Provjera koncepta decentralizirane aplikacije za izdvajanje izračuna na blockchainu

Fajdetić, Kristijan

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Preddiplomski sveučilišni studij računarstva

Završni rad

PROVJERA KONCEPTA DECENTRALIZIRANE APLIKACIJE ZA IZDAVANJE IZRAČUNA NA BLOCKCHAINU

Rijeka, rujan 2022.

Kristijan Fajdetić

0069088155

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Preddiplomski sveučilišni studij računarstva

Završni rad

PROVJERA KONCEPTA DECENTRALIZIRANE APLIKACIJE ZA IZDAVANJE IZRAČUNA NA BLOCKCHAINU

Mentor: prof. dr. sc. Kristijan Lenac

Rijeka, rujan 2022.

Kristijan Fajdetić

0069088155

Umjesto ove stranice umetnuti zadatak
za završni ili diplomski rad

Izjava o samostalnoj izradi rada

Izavljujem da sam samostalno izradio ovaj rad.

Rijeka, rujan 2022.

_____

Kristijan Fajdetić

ZAHVALA

Zahvaljujem se svim profesorima koji su me pratili tijekom studija, pogotovo prof. dr. sc. Kristijanu Lencu kao i cijelom blockchain timu Tehničkog fakulteta u Rijeci na iskazanom povjerenju, razumijevanju i posvećenom vremenu, ukazanoj prilici te iskustvu za razvoj karijere kroz blockchain tehnologiju.

SADRŽAJ

Popis slika

Popis primjera koda

Uvod

Rastućim problemom zaštite privatnosti osobnih podataka, kriptovalute, kao digitalni novac kreiran u digitalnom obliku, postale su sve aktualnije sredstvo digitalne razmjene bez posredovanja treće strane. Za privatnost i rad kriptovaluta zaslužna je blockchain tehnologija. Blockchain je decentralizirana i distribuirana knjiga (eng. ledger), koja olakšava proces bilježenja i praćenja transakcija.[1] Razvoj blockchaina omogućio je njegovu primjenu ne samo u financijama već i u brojnim drugim sektorima poput medicine, logistike, upravljanja podacima i drugih. Ovo je pridonijelo razvoju decentraliziranih aplikacija (DAPP), koje za razliku od tradicionalnih, centraliziranih aplikacija, za svoj rad koriste blockchain. Blockchain aplikacijama omogućuje veću sigurnost i privatnost pomoću peer-to-peer (P2P) mreže gdje je svaki čvor anoniman i sadrži kopiju blockchaina pa samim time i stanja aplikacije. Svi podaci su enkriptirani, dok se za komunikaciju s ostalim čvorovima i pristupanje mreži koriste brojne kriptografske metode.[2]

Kako bi se postigla sinkronizacija i dogovor između čvorova na mreži, blockchain koristi razne mehanizme za postizanje konsenzusa. Kriptovalute poput bitcoina koriste proof-of-work (PoW) mehanizam, koji za svoj rad zahtjeva veliku količinu električne energije.

Kao alternativa ovom konsenzusu sve se češće koristi proof-of-stake (PoS), ali ovaj mehanizam smanjuje ukupnu sigurnost blockchaina budući da se ne oslanja na realni vanjski resurs već na uloženu količinu kriptovalute, čime se smanjuje gubitak za zlonamjerne čvorove.

Kako bi se riješili ovi problemi konsenzusnih mehanizama osmišljen je održivi mehanizam za postizanje konsenzusa zvani sustainable proof-of-work (SPOW). Ovaj rad bavi se razvojem decentralizirane aplikacije, kojom se omogućava daljini razvoj održivog mehanizma za postizanje konsenzusa. Za svrhe implementacije i testiranja aplikacije postavljena je testna blockchain mreža.

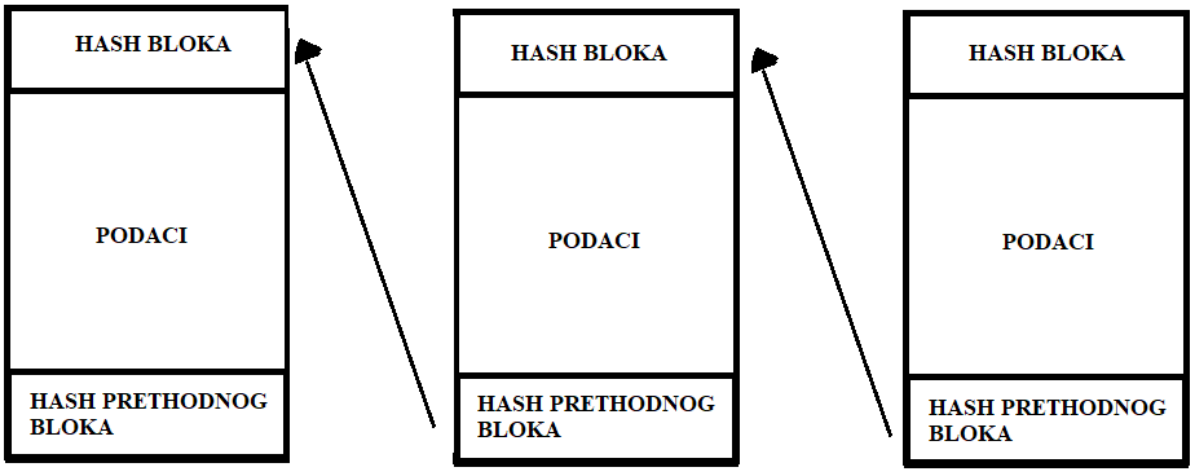
U prvom poglavlju opisana je blockchain tehnologija i konsenzusni mehanizmi uključujući SPOW kao glavnu motivaciju za izradu ove aplikacije. Drugo poglavlje rada objašnjava tehnologije koje je potrebno poznavati kako bi se razumjela arhitektura i princip rada aplikacije. U trećem poglavlju opisan je postupak postavljanja testne mreže te implementacija same aplikacije. Kroz četvrto poglavlje prikazano je pokretanje testne blockchain mreže i aplikacije te interaktivno tekstualno korisničko sučelje, a u zaključku su opisani problemi i ograničenja te predložena poboljšanja za daljnji razvoj decentralizirane aplikacije.

1. BLOCKCHAIN

Blockchain je temeljna tehnologija kriptovaluta poput bitcoina i ethereuma. Ovaj jedinstveni način sigurne pohrane i prijenosa informacija ima široke primjene i u drugim domenama izvan kriptovaluta. Za njegovu pojavu najzaslužniji je pseudonim Satoshi Nakamoto sa svojim radom iz 2009. godine.[3] Blockchain vrsta je distribuirane knjige. Tehnologija distribuirane knjige (DLT) omogućuje vođenje zapisa na više računala, poznatih kao čvorovi. Čvorovi provjeravaju, odobravaju i pohranjuju podatke unutar glavne knjige. Ovo se razlikuje od tradicionalnih metoda vođenja evidencije, koje pohranjuju podatke na središnji računalni poslužitelj.

1.1. Princip rada blockchaina

Blockchain se sastoji od skupine blokova koji su međusobno povezani u lanac. Svaki blok čini logičku organizaciju provedenih transakcija, a veličina bloka ovisi o načinu implementacije blockchaina. Veličina bloka i informacije koje se spremaju ovise o implementaciji i potrebama blockchaina. Struktura bloka također ovisi o tipu blockchaina i načinu implementacije, a glavni atributi koje svaki blok mora sadržavati su: zaglavlje, brojač transakcija i listu transakcija. Zaglavlje sadrži pokazivač na vrijednost hash funkcije prethodnog bloka kako je prikazano na slici 1.1, vremensku oznaku kreiranja bloka te Merkle stablo izvedeno iz hash vrijednosti svih transakcija uključenih u ovaj blok, čime se osigurava da se nijedna od tih transakcija ne može modificirati bez modificiranja zaglavlja bloka. Blokovi su obično adresirani prema njihovoj visini u lancu odnosno prema indeksu. Posebno prvi blok visine nula ili blok geneze (eng. genesis block) nema poveznicu na prethodni blok. Pri svakom kreiranju novog bloka, čvor uzima transakcije iz skupine neobrađenih transakcija koje se čuvaju u memorijskom bazenu (eng. memory pool) ili mempool. Prema pravilima protokola, kako bi se potvrdio blok transakcija na čekanju, čvor provjerava svoju kopiju glavne knjige koja uključuje sve prethodne transakcije, kako bi osigurao da postoji dovoljna količina novčanih sredstava odnosno kriptovalute sudionika koji ih šalje.



Slika 1.1 Pojednostavljeni prikaz niza povezanih blokova u blockchainu

Kako bi se dodao na blockchain, ovaj blok transakcija raspršuje se putem hash funkcije. Hash funkcija iz ulaznih podataka stvara jedinstveni niz znamenki. Ulaz u hash funkciju razlikuje se za pojedini blockchain; ono što je nužno je da jedan od ulaza bude hash vrijednost prethodnog bloka. Budući da se hash svakog bloka izračunava koristeći hash bloka prije njega, svaki je blok povezan sa svojim prethodnim blokom i svim blokovima prije njega, uključujući njihove transakcije. Mala promjena u bilo kojem bloku (kao što je dodavanje lažne transakcije) potpuno bi promijenila hash svakog sljedećeg bloka. Kao rezultat toga, kako bi se osiguralo da je blockchain valjan, čvorovi ne moraju ispitati svaki prethodni blok, već samo hash najnovijeg te izbaciti neispravni iz mreže.

1.2. Vrste blockchaina

Kroz godine blockchain se razvio u više smjerova, zbog čega su se razvili više tipova blockchaina s različitim, ali i često sličnim karakteristikama. Razlikujemo: javni, privatni i konzorcijski ili federalni blockchain.

Javni blockchain je otvoren javnosti i unutar njega svatko može sudjelovati kao čvor u donošenju odluka, a korisnici mogu, ali ne moraju biti nagrađeni za njihov doprinos. Svaki korisnik ima kopiju javne knjige (eng. ledger) na svom lokalnom čvoru i koristi distribuirani mehanizam konsenzusa kako bi se donijela odluka o daljnjem stanju blockchaina.

Privatni blockchain je otvoren samo određenoj grupi ljudi ili organizacijama koje su odlučili svoje transakcije vršiti putem blockchaina. Jedino sudionici koje je ovlastio centralni entitet imaju pravo sudjelovanja te je njihov identitet javan.

Konzorcijski ili federalni blockchain je sličan privatnom blockchainu, ali umjesto da je jedan čvor glavni, postoji više glavnih. Radi se o skupini predstavnika ili poduzeća koji donose najvažnije odluke za cijelu mreže.[4]

1.3. Konsenzus

Decentralizacija jedna je od glavnih značajki blockchaina, odnosi se na eliminaciju potrebne treće strane, to jest, nema glavnog autoriteta koji nadzire sustav. Zbog velikih vrijednosti transakcija određen broj sudionika sklon je prevarama kako bi pridobio financijsku korist i naštetio funkcioniranju sustava. Kako bi cijeli sustav, gdje je svaki sudionik povezan s ostalima bez autoritativnog djelovanja među njima funkcionirao, potrebno je postići konsenzus među sudionicima o ispravnosti bloka. Uvode se konsenzusni algoritmi.

Kod donošenja odluke o konačnom stanju blockchaina razlikuju se dva načina postizanja konsenzusa: bazirano na dokazu (eng. proof-based) i bazirano na glasanju (eng. voting-based). Kod konsenzusa baziranog na dokazu ako postoji lanac koji je riješio teži problem odnosno koji je uložio veću količinu rada, onda je taj lanac valjani lanac. Primjer ovakvih konsenzusnih mehanizama su PoW i PoS koje ćemo objasniti u nastavku. Kod konsenzusa baziranog na glasanju konačna odluka se donosi brojanjem glasova čvorova. Ova vrsta mehanizma se ne primjenjuje samostalno već u kombinaciji s nekim drugim jer bi čisti mehanizmi baziran na glasanju bio neefikasan. U ovu kategoriju spadaju konsenzusni algoritmi BFT stila, koji se temelje na PBFT-u. Tendermint je najbolji primjer BFT algoritma koji jamči sigurnost te će biti objašnjeni u nastavku rada.

Konsenzus treba pružiti dva bitna svojstva:

- Sigurnost (eng. safety) - sigurnost podrazumijeva postojanje jednog glavnog lanca u kojem ne postoje dva ili više konkurentskih lanaca s valjanim transakcijama. Konsenzusni mehanizam može biti siguran kada su blokovi konačni. Ovo svojstvo preferira se kod konsenzusa baziranih na dokazu.
- Dostupnost (eng. live) – ako protokol reagira na propagaciju novih transakcija te validne transakcije na kraju stignu na blockchain kažemo da je konsenzus dostupan. Pogreška dostupnosti javlja se kada se među brojnim kršenjima uoči izostavljanje transakcije, uskraćivanje informacija ili preuređivanje poruka.[11] Ovo svojstvo preferira se kod konsenzusa baziranih na dokazu.

1.3.1. Proof-of-Work

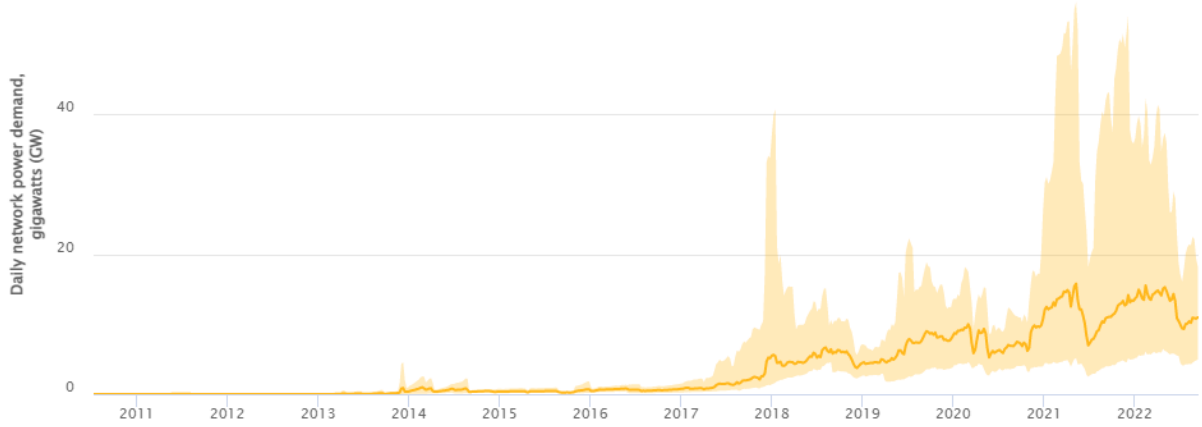
PoW (dokaz o radu) mehanizam je za postizanje konsenzusa zahtjeva da se za svaki blok koji se želi dodati na blockchain uloži određena količina računalnog rada. Ovime se osigurava da zlonamjerni čvorovi koji žele modificirati prijašnje blokove trebaju imati više računalne snage od poštenih čvorova. Taj posao čvoru nameće velike troškove, a time ga potiče na objavljivanje iskrenih informacija. [5]

Računalni rad uključuje uzastopno pokušavanje (eng. brute force) izračuna hash vrijednosti koja zadovoljava određene uvjete. Za bitcoin blockchain potrebno je da za proizvoljan jednokratni niz (eng. nonce) koji se kombinira s ostalim podacima bloka dobije hash koji počinje s određenim brojem nula.[3] Kada čvor pronađe odgovarajući hash, on prenosi blok na kojem je radio ostalim čvorovima koji zatim provjeravaju odgovaraju li dobiveni podaci u bloku hash vrijednosti tog bloka. Čvor prihvaća blok tako što ga dodaje u svoju kopiju blockchaina. Ako je postignut konsenzus između čvorova, čvor koji je napravio blok dobiva nagradu za svoj rad putem specijalne transakcije zvane coinbase, koja sadrži određeni broj novonastalih bitcoina. Nagrada potiče sudjelovanje na mreži.

Što je veća računalna snaga čvora, to je veća vjerojatnost pronalaska hash vrijednosti jer se povećava broj pokušaja za pronalazak hasha svake sekunde. Kako bi tijek nagrada bio stabilan, mreža samostalno prilagođava težinu izračuna hasha tako da se novi blokovi stvaraju, u prosjeku, jednom svakih 10 minuta.

Proces traženja hash vrijednosti često se uspoređuje s rudarenjem zlata odakle dolazi i izraz bitcoin mining. U slučaju bitcoina ovaj izraz označava procesorsko vrijeme i potrošnju električne energije. Zbog ovoga se čvorovi koji gađaju hash vrijednosti zovu rudarima.

Priroda ovog mehanizma izaziva utrku među čvorovima što izaziva potrebu za velikom količinom hardvera i električne energije. Na slici 1.2 prikazana je potrošnja električne energije Bitcoina u gigavatima; za vrijeme pisanja ovog rada vrijednosti nadmašuju potrošnju električne energije nekih zemalja. Ovo je izazvalo brojne debate posljednjih nekoliko godina jer zahtijeva veliku količinu izračuna za krajnji cilj obrade financijskih transakcija bez posrednika.[6] Ovo čini potrošnju električne energije uzaludnom jer se ne radi o korisnom izračunu i tako stvara negativan utjecaj na okolinu.[7]



Slika 1.2 Potrošnja električne energije za rudarenje bitcoina[8]

Za razliku od ostalih mehanizama za postizanje konsenzusa, PoW koristi resurse iz stvarnog svijeta, čime se dobiva na dodatnoj vrijednosti i sigurnosti cijelog sustava jer je za prevaru potrebna velika količina električne energije i snažan hardver, pa time postaje manje isplativo.

Rudarski bazen (eng. mining pool) je grupa rudara koja kombinira svoju računalnu snagu te proporcionalno radu, podijeli se nagrada. Ova ideja prethodila je formiranju nekoliko velikih rudarskih bazena. Niti jedan rudarski bazen ne kontrolira većinu ukupne hash snage mreže, no kada bi se nekolicina rudarskih bazena spojila u jedan veći rudarski bazen, preuzeli bi kontrolu nad mrežom (51% napad).

1.3.2. Proof-of-Stake

PoS se temelji na ulogu (eng. stake) (obično kriptovaluta) svakog čvora kojeg nazivamo validator. Konsenzus se događa kroz nekoliko koraka: prvo se deterministički odabire validator (vjerojatnost odabira povećava se s većim ulogom) koji predlaže blok ostalim čvorovima na mreži. Ostali validatori imaju zadatak potvrditi ispravnost bloka koji dobiju putem glasanja. Ako validator koji predlaže blok zvani predlagatelj (eng. proposer) generira maliciozni blok ili ako validator potvrdi ispravnost malicioznog bloka, gube dio svog uloga. Ovo podrazumijeva da protokol ima više povjerenja u validatora s većim ulogom te je njihov glas vrjedniji od ostalih čvorova budući da se u slučaju zlonamjernog postupka gubi veći dio uloga. Prilikom pridruživanja mreži, ulog validatora se zaključava dok se blok ne pokaže validnim. Sigurnost PoS sustava bazirana je na veličini samog sustava odnosno kako bi se proveo 51% napad, validator treba imati ulog veći od pola vrijednosti sustava.

Prednost PoS mehanizma u odnosu na PoW je manja potrošnja resursa jer samo jedan validator predlaže blok i nema natjecanja, što povećava propusnost i brzinu mreže.

Nedostatak ovog konsenzusa je činjenica da validator s većim ulogom je češće odabran za predlaganje bloka te njegov ulog raste, što onda vodi prema 51% napadu odnosno centralizaciji. Budući da se pomoću PoS mehanizma blokovi lakše i brže generiraju, postoji mogućnost napada dalekog dometa (eng. long range attack) jer nije potrebno uložiti veliku količinu rada kako bi se stvorio novi lanac koji je duži od glavnog.[9] Još jedna razlika od PoW je da ne postoji vanjski resurs i samim time jedini gubitak je uložena kriptovaluta.

1.3.3. Sustainable Proof-of-Work

SPOW je mehanizam za postizanje konsenzusa koji se razvija u sklopu blockchain tima na Tehničkom fakultetu u Rijeci. Mehanizam ima većinu karakteristike PoW sustava, ali dio energije koja je utrošena za pogađanje hash vrijednosti bila bi utrošena na korisne zadatke i time poboljšala energetska učinkovitost mehanizma. Zadatak se potom prijavljuje od strane čvora ili krajnjeg korisnika na decentraliziranoj aplikaciji. Primljeni zadatak aplikacija dijeli na manje dijelove, to jest na podzadatke. Podzadaci se dijele čvorovima ovisno o njihovoj računalnoj snazi i kompleksnosti samog zadatka. Bitno je napomenuti da su rudari proporcionalno nagrađeni uloženom poslu te se time smanjuje natjecanje između rudara, odnosno solvera.

Decentralizirana aplikacija bit će opisana kasnije u radu, a njena implementacija omogućava implementaciju SPOW mehanizma. Aplikacija služi kao mjesto za registraciju korisnika i zadatka, prijavu resursa za rješavanje, omogućuje kooperativno izvođenje takvih zadataka te provjeravanje rješenja.

1.3.4. BFT

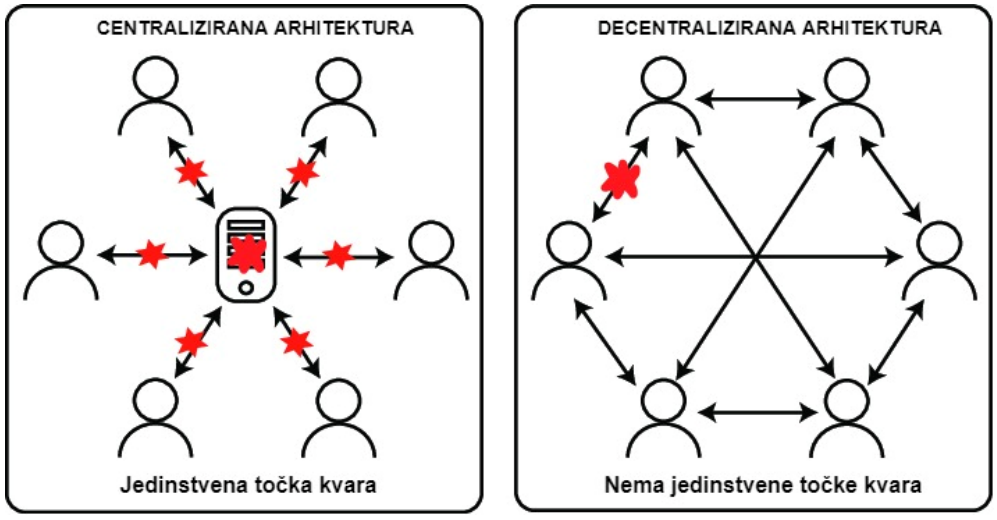
U potpuno pouzdanim, kontroliranim okruženjima gdje se neispravni procesi strogo prate i brzo ispravljaju, postizanje konsenzusa je jednostavno. Međutim, decentralizirani protokoli u stvarnom svijetu podložni su znatno većem broju proizvoljnih kvarova, zahtijevajući puno veća jamstva pouzdanosti kako bi bili korisni. Jedan od glavnih problema blockchaina je sposobnost da nastavi s radom čak i ako neki od čvorova zakažu ili djeluju zlonamjerno. Ovo predstavlja koncept problema bizantinskih generala (eng. Byzantine Generals' problem).[10]

Svaki konsenzusni dogovor koji se na kraju postigne mišljenje je većine. Protokol konsenzusa na kojem funkcionira blockchain sustav nije iznimka. Kod mehanizama baziranih na glasanju (eng. voting-based) konačna odluka o postizanju konsenzusa donosi se brojanjem glasova čvorova. Ova vrsta mehanizma ne primjenjuje se samostalno već u kombinaciji s nekim drugim jer bi mehanizmom poput PoS, budući da bi sustav baziran samo na glasanju, bio neefikasan.

Svojstvo blokchaina da tolerira kvarove i zlonamjernost čvorova koji sudjeluju u konsenzusu, poznata je kao tolerancija bizantinskih grešaka (eng. Byzantine Failure Tolerance) ili kraće BFT. Blockchain tehnologija samo je formlizacija BFT-a u modernijem okruženju, s naglaskom na peer-to-peer umrežavanje i kriptografsku autentifikaciju. U praksi, blockchain struktura podataka zapravo optimizira BFT dizajn. Byzantine Fault Tolerance (BFT) mehanizmi za postizanje konsenzusa omogućuju ispravan rad mreže sve dok je broj bizantinskih čvorova (čvorovi podložni kvaru ili zlonamjerni čvorovi) manji od jedne trećine.

1.4. Decentralizirane aplikacije (DAPP)

Centralizirana aplikacija u vlasništvu je jedne tvrtke te se nalazi na jednom ili više poslužitelja kojima upravlja tvrtka. Svi upiti i pohrana podataka vrše se na jednom centraliziranom poslužitelju. Ovo omogućava razvojnom programeru potpunu kontrolu nad aplikacijom i njenim korištenjem. Također, centralizirane aplikacije vrlo se lako mogu nadograđivati i ažurirati. Decentralizirana aplikacija radi na blockchainu ili peer-to-peer mreži. Blockchain tehnologija pruža aplikaciji veću sigurnost pohranjivanjem podataka na svaki čvor decentralizirane mreže umjesto na jedan centralizirani poslužitelj. Ovime se eliminira problem jedinstvene točke pogreške (eng. single point of failure) odnosno ako zakaže jedan dio sustava, on neće zaustaviti rad ostatka sustava kao kod centraliziranih aplikacija kako je prikazano na slici 1.3, te sustav čini otpornijim na curenje podataka i hakiranje.



Slika 1.3 Usporedba centraliziranog i decentraliziranog sustava u slučaju kvara

Nadalje blockchain omogućuje korisnicima aplikacije da šalju transakcije izravno jedni drugima bez oslanjanja na središnje tijelo. U decentraliziranim aplikacijama korisnici ne moraju slati svoje osobne podatke kako bi koristili funkciju koju aplikacija pruža. Budući da se osjetljive

podatke ne pohranjuje, hakeri im nemaju načina pristupiti. Korisnici mogu komunicirati s decentraliziranom aplikacijom putem pametnih ugovora[12] bez potrebe da otkrivaju svoj identitet, a automatizacija transakcija između anonimnih korisnika čini protokol sigurnijim od krađe identiteta i hakiranja. Budući da korisnik pristupa podacima s više čvorova u mreži, od svakog čvora preuzima dio podataka što povećava propusnost mreže za razliku od centraliziranog poslužitelja koji je limitiran određenom propusnošću preuzimanja.

Decentralizirana aplikacija ima četiri svojstva:

1. Nema prekida rada - mreža uvijek može služiti potrebama klijenata koji žele komunicirati s aplikacijom. Također osigurava da zlonamjerni akteri ne mogu pokrenuti napade uskraćivanja usluge na aplikaciju budući da se radi o *peer-to-peer* distribuiranoj mreži te će nastaviti funkcionirati čak i ako dijelovi mrežne arhitekture ne rade.
2. Otpor cenzuri – blockchain ne dopušta kontrolu nad podacima i procesima jednom entitetu. Iz tog razloga je nemoguće da bilo koji entitet blokira korisnike u podnošenju transakcija ili implementaciji aplikacije, pa čak i čitanju podataka iz blockchaina. Bez ikakvog određenog pojedinca ili organizacije koja kontrolira decentraliziranu aplikaciju, korisnici imaju prednost potpune slobode.
3. Privatnost – decentralizirana aplikacija ne zahtijeva stvarni identitet osobe pa za razliku od tradicionalnih aplikacija ne može zloupotrebiti osobne podatke.
4. Integritet – pomoću kriptografije osigurava se sigurna pohrana podataka na blockchainu te dostupnost provjere transakcije na javnom blockchainu pruža pouzdanost u zapisima podataka.[12]

2. RAZVOJNO OKRUŽENJE I PRINCIP RADA APLIKACIJE

Kako bi se razumio način rada aplikacije potrebno je razumjeti princip rada Tendermint blockchain protokola i ostalih tehnologija korištenih za razvoj aplikacije i njeno funkcioniranje. Stoga, u ovom poglavlju opisani su korišteni alati i tehnologije za razvoj decentralizirane aplikacije te je objašnjena povezanost decentralizirane aplikacije i Tendermint protokola.

2.1. Tendermint

Tendermint je blockchain protokol koji se koristi za repliciranje i pokretanje blockchain aplikacija na strojeve. Softver je otvorenog programskog koda koji omogućuje pisanje aplikacija u svim programskim jezicima. Nastao je kao zamisao arhitekta Jae Kwona i internetskog biofizičara Ethana Buchmana.[14] S Tendermintom možete neprimjetno stvoriti bilo koji blockchain sustav. Pomaže prevladati dugotrajnu fazu tehničkog postavljanja kako biste se mogli usredotočiti na samu aplikaciju. Ovo programerima omogućuje zaobilaženje implementacije blockchaina i preusmjerava njegov fokus na aplikacijski sloj. Tendermint može komunicirati s blockchainom unutar mreže i s vanjskim blockchainom. Ovo je moguće zahvaljujući troslojnoj arhitekturi Tendermint protokola:

- Aplikacijski sloj, koji obrađuje transakcije i održava stanje mreže.
- Mrežni ili komunikacijski sloj omogućava komunikaciju između aplikacijskog i konsenzusnog sloja
- Konsenzusni sloj određuje stanje mreže uspostavljanjem konsenzusa među čvorovima.

Kombinacija sva tri sloja pojednostavljuje razvojni proces. Ovakvom arhitekturom programske komponente su nezavisne i omogućuje se ažuriranje aplikacije bez bez izazivanja značajnih promjena u cijelom sustavu.

Tendermint se sastoji od dvije ključne komponente:

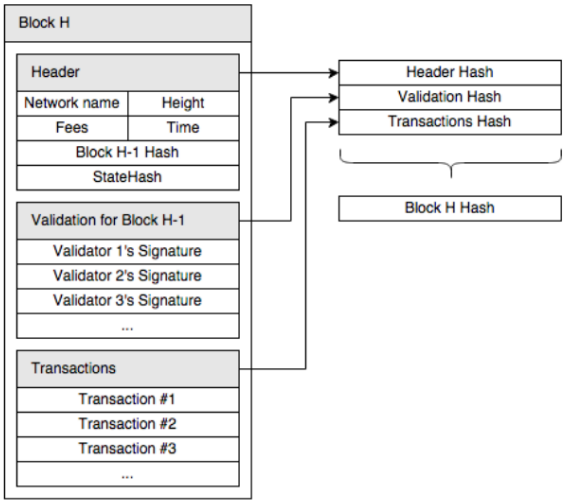
- Tendermint Core programskog okvira za postizanje konsenzusa koji služi kao konsenzusni sloj
- ABCI generičkog aplikacijskog sučelja, koji služi kao komunikacijski sloj

2.1.1 Čvorovi

Uz čvorove validatora, Tendermint konsenzus koristi tri glavne vrste čvorova:

- Puni čvor (eng. full node) odgovoran je za pohranu i distribuciju kopije cijelog blockchaina uključujući i stanje aplikacije i njene podatke.
- Stražarski čvorovi (eng. sentry nodes) povezuje se na jednog ili više validatora sa punim čvorom, budući da se na mreži ne koristi IP adresa, na validatoru je odgovornost da održava puni čvor otporan na greške. Pretpostavka je da će validator htjeti poduzeti sve mjere predostrožnosti kako bi održao toleranciju na greške, jer ako to ne učine, bit će izbačeni iz skupa validatora jer će se smatrati nedostupnim.
- Početni čvorovi (eng. seed node) pružaju popis ostalih čvorova putem *seeding* procesa na koje se čvor može povezati.[15]

Validatori su odgovorni za predlaganje novih blokova te emitiranju glasova u konsenzusnom mehanizmu. Pokretanjem blockchaina na mrežu, početni čvor šalje poruke s listom svih validatora koja se dalje ne mijenja. Razlog ovome je limitirani broj validatora na tendermintu. Zbog veličine strukture bloka Tenderminta prikazane na slici 2.1, postoje hardverska ograničenja zbog kojih brzina mreže pada s brojem validatora.[16]



Slika 2.1 Struktura bloka Tendermint blockchaina

Najbitnija je lista validatora sadrži manje od jedne trećine validatora koji su bizantinski. Ako se lista želi promijeniti ili implementirati, mehanizam kojim se nasumično odabire manja skupina validatora iz veće skupine i tako uvede veća decentralizacija sustava, dovoljno je označiti u protokolu specifičnu transakciju kojom čvor šalje poruku da želi postati validator. Predefinirana lista validatora nalazi se u json datoteci *gensis.json*. Budući da je potrebno imati tri stabilna validatora da bi konsenzus funkcionirao, u aplikaciji svaki validator ima svoj direktorij i svoj *gensis.json* kojim se veže na ostale validatore. Primjer liste validatora iz aplikacije prikazan je na primjeru 2.1.

```
"validators": [
  {
    "address": "CA8924E638EEED1C2EA5CF54052A13CC3ABB0040",
    "pub_key": {
      "type": "tendermint/PubKeyEd25519",
      "value": "3ozOUMmEIoclxWKnkXxrJRH0jaqSUknVoN//1gYq9oY="
    },
    "power": "100",
    "name": "node 1"
  },
  {
    "address": "D63C470C6F2779A01777DC14A4204845ED9121D1",
    "pub_key": {
      "type": "tendermint/PubKeyEd25519",
      "value": "gje+/5kyTmh4PmbW27yMoyqwkhdtVQwO7fTXy+roo00="
    },
    "power": "100",
    "name": "node 2"
  }
]
```

Primjer koda 2.1 Lista validatora koja se nalazi u genesis.json početnog čvora

2.1.2. Konsenzus

Tendermint je prva adaptacija konsenzusa Proof-of-Stake izvedena iz practical byzantine fault tolerant (PBFT) algoritma koji su predstavili Castro i Liskov 1999. godine.[18] Protokol se bazira na glasanju, a pokreće se za svaku novu visinu blockchaina, kako bi se odlučilo o idućem bloku. Svaki krug protokola sastoji se od tri koraka:

1. Predlaganje bloka
2. Prevote[1]
3. Precommit[1]

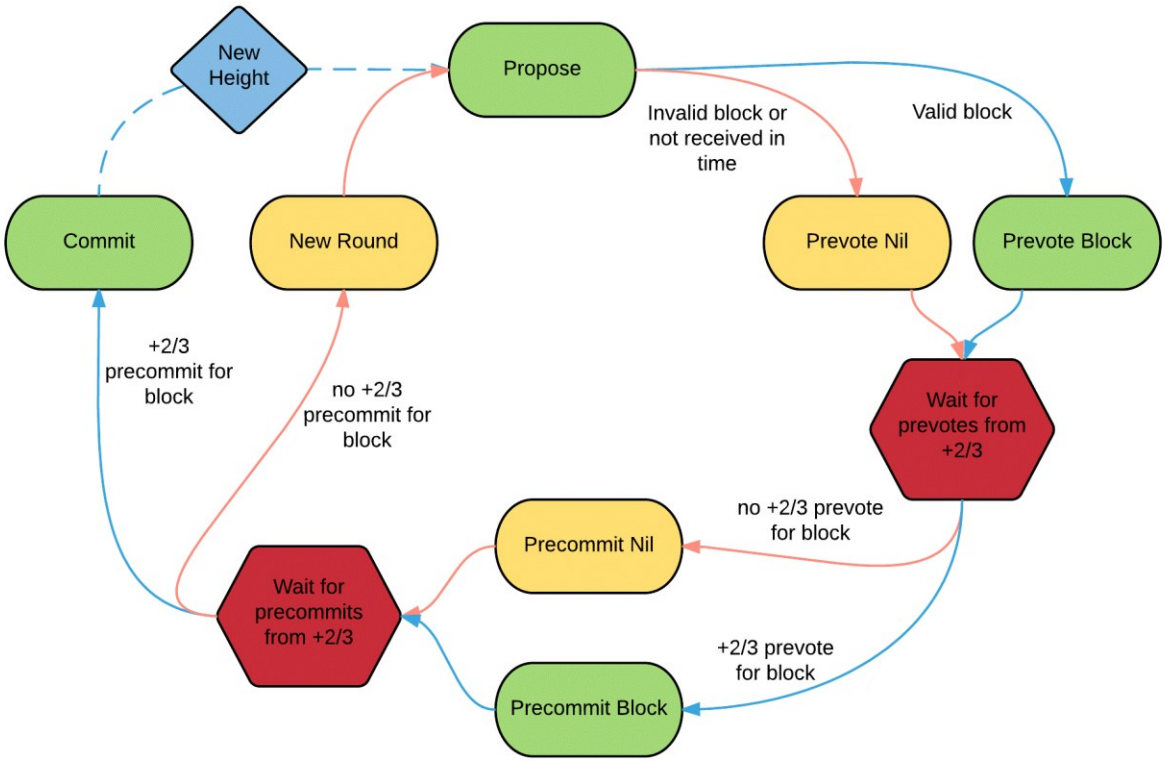
Tijekom prevote i precommit koraka postoji glasanje, svaki korak mora prikupiti više od dvije trećine glasova kako bi se postigao konsenzus. Budući da se glasa za svaki blok nikada neće postojati situacija u kojoj se dva različita bloka mogu stvoriti istovremeno, a svi blokovi nastali naknadno, a imaju istu visinu, odbacuju se. Ovim pristupom postizanja konsenzusa nikada neće doći do račvanja (eng. fork) lanca, odnosno nikad neće postojati konkurentski lanac onom glavnom. Međutim, ako se konsenzus nikada ne postigne, zbog asinkrone mreže, gdje nije zajamčeno da će poruke stići na vrijeme, blok se možda nikada neće ni stvoriti. Stoga Tendermint svojim konsenzusom jamči sigurnost, ali ne i dostupnost.

[1] Zbog nedostatka odgovarajućeg prijevoda korišten je engleski izraz

Predlaganje bloka odabir je validatora koji će generirati blok. Predlagatelj se bira na principu round-robin algoritma proporcionalno vrijednosti težine validatora kojeg razvojni programer sam bira (obično kriptovaluta). Aplikacija koristi jednaku težinu vrijednosti validatora, a budući da se radi o testnoj mreži, nema opasnosti od napada.

U prevote fazi svaki validator daje svoju procjenu bloka, a u najjednostavnijem slučaju šalje se poruka o suglasnosti za blok.

Nakon određenog vremena predviđenog za prevote korak (pet sekundi), svaki validator provjerava koliko je prevote poruka primio od drugih čvorova. Ako je broj veći od dvije trećine ukupnog broja validatora, validator šalje Precommit transakciju. Konsezus koristi automat stanja (eng. state machine) prikazan na slici 2.2.



Slika 2.2 Automat stanja za postizanje konsenzusa na Tendermintu[15]

Protokol naizmjenice odabire predlagatelj iz liste validatora koji predlaže blok transakcija. Ostali validatori provjeravaju ispravnost bloka, te na temelju toga glasaju o prihvaćanju bloka. Ako je predlagatelj bizantin, protokol ponovo bira predlagatelj te poveća dužinu trajanja za prevote i precommit korak budući da postoji mogućnost da je mreža spora.

Za svaki blok potrebne su dva kruga glasanja: prevote i precommit. Čvor ne dopušta prevote glasanje za drugčije blokove iste visine. Kada validator skupi dvije trećine pozitivnih prevote glasova, platforma poziva nad svim validatorima koji su pozitivno glasali proof-of-lock-change (dokaz o promjeni) ili PoLC. Kada čvor vidi PoLC kreće treći korak. Ako je dvije trećine čvorova pozitivno glasalo za precommit, blok se dodaje na blockchain. PoLC mehanizmom garantirano je da će protokol sigurno raditi, odnosno da validator nikad neće potvrditi kofliktan blok na istu visinu iako je jedna trećina validatora bizantinskih.[19]

Prije potvrde svakog bloka validatori čekaju određeno vrijeme da predlagatelj napravi blok i proslijedi mrežom. Ovo zadržavanje čini konsenzus sinkronim iako je ostatak protokola asinkron.

Mehanizam zaključavanja jamči sigurnost bez ikakvih vremenskih pretpostavki, što je nužan uvjet u sustavu velikih razmjera kada su kašnjenja u komunikaciji nepredvidljiva. Unatoč tome, algoritam mora koristiti vremensko odbrojavanje svakog koraka kako bi osigurao živost i izbjegao nemogućnost postizanja konsenzusa s jednim neispravnim procesom

Na aplikacijskom sloju može se definirati valuta ili neka druga težina prema kojoj Tendermint denominira glasačku moć. Validatori se kroz aplikaciju mogu vezati za valutu pomoću uloga i tako ih se može kazniti ako se utvrdi da se zlonamjerno ponašaju kroz konsenzus. Ovo je moguće budući da Tendermint replicira proizvoljnu aplikaciju.

2.2. Tendermint Core

Tendermint Core je blockchain platforma koja nudi programski okvir (framework) za postizanje konsenzusa. Radi se o djelomično asinkronoj mreži koja koristi kombinaciju BFT i PoS mehanizama za postizanje konsenzusa, pruža ekvivalent web-poslužitelja, baze podataka i pratećih knjižica za blockchain aplikacije napisane u bilo kojem programskom jeziku. Poput web poslužitelja koji poslužuje web aplikacije, Tendermint poslužuje blockchain aplikacije.[15]

2.3. Tendermint Application blockchain interface (ABCI)

Tendermint odvaja konsenzusni protokol i distribuiranu mrežu od detalja stanja aplikacije. Ovo je izvedeno apstrahiranjem dijelova koda aplikacije u sučelje te njegove primarne implementacije Tendermint Socket Protocol (TSP) koja služi za prijenos podataka. Ovo sučelje zove se ABCI i omogućuje komunikaciju aplikacije i Tendermint Core programskog okvira.

Ukoliko se ABCI aplikacija pokreće u istom procesu kao i Tendermint Core, Tendermint poziva ABCI aplikacijsku metodu direktno kao poziv na metode.

Postoji nekoliko metoda ovisno na kojim mehanizam rukuju:

- Metode za konsenzus - *InitChain, BeginBlock, DeliverTx, EndBlock,* i *Commit.*
- Metode za mempool - *CheckTx.*
- Metode za upite - *Info* i *Query.*
- Metode za snimke stanja - *istSnapshots*, *LoadSnapshotChunk*, *OfferSnapshot*, i *ApplySnapshotChunk.*[20]

DeliverTx Tendermint isporučuje aplikaciji transakcije koje je potrebno izvršiti. Kada Tendermint primi *ResponseDeliverTx* poruku s kodom greške, transakcija je već bila uključena u blok, tako da kod ne utječe na konsenzus Tenderminta. Aplikacija treba provjeriti i potvrditi ispravnost svake transakcije primljene ovom porukom. Potvrđena transakcija zatim ažurira stanje aplikacije.

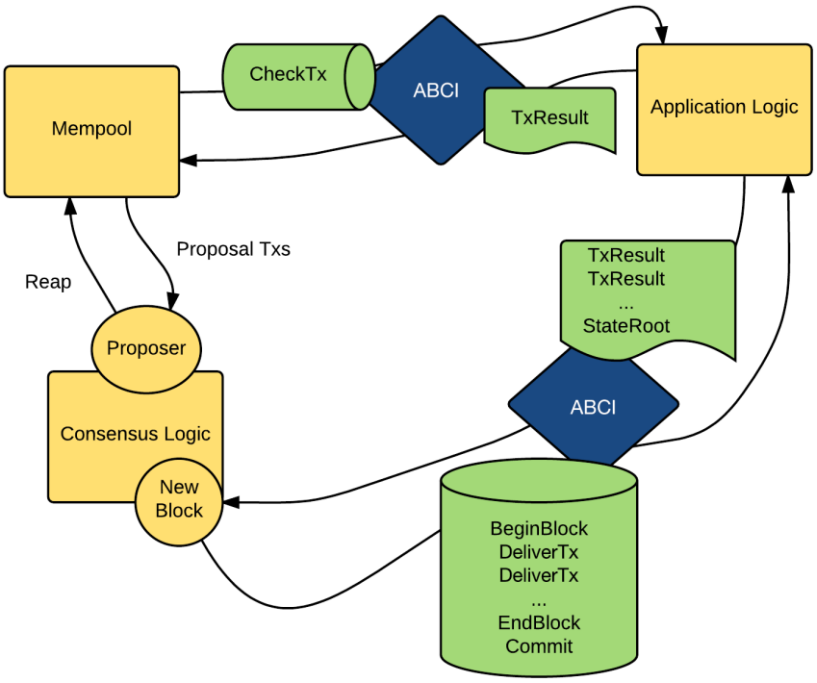
CheckTx služi za provjeru valjanosti transakcije koje se razmatraju za dodavanje u mempool. Kada Tendermint primi *ResponseCheckTx* s kodom greške, povezana transakcija neće biti dodana u mempool ili će biti uklonjena ako je već uključena.

Commit poruka računa kriptografsku vrijednost trenutnom stanju aplikacije koja se zatim postavlja u zaglavlje idućeg bloka.

Query metoda šalje upit aplikaciji za određenom informacijom o stanju aplikacije.

Kada se prvi put pokreće aplikacija, tendermint poziva *InitChain*. Zatim se za svaki blok poziva *BeginBlock,* a za svaku transakciju unutar bloka *DeliverTx* nakon obrade transakcija poziva se *EndBlock* i *Commit.* Ove metode se koriste za provjeru stanja aplikacije ili neku drugu logiku nakon promjene stanja aplikacije.

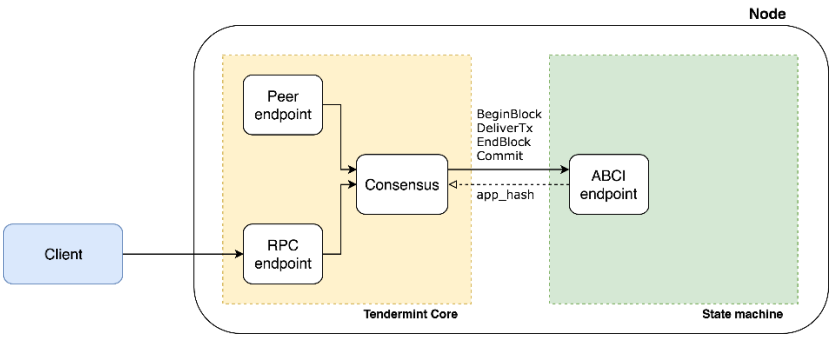
Kako se šalju poruke i slijed vidljivo je na slici 2.3. Prvo za svaku transakciju koja dolazi na mrežu Tendermint preko ABCI sučelja poziva *CheckTx* metodu kojom aplikacija prima transakciju i provjerava valjanost te vraća rezultat kroz *TxResult*. Kada konsenzus odabere predlagatelja, on sastavlja blok iz transakcija na mempoolu te šalje BeginBlock. Potom za svaku transakciju šalje *DeliverTx* ovom metodom aplikacija obrađuje transkaciju (prijenos kriptovalute, izvođenje nekog zadatka i slično). Nakon obrade transakcije na Tendermint Core se šalje poruka *TxResult*. Nakon obrade svih transakcija stanje aplikacije se sprema na blockchain.



Slika 2.3 Slijed slanja poruka između aplikacija i Tendermint komponenti[15]

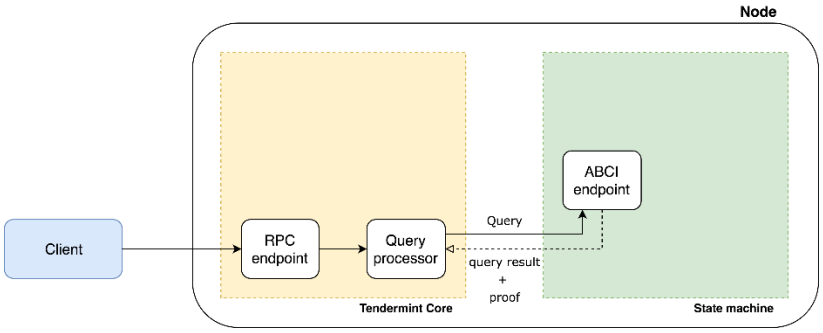
2.4. RPC poslužitelj

Za izvođenje operacija, klijent se obično povezuje s jednim čvorom. Upitne zahtjeve kao što je get u potpunosti obrađuje jedan čvor. Stroj stanja jednog čvora ima sve potrebne podatke za obradu upita jer se stanje u potpunosti replicira. Blockchain je također u potpunosti repliciran i svaki blok sadrži potpise čvorova koji su ga prihvatili, tako da klijent može provjeriti da *app_hash* koji vraća čvor nije lažan. Obrada učinkovitih zahtjeva kao što je put zahtijeva više čvorova da bi se postigao konsenzus, međutim klijent i dalje šalje transakciju jednom čvoru. Tendermint je odgovoran za širenje ove transakcije na druge čvorove i postizanje konsenzusa. Tipičan slijed zahtjeva vidljiv je na slici 3.1. Primjer get metode za blok određene visine je *"/block?height=1"* .



Slika 2.4 Slijed za get ili put upit

Krajnja točka za upite na RPC poslužitelju koja se odvaja od ostalih je *Query*. *Query* kao parametre ima traženi ključ i visinu. Na početku ih obrađuje Tendermint procesor upita (eng. Tendermint processor) koji ih preusmjerava na stanje stroja. Stroj stanja obrađuje upit tražeći ciljni ključ u stanju koje odgovara bloku specificirane visine. Ovaj postupak vidljiv je na slici 3.2. To znači da stroj stanja mora održavati nekoliko stanja (koja odgovaraju različitim visinama blokova) kako bi mogao služiti različitim upitima. Stoga *query* metoda ne zahtijeva Tendermint Core za svoje funkcioniranje. Primjer query metode je *"/abci_query?data="*.



Slika 2.5 Slijed za query upit

2.5. Programski jezik GO

Go ili Golang programski je jezik opće namjene i otvorenog koda, dizajniran za produktivnost i relativnu jednostavnost poput Pythona s brzinom C-a. Problema koje Go rješava su sporo vrijeme izrade, nekontrolirane ovisnosti, dupliciranje napora, poteškoće u pisanju automatskih alata i razvoj na više programskih jezika (eng. cross-language development).

Go koristi go rutine, odnosno lagane dretve (eng. thread) kojima upravlja Go tijekom izvođenja, što je korisno kod blockchain razvoja jer omogućuje pokretanje nekoliko dijelova programa asinkrono ili paralelno, što poboljšava propusnost i rukovanje paralelnim akcijama što je nužno za distribuirane sustave poput blockchaina.

Budući da su Tendermint Core i ABCI razvijeni u Go jeziku, najviše je pomoćnih knjižica razvijeno za njega. Dodatno korištene su knjižice poput:

- SHA256 ova knjižica omogućuje raspršivanje podataka i dio je crypto modula
- BadgerDB ugradiva je trajna i brza baza tipa ključ-vrijednost (KV). Zamišljeno je da bude učinkovita alternativa pohrani KV podataka koje se ne temelje na Go jeziku.

2.6. Docker

Docker je alat osmišljen kako bi programerima olakšao razvoj, isporuku i pokretanje aplikacija pomoću docker spremnika (eng. container). Spremnici omogućuju razvojnim programerima da pakiraju aplikaciju sa svim njezinim zahtjevima i konfiguracijama i drugim ovisnostima, te da je implementiraju kao jedan paket. Ovo omogućuje Tendermintu i ostalim blockchain tehnologijama pokretanje na svim računalima neovisno o operacijskom sustavu ili hardveru. Ideja iza korištenja dockera je napisati skriptu kojom bi se aplikacija pokretala i bez instaliranja Go jezika ili postavljanja Tendermint testne mreže. Također Tendermint nudi pokretanje lokalne testne mreže jednom naredbom dockera.[21]

3. RAZVOJ APLIKACIJE

Razvoj protokola na blockchainu može se razdvojiti na tri faze: testna, razvojna i glavna mreža. Glavna mreža je uvijek aktivna, javno dostupna verzija protokola. Testne i razvojne mreže pružaju alternativna okruženja koja oponašaju funkciju glavne mreže kako bi razvojnim programerima omogućili izgradnju i eksperimentiranje s projektima bez utjecaja na glavnu mrežu.

Kako je za razvoj aplikacije potrebno postaviti i koristiti vlastitu testnu mrežu, u nastavku je objašnjeno postavljanje Tendermint testne blockchain mreže i njena konfiguracija, a potom implementacijski detalji aplikacije.

3.1. Postavljanje Tendermint testne blockchain mreže

3.1.1. Instalacija

Za pokretanje testne blockchain mreže i decentralizirane aplikacije potrebno je instalirati programski jezik GO i Tendermint blockchain. Ovo je najlakše pomoću Linux paketa brew. Instrukcije su vidljive na primjeru 3.1.

```
// Instalacija go programskog jezika
brew update&& brew install golang
// ili putem go instalacijskog alata
https://go.dev/doc/install

// Instalacija tendermint blockchain tehnologije
brew install tendermint
// ili Pomoću make install iz github repozitorija
git clone https://github.com/tendermint/tendermint.git
cd tendermint
Make install
```

Primjer koda 3.1 Instalacija go programskog jezika i tenderminta

3.1.2. Konfiguriranje čvorova

Kako bi inicijalizirali validatora koristimo komandu *tendermint init*. Ovo će stvoriti novi privatni ključ (*priv_validator_key.json*) koja sadrži pridruženi javni ključ, genesis datoteku *genesis.json*, koja sadrži prikazanu listu početnih validatora koje se šalju kroz početni blok, parametre poput početne visine bloka, dužine trajanja prevote koraka konsenzusa i lista poznatih validatoru poznatih čvorova. Generira se i TOML datoteka kojom se konfiguriraju postavke Tendermint Corea poput IP adrese validatora i adrese i porta za RPC poslužitelj. Zadani direktorij za blockchain podatke je *~/.tendermint*. Ovo zaobilazimo tako da postavljanjem parametar okruženja *–tm-home ./lokalni-čvor* tako da kroz inicijalizaciju čvora presumjerimo podatke na direktorij

aplikacije gdje je testna mreža, prikazano u primjeru 3.2. Sada aplikacija može koristiti postavljenu konfiguraciju čvorova te se svi podatci o čvorovima i blockchainu spremaju na postavljenu testnu mrežu.

```
func init() {
        flag.StringVar(&homeDir, "tm-home", "", "Path to the tendermint
config directory (if empty, uses $HOME/.tendermint)")
}

func main() {
        fmt.Print("Loading tendermint home directory...")
        flag.Parse()
        if homeDir == "" {
                homeDir = os.ExpandEnv("$HOME/.tendermint")
        }
        println("loaded")
        //....
        config.SetRoot(homeDir)
        app := NewKVStoreApplication(db, homeDir, config, resource)
}
```

Primjer koda 3.2 Postavljanje parametre tm-home za pronalzak konfiguracije čvora

3.1.3. Lokalna Tendermint testna mreža

U projektu je postavljena mreža od dva validatora i jednog početnog (seed) čvora koji ih povezuje.

Početni čvor sadrži listu validatora prikazanu u prethodnom poglavlju. Pri prvom pokretanju blockchaina za svaku adresu validatora prosljeđuje se lista. Kada validator primi konfiguraciju adrese drugog validatora pokušava se povezati. Početni čvorovi stalno indeksiraju mrežu kako bi dobili informacije o stanju sudionika. U osnovi, posao početnih čvorova je samo prenijeti adrese svih sudionika; obično samo pri prvom pokretanju. Pokretanje početnog čvora vrši se nakon pokretanja svih validatora komandom *tendermint start --home ./tendermint-seed*.

Validatorski čvorovi primaju blokove i transakcije te ih obrađuju. Za pokretanje prvog validatora koristimo komandu *./poc-test -tm-home ./tendermint-validator-1*. Vidimo da za validatora definiramo parametar *-tm-home* koji pokazuje na direktorij validatora *./tendermint-validator-1* kako bi se uzela potrebna konfiguracija i spremali podaci.

Klijent ima zasebnu aplikaciju koja se pokrene s go run čime se otvara sučelje za slanje zadataka odnosno specijalnih transakcija na mrežu.

Za izradu novog validatora potrebno je promijeniti polja *_laddr* na željenu adresu gdje se pokreće čvor, adresu RPC poslužitelja te dodati potrebne informacije u listu validatora početnog

čvora. Parametar *addr_book_strict* mora imati vrijednost =*false*, inače će Tendermintova peer-to-peer knjižnica zabraniti povezivanje s peerovima s istom IP adresom.

Tendermint prema zadanim postavkama stvara prazne blokove, to jest, nije potrebno poslati transakciju kako bi se aplikacija pokrenula. Ovo se može konfigurirati u genesis.json tako što se postavi parametar *create-empty-blocks* na *false*. Također se može povećati ili smanjiti interval između stvaranja praznih blokova.

Resetiranje cijelog blockchaina moguće je s parametrom --*unsafe_reset_all.*

3.2. Implementacija ključ-vrijednost baze

Nakon postavljanja tenderminta i testne mreže, može se krenuti s implementacijom same aplikacije. Tendermint Core komunicira s aplikacijom putem Application BlockChain Interface (ABCI). Sve vrste poruka koje će se koristiti za komunikaciju definirane su u prethodnom poglavlju.

Ideja je da key-value store aplikacije omogući spremanje ključa i vrijednosti koje dobiva u transakciji putem RPC poslužitelja. Slanjem zahtjeva na *broadcast_tx_commit* transakcija će se propagirati na tendermint. Jedan takav zahtjev prikazan je na primjeru 3.3.

```
1 curl -s 'localhost:26657/broadcast_tx_commit?tx="name=satoshi"'
```

Primjer koda 3.3 Adresa na koju se šalje transakcija

Aplikacija će iz transakcije izvući ključ i vrijednost i spremiti u badger bazu podataka. Za primjer 2.1 sprema se riječ „name“ a za value riječ „satoshi“. Na ovaj način mogu se slati i podaci u JSON obliku. Kostur same aplikacije su poruke koje šaljemo na ABCI definirane u *app.go* prikazane na primjeru 3.4. Svi ključevi i vrijednosti šalju se kao byte tip podataka koji je su enkodirani metodom Base64.

```
package main

import (
        abcitypes "github.com/tendermint/tendermint/abci/types"
)

type KVStoreApplication struct {}

var _ abcitypes.Application = (*KVStoreApplication)(nil)

func NewKVStoreApplication() *KVStoreApplication {
        return &KVStoreApplication{}
}

func (KVStoreApplication) Info(req abcitypes.RequestInfo)
abcitypes.ResponseInfo {
        return abcitypes.ResponseInfo{}
}
func (KVStoreApplication) DeliverTx(req abcitypes.RequestDeliverTx)
abcitypes.ResponseDeliverTx {
        return abcitypes.ResponseDeliverTx{Code: 0}
}

func (KVStoreApplication) CheckTx(req abcitypes.RequestCheckTx)
abcitypes.ResponseCheckTx {
        return abcitypes.ResponseCheckTx{Code: 0}
}

func (KVStoreApplication) Commit() abcitypes.ResponseCommit {
        return abcitypes.ResponseCommit{}
}

func (KVStoreApplication) Query(req abcitypes.RequestQuery)
abcitypes.ResponseQuery {
        return abcitypes.ResponseQuery{Code: 0}
}

func (KVStoreApplication) InitChain(req abcitypes.RequestInitChain)
abcitypes.ResponseInitChain {
        return abcitypes.ResponseInitChain{}
}

func (KVStoreApplication) BeginBlock(req abcitypes.RequestBeginBlock)
abcitypes.ResponseBeginBlock {
        return abcitypes.ResponseBeginBlock{}
}

func (KVStoreApplication) EndBlock(req abcitypes.RequestEndBlock)
abcitypes.ResponseEndBlock {
        return abcitypes.ResponseEndBlock
```

Primjer koda 3.4 Kostur aplikacije je skup metoda koje se direktno pozivaju preko ABCI sučelja

Svaki put kad se transakcija doda na Tendermint Core šalje se CheckTx prema aplikaciji. Potrebno je provjeriti validaciju formata i potpisa transakcija. Provjeravamo postoji li ključ i

vrijednost za svaku transakciju prikazano na primjeru 3.5. Ako transakcija nema oblik {bytes}={bytes}, vraćamo grešku.

```
func (app *KVStoreApplication) validateTx(tx []byte) uint32 {
        parts := bytes.SplitN(tx, []byte("="), 2)

        // check that the transaction is not malformed
        if len(parts) != 2 || len(parts[0]) == 0 {
                return 1
        }
        return 0
}
```

Primjer koda 3.5 Provjera valjanosti transakcija koje dolaze na mempool

3.3. Registracija sudionika

Za svaki čvor definirana je *Peer* struktura prikazana na primjer 3.6.

```
type Peer struct {
        roles    []string
        ID       string
        username string
        resource int
}
```

Primjer koda 3.6 Struktura čvora

Polje *ID* svakom čvora dodjeljuje aplikacije, a *resource* i *roles* su nužna polja za registraciju čvorova. Registracija se vrši slanjem specijalne transakcije. U aplikaciji specijalna transakcija predstavlja par ključ i vrijednost. Ključ je JSON oblika i na sebi sadrži polje *type* za tip transakcije te polje *node* gdje se sprema javni ključ čvora. Polje type ispunjeno je *TypeRegistration* konstantom a *node* javnim ključem čvora. Za vrijednost se postavlja *Peer* struktura u JSON obliku. Navedeni postupak prikazan je u primjeru 3.7.

```
key.Type = TypeRegistration
nodeKey, _ := getNodeKeyIDFromDirectory(app.homeDir)
key.Node = string(nodeKey)
if req.Resources == nil {
        *value.Resources = DefaultResource
} else {
        *value.Resources = *req.Resources
}
if req.Username != nil {
        *value.Username = *req.Username
}

http.Get("http://" + addr + "/broadcast_tx_commit?tx=" +
   strconv.Quote(string(keyBytes)+"="+string(vlueBytes)))
```

Primjer koda 3.7 Slanje specijalne transakcije za registraciju

Nakon što se pošalje zahtjev, transakcija se nalazi u mempoolu. Kada Tendermint Core započne blok metodom beginBlock, te za obradu transakcije pošalje deliverTx prema aplikaciji koja prepozna da se radi o ključu i vrijednosti te dekodira polja. Dekodiranjem polja provjeri se polje *type* ključa i prepozna radi li se o specijalnoj transakciji za registraciju ili o prijedlogu zadatka. Potom se informacije o registraciji spreme u badger bazu. Ovo je vidljivo na primjeru 3.8.

```
        errKey := json.Unmarshal(key, &transactionKey)
        if errKey != nil {
                fmt.Println(errKey)
        }
        if transactionKey.Type == TypeRegistration {
                errValue := json.Unmarshal(value, &registrationValue)
                if errValue != nil {
                        fmt.Println(errValue)
                }
                if err := app.pendingBlock.Set(key, value); err != nil {
                        log.Panicf("Error reading database, unable to
verify tx: %v", err)
                }
                return abcitypes.ResponseDeliverTx{Code: 0}
        }
```

Primjer koda 3.8 Deokodiranje primljenih podataka i provjera tipa transakcije

Za prijavu korisnika potrebno je pretražiti bazu za postojeći ključ čvora, ovo je moguće za preko krajne točke RPC poslužitelja za upite na bazu: *abci_query*. Kod za prijavu korisnika prikazan je na primjeru 3.9.

```
    key.Type = TypeRegistration
    nodeKey, _ :=
    getNodeKeyIDFromDirectory(app.homeDir)
    key.Node = string(nodeKey)
    keyBytes, err := json.Marshal(key)
    if err != nil {
         fmt.Println(err.Error())
    }
    addr := getRPCAddress(app)
    resp, err := http.Get("http://" + addr +
    "/abci_query?data=" +
    strconv.Quote(string(keyBytes)))
    if err != nil {
         fmt.Println(err)
         fmt.Println(err.Error())
    }
```

Primjer koda 3.9 Provjera postoji li korisnik u bazi

3.4. Predlaganje i delegacija zadatka

Kako bi predložio zadatak, klijent se prethodno treba registrirati. Prijavom korisnika u aplikaciju nudi mu se opcija predlaganja zadatka čija je struktura vidljiva na primjeru 3.10.

```
type taskProposal struct {
        id              string //UUID STRING
        name            string
        description     string
        executablePath string
        input           Input
}
```

Primjer koda 3.10 Struktura prijedloga zadatka

Trenutno jedini podržani zadatak za aplikaciju je hashcash odnosno validator raspršivanjem jednokratnog niza s ostalim podacima bloka želi postići hash s određenim karakteristikama. Algoritam za ovaj zadatak vidljiv je na primjeru 3.11.

```
func calculateHash(app *KVStoreApplication) string {
        i := 0
        h := sha256.New()
        bestSol := hex.EncodeToString(currentBlockHash)
L:
        for i < 100*app.resource && app.nonce < ^uint32(0) {
                blockHash := hex.EncodeToString(currentBlockHash)
                nodeKey, _ := getNodeKeyIDFromDirectory(homeDir)
                app.nonce++
                nonce := strconv.Itoa(int(app.nonce))
                h.Write([]byte(blockHash + string(nodeKey) + nonce +
string(i)))
                hash := h.Sum(nil)
                hexString := fmt.Sprintf("%x", hash)
                first4Chars := hexString[0:4]
                if strings.CountPrefix(bestSol, '0') <
strings.CountPrefix(hexString, '0') bestSol = hexString
                if first4Chars == "0000" {
                        fmt.Println("**********************")
                        fmt.Println("CORRECT HASH!")
                        fmt.Println("**********************")
                        println(hexString)
                        currentBlockHash = []byte(hexString)
                        sendAwardTransaction(blockHash, string(nodeKey),
nonce, i, app)
                        return hexString
                }
        }
        i++
        if i < 100 { goto L }
        else sendAwardTransaction(bestSol, string(nodeKey), nonce,
string(i), app)
}
```

Primjer koda 3.11 Rješenje za hashcash zadatak

Bitna karakteristika aplikacije je rudarenje svih čvorova odnosno delegiranje dijelova zadatka svim čvorovima koji zatim rješavaju manji dio zadatka koji je njima pripao. Za primjer hashcash zadatka, čvorovi ovisno o dostupim resursima izvršavaju izračun hash vrijednosti onoliko puta koliko im je aplikacija zadala. Radi jednostavnosti dokaza koncepta trenutno su sve vrijednosti konstante pa tako točan hash mora započeti s četiri nule.

Slično specijalnoj transakciji registracije na isti način aplikacija prepoznaje da se radi o specijalnoj transakciji za prijedlog zadatka, samo što je za polje *type* ključa vrijednost *TypeHashCashTask*.

3.6. Nagrađivanje i spremanje rješenja zadatka

Kada čvor pogodi točnu vrijednost hasha poziva se metoda *sendAwardTransaction*, koja šalje dvije specijalne transakcije koje se dodaju na idući blok.

Prva specijalna transakcija je spremanje rješenja zadatka u bazu gdje se za tip ključa koristi *TaskID,* a za *node* polje, javni ključ čvora. Validatori u idućem bloku prepoznaju na temelju ključa o kojem se zadatku radi te o kojem specifičnom čvoru, budući da svi čvorovi dobivaju nagradu proporcionalno njihovim resursima odnosno količini rada. Ako neki čvor ne pogodi željene karakteristike hasha, šalje se njegov najbolji pokušaj (hash koji je počinjao s najviše nula).

Druga specijalna transakcija je dodjela nagrade gdje validator šalje tip TypeCorrectHash i svoj javni ključ, a za vrijednost dobiveni hash i podatke čijim je raspršivanjem dobio navedeni hash. Validatori za ovaj tip transakcije pomoću dobivenih podataka provjeravaju vrijednost hasha. Kako je vidljivo na primjeru koda 3.12, ako je hash ispravan, aplikacija šalje događaj (eng. event). Događaji su objekti koji sadrže informacije o izvršavanju aplikacije. Uglavnom ih koriste pružatelji usluga poput block explorera i novčanika za praćenje izvršenja raznih poruka i indeksnih transakcija.

```
  blockHash := correctHashValue.Block
1 nodeKey := transactionKey.Node
2 nonce := correctHashValue.Nonce
3
4 hash := sha256.Sum256([]byte(blockHash + nodeKey + nonce))
5 hexString := fmt.Sprintf("%x", hash)
6 first4Chars := hexString[0:4]
7
8 if first4Chars == "0000" {events := []abcitypes.Event{
9    {Type: "tx", Attributes: []abcitypes.EventAttribute{
10        {Key: strconv.Itoa(transactionKey.Type), Value:
11 transactionKey.Node, Index: true},
12        {Key: "type", Value: strconv.Itoa(transactionKey.Type), Index:
13 false},
14        {Key: "node", Value: transactionKey.Node, Index: false},
15        {Key: "block", Value: correctHashValue.Block, Index: false},
16        {Key: "nonce", Value: correctHashValue.Nonce, Index: false},
          },},}
```

Primjer koda 3.12 Validacija i nagrađivanje

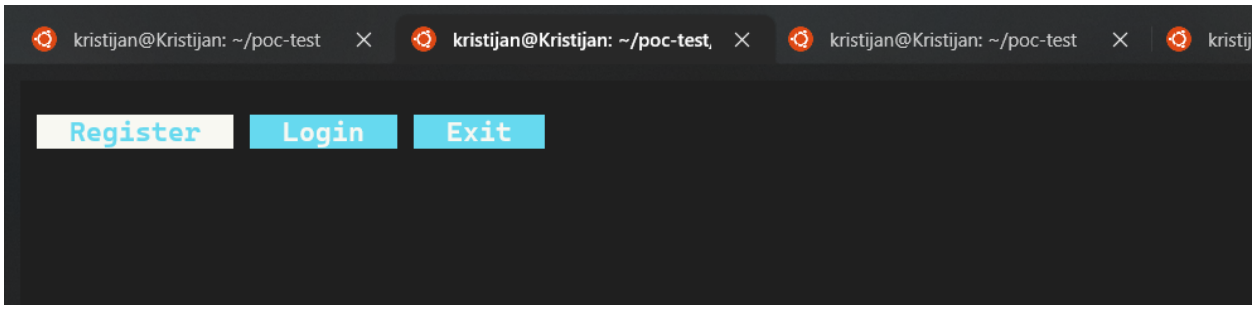
4. INTERAKTIVNO TEKSTUALNO KORISNIČKO SUČELJE

Budući da je Tendermint pisan u Go jeziku lako je pokrenuti aplikaciju u zajedničkom procesu sa Tendermintom, da se za aplikaciju koristio neki drugi jezik aplikacija bi bila zaseban proces. Interaktivno sučelje za aplikaciju pokreće se kao terminal te omogućuje korištenje miša i tipkovnice.

4.1. Predlagatelj

Predlagatelj u decentraliziranoj aplikaciji predstavlja čvor ili korisnika koji želi predložiti zadatak ili poslati transakciju. Predlagatelj se nalazi u odvojenoj datoteci *client.go* i pokreće se odvojeno od ostatka mreže pomoću *go run.*

Početni ekran za predlagatelje služi za slanje registracijske transakcije odnosno za prijavu. Prikaz ekrana može je na slici 4.1.

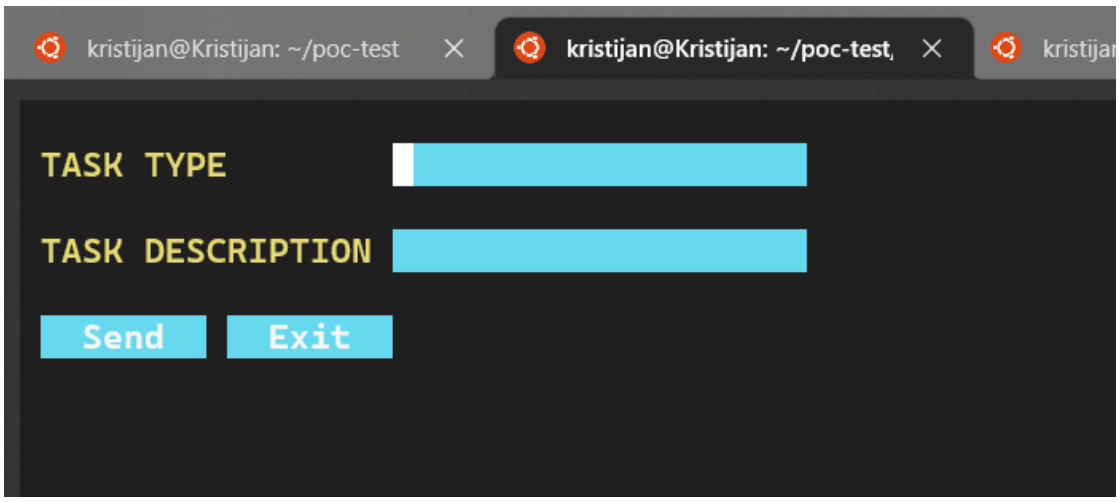


Slika 4.1 Početni ekran korisnika

Ekran za registraciju predlagatelja zahtijeva samo *username* polje koje je neobavezno, dovoljno je poslati transakciju pritiskom na send te će aplikacija dodijeliti korisniku ID. Prikaz ekrana prikazan je na slici 4.2.

Slika 4.2 Ekran za registraciju

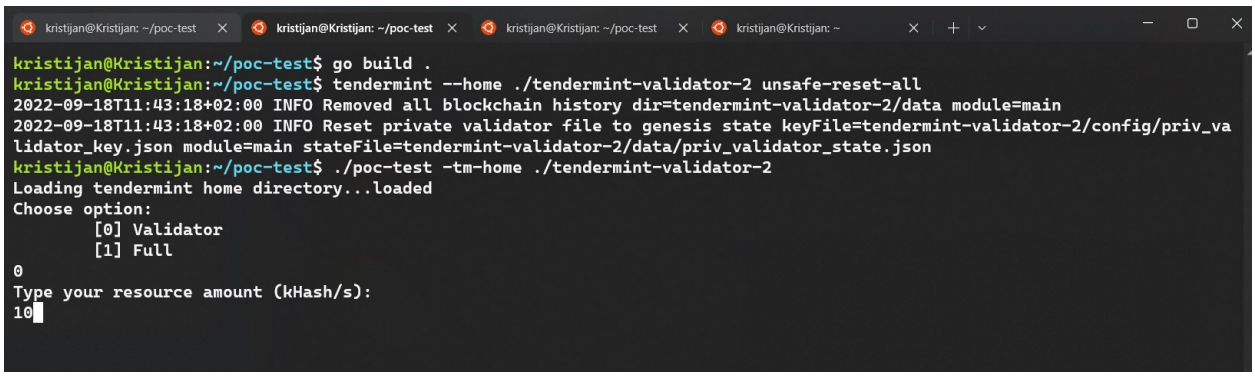
Ekran za prijavu vodi na ekran za predlaganje zadatka (slika 4.3); radi dokaza koncepta aplikacije trenutno se koristi samo hashcash kao tip zadatka. Ostali parametri su neobavezni i ne utječu na logiku aplikacije. Svaki zadatak na sebi ima ID koji aplikacija sama dodijeli.



Slika 4.3 Ekran za predlaganje zadatka

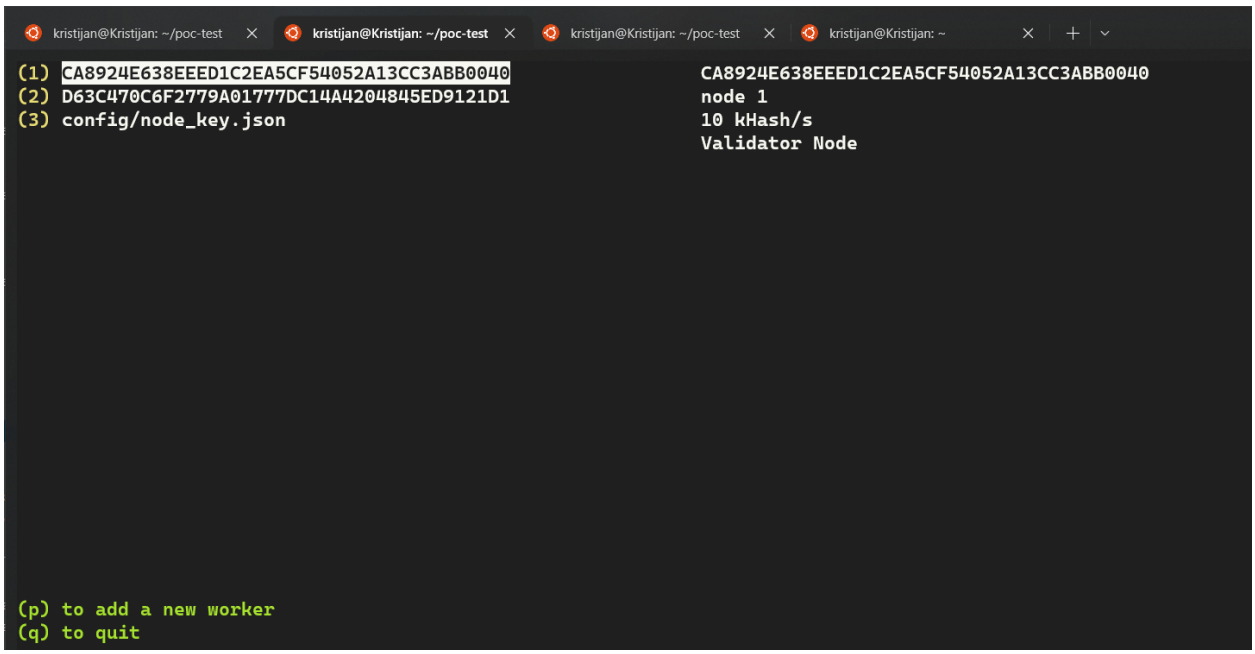
5.2. Solver

Solver predstavlja čvora koji koristi svoju računalnu snagu kako bi riješio određeni zadatak. Prema objašnjenim instrukcijama pokreće se validator. Aplikacija potom putem tekstualnog sučelja traži dodatne parametre za postavku čvora kako je prikazano na slici 4.4. Prvo traži ulogu čvora; radi li se o validatoru ili o punom čvoru. Za sad puni čvorovi nemaju veću ulogu u funkcioniranju aplikacije. Drugi parametar je računalna snaga čvora koja se koristi za izvršavanje zadatka.



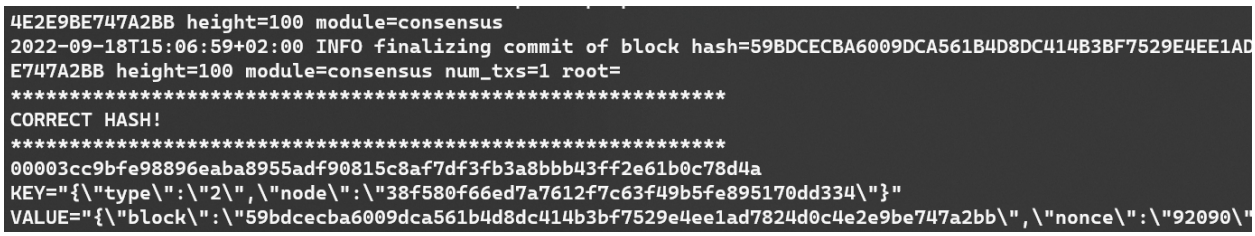
Slika 4.4 Pokretanje tendermint validatora

Nakon pokretanja čvora, aplikacija pokreće ekran za pregled čvorova prikazan na slici 4.5. Sučelje prikazuje sve validatore koji se nalaze u listi validatora početnog čvora. Za svakog čvora s desne strane vidljivi su njegovi parametri. Prvo je *ID* validatora, zatim ime pa računalna snaga, a posljednje su prikazani sve uloge (eng. roles) čvora. Ovaj ekran kasnije se može preraditi za prikaz svih radnika (eng. worker) koje određeni korisnik aplikacije pokreće.



Slika 4.5 Ekran za prikaz čvorova

Pritiskom tipke „q“ korisnik može izaći iz interaktivnog sučelja i ulazi u sučelje gdje se ispisuje tijek događaja aplikacije. Jedan primjer je kada čvor pogodi hash prikazan na slici 4.6.



Slika 4.6 Ispis aplikacije, pogođen traženi hash

5. ZAKLJUČAK

U ovom radu objašnjen je razvoj decentralizirane aplikacije na Tendermint testnoj blockchain mreži. Decentralizirana aplikacija omogućava implementaciju održivog mehanizma za postizanje konsenzusa. Osmišljeni održivi mehanizam za postizanje konsenzusa posjeduje većinu karakteristika klasičnog PoW mehanizma. Glavna razlika je u tome što se u klasičnom PoW mehanizmu troši velika količina električne energije samo na pogađanje hash vrijednosti, dok kod SPOW, dio te energije odvojio bi za izvođenje zadataka prijavljenih na decentraliziranoj aplikaciji Tako bi se uštedjelo na električnoj energiji, jer bi prijavljeni zadatak (za čije rješavanje bi se svakako potrošila električna energija) riješili u svrhu postizanja konsenzusa te tako zadatak riješio bez upotrebe dodatnih resursa.

Decentralizirana aplikacija ima mogućnost registracije korisnika, predlaganje zadataka, prijavljivanje vlastitih resursa za rješavanje predloženih zadataka, delegiranje zadataka i dijeljenje zadataka na manje dijelove, provjeravanje rješenja, te zapisivanje svih zadataka i rješenja u ključ-vrijednost bazu podataka. Implementirano je i interaktivno sučelje te sučelje za praćenje obavijesti aplikacije.

Mogući daljnji radovi na aplikaciji uključuju aplikacijsko programsko sučelje za učitavanje programskog koda ili izvršne datoteke za nove vrste zadataka. Za razliku od poslovne logike aplikacije, koja od programera zahtijeva razumijevanje Tenderminta, na strani klijenta može se uvesti web korisničko sučelje, budući da web preglednici putem ekstenzije kao što je MetaMask, omogućuju lakšu interakciju s blockchainom, poput potpisivanja poruka i slanja transakcija.

U realizaciji ovog projekta pružila mi se mogućnost upoznavanja i korištenja novih tehnologija i znanja o blockchainu. Stekao sam i zanimljivo iskustvo te poticaj za daljnje istraživanje i surađivanje u sklopu blockchain tima na Tehničkom fakultetu u Rijeci.

Literatura

[1] Gupta, M.; „IBM Blockchain for Dummies“, John Wile & Sons, New Jersey, 2017.

[2] Emurgo, „How blockchain solutions can improve data security for enterprises and users“, s Interneta https://emurgo.io/how-blockchain-solutions-can-improve-data-securityfor-enterprises-users-a-general-overview/, 7. srpnja 2022

[3] Nakamoto, S.; „Bitcoin: A Peer-to-Peer Electronic Cash system“, s Interneta https://bitcoin.org/bitcoin.pdf, 2008.

[4] Bashir, I.; “Mastering Blockchain: Deeper insights into decentralization, cryptography, Bitcoin, and popular Blockchain frameworks”, Packt Publishing, London, 2017.

[5] Bitcoin. Bitcoin Developer Guides, s Interneta https://developer.bitcoin.org/devguide/block_chain.html, 5.srpnja 2022.

[6] Vires, A.; „Bitcoin's Growing Energy Problem“, s Interneta https://www.cell.com/joule/fulltext/S2542-4351(18)30177-6?_, 6. srpnja 2022.

[7] Hinsdale, J.; „Cryptocurrency's Dirty Secret; Energy Consumption“, s Interneta https://news.climate.columbia.edu/2022/05/04/cryptocurrency-energy/#., 6. srpnja 2022.

[8] University of Cambridge, Cambridge Electricity Consumption Index, s Interneta https://ccaf.io/cbeci/index, 5. srpnja 2022.

[9] Buterin, V.; „Long-Range Attacks: The Serious Problem With Adaptive Proof of Work“, s Interneta https://ghoststaking.com/nothing-at-stake-long-range-attacks/, 15.srpnja 2022.

[10] River Financial, „what-is-the-byzantine-generals-problem“, s Interneta https://river.com/learn/what-is-the-byzantine-generals-problem/ , 10. rujna 2022.

[11] Interchain, „Understanding the Basics of a Proof-Of-Stake Security Model“, s Interneta https://blog.cosmos.network/understanding-the-basics-of-a-proof-of-stake-security-model-de3b3e160710, 9. rujna 2022.

[12] IBM, „Smart contracts“, s Interneta https://www.ibm.com/topics/smart-contracts, 20. rujna 2022.

[13] Anwar, H.; „What Is A Decentralized Application (DApp)“, s Interneta https://101blockchains.com/what-is-dapp/, 7. srpnja 2022.

[14] Kwon, J.; „Tendermint: Consensus without Mining“ , s Interneta https://tendermint.com/static/docs/tendermint.pdf 2014.

[15] Tendermint Core documentation, „What is Tendermint“ s Interneta https://docs.tendermint.com/v0.34/introduction/what-is-tendermint.html, 8. rujna 2022.

[16] Cosmos Forum, s Interneta https://forum.cosmos.network/t/performance-of-tendermint-in-case-of-large-number-of-nodes/3442, 15. rujna 2022.

[17] Alqahtani, S.; „A paper review: The design, architecture, and performance of the Tendermint Blockchain Network“ s Interneta https://salemal.medium.com/a-paper-review-the-design-architecture-and-performance-of-the-tendermint-blockchain-network-7402e0179bd4, 13. rujna 2022.

[18] Casto, M., Liskov, B.; „Practical Byzantine Fault Tolerance“ s Interneta, https://pmg.csail.mit.edu/papers/osdi99.pdf, 15. Rujna 2022.

[19] Amoussou-Guenou, Y.; „Correctness and fairness of tendermint-core blockchains“, s Interneta https://eprint.iacr.org/2018/574.pdf, 13. rujna 2022.

[20] Tendermint, Tendermint GitHub repostiory, s Interneta https://github.com/tendermint/tendermint/blob/v0.34.x/spec/abci/abci.md, 16. rujna 2022.

[21] Tendermint Core, Docker compose, s Interneta https://docs.tendermint.com/v0.34/networks/docker-compose.html, 16. rujna 2022.

Pojmovnik

P2P – Peer-to-Peer

POS – Proof-of-Stake

POW – Proof-of-Work

SPOW – Sustainable Proof-of-Work

DAPP – Decentralized application

BFT – Byzantine fault tolerance

PBFT – Practical byzantine fault tolerance

ABCI – Application blockchain interface

RPC – Remote procedure call

IP – Internet Protocol

API – Application Programming Interface

Sažetak

Decentralizirana aplikacija je aplikacija koja za svoje izvršavanje koristi blockchain ili drugi sustav distribuirane knjige. U ovom radu predstavljen je koncept decentralizirane aplikacije razvijene u Go programskom jeziku koja se izvršava na Tendermint blockchain protokolu. Koncept aplikacije predstavlja način rada održivog mehanizma za postizanje konsenzusa u blockchainu, zvanog Sunstainable Proof-of-Work koji se razvija u sklopu blockchain tima na Tehničkom Fakultetu u Rijeci. Osnovne funkcionalnosti aplikacije su registracija sudionika, predlaganje zadataka, delegacija zadatka ili dijelova zadatka sudionicima te provjera ispravnosti zadatka. Predstavljeni su mogući problemi, ograničenja i prijedlozi poboljšanja aplikacije.

Ključne riječi: blockchain, konsenzusni mehanizam, decentralizirana aplikacija, Tendermint

Abstract

A decentralized application is an application that runs on blockchain or other distributed ledger system. This thesis presents the concept of a decentralized application developed in the Go programming language which runs on the Tendermint blockchain protocol. The concept of the application represents the working method of a sustainable mechanism for achieving consensus in blockchain, called Sustainable Proof-of-Work, which is being developed as part of the blockchain team at the Technical Faculty in Rijeka. The basic functionalities of the application are registration of participants, proposing tasks, delegation of tasks or parts of the task to participants, and validation of tasks. Possible problems, limitations and suggestions for improving the application are presented.

Keywords: blockchain, consensus mechanism, decentralized application, Tendermint