break;
```

Kreira se poslužitelj na *portu* odabranom ovisno o konfiguraciji te se pokreće funkcija `setInterval()` koja određenu funkciju ponavlja svakih *n* sekundi. Sve funkcije koje se pozivaju od tog trenutka su asinkrone te se mora kreirati *Promise chain* kako bi se funkcije pravilno izvršavale. *Promise chain* funkcionira na principu da tek kada se nešto obavi i dobije nazad razriješeno obećanje (eng. *Resolved promise*) se kreće dalje u `.then()` dio. Kako bi se simulirao sustav, a ujedno simulacija bila nepredvidljiva, aplikacija uzima slučajnu liniju datoteke `inputs.csv` u kojoj su zapisani svi trenutni korisnici u *blockchainu*.

Nakon što su potrebni podatci izvučeni iz datoteke te je JMBAG-ov niz byte znakova iz datoteke pretvoren u JMBAG, hashira se JMBAG te se *hashira* oznaka komponente

sveučilišta koja šalje transakciju. Transakcija se tada šalje na *blockchain*. Primjer koda koji šalje transakcije na čvor kako bi se izvršio te prima nazad vrijednost i u ovisnosti o toj vrijednosti ispisuje rezultat na konzolu:

```
piiSZG.methods.callAccessTransaction(jmbagHashed, universityKeyHashed,
ttime, n, _tid).send({
    from: account,
    gas: numGas
}).then(function (retValue) {
if(JSON.stringify(retValue.events.ControlEvent.returnValues["0"]) ==
"null")
console.log("Access false for hashed jmbag " + jmbagHashed + " at hashed
university " + universityKeyHashed + " in " + Date(n).toLocaleString())
else
console.log("Access " + JSON.stringify
(retValue.events.ControlEvent.returnValues["0"]) + " for hashed jmbag "
+ jmbagHashed + " at hashed university " + universityKeyHashed + " in "
+ Date(n).toLocaleString())
})
```

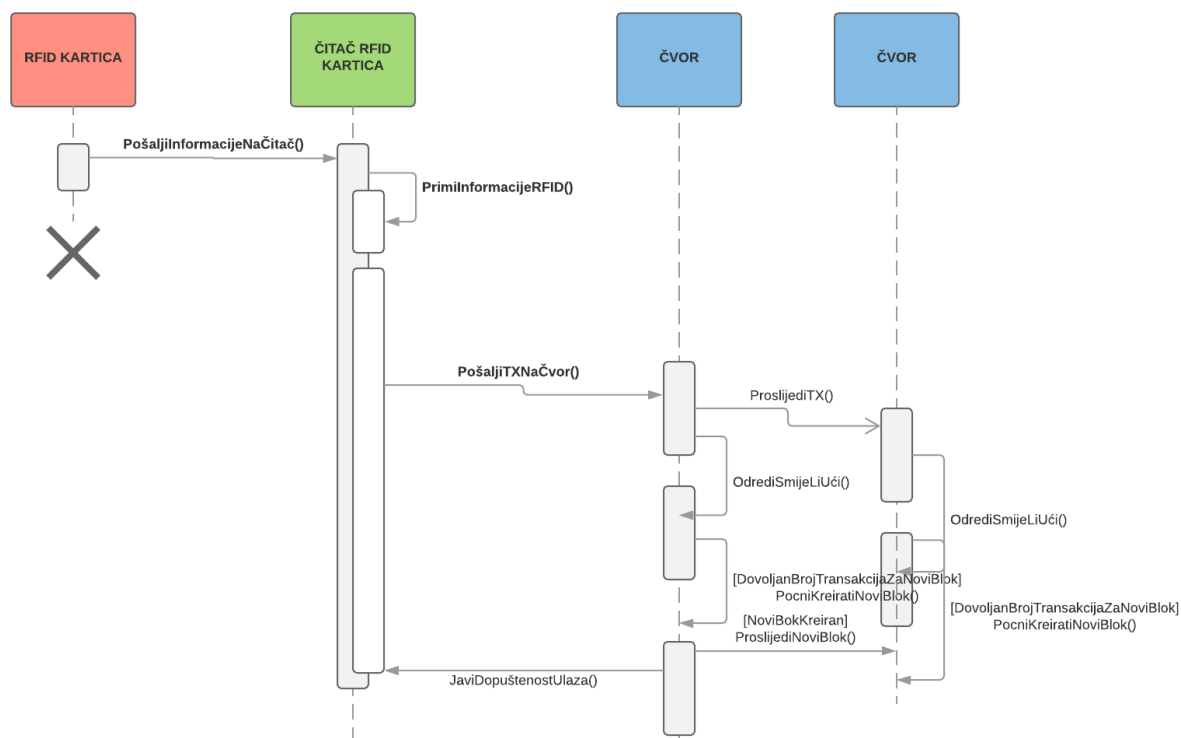
The screenshot displays three terminal windows from a user named 'kresho' on a machine 'kresho-Vostro-5568'. The windows show the execution of a Node.js script named 'index.js' which simulates access control on the Ethereum blockchain. The script sends transactions to a smart contract and receives responses.

Terminal 1 (Left): Shows the script running 'node index.js serverStart FSB'. The output indicates the server is listening on port 3007. It then shows a transaction being sent to a smart contract (E200341201301700026A6B33,303033363131313133330000) and the response 'Access true for hashed jmbag 0x8e2a2b9cb7ecd9104a28f0f06ea59f470de9ff28 at hashed university 0x88c79c1dc9d13b2b592b9057730df504809a43c1 in Fri Jun 08 2018 14:58:41 GMT+0200 (CEST)'. It then shows another transaction being sent to the same smart contract and the response 'Access false for hashed jmbag 0x7827bca6cc10568432cbff663192f7e7fabe0e08 at hashed university 0x88c79c1dc9d13b2b592b9057730df504809a43c1 in Fri Jun 08 2018 14:58:50 GMT+0200 (CEST)'. Finally, it shows a transaction being sent to the same smart contract and the response 'Access true for hashed jmbag 0x4bbf99875a39fc1f858d4267f77fe877f7f8574e at hashed university 0x88c79c1dc9d13b2b592b9057730df504809a43c1 in Fri Jun 08 2018 14:58:52 GMT+0200 (CEST)'.

Terminal 2 (Middle): Shows the script running 'node index.js serverStart FER'. The output indicates the server is listening on port 3005. It then shows a transaction being sent to a smart contract (E200341201301700026A6B33,303033363131313133330000) and the response 'Access true for hashed jmbag 0x4bbf99875a39fc1f858d4267f77fe877f7f8574e at hashed university 0x88c79c1dc9d13b2b592b9057730df504809a43c1 in Fri Jun 08 2018 14:58:39 GMT+0200 (CEST)'. It then shows another transaction being sent to the same smart contract and the response 'Access true for hashed jmbag 0xa7cb41cdf0f7a8d70afcb05e415d64d46429056c at hashed university 0x88c79c1dc9d13b2b592b9057730df504809a43c1 in Fri Jun 08 2018 14:58:49 GMT+0200 (CEST)'. Finally, it shows a transaction being sent to the same smart contract and the response 'Access true for hashed jmbag 0x4bbf99875a39fc1f858d4267f77fe877f7f8574e at hashed university 0x88c79c1dc9d13b2b592b9057730df504809a43c1 in Fri Jun 08 2018 14:58:52 GMT+0200 (CEST)'.

Terminal 3 (Bottom): Shows the script running 'node index.js serverStart FFZG'. The output indicates the server is listening on port 3006. It then shows a transaction being sent to a smart contract (E200341201301700026A6B33,303033363131313133330000) and the response 'Access false for hashed jmbag 0xaa3d448f807d2053428db04c63e9aaac1f400807 at hashed university 0x011c945f30ce2cbaf452f39840f025693339c42 in Fri Jun 08 2018 14:58:42 GMT+0200 (CEST)'. It then shows another transaction being sent to the same smart contract and the response 'Access true for hashed jmbag 0x4bbf99875a39fc1f858d4267f77fe877f7f8574e at hashed university 0x011c945f30ce2cbaf452f39840f025693339c42 in Fri Jun 08 2018 14:58:52 GMT+0200 (CEST)'.

Slika 3.14: Prikaz simulacije triju uređaja za kontrolu ulaza na različitim fakultetima na platformi Ethereum



Slika3.15: Sekvencijski dijagram aplikacije za kontrolu ulaza na Ethereumu

4. Usporedba aplikacije na Ethereumu i Hyperledgeru

4.1. Usporedba na temelju performansi

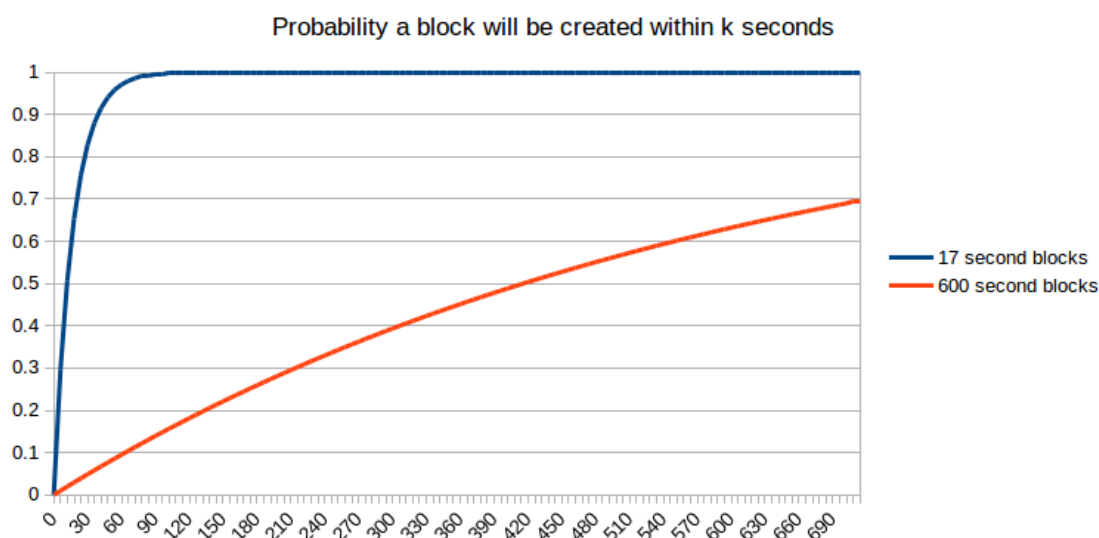
Mreža na Ethereumu je poprilično spora ako se usporedi s tradicionalnim SQL i NoSQL bazama podataka. SQL i NoSQL baze podataka mogu podnijeti i do 15000 modifikacija po sekundi u sigurnom načinu rada, a do 100000 tps na SSD diskovima ako je isključen siguran način rada. Nasuprot tome u javnom okruženju Ethereumovih 15 tps odnosno privatnom okruženju 45 transakcija po sekundi izgleda smiješno. Međutim brzina čitanja stanja na blockchainu ovisi samo o brzini diska na lokalnom čvoru. Hyperledger postiže nešto bolje rezultate i može doći do 3000 tps po kanalu, a sustav se može dizajnirati tako da ima više kanala po potrebi.

Zbog brzine na Ethereum mreži može se doći do zaključka da bi nekoliko aplikacija srednje veličine na Ethereum mreži zagušilo mrežu[24] čak i ako se radi o privatnoj mreži Sveučilišta u Zagrebu u kojem bi možda u trenutcima gužvi bilo više od 45 tps. Nasuprot tome Hyperledger ima fleksibilnu mogućnost kanala te ne moraju sve komponente sveučilišta biti umrežene kako bi se postigla efikasnost privatnosti podataka i brzine.

How nodes are distributed	One-hop node-to-node latency in the cluster (t_{hop})	Minimum internal transaction latency in the cluster (minimum $t_{internal}$)
In one data center	≈ 0.25 ms	≈ 2.25 ms + $3 \cdot t_{qp}$
In one region (e.g. America)	≈ 70 ms	≈ 630 ms
Spread globally	≈ 150 ms	≈ 1350 ms

Tablica 4.1 Vrijeme čekanja na pojavu nove transakcije na jednom čvoru/skupu čvorova

Prosječno vrijeme čekanja primanja nove transakcije u skupini čvorova u nekoj regiji je 70ms međutim prosječno vrijeme čekanja kreiranja novog bloka transakcija u Ethereum mreži je 15 sekundi, te treba pričekati još nekoliko blokova da se može sa sigurnošću potvrditi da je transakcija ušla u blok. Vrijeme čekanja u najboljem slučaju u pravoj mreži je ~17 sekundi što je za upotrebu u sustavu kontrole ulaza koji zahtjeva odaziv u realnom vremenu previše.



Slika 4.1: Vjerojatnost kreiranja novog bloka za k broj sekundi ako u mreži dogovoren algoritam za kreiranje svakih n sekundi

Jedan od načina ubrzanja je kreirati podlanac koji će imati brz odaziv, te će se redovito sinkronizirati s pravim lancem. Takav sustav bi, ako se uspoređuju slične aplikacije na mreži, mogao imati vrijeme čekanja do maksimalno jedne sekunde te bi podlanac ovisio o PoA algoritmu.

Hyperledger platforma je značajno brža od Ethereumove. Sveukupno vrijeme čekanja transakcija je oko 500ms u najgorem slučaju totalne popunjenosti i to je u slučaju jednog kanala. U tablici 5.1 se mogu vidjeti statistički podatci poredani po fazama. Moguće je postaviti više kanala, jedan za svaki fakultet ili čak nekoliko za svaki ulaz po jedan te nekoliko čvorova za slaganje transakcija te bi se time dodatno smanjilo vrijeme čekanja.

	avg	st.dev	99%	99.9%
(1) endorsement	5.6 / 7.5	2.4 / 4.2	15 / 21	19 / 26
(2) ordering	248 / 365	60.0 / 92.0	484 / 624	523 / 636
(3) VSCC val.	31.0 / 35.3	10.2 / 9.0	72.7 / 57.0	113 / 108.4
(4) R/W check	34.8 / 61.5	3.9 / 9.3	47.0 / 88.5	59.0 / 93.3
(5) ledger	50.6 / 72.2	6.2 / 8.8	70.1 / 97.5	72.5 / 105
(6) validation (3+4+5)	116 / 169	12.8 / 17.8	156 / 216	199 / 230
(7) end-to-end (1+2+6)	371 / 542	63 / 94	612 / 805	646 / 813

Tablica 4.2: Usporedba vremena čekanja 5 faza Hyperledger Fabrica u ms (milisekundama)

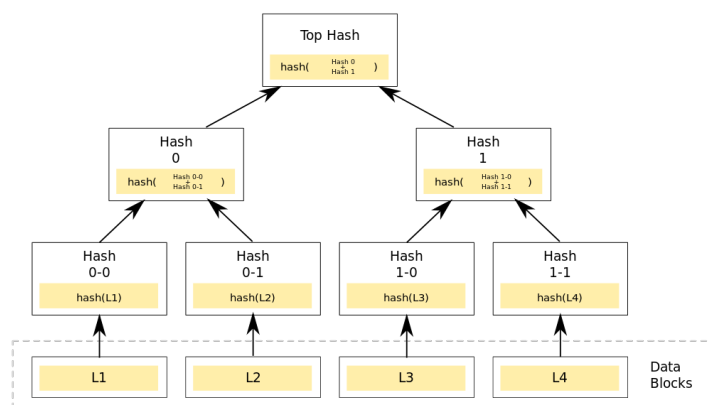
4.2. Usporedba na temelju zahtijevanih razina povjerenja

Ethereum je mreža otvorenog tipa te se zbog toga niti jednom čvoru ne smije vjerovati. Svi čvorovi zbog toga moraju provjeravati svaku transakciju te ako je neki čvor zlonamjerman i želi lošu transakciju poslati na, ostali čvorovi će to vidjeti i odbaciti, sve dok zlonamjerni čvor ne skupi preko 51% računalne moći. Međutim u aplikaciji za kontrolu ulaza to znači da čvorovi ne mogu dopuštati ulaz osobama čak ni ako su zlonamjerne osobe preuzele vlast nad čvorom za kontrolu ulaza.

Hyperledger mreža je zatvorenog tipa, pa se čvorovima u kanalu može vjerovati da su poslali dobronamjernu transakciju. Ipak ako je transakcija u sukobu s lokalnom bazom čvora ili logika transakcije nalaže da se transakcija poništi, lokalni čvor tu transakciju neće potpisati kao valjanu nakon izvršavanja te čvor za slaganje transakcija ne mora tu transakciju izvršiti uvrstiti u blockchain. U aplikaciji za kontrolu ulaza se može staviti restriktivni način da svi sudionici mreže moraju potpisati transakcije te se time dobije mreža kao na Ethereumu, međutim zlonamjerman čvor ne mora potpisati transakciju te može poništiti valjanu transakciju. Zbog toga se predlaže manje restriktivan način u aplikaciji za kontrolu ulaza tako da transakciju potvrdi više od dvije trećine čvorova te se time dobiva BFT algoritam konsenzusa.

4.3. Usporedba na temelju sigurnosti podataka i privatnosti

Objekti platforme koriste inačicu distribuirane knjige zapisa zvanu *blockchain*. *Blockchain* platforme obično koriste računala koja obavljaju neke izračune, kako bi dokazali da su validni. Zatim podatke koje su im dostavili ostali čvorovi spremaju u blokove koje



Slika 4.2: Merkle tree struktura podataka koju koristi *blockchain*

povezuju s prijašnjim blokovima preko njihovog *hasha*, zato se i zove Blok lanac (eng. *blockchain*) koja je zapravo Merkle tree struktura podataka opisana 80tih godina 20. st., te prosljeđuju novo stanje svim ostalim čvorovima. Zbog toga što su svi blokovi međusobno povezani *hashevima* prošlog bloka, gotovo je ne moguće promijeniti blokove koji su daleko u lancu pošto bi se moralo promijeniti i svaki blok nakon njega. Jedna mogućnost mijenjanja blokova je 51% napad (eng. 51% *attack*). Napad u kojemu napadač ima utjecaj na 51% mreže, te promijeni blokove, a svi koji se ne slažu s mijenjanjem blokova, a to je manjina, budu odbačeni od strane glavne mreže.

Ethereum je baziran na PoW algoritmu te je zbog otvorenosti pristupa mreži izložen 51% napadu. Princip Hyperledger Fabric mreže je da čvorovi moraju tražiti pristup kanalu i ako im administrator kanala odobri pristup i potvrdi da su akreditacije izdane od valjanog CA čvora, onda imaju pravo sudjelovati u mreži. Time što je potvrđen identitet čvora, smatra se da je čvor siguran i neće pokušati sabotirati mrežu. Ako čvor ipak pokuša sabotirati mrežu, ostali čvorovi u mreži, ako nisu pod utjecajem napadača, će odbaciti transakcije kao nevaljale. Podatci su s toga na Ethereum platformi kriptografski sigurni od mijenjanja te ako napadač dođe do 51% kontrole na mreži, morao bi utrošiti mnogo računalne snage da promijeni sve podatke. Ako napadač dođe do 51% računalne snage, može unositi u *blockchain* transakcije koje nisu valjane. [24] Ako se uvede podlanac (eng. *subchain*) te bude mnogo čvorova za Ethereum mrežu, teško će doći do prikupljanja 51% računalne snage koja bi srušila aplikaciju za kontrolu ulaza.

Na Hyperledger Fabric platformi su sigurni od mijenjanja sve dok napadač ne kontrolira sve čvorove u kanalu te tako izmijeni blockchain kompletne mreže. Nove transakcije koje nisu valjane u mrežu napadač može poslati ako kontrolira sve čvorove koji moraju potpisati transakciju da je valjana. Međutim Fabric algoritam za slaganje novih transakcija u blokove je ovisan o čvoru za slaganje transakcija te kao takav nije BFT, a u slučaju da postoji samo jedan čvor za slaganje transakcija, algoritam nije ni CFT. Nadogradnja algoritma kako bi bio BFT je planirana. [23] Ako se uvede pravilo potvrđivanja transakcija tako da svi čvorovi moraju potvrditi transakcije u mreži, aplikacija za kontrolu ulaza na Hyperledger Fabricu je sigurna od vanjskih napada.

U Ethereum platformu mogu doći bilo koji čvorovi te vidjeti sve podatke unutar mreže. Zbog toga se podatci koji trebaju biti zaštićeni, a da ih je moguće kasnije pročitati, kriptiraju, a podatci za koje je bitno samo da nisu mijenjani se *hashiraju*. U aplikaciji

kontrole ulaza JMBAG i UniversityKey su hashirani pošto original podatci se dohvaćaju s RFID kartice, a hashirani podatci služe za provjeru.

U Hyperledger Fabric platformi postoji sustav dozvola zbog kojeg u mreži sudjeluju samo poznati provjereni čvorovi. Nema potrebe kriptirati ili *hashirati* podatke zbog toga te se podatci u *chaincode* pri izvršavanju te u *blockchain* pri spremanju unose kakvi jesu. U aplikaciji kontrole ulaza svi podatci su u *blockchain* zapisani kako su dohvaćeni s RFID kartice.

4.4. Usporedba na temelju složenosti aplikacije

Složenost dizajniranja aplikacije za Ethereum platformu je veća od dizajniranja klasičnih aplikacija s klasičnim bazama podataka. Naime mora se voditi računa o skrivanju te kriptiranju osjetljivih podataka te treba naučiti novi programski jezik Solidity i znati optimizirati pametni ugovor. Povezivanje WEB sučelja ili server sučelja s *blockchainom* se najčešće odvija preko biblioteke Web3 koju je potrebno savladati. Međutim u pametnom ugovoru se na jednome mjestu mogu definirati svi potrebni modeli podataka, ograničenja nad pogledom podataka te logika transakcija. Čvorovi za transakcije se lagano kreiraju i dodaju u mrežu pošto ne treba tražiti nikakvu dozvolu. Pametni ugovor aplikacije kontrole ulaza je jednostavan za shvatiti te povezivanje korištenih sučelja na blockchain je relativno jednostavno.

Hyperledger Fabric Composer platforma se za logiku transakcija te povezivanje WEB ili serverskog sučelja koristi popularnim programskim jezikom JavaScript te je u prednosti nad većinom *blockchain* platformi. Međutim ostale karakteristike poput modeliranja baze te dozvola pristupa podacima se programiraju odvojeno i s posebnim pravilima koje je potrebno savladati. Čvorovi za kreiranje transakcija se teško dodaju u mrežu jednom kada je mreža napravljena te treba tražiti odobravanje pristupa pošto je Fabric mreža s dozvolom za pristup. Iako se pametni ugovori aplikacije za kontrolu ulaza lako shvate, ostatak mreže je teže posložiti.

4.5. Zaključak usporedbi

Aplikacija za kontrolu ulaza zahtjeva provjeru ulaza u realnom vremenu što na Ethereum platformi na javnoj mreži nije moguće. Štoviše za provjeru ulaza na javnoj

mreži, po cijeni od 520\$ za jedan Ether na dan 11.6.2018, trebalo bi se platiti 0.93\$. Rješenje su podlanci koji bi se sinkronizirali s javnom mrežom, ali bi ti podlanci koristili ili privatne čvorove ili čvorove koji koriste PoA algoritme kako bi ubrzali izvođenje. Međutim korištenje PoA algoritama ili originalni Ethash PoW algoritam na maloj težini, kako bi rješavanje kriptografske zagonetke bilo brzo za korištenje u realnom vremenu, potkopava sigurnost aplikacije. Hyperledger Fabric je značajno brži zbog korištenja drugačije vrste algoritma za potvrdu transakcija koji je sličan Ethereumovom PoA algoritmu. Osim toga Fabric je značajno fleksibilniji što se tiče skalabilnosti od Ethereum. Ako se postavje jednaki uvjeti: korištenje jednog podlanca u Ethereum platformi te kreiranje jednog kanala za vezu aplikacije za kontrolu ulaza s glavnim sveučilišnim centrom, Hyperledger Fabric ima značajnu prednost u brzini transakcija i broju transakcija po sekundi kao i zaštitu od modificiranja blockchaina. Ako se poveća složenost računanja Ethash algoritma na Ethereumu, Ethereum postaje sigurniji što se tiče mijenjanja zapisanih podataka.

Ethereum platforma ne zahtijeva nikakvu razinu povjerenja te su svi bitni podatci ili kriptirani ili *hashirani*, odnosno svaki čvor ne mora nikome drugome vjerovati osim sebi. Ako dođe do račvanja mreže, čvor se može prikloniti bio kojoj strani mreže, ali za slučaj aplikacije kontrole ulaza za određeno sveučilište se ne želi doći do takve situacije prema tome fleksibilnost povjerenja u pravoj mreži je veliki plus, ali u ovakvoj mreži bi bio minus. Ako određena transakcija ima pogrešku i sustav ju želi izmijeniti mora doći do 100 postotnog prihvaćanja čvorova kako bi se promjena izvela. Aplikacija za kontrolu ulaza ima sigurnosne zahtjeve u pametnom ugovoru koji onemogućavaju slanje transakcija i novih korisnika za račune drugih fakulteta, pa ako napadač šalje takve transakcije i smatra da su valjane, tada će on ostati sam na mreži, a mreža će se računati.

Ako je transakcija zlonamjerna i svi čvorovi su zlonamjerni, oni mogu promijeniti stanje u mreži prije nego što se podlanac sinkronizira s glavnim lancem. Primjer bi bio vratiti sustav u stanje prije nego što je osoba ušla na fakultet kao da nikada nije ušla.

U Hyperledger Fabric čvorovi moraju vjerovati čvoru za slaganje transakcija te ako certifikati nisu izdani pri kreiranju kanala odnosno postoji CA čvor koji izdaje certifikate i njemu. Ostalim čvorovima u kanalu je potrebno donekle vjerovati, ovisi o jačini konsenzusa koji se koristi. [25] Fabric je fleksibilan oko konsenzusa valjanosti transakcije te ako se u kanalu namjesti opcija da svi čvorovi u kanalu moraju potvrditi transakcije onda moramo vjerovati da su svi dobronamjerni i ne postoji jedan čvor koji ne želi dati

potpis valjanoj transakciji. Ako se u kanalu namjesti opcija da je samo jedan čvor potreban za potvrđivanje transakcija, onda se svim čvorovima treba vjerovati da su dobronamjerni. Iako ne valjale transakcije neće ući u blockchain na dobronamjnim čvorovima, oni će se ipak zapisati na čvorovima pod utjecajem napadača, a mreža se neće odvojiti kao što je slučaj u Ethereumu. Aplikacija za kontrolu ulaza ima podešene postavke sigurnosti slanja transakcija i novih korisnika na *blockchain* takav da će zlonamjerne transakcije biti valjale samo na napadačevom čvoru.

S obzirom na to da je čista aplikacija kontrole ulaza na Ethereum mreži nedovoljno dobra za izvršavanje u realnom vremenu te bi se morali koristiti podlanci i sakrivanje podataka, složenost raste te dolazi do ranga Hyperledger Fabric aplikacije koja sve već ima ukomponirano u modularnu cjelinu.

Zaključak

Aplikacije s konvencionalnom bazom podataka muči problem mijenjanja baze od strane administratora sustava ili napadača koji može dobiti administratorska prava. Aplikacija za kontrolu ulaza temeljena na blockchain tehnologiji je zbog toga potrebna kako bi se osigurali podaci od mijenjanja te kako logika transakcije ne bi bila vezana uz čvorove, a istovremeno bi bila sigurna od napada na sustav. RFID nije najbolje rješenje za identifikaciju na blockchainu pošto je u RFID kartice moguće spremati samo 12B koji su nedostatni za spremanje adrese na Ethereum blockchainu ili privatni ključ na Hyperledgeru. Ipak spremanjem identifikacijskog podatka JMBAG na RFID te čitanje istog na Hyperledgeru ili naknadnim *hashiranjem* JMBAG-a na Ethereumu omogućena je sigurnost jednaka ZKP algoritmima koji se koriste za konvencionalne baze.

Razine povjerenja za aplikaciju kontrole ulaza na Ethereum i Hyperledger blockchainu su fleksibilne te su sigurno bolje od konvencionalnih baza u kojima se mora vjerovati glavnom administratoru baze. Brzina transakcija i čekanje na njihovo izvršavanje su ipak značajno sporije od konvencionalnih baza podataka te su mnoga buduća rješenja, pogotovo za Ethereum, usmjerena na ubrzavanje mreže, a da pritom sigurnost podataka od mijenjanja ostane ista.

Sigurnost privatnih blockchain sustava od strane zlonamjernih sudionika nije narušena ako se aplikacija dobro napravi te ako se ograniče ovlasti administratora za svaki čvor odnosno kanal. Hyperledger Fabric sustav privatni je *blockchain* sustav koji je uspio u svojoj namjeri, a to je da postane *blockchain* sustave za jeftinu, brzu i relativno sigurnu interakciju između tvrtki. Ethereum sustav je pokazao svoje slabosti u izvršavanju u realnom vremenu te nepraktičnost sustava koji bez dozvole u korporativnom okruženju. U trenutku pisanja rada ako se određene postavke promijene da sustav bude dovoljno brz, gubi na sigurnosti. Razni Ethereum programski okviri koji bi trebali jamčiti velik broj sigurnih i nepromjenjivih transakcija za privatno okruženje uskoro dolaze na tržište među njima je Ethereum Quantum koji uzima princip Hyperledgera te prebacuje na Ethereum platformu, a i Ethereum glavna mreža uskoro prelazi na novi PoS algoritam potvrđivanja transakcija, pa Ethereum u budućnosti još uvijek stigne pokazati da je bar jednako dobro rješenje kao Hyperledger.

Literatura

- [1] Hyperledger – Fabric docs,
<http://hyperledger-fabric.readthedocs.io/en/latest/whatis.html>, Preuzeto s interneta, 30.5.2018
- [2] MEDIUM Tutorial Blog,
<https://medium.com/@philippsandner/comparison-of-ethereum-hyperledger-fabric-and-corda-21c1bb9442f6>, Preuzeto s interneta, 30.5.2018.
- [3] Hyperledger Fabric performance benchmark paper,
<https://arxiv.org/pdf/1801.10228v2.pdf>, Preuzeto s interneta, 21.4.2018
- [4] Hyperledger – Fabric Network docs
<http://hyperledger-fabric.readthedocs.io/en/latest/network/network.html>, Preuzeto s interneta, 30.5.2018
- [5] Ethereum project wiki
<https://github.com/ethereum/wiki/wiki/>, Preuzeto s interneta, 30.5.2018
- [6] Ethereum yellow paper
<https://ethereum.github.io/yellowpaper/paper.pdf>, Preuzeto s interneta, 30.5.2018
- [7] Transaction speed Ethereum
<https://ethereum.stackexchange.com/questions/36565/what-is-a-private-proof-of-authority-transaction-speed>, Preuzeto s interneta, 30.5.2018
- [8] Next Ethereum version Serenity
<https://cointelegraph.com/news/is-serenity-the-solution-to-ethereums-difficulty-bomb>, Preuzeto s interneta, 30.5.2018
- [9] PoS consensus algorithm CASPER
<https://cryptodisrupt.com/ethereums-hybrid-pow-pos-consensus-algorithm-casper/>, Preuzeto s interneta, 30.5.2018
- [10] Smart Contracts, explained
<https://cointelegraph.com/explained/smart-contracts-explained>, Preuzeto s interneta, 31.5.2018
- [11] Smart Contracts, Ethereum
<https://www.cryptocoinsnews.com/counterparty-brings-ethereum-smart-contracts-to-the-bitcoin-blockchain/>, Preuzeto s interneta, 31.5.2018
- [12] Solidity docs
<https://solidity.readthedocs.io/en/latest/>, Preuzeto s interneta, 31.5.2018
- [13] Ethereum white paper
<https://github.com/ethereum/wiki/wiki/White-Paper>, Preuzeto s interneta, 31.5.2018
- [14] Permissionless Ethereum system
<https://coincenter.org/entry/what-does-permissionless-mean>, Preuzeto s interneta, 31.5.2018

- [15] Ethereum transactions
<https://medium.com/blockchannel/life-cycle-of-an-ethereum-transaction-e5c66bae0f6e>, Preuzeto s interneta, 31.5.2018
- [16] Hyperledger Composer Github page
<https://hyperledger.github.io/composer/latest/introduction/introduction.html>,
Preuzeto s interneta, 5.6.2018
- [17] Hyperledger Composer Github page, deploying to peers
<https://hyperledger.github.io/composer/latest/tutorials/developer-tutorial.html>,
5.6.2018
- [18] Hyperledger Composer Github page, Modeling language
https://hyperledger.github.io/composer/latest/reference/cto_language.html, Preuzeto
s interneta, 5.6.2018
- [19] Hyperledger Composer Github page, Query language
<https://hyperledger.github.io/composer/latest/tutorials/queries>, Preuzeto s interneta,
5.6.2018
- [20] Hyperledger Composer Github page, Permissions
https://hyperledger.github.io/composer/latest/reference/acl_language, Preuzeto s
interneta, 6.6.2018
- [21] Ethereum Truffle docs page, <http://truffleframework.com/docs/>, Preuzeto s interneta
7.6.2018
- [22] Web3 library, <https://web3js.readthedocs.io/en/1.0/web3-eth.html> , Preuzeto s
interneta 8.6.2018
- [23] Consensus algorithm Hyperledger, https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf, Preuzeto
s interneta 10.6.2018
- [24] Ethereum transaction per second too slow, <https://medium.com/gochain/if-1000-dapps-on-ethereum-need-to-do-1-transaction-per-second-what-happens-9e6247e0beca>, Preuzeto s interneta 10.6.2018
- [25] Trust on Hyperledger Fabric, <https://arxiv.org/pdf/1805.08541.pdf>, Preuzeto s
interneta 10.6.2018

Sažetak

Primjena raspodijeljene knjige zapisa za kontrolu ulaza

Rad uspoređuje dvije po konceptu veoma različite *blockchain* platforme, te na kraju donosi zaključak o uspješnosti platformi na temelju aplikacije za kontrolu ulaza. Ethereum platforma je primjer prve platforme koja je osmišljena da podržava pametne ugovore, a u mrežu se može ući bez dozvole i čitati sve podatke na blockchainu. Hyperledger Fabric je platforma koja također podržava pametne ugovore, međutim u mrežu se ne može ući bez dozvole i ne mogu se čitati svi podaci već oni za koje čvor ima pristup. U Ethereum platformi se ne mora nikome vjerovati te je osmišljena kao javna mreža, a u Hyperledger platformi se mora vjerovati određenim čvorovima i mreža je zamišljena za tvrtke. Krajnja usporedba donosi i zaključke o potrebi blockchaina za kontrolu ulaza naspram konvencionalnih baza podataka.

Ključne riječi: Blockchain, Raspodijeljena knjiga zapisa, Ethereum, Hyperledger Fabric, javni vs. privatni blockchain, Kontrola ulaza

Summary

Use of distributed ledger for access control

This work compares two concepts of different blockchain platforms, and ultimately brings conclusion on the platform's performance based on access control application. The Ethereum Platform is an example of the first platform designed to support smart contracts and access on the network is permissionless plus nodes can read all the data in blockchain freely. Hyperledger Fabric is a platform that also supports smart contracts, but the network is permissioned plus nodes can read the data on blockchain for which they have access. In the Ethereum platform, other nodes does not need to trust anyone and the network is designed as a public network. In the Hyperledger platform other nodes have to trust certain nodes and the network is designed for corporations. The final comparison also draws conclusions for the need of blockchain for access control applications versus conventional databases.

Keywords: Blockchain, Distributed ledger, Ethereum, Hyperledger Fabric, Public vs. Private blockchain, Access Control

Privitak