Razvoj blockchain aplikacije koristeći TypeScript i Angular

Oremuš, Tihana

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Tihana Oremuš

RAZVOJ BLOCKCHAIN APLIKACIJE KORISTEĆI TYPESCRIPT I ANGULAR

Diplomski rad

Voditelj rada:
v. pred. dr. sc. Goran Igaly

Zagreb, veljača, 2022.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

Termin blockchain usko povezujemo s najpoznatijom kriptovalutom današnjice, a to je Bitcoin. Uzrok tome je što Bitcoin u pozadini koristi blockchain tehnologiju za spremanje i provođenje transakcija među korisnicima mreže. Jedan od razloga zašto su kriptovalute postale iznimno popularne u poslijednjih nekoliko godina jest decentralizacija, odnosno odsutnost posrednika koji nagledava i odobrava transakcije. Blockchain tehnologija zasniva se na ideji da se digitalni podaci razmjenjuju među svim čvorovima u određenom sustavu. Svaki čvor sadrži vlastitu kopiju podataka, te se na taj način gubi potreba za centralnim autoritetom.
Osim spomenute decentralizacije i transparentnosti, blockchain ima karakteristike nepromjenjivosti i ireverzibilnosti. Svaki podatak zapisan u blockchain ostaje ondje zauvijek, te ne postoji način da ga se izmijeni. Takvo nešto je moguće zbog strukture blockchaina i načina na koji se podaci u njemu spremaju. Struktura blockchaina jednostavno se može opisati kao lanac blokova. Riječ je o podatkovnim blokovima koji su kriptografijom povezani u jednosmjerni lanac u kojem svaki novi blok zavisi o vrijednosti prethodnog bloka. Zbog toga, ako netko poželi promijeniti vrijednost jednog bloka, morat će promijeniti sve blokove koji su mu prethodili, ali i sve one koji slijede nakon njega. Taj postupak je gotovo nemoguć jer za dodavanje jednog bloka u blokchain korisnik, odnosno *rudar*, mora napraviti takozvani *Proof of work*, tj. mora izvršiti određeni algoritam te na taj način potrošiti neku količinu resursa i vremena. A za velike brojeve blokova u lancu, kao u primjeru Bitcoin mreže, takav pothvat nije izvediv. Time smo opisali jednu od najbitnijih značajki blockchaina, što je sigurnost.

Cilj ovog rada je implementirati i prikazati aplikaciju koja demonstrira rad jednostavne Bitcoin mreže. Aplikacija je implementirana korištenjem razvojnog okvira Angular, pa će se dio rada fokusirati i na osnove Angulara, odnosno bit će opisana njegova arhitektura i osnovni principi na kojima se bazira.

Poglavlje 1

Blockchain tehnologija

1.1 Povijest

Ideja koja stoji iza blockchain tehnologije opisana je već 1991. godine, kada su znanstvenici Stuart Haber i W. Scott Stornetta predstavili računalno praktično rješenje za vremensko žigosanje digitalnih dokumenata kako ih se ne bi moglo krivotvoriti. Sustav je koristio kriptografski osiguran lanac blokova za pohranjivanje vremenski označenih dokumenata, a 1992. godine ugrađeno je u dizajn Merkleovo stablo, što je učinilo cijeli proces učinkovitijim dopuštajući prikupljanje nekoliko dokumenata u jedan blok.

Ipak, blockchain kakvog poznajemo danas, opisan je i definiran 2008. godine kada je jedna neidentificirana osoba pod pseudonimom Satoshi Nakamoto objavila web stranicu zvanu "Bitcoin: A Peer-to-Peer Electronic Cash System", gdje je taj sistem opisan kao ravnopravna verzija elektroničkog plaćanja koja bi omogućila izravno slanje online gotovine određenom primatelju bez prolaska kroz financijsku instituciju. Blockchain pruža alternativu klasičnom sustavu te zaobilazi treću stranu, odnosno centraliziranog posrednika. U blockchainu, taj posrednik se zamjenjuje decentraliziranom mrežom anonimnih računala koja potvrđuju transakcije na bazi algoritma. Iza računala se nalazi bilo koji pojedinac, takozvani rudar (eng. *miner*), koji za potvrđivanje transakcije prima kao nagradu određenu svotu kriptovalute.

1.2 Kriptografija

Blockchain tehnologija primjenjuje kriptografiju za bitne koncepte poput povezivanje blokova u blockchainu, algoritme konsenzusa i digitalnih potpisa te će se u ovom poglavlju predstaviti osnovne kriptografske tehnike koje se koriste u blockhain tehnologiji.

Kriptografija je grana matematike koja ima veliku ulogu u računalnoj sigurnosti, a metode zaštite podataka na mreži poznate su pod nazivom kriptozaštita javnim ključem (eng.

Public-key cryptography). Kriptozaštita javnim ključem ispunjava sljedeće uvjete:

- **šifriranje/dešifriranje**:
  Omogućuje dvjema stranama da prikriju poslanu poruku. Pošiljatelj šifrira poruku i kao takvu je šalje do primatelja koji ju onda dešifrira. U prijenosu, dok je poruka šifrirana, potpuno je nečitljiva.
- **detekcija neovlaštenog mijenjanja podataka**:
  Omogućuje primatelju da utvrdi je li poruka promijenjena. Svaki pokušaj promjene ili zamjene poruke ili bilo kojeg njenog dijela bit će otkriven.
- **autorizacija**:
  Omogućuje primatelju da utvrdi identitet pošiljatelja.
- **neoporecivost poslanih podataka**:
  Sprječava pošiljatelja poruke da kasnije tvrdi kako nije poslao poruku.

Većina modernih kriptografskih metoda ne zasniva se na tajnosti kriptografskih algoritama, koji su dobro znani, već na broju koji se naziva ključ, a bez kojeg je vrlo teško, odnosno nemoguće dešifrirati poruku.

Razlikuju se tri vrste šifriranja:

- šifriranje simetričnim ključem (eng. *Symmetric-key encryption*)
- šifriranje javnim ključem (eng. *Public-key encryption*)
- jednosmjerno hash šifriranje (eng. *one-way hash encryption*)

Pri šifriranju simetričnim ključem ključ za šifriranje se može izračunati iz onog za dešifriranje i obratno. Većina primjena ipak koristi jedan jedinstveni ključ. Implementacije sa simetričnim ključem su vrlo učinkovite, a korisnici ne gube previše vremena dok se obavljaju šifriranje i dešifriranje. Nadalje omogućuje se i određeni stupanj autorizacije pošto poruka šifrirana s jednim simetričnim ključem ne može biti dešifrirana s drugim. Ova komunikacija je sigurna sve dok se ključ drži u tajnosti. Problem se javlja ako netko dozna za taj ključ. Na taj način on može ne samo čitati podatke već slati i svoje podatke objema stranama koje to ne bi mogle otkriti.

Šifriranje javnim ključem (često zvano i šifriranje asimetričnim ključem) uključuje par ključeva, jedan javni i virtualno poznat svima i jedan tajni poznat samo primatelju. Podatak šifriran s javnim ključem može se dešifrirati samo s tajnim ključem primatelja, a to znači

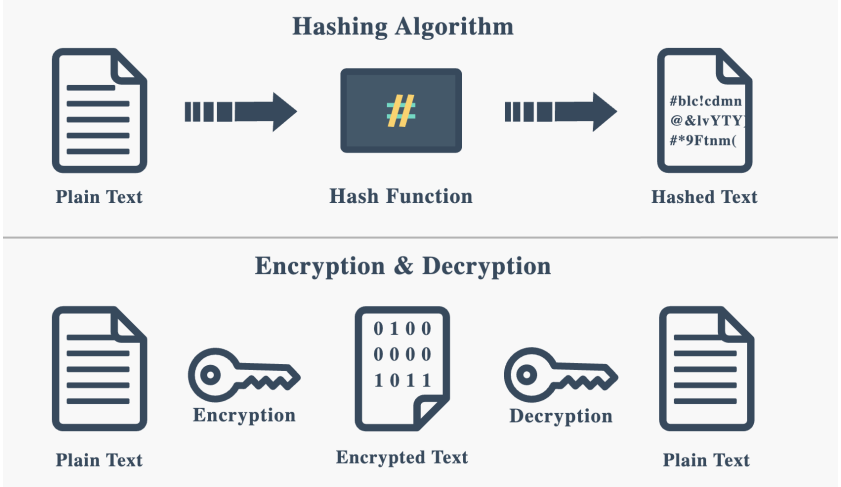
da slobodno može objavljivati svoj javni ključ. Treba napomenuti da obrat također vrijedi. Moguće je šifrirati podatke sa tajnim ključem koji se onda mogu dešifrirati s javnim ključem. To naravno nije preporučljivo jer svatko tko zna vaš javni ključ, a koji je po definiciji objavljen može dešifrirati vašu poruku. Ipak, takvo šifriranje je korisno jer to znači da je moguće iskoristiti tajni ključ za utvrđivanje identiteta pošiljatelja. Ova se metoda koristi zajedno s još nekim parametrima za digitalni potpis u transakcijama kriptovaluta.

Slika 1.1: Šifriranje simetričnim i javnim ključem

Hash funkcija

Hash funkcija je funkcija koja uzima ulaznu vrijednost i pretvara ju u izlaznu vrijednost unaprijed određenu ulaznom vrijednošću. Odnosno, kada bi se hash funkcija pokrenula s bilo kojom, ali istom vrijednošću nekoliko puta, rezultat bi svaki put bio isti. Funkcija kao ulaz može uzeti bilo koji podatak (string, broj, datoteku itd.), a ispisuje hash (obično se prikazuje kao heksadecimalni broj). Postoje razni algoritmi hashiranja, a za potrebe blockchaina koristi se SHA-256 (*Secure Hashing Algorithm*) koji kao izlaz vraća 256-bitni hash.

Prva važna karakteristika hash funkcije je da promjena ulazne vrijednosti, čak i brisanje ili promjena jednog znaka rezultira potpuno drugačijom vrijednošću. Druga karakteristika je da se iz hasha ne može doznati ulazna vrijednost, zbog čega se i zove jednosmjerna funkcija.

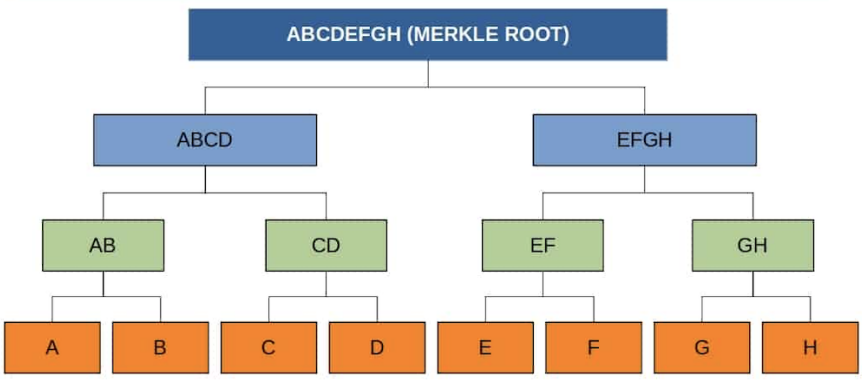


Slika 1.2: Hash funkcija i šifriranje/dešifriranje

Merkleovo stablo

Treba spomenuti još jedan koncept koji ima veliku ulogu u blockchain tehnologiji, a to je Merkleovo stablo. Merkleovo stablo, poznato još i kao binarno hash stablo, je struktura podataka koja se često koristi u računalnoj znanosti. To je zapravo način organizacije podataka koji omogućuje brže verificiranje informacija za velike skupove podataka.

Pomoću Merkleovog stabla, veliki broj transakcija u nekom bloku se lakše i brže verificira od strane svih sudionika mreže. Tu igra ulogu Merkleov korijen, odnosno jednostavna matematička metoda kojom se učinkovitije mogu zapisati i potvrditi podaci na cijelom stablu.



Slika 1.3: Merkleovo stablo

Merkleov korijen se dobiva na sljedeći način: podaci koje je potrebno kriptirati razbijaju se u blokove, ti blokovi se zatim hashiraju i dobiveni hashevi predstavljaju listove stabla. Iduća se razina stabla dobiva konkateniranjem parova listova te njihovim hashiranjem. Postupak se ponavlja sve dok na kraju ne ostane samo jedan čvor, a to je Merkleov korijen.

Detaljniji opis primjene Merkleovog stabla u blockchain tehnologiji slijedi u idućim poglavljima.

1.3 Vrste blockchaina

Postoje tri osnovne vrste blockhcaina, ovisno o ograničenju pristupa podacima. Razlikujemo javni, privatni i konzorcijski blockchain.

Slika 1.4: Javni, privatni i konzorcijski blockchain

Javni blockchain

U javnom blockchainu, blokovi su slobodno dostupni javnosti (eng. *open source*). Dopuštaju svakome da sudjeluje kao korisnik, rudar, programer ili član zajednice. Sve transakcije koje se odvijaju na javnim blokovima su potpuno transparentne, što znači da svatko može pregledati pojedinosti transakcije. Ova vrsta blockchaina omogućila je i nagradu (ekonomske poticaje) za one koji koriste neki od algoritama vrste *proof-of-stake* ili *proof-of-work*.

Privatni blockchain

Privatni blockchaini, također poznati kao upravljani, su blockchaini koje posjeduje jedna organizacija. U privatnom blockchainu, središnji autoritet određuje tko smije biti sudionik u mreži. Središnji autoritet također ne dodjeljuje svakom sudioniku nužno jednaka prava

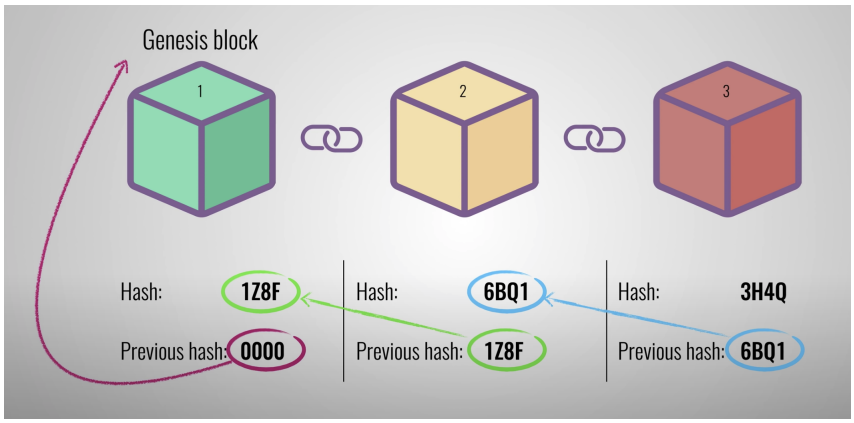
za obavljanje svih funkcija. Privatni blockchaini su samo djelomično decentralizirani pošto je javni pristup informacijama na blockchainu ograničen.

Konzorcijski blockchain

Konzorcijskim blockchainima upravlja grupa organizacija, a ne samo jedan entitet kao u slučaju privatnog blockchaina. Stoga konzorcijski blockchaini posjeduju veću razinu decentralizacije od privatnih, što rezultira u većoj razini sigurnosti. Doduše, uspostavljanje konzorcijskih blockchaina može biti vrlo naporan proces jer zahtijeva suradnju između brojnih organizacija.

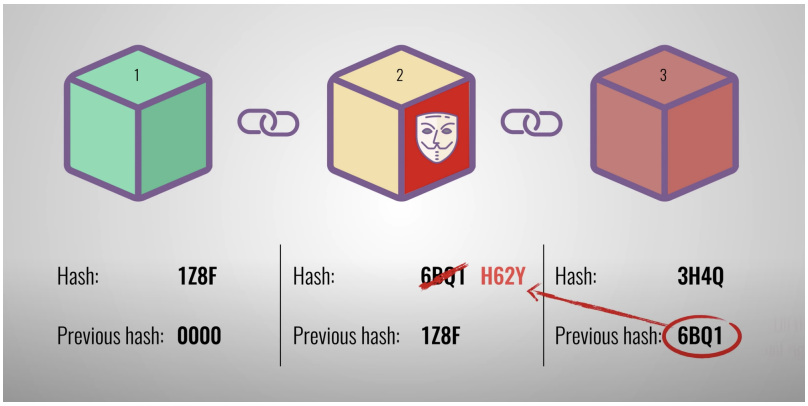
1.4 Struktura bloka

Kao što je rečeno u uvodu, blockchain je struktura blokova koji sadrže podatke, povezanih u lanac. Svaki blok predstavlja strukturu podataka unutar koje su zapisane informacije koje se dijele putem blockchaina. Tip informacija koje se spremaju u blok ovisi o tipu blockchaina. Primjerice, bitcoin sprema detalje transakcije između dva korisnika, pa bi informacije na bloku bile adresa pošiljatelja, adresa primatelja i iznos koji se treba poslati.



Slika 1.5: Struktura lanca

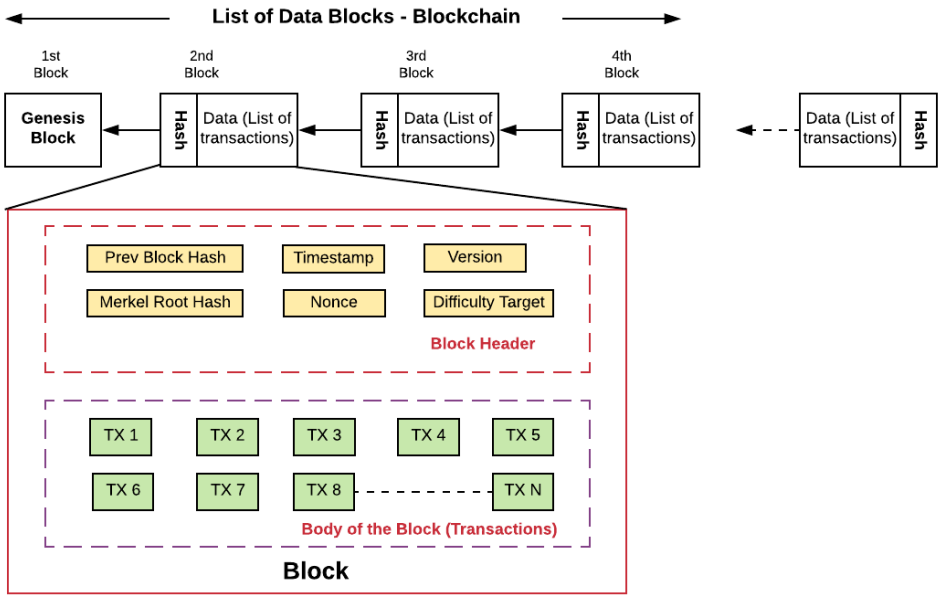
Tri najbitnija atributa koje sadrži jedan blok su hash tog bloka, hash bloka koji mu prethodi i podaci. Prvi hash ne sadrži hash prethodnog bloka i njega zovemo izvorni blok (eng. *genesis block*). Time blokovi zapravo čine lanac, svaki blok pokazuje na prethodni pomoću njegovog hasha. Ovdje hash bloka, zbog svoje jedinstvenosti, djeluje kao otisak prsta. Svaki izmjenjeni podatak na bloku rezultirat će promjenom njegovog hasha, pa blok koji slijedi nakon njega više neće imati valjani hash tog bloka.



Slika 1.6: Izmijenjen podatak u bloku

Zaglavlje bloka

U načinu na koji se blokovi povezuju u lanac veliku ulogu ima zaglavlje bloka. Zaglavlje bloka je veličine 80 bajtova, a sadrži metapodatke o samom bloku kao i informacije potrebne za povezivanje blokova u lanac. Metapodaci su zapravo podaci koji pružaju informacije o drugim podacima. Zaglavlje bloka sastoji od tri skupa metapodataka.

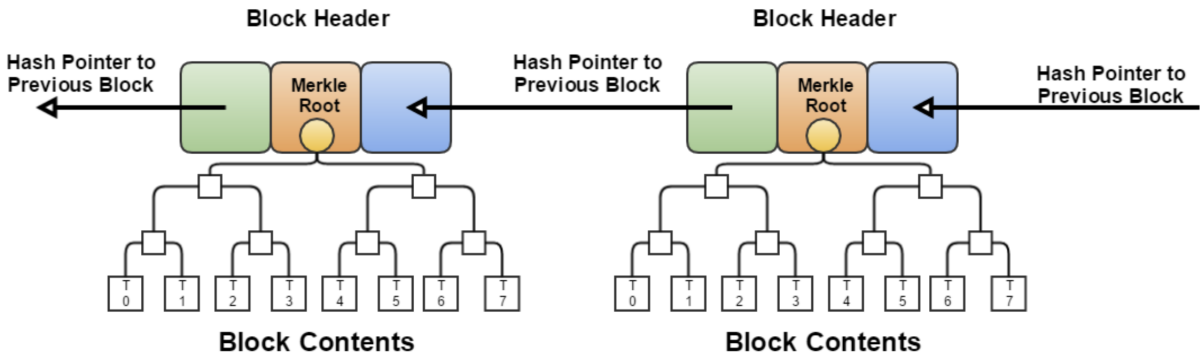


Slika 1.7: Struktura jednog bloka

Prvi skup metapodataka se sastoji od reference na hash prethodnog bloka u lancu koji služi za povezivanje postojećeg bloka s prethodnim.

Drugi skup metapodataka odnosi se na rudarenje, a uključuje broj verzije (eng. *version*), vremensku oznaku (eng. *timestamp*), težinsku oznaku (eng. *difficulty target*), te broj *nonce*. Broj verzije koristan je radi praćenja promjena i ažuriranja u cijelom protokolu. Vremenska oznaka uključena je tako da svi sudionici mogu vidjeti trajni, kodirani zapis o točnom vremenu kada se određena transakcija provela. Težinska oznaka određuje koliko će rudarima biti teško (koliko vremena i resursa moraju potrošiti) da stvore novi blok. *Nonce* je broj kojeg rudari mijenjaju i pritom stvaraju nove permutacije hasha da bi došli do zadovoljavajućeg hasha i generirali novi blok.

Posljednji, odnosno treći skup metapodataka sadrži ranije spomenut Merkleov korijen. On predstavlja sve transakcije tog bloka hashirane na način opisan u poglavlju 1.2



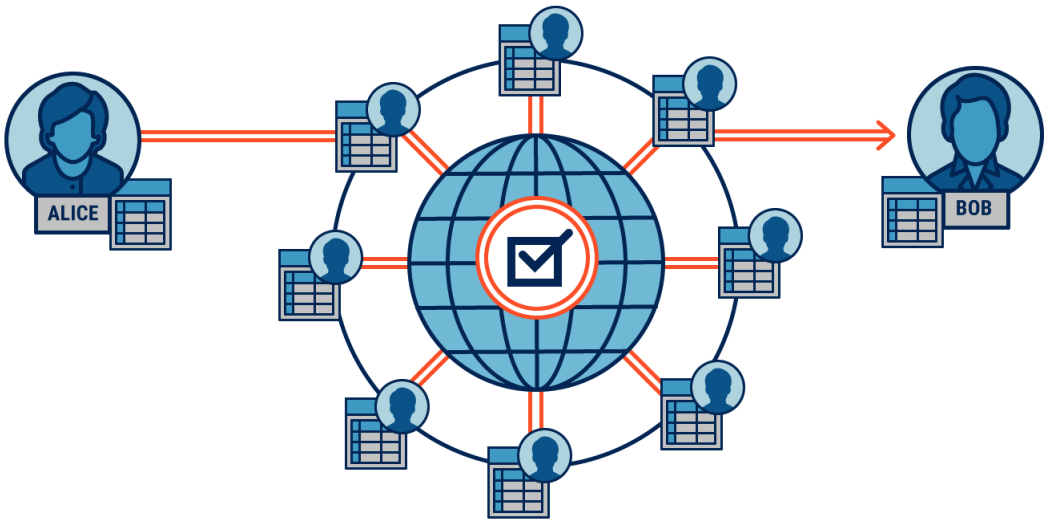
Slika 1.8: Merklovo stablo u blockchainu

1.5 Decentralizirani sustav

Sustav ravnopravnih partnera, odnosno sustav građen prema modelu ravnopravnih partnera (eng. *peer-to-peer*) sastoji se od velikog broja istovrsnih procesa, takozvanih partnera. Partneri obavljaju zadaće prema potrebama svojih korisnika. Ako je partneru pri obavljanju neke zadaće potrebna pomoć, on stupa u komunikaciju sa svojim susjedima, a ti susjedi sa svojim susjedima i tako se komunikacija odvija na razini cijelog sustava. Gore spomenuti sustavi mogu biti koncipirani na centralizirane i decentralizirane sustave.

Centralizirani sustavi su sustavi koji imaju jedno mjesto s kojeg se upravlja cijelim sustavom dok decentralizirani sustavi imaju mogućnost samostalnog rada kao podsustav unutar velikog sustava. U ovom slučaju to bi značilo da u centraliziranom sustavu postoji poslužitelj koji ima za funkciju povezivanja klijenata kako bi oni bili u vezi i kako bi mogli međusobno komunicirati.

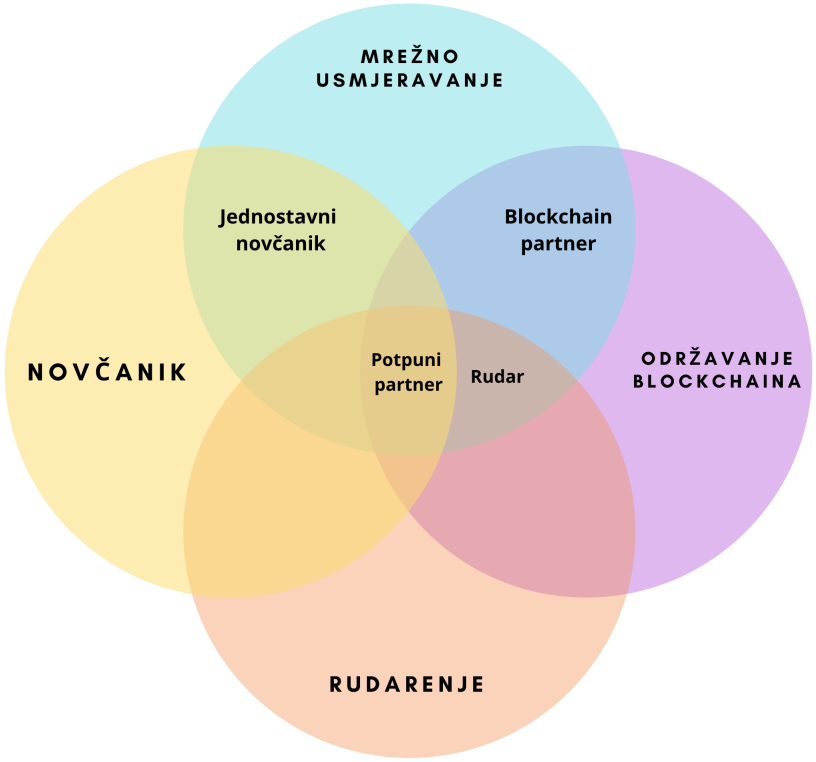
U decentraliziranom sustavu situacija drugačija, što znači da u takvom sustavu nema određenog, glavnog poslužitelja što ubrzava i pojednostavljuje komunikaciju.



Slika 1.9: Decentralizirani sustav

U privatnim blockchain sustavima, svaki od partnera će obavljati sve funkcije, dok ćemo u javnim blockchain sustavima prema funkcijama koje partneri obavljaju razlikovati i samu vrstu partnera. Partneri se dijele na rudare, potpune partnere, blockchain partnere te novčanike (eng. *wallets*). Sve vrste partnera obavljaju zadaću mrežnog usmjeravanja. Razlog tome je potreba svakog partnera za uspostavljanjem i održavanjem veza s nekim od ostalih partnera u modelu ravnopravnih partnera. Isto tako svaki od partnera koji sudjeluje u sustavu zadužen je za validiranje i difuziju (eng. *broadcast*) novih zapisa i novih blokova. U sljedećim potpoglavljima bit će objašnjene uloge blockchain partnera, jednostavnog novčanika i rudara. Potpuni partner može obavljati sve uloge ostalih partnera.

Koje konkretne funkcije ima pojedini partner, prikazano je na sljedećem dijagramu:



Slika 1.10: Uloge partnera

Jednostavni novčanik

U javnim sustavima koji koriste blockchain zbog velike količine podataka nema svaki korisnik mogućnost pohraniti čitav blockchain. Takav korisnik tada u sustavu sudjeluje kao jednostavan novčanik. Glavna zadaća koju jednostavni novčanik obavlja je kreiranje novih zapisa u skladu s protokolom koji propisuje sustav. U svrhu potvrde vlasništva nad digitalnim novcem ili nekom drugom digitalnom informacijom čiji integritet želimo zaštiti pohranjivanjem na blockchain, novčanici pohranjuju parove javnih i privatnih kriptografskih ključeva. Iz javnog ključa se generira adresa koja služi za primanje novca od ostalih korisnika, analogno broju bankovnog računa. Privatni ključevi su potrebni za pristup adresi i novčanim sredstvima. Njih možemo usporediti s osobnim identifikacijskim brojem (pinom) bankovnog računa i za razliku od adresa nije ih preporučljivo dijeliti s ostatkom sustava.

Jednostavni novčanik provodi pojednostavljen način verificiranja jer sam ne sadrži zapis o svim prethodnim transakcijama. Jednostavna metoda verificiranja podrazumijeva

pohranjivanje samo zaglavlja blokova umjesto cijelog blockchaina koji sadrži sve zapise. Budući da nemaju punu sliku svih transakcija koje su se dogodile prije one koju žele validirati, oslanjaju se na ostale partnere koji im na njihov zahtjev pružaju uvid u dio blockchaina za određenu transakciju.

Rudar

Partneri rudari preuzimaju nove transakcije koje su kreirali novčanici, formiraju ih u blokove i dodaju u blockchain. U Bitcoin protokolu dodavanje novih transakcija zahtijeva korištenje računalnih resursa. Kada rudar doda blok u blockchain, rješavajući algoritam pod nazivom *proof-of-work*, objašnjen u poglavlju 1.7, i koristeći svoje računalne resurse, tada sve transakcije u tom bloku postaju potvrđene, a rudar za nagradu dobiva određeni broj novih bitcoina.

Blockchain partner

Blockchain partner održava blockchain i sve njegove zapise, počevši od prvog, izvornog bloka nakon kojeg slijede svi ostali blokovi sve do zadnjeg. Za razliku od jednostavnog novčanika blockchain partner nema potrebe za oslanjanjem na ostale partnere u svrhu pretraživanja blockchaina ili provjere integriteta podataka. Ako je riječ o zapisu transakcija u blockchain, blockchain partner u svrhu validacije nove transakcije ima mogućnost provjeriti pripadaju li sredstva koja korisnik želi potrošiti u novoj transakciji zaista tom korisniku. To će napraviti na način da poveže novu transakciju sa svim prijašnjim transakcijama tog korisnika sve do generičkog bloka. Takav partner oslanja se na ostatak mreže samo kako bi u realnom vremenu primio novokreirane blokove koje nakon toga verificira i nadovezuje na svoju lokalnu kopiju blockchaina.

1.6 Rudarenje

U suštini, ruderenje je proces zarade neke svote kriptovaluta potvrđivanjem transakcija na javnom digitalnom zapisu transakcija koji zovemo blockchain, a postiže se primjenom računala te rješavanjem kriptografskih jednadžbi. U zamjenu za složene matematičke probleme koje rješavaju, rudari su nagrađeni kripovalutama.

Rudarenje se odvoja u nekoliko koraka:

1. **Čvorovi potvrđuju legitimitet transakcija**

   Primjerice, čvor A želi poslati neki iznos bitcoina čvoru B. No transakcija prvo mora proći proces verifikacije prije nego što bude izvršena.

2. **Svaka zasebna transakcija se dodaje u listu transakcija kako bi se stvorio jedan blok**

   Verifikacijom transakcija sprječava se dvostruka potrošnja, odnosno pojava u kojoj bi pošiljatelj potencijalno mogao slati novac koji ne posjeduje. Primjerice, čvor A na računu ima X bitcoina, te pošalje tih X bitcoina čvoru B. No kako transakcija još nije provedena, taj iznos je i dalje na računu čvora A, te on može napraviti još jednu transakciju i pokušati poslati X bitcoina čvoru C. Međutim, tijekom verifikcije transakcija, druga transakcija se neće validirati zbog prve transakcije u kojoj je čvor A potrošio cijeli iznos s računa. Na taj način se blockchain štiti od dvostruke potrošnje, a pritom uplate ostaju trajno zabilježene na nepromjenjivom lancu.

3. **Hash bloka i ostali podaci u zaglavlju se dodaju u nepotvrđeni blok**

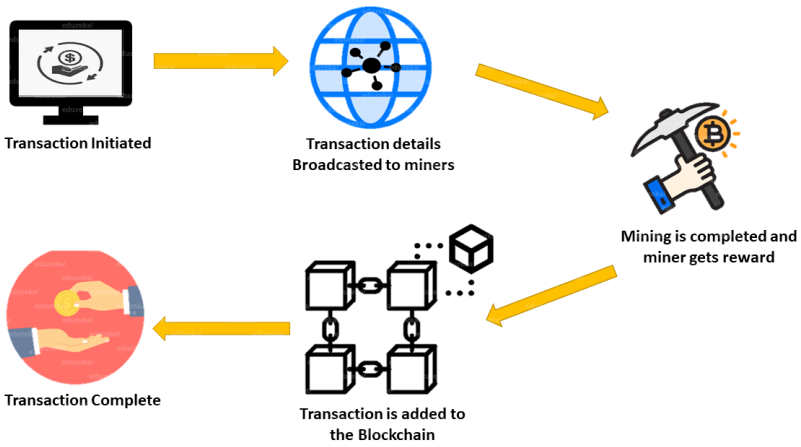
   U trenutku kada blok ima dovoljan broj transakcija, u njegovo zaglavlje dodaju se svi potrebni podaci da bi se on dodao u lanac, kao što je hash prethodnog bloka, vremenska oznaka, nonce itd. Hash se računa nakon što su svi podaci dodani te se dodaje u taj, još uvijek nepotvrđeni, blok.

4. **Rudari verificiraju hash novog bloka**

   U ovom koraku ostali rudari provjeravaju hash nepotvrđenog bloka kako bi se osigurao njegov integritet.

5. **Nakon što je potvrđen, blok se dodaje u blockchain**

   Ciklus je završen nakon što je hash riješen i drugim sudionicima je dokazano da je rješavanje učinjeno na autentičan način kojeg je moguće provjeriti.



Slika 1.11: Proces rudarenja

1.7 Algoritmi konsenzusa

Algoritmi konsenzusa su procesi donošenja odluka za grupu u kojem pojednici iz grupe konstruiraju i podrže odluku koja najbolje funkcionira za ostale u grupi. To je oblik rješenja u kojem pojedinci moraju poduprjeti odluku većine, svidjelo im se to ili ne.

Jednostavno rečeno to je samo metoda odlučivanja unutar grupe. Algoritmi konsenzusa ne slažu se samo s većinom glasova, već i s onom koja koristi svima. U konačnici je odluka uvijek pozitivna stvar za cijelu mrežu.

Modeli konsenzusa u blockchainu su metode za stvaranje jednakosti i pravde u online svijetu. Sustavi konsenzusa korišteni za takve sporazume nazivaju se Teorem o konsenzusu. Takvi modeli sastoje se od nekih specifičnih ciljeva, kao što su:

1. **Dogovaranje**

   Mehanizam prikuplja sve dogovore iz grupe koliko je u mogućnosti

2. **Suradnja**

   Svaki pojedinac iz grupe teži boljem dogovoru koji je u interesu grupe kao cjeline

3. **Timski rad**

   Svaki pojedinac će ostaviti svoje interese po strani za dobrobit grupe

4. **Jednakost**

   Svaki sudionik grupe ima jednaku vrijednost u glasovanju, odnosno glas svake osobe je bitan

5. **Sudjelovanje**

   Svi unutar mreže moraju sudjelovati u dogovoru, nitko ne može biti izostavljen i bez doprinosa

6. **Aktivnost**

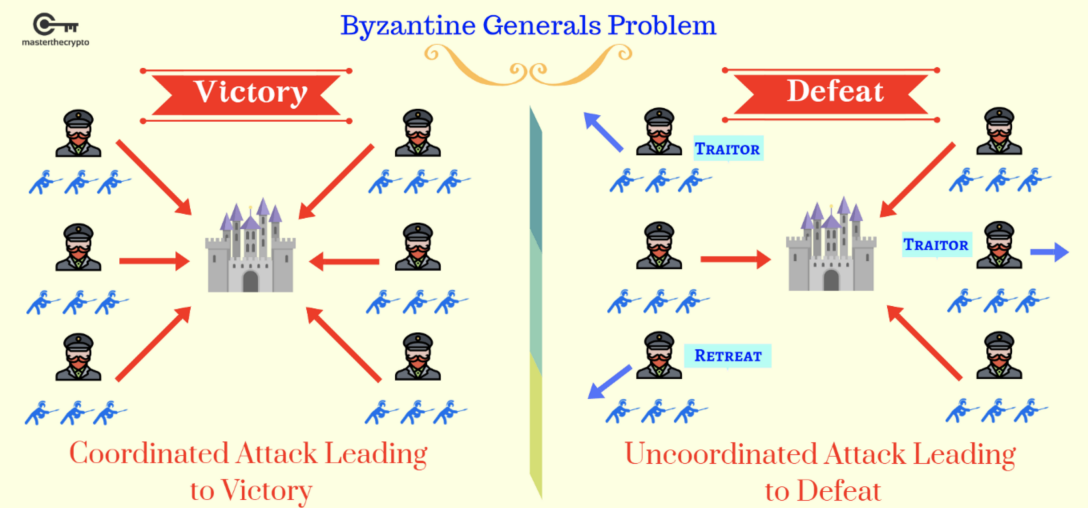
   Svaki član grupe je jednako aktivan, ne postoje pojedinci s više odgovornosti od ostalih

Problem bizantskih generala

Najvažnija stavka sigurnosti u blockchainu jest postizanje konsenzusa između sudionika ili partnera. Usuglašavanje oko stanja blockchaina spada u kategoriju problema koji je u računarstvu poznat pod nazivom problem usuglašavnja bizantskih generala.

Problem bizantskih generala možemo zamisliti kao situaciju u kojoj nekoliko divizija Bizantske vojske okružuje neprijateljski grad. Svaka divizija ima svog generala. Generali međusobno komuniciraju putem poruka i donose zajednički plan akcija o napadu ili

povlačenju. Ali neki od generala mogu biti nelojalni i pokušavati djelovati protiv dogovora lojalnih generala. Stoga generali moraju imati algoritam koji će im garantirati postizanje ispravnog konsenzusa. Prema istraživanju, potrebno je 3n + 1 generala da se obračunaju s n izdajnika. Dakle, trebalo bi četiri generala da se nose samo s jednim izdajnikom, što problem čini pomalo nezgodnim.



Slika 1.12: Problem bizantskih generala

Promatrajmo sustav koji koristi blockchain tehnologiju u terminima prije opisanog problema. Partneri u distribuiranom sustavu predstavljaju generale. Digitalni podaci koje želimo upisati u blockchain su poruke među generalima. Blockchain ima ulogu pohrane dogovorenog vremena napada. Pojedini partner ne zna broj ostalih partnera kao ni koji su od partnera izdajnici. Izdajnicima je u interesu upisati u blockchain podatke koji nisu istiniti. Algoritmi za postizanje konsenzusa omogućuju partnerima da u ovakvim uvjetima budu sigurni da su podaci koji se upisuju u nove blokove točni te da su podaci prije zapisani u blockchain istiniti i nepromijenjeni. Postoje razni tipovi algoritama konsenzusa, neki od njih su:

- *Proof-of-Work*
- *Proof-of-Stake*
- *Leased Proof-Of-Stake*
- *Practical Byzantine Fault Tolerance*
- *Proof-of-Activity*
- *Proof-of-Capacity*

U idućim poglavljima, obradit će se prva dva, ujedno i najčešće korištena algoritma.

Proof-of-Work

Dokaz o radu ili *Proof-of-work* algoritam je prvi blockchain algoritam uveden u blockchain mrežu. Mnoge blockchain tehnologije koriste ovaj model konsenzusa u svrhu potvrde transakcija i dodavanja relevantnih blokova u mrežni lanac. Iako decentralizirani sustav sam prikuplja sve informacije vezane uz blokove, potrebno je posebno obratiti pozornost na transakcijeske blokove. Ta odgovornost pada na partnere spomenute u poglavlju 1.5, odnosno rudare. Kao što je rečeno ranije, središnji princip rudarenja jest rješavanje složenih matematičkih problema i jednostavno davanje njihovih rješenja.

Takvi matematički problemi zahtijevaju veliku računalnu snagu. Primjerice, na koji način saznati izlaz hash funkcije ne znajući njen ulaz ili cjelobrojna faktorizacija. Složenost takvih problema uvelike ovisi o broju korisnika i ukupnom opterećenju mreže.

Međutim, ovaj algoritam ima svojih ograničenja. Kako mreža brzo raste, zahtijeva sve više resursa. U tom procesu se povećava sveukupna osjetljivost sustava. Proces konsenzusa u blockchainu uglavnom se osjanja na točnost informacija, no brzina sustava lako postaje nedostatna. Ako matematički problem postane prekompleksan, potrebno je mnogo vremena za generiranje novog bloka. Kada se izvršavanje transakcija uspori, cijeli tijek rada osjeti zastoj. Ako problem generiranja blokova ne može biti riješen unutar određenog vremena, onda se gubi smisao mreže. Međutim, ako problem postane prelako riješiv, tada će sustav postati sklon DDoS napadima, odnosno napadima uskraćivanjem usluga u kojem su izvori mrežnog prometa (napada) distribuirani na više mjesta diljem Interneta. Takav napad karakterizira namjerno generiranje velike količine mrežnog prometa da bi se zasitili mrežni resursi i poslužitelji. Zbog prevelikog opterećenja oni više nisu u stanju pružati namijenjene usluge.

Također, s vremenom veza unutar zajednice rudara postaje sve jača, te postoji opasnost od njihovog udruživanja. U tom slučaju bi moglo doći do prevelike centralizacije unutar decentralizirane mreže.

Proof-of-Stake

Dokaz o udjelu ili *Proof-of-stake* je algoritam konsenzusa koji se bavi glavnim nedostacima *Proof-of-work* algoritma. Svaki blok provjerava prije nego što se doda u mrežu, no u ovom algoritmu rudari se mogu pridružiti tom procesu koristeći određeni iznos svojih kriptovaluta kao ulog.

Dokaz udjela je nova vrsta koncepta u kojem svaki pojedinac može rudariti ili čak validirati nove blokove samo na temelju kriptovalute koje posjeduje. Dakle, u ovom algoritmu što više novaca ima neki korisnik, veće su mu šanse sudjelovanja.

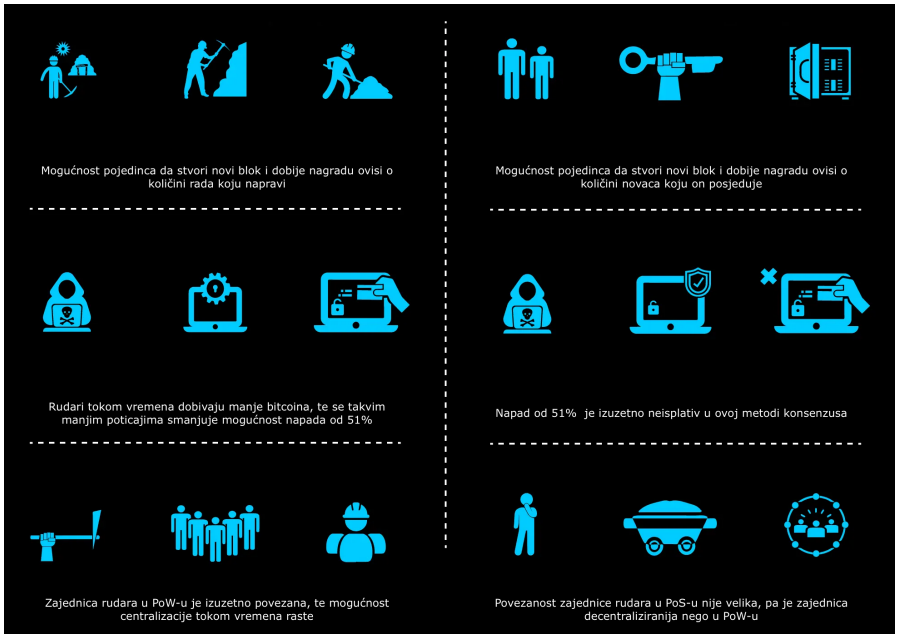
U ovom algoritmu, rudari se unaprijed izabiru. Iako je postupak odabira potpuno nasumičan, ne može svaki pojedinac sudjelovati u ulaganju. Svi sudionici mreže su birani na

nasumičan način, a ako pojedinac ima dovoljnu količinu novaca u svom novčaniku, on je kvalificiran da bi postao čvor u mreži. Nakon što je postao čvor, da bi postao kvalificiran za rudarenje, pojedinac mora dati određenu svotu kriptovalute kao polog. Zatim slijedi sustav glasanja u kojem će se odrediti validatori za blok.

Validator tada provjerava točnost transakcija i, ako odredi da su podaci točni, dodaje novi blok u blockchain te dobiva nagradu za svoj doprinos. No ako validator predloži dodavanje bloka s netočnim informacijama, gubi sav novac koji je uložio kao kaznu.

Već je rečeno da što manje novaca pojedinac uloži, to su mu manje šanse da bbude izabran kao validator. Iz tog razloga se sudionici pridružuju takozvanim platformama za ulaganje (eng. *staking pools*). Ako vlasnik platforme za ulaganje postane čvor u mreži, grupa ljudi može zajednički uložiti svoj novac kako bi imali bolju šansu postati validatorski čvor. Tada nagrada za stvaranje novog bloka biva podijeljena među sudionicima koji su uložili novac.

Ovaj tip algoritma konsenzusa ne zahtjeva nikakvu posebnu količinu hardverske opreme, potrebno je imati funkcionalan računalni sustav i stabilnu internetsku vezu. Svaka osoba s dovoljno novaca u novčaniku može sudjelovati u rudarenju, što je prednost pred dokazom o radu. Glavni nedostatak ovog algoritma jest što s njim potpuna decentralizacija još nije moguća. To je jednostavno zato što samo nekoliko čvorova može sudjelovati u ulaganjima. Tako pojedinci s najviše novaca u konačnici kontroliraju većinu sustava.



Slika 1.13: Proof-of-work vs Proof-of-stake

1.8 Primjene blockchaina

Prvobitno nastala kao podloga za potrebe kriptovalute Bitcoin, blockchain tehnologije danas imaju puno širu primjenu. Iako je trenutno najznačajnija za financijski sektor, blockchain tehnologija se može primijeniti u nizu brojnih sektora koji bi nam omogućili ne samo niže troškove i čekanje u raznoraznim redovima na šalterima, već i lakšu svakodnevicu.

Kriptovalute

Kao što je već spomenuto, Bitcoin je postao prva kriptovaluta ikada kada je objavljena 2009. Međutim, tek nekoliko godina kasnije, kada se stvorilo sve više i više kriptovaluta, ljudi su počeli trgovati, Ideja je vrlo jednostavna, trguje se jednom kriptovalutom za drugu, uz nadu da novčić koji pojedinac kupi poveća svoju vrijednost s vremenom. Koncept je isti kao na burzi. Kada ljudi trguju, trebaju koristiti razmjenu kriptovaluta. Tako se kupci i prodavači mogu podudariti. Primjerice, posjedujemo Bitcoin i želimo ga prodati za Ethereum, razmjena će nam pomoći naći prodavača Ethereuma s kojim možete trgovati. Burze će nam naplatiti naknadu za to, što obično košta oko 0.1% za svaku razmjenu. Trgovanje kriptovalutama sada je zaista popularno, jer se dnevno kupuju i prodaju u vrijednosti milijarde dolara.

Kriptovaluta je u osnovi digitalni način zadržavanja i prijenosa vrijednosti na mreži. To je internetski medij razmjene koji koristi kriptografske funkcije za obavljanje financijskih transakcija. Kriptovalute podupiru blockchain tehnologiju za postizanje decentralizacija, transparentnosti i nepromjenjivosti.

Dostupno je nekoliko desetaka različitih kriptovaluta, a najveće i najpoznatije su Bitcoin i Ethereum. Vrijednost bilo koje kriptovalute u bilo kojem trenutku ovisi o ponudi i potražnji. Obično je u bilo kojem trenutku dostupan fiksni iznos bilo koje valute, pa što je više ljudi želi iskoristiti, to je viša cijena. Primjerice, krajem 2017. godine, cijena jednog bitcoina porasla je otprilike 20 000 USD, a zatim je pala na cijenu od oko 4000 USD.

Odnos državnih institucija prema Bitcoinu varira od zemlje do zemlje. U Njemačkoj Bitcoin ima status privatnog novca, u Danskoj i Ujedinjenom Kraljevstvu se na trgovanje bitcoinom ne plaća porez, a u većini zemalja EU se bitcoin slobodno koristi. No, neke vlade su neprijateljski nastrojene prema bitcoinu. Rusija je zabranila korištenje bitcoina, a u Kini kupnja bitcoina podliježe određenim ograničenjima, primjerice, kineske banke ne smiju poslovati s bitcoin tvrtkama.

Pametni ugovori

Pametni ugovori u ovom se vremenu mogu nazvati najkorištenijom primjenom blockchain tehnologije. Koncept pametnih ugovora uveo je Nick Szabo, pravni znanstvenik i kriptograf 1994. godine. Zaključio je da se bilo koja decentralizirana knjiga može koristiti kao ugovor koji je kasnije nazvan pametnim ugovorom. Ovi digitalni ugovori mogu se pretvoriti u kodove i dopustiti im da se izvode na blockchainu. Iako je ideja pametnih ugovora nastala davno, trenutni svijet u kojem živimo djeluje na papirnatim ugovorima. Čak i ako se koriste digitalni ugovori, uključivanje pouzdane treće strane iz sustava ne može se isključiti.

Iako smo definirali sustav funkcioniranja ovom metodom, ne možemo sa sigurnošću reći je li uvijek glatka. Uključivanje treće strane moglo bi dovesti do sigurnosnih pitanja ili lažnih aktivnosti zajedno s povećanom naknadom za transakcije.

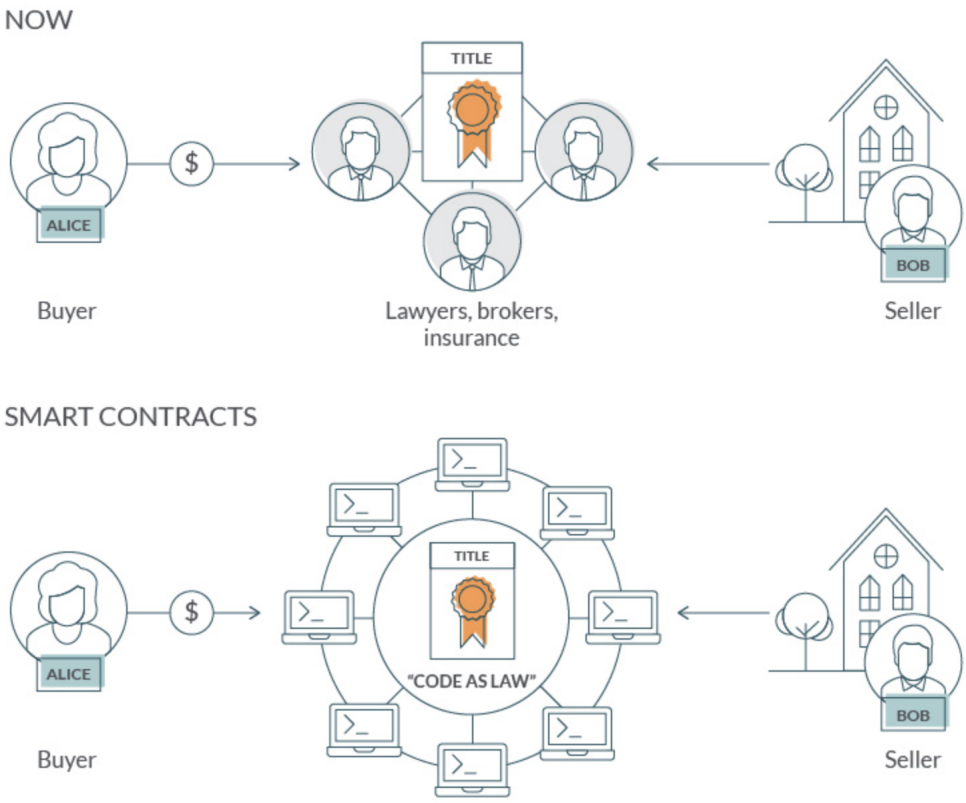
Pametni ugovor predstavlja kod u nekom programskom jeziku koji olakšava razmjenu novca, nekretnina, dionica ili bilo kakvih vrijednosti. Možemo reći da pametni ugovori služe za reguliranje nekog poslovnog odnosa između stranaka među kojima ne postoji uzajamno povjerenje. Takav kod se može zapisati na blockchain i izvršavati na bilo kojem računalu u distribuiranoj mreži. Pametan ugovor se automatski izvršava kada su zadovoljeni specifični uvjeti.

Proces kreiranja pametnog ugovora može se podijeliti u tri koraka:

1. Moguća razmjena dobara između dva ili više partnera zapisuje se kao programerski kod i pohranjuje na blockchain. Partneri ostaju anonimni, no sadržaj pametnog ugovora javno je dostupan svim partnerima u sustavu.

2. Varijable poput datuma ili određene količine novca potiču izvršavanje ugovora prema pravilima definiranim u kodu

3. Ostali korisnici sustava mogu pretražiti blockchain, kako bi razumjeli aktivnosti definirane ugovorom ili provjerili rezultat izvršavanja ugovora



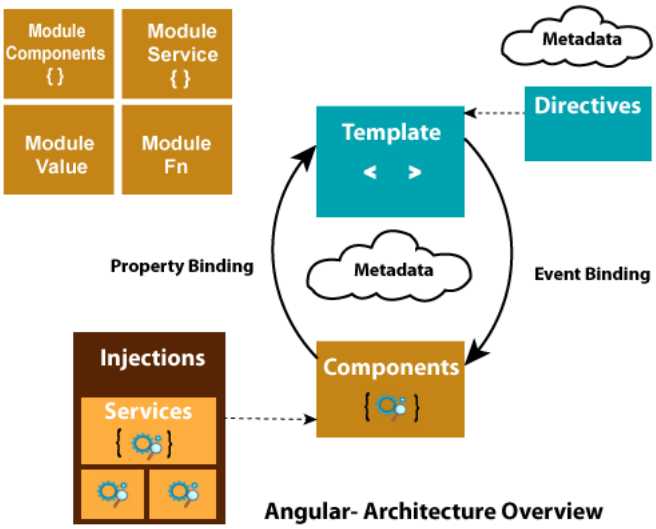
Slika 1.14: Pametni ugovori

Poglavlje 2

Razvojni okvir Angular

2.1 Uvod

Angular je platforma i razvojni okvir za razvoj jednostraničnih klijentskih aplikacija koji koristi HTML i programski jezik Typescript. Također, sam okvir je napisan u Typescriptu. Sve osnovne i opcionalne funkcionalnosti implementirane su kao skup Typescript biblioteka koje se onda uvoze u aplikaciju koja se razvija.
Arhitektura Angular aplikacije oslanja se na određene temeljne koncepte. Osnovni građevni blokovi Angular okvira su komponente koje su organizirane u module. Moduli prikupljaju povezane dijelove koda u funkcionalne skupove, cijela aplikacija je zapravo definirana skupom modula. Angular aplikacija uvijek ima barem korijenski modul (eng. *root module*) i obično ima više modula značajki (eng. *feature modules*).
Metapodaci neke komponente pridružuju toj komponenti predložak (eng. *template*) koji definira pogled (eng. *view*), odnosno dio zaslona kojim komponenta upravlja. Predložak kombinira obični HTML kod s Angular direktivama koje dopuštaju Angularu da modificira HTML prije nego što ga prikaže.
Komponente aplikacije obično definiraju mnogo takvih pogleda koji su raspoređeni u hijerarhiju. Angular pruža uslugu usmjerivača (eng. *router*) koji pomaže u navigaciji između različitih pogleda.



Slika 2.1: Arhitektura Angular aplikacije

2.2 Typescript

TypeScript je besplatan jezik otvorenog koda, razvijen i održavan od strane Microsofta. Sintaksa mu je strogi nadskup JavaScripta. TypeScript podržava pisanje programskog koda u JavaScriptu, korištenje biblioteka za JavaScript i pozivanje TypeScript koda iz samog JavaScripta. Drugim riječima, svaki JavaScript program je zapravo i TypeScript program. TypeScript se pomoću specijalnog programa, takozvanog transpilera prevodi u čisti i jednostavan JavaScript programski kod koji se lako pokreće u bilo kojem internetskom pregledniku i na bilo kojem JavaScript pokretaču (eng. *engine*) koji podržava standard ECMAScript 3 ili noviji. TypeScript se koristi za razvoj klijentskih JavaScript aplikacija, ali i serverskih (Node.js) aplikacija.

Glavne prednosti programskog jezika TypeScript su to što je objektno orijentiran i to što su tipovi svih varijabli poznati u vrijeme prevođenja. Na taj način prevoditelj (eng. *transpiler*) može otkriti velik broj trivijalnih pogrešaka već u najranijim fazama razvoja. Ova svojstva olakšavaju razvoj srednjih i velikih aplikacija. S obzirom da je JavaScript podskup TypeScripta, programeri dobivaju dodatne mogućnosti kombiniranjem elemenata ta dva programska jezika. TypeScript također donosi mnoge mogućnosti standarda ECMAScript, kao sto su lambda funkcije, moduli i klase.

2.3 Arhitektura Angulara

Moduli

Angular aplikacije su modularne i Angular ima vlastiti sustav modularnosti nazvan NgModuli (eng. *NgModules*). NgModuli su zapravo spremnici za neki kohezivni blok koda posvećen radnom toku ili domeni aplikacije. Ti spremnici mogu sadržavati komponente, servise ili druge datoteke čiji je opseg definiran NgModulom koji sadrži. Također, oni mogu i uvesti (eng. *import*) određenu funkcionalnost koja je izvezena iz drugih NgModula ili odabranu funkcionalnost izvesti (eng. *export*) za korištenje od strane drugih NgModula. Kao što je rečeno u uvodu, svaka Angular aplikacija ima barem jednu klasu NgModule, a to je korijenski modul, koji se konvencionalno zove AppModule i nalazi se u datoteci pod nazivom app.module.ts. Svaka aplikacija se pokreće podizanjem tog korijenskog modula, koji je nazvan tako iz razloga što može uključiti sve podređene NgModule (eng. *child modules*) u hijerarhiju proizvoljne dubine.

NgModuli pružaju kompilacijski kontekst svojim komponentama, pa komponente koje pripadaju istom NgModulu dijele isti kompilacijski kontekst. Svaki NgModule je definiran klasom dekoriranom s `@NgModule()`. Dekorator `@NgModule()` je funkcija koja uzima jedan parametar, metapodatak, čija svojstva opisuju taj modul. Najvažnija svojstva su sljedeća:

1. **deklaracije:** predstavljaju komponente, direktive i cijevi (eng. *pipes*) koje pripadaju tom modulu
2. **izvozi**: podskup deklaracija koji bi trebao biti vidljiv i upotrebljiv u predlošcima komponenti drugih NgModula
3. **uvozi**: ostali moduli čije su izvezene klase potrebne predlošcima komponenti deklariranim u ovom NgModulu
4. **servisi**: kreatori usluga koji postaju dostupni u svim dijelovima aplikacije
5. **bootstrap**: glavni pogled dane aplikacije u kojoj se nalaze svi ostali pogledi, samo bi korijenski modul trebao imati postavljeno ovo svojstvo

```
@NgModule({
  declarations: [
    AppComponent,
    WalletComponent,
    BlockchainComponent,
    TransactionPoolComponent,
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule,
    FormsModule
  ],
  providers: [NzNotificationService, NzMessageService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Slika 2.2: Metapodaci jedne NgModule klase

Komponente

Komponente kontroliraju dio zaslona koji se zove pogled. Aplikacijsku logiku komponente definiramo unutar klase, odnosno to su sva svojstva i metode koje komponenta treba da bi upravljala pogledom.

Angular stvara, ažurira i uklanja komponente kako se korisnik kreće kroz aplikaciju, što je postignuto metodama životnog tijeka (eng. *lifecycle hook methods*) komponente. To se postiže na način da se aplikacija uključi u ključne događaje u životnom ciklusu komponente kako bi inicijalizirala novu instancu komponente, ažurirala neke nove promjene ili očistila što je potrebno prije brisanja te instance. Primjer jedne takve metode je `ngOnInit()`, metoda koja se poziva tijekom stvaranja komponente. Sljedeći primjer pokazuje metodu koja postavlja vrijednost svojstva `blockchain` tijekom instanciranja komponente `BlockchainComponent`.

```
export class BlockchainComponent implements OnInit, OnDestroy {

  blockchain!: Block[]
  subscription = new Subscription();

  constructor(private service: AppService) {}

  ngOnInit(): void {
    this.subscription.add(
      this.service.getBlockchain().subscribe( next: blockchain => {
        this.blockchain = blockchain;
      })
    );
  }
```

Slika 2.3: Metoda `ngOnInit`

Predlošci i direktive

Predlošci i direktive koriste se uzajamno, a zajedno definiraju na koji način se dio aplikacije treba prikazati na zaslonu. Način na koji se direktive koriste ovisi o vrsti direktive. Postoje tri osnovne vrste direktiva:

1. **komponente**: direktive s vlastitim predloškom
2. **atributne direktive**: mijenjaju izgled ili ponašanje nekog elementa ili komponente u predlošku
3. **strukturalne direktive**: mijenjaju izgled dokumenta dodavanjem ili uklanjanjem nekog elementa iz dokumenta

Primjer jedne ugrađene atributne direktive je `NgStyle`, koja se koristi za postavljanje stila elementa, ovisno o trenutnom stanju komponente. Za upotrebu te direktive, potrebno je dodati uvjet u čijoj ovisnosti će se primijeniti određeni stil. Na slijedećem primjeru boja pozadine ovisi o vrijednosti svojstva `person.country`.

Još jedna vrlo česta direktiva je `*ngFor`, ona pripada kategoriji strukturalnih direktiva. Primjenjuje se u slučajevima kada je potrebno prikazati sve elemente neke liste i tada se direktiva postavlja na roditeljski element.

```
<span [ngStyle]="{'background-color' : person.country === 'UK' ? 'green' : 'red' }"></span>
```

Slika 2.4: Direktiva `ngStyle`

```
<div *ngFor="let item of itemList; let i = index">
  <p>{{ item.name }}</p>
</div>
```

Slika 2.5: Direktiva `ngFor`

2.4 Osnovni koncepti

Vezanje podataka

Vezanje podataka jedan je od temeljnih koncepata u Angularu i služi za definiranje komunikacije između komponente i DOM-a, što izradu interaktivnih aplikacija čini puno lakšom. Vezanje podataka automatski održava zaslon aplikacije ažurnim na temelju trenutnog stanja aplikacije. Koristi se za definiranje svojstava kao što su stanje nekog gumba ili podaci određenog korisnika. Postoje četiri oblika vezanja podataka koji se razlikuju prema načinu protoka podataka.

- **interpolacija** `{{ value }}`
  Predlošku se šalje vrijednost svojstva `value` iz komponente

```
<p>Name: {{ user.name }}</p>
```

Slika 2.6: Interpolacija podataka

- **vezivanje svojstva** `[property] = "value"`
  Vrijednost svojstva `value` iz komponente se dodjeljuje svojstvu `property`

```
<input type="email" [value]="user.email">
```

Slika 2.7: Vezivanje svojstva

- **vezivanje događaja** `(event) = "function"`
  U slučaju nekog događaja u DOM-u, primjerice klika na gumb, metoda `function` se pozove iz komponente

```
<button (click)="showUserData()"></button>
```

Slika 2.8: Vezivanje događaja

- **dvosmjerno vezivanje podataka** `[(ngModel)] = "value"`
  Omogućuje protok podatka u oba smjera, odnosno promijeni li se neko svojstvo u komponenti, to je odmah vidljivo u predlošku i obratno, promijeni li korisnik neko svojstvo u predlošku, svojstvo u komponenti se automatski ažurira na novu vrijednost.

```
<input type="email" [(ngModel)]="user.email">
```

Slika 2.9: Dvosmjerno vezivanje podataka

Injekcija ovisnosti

Ovisnosti (eng. *dependencies*) u aplikaciji su servisi ili objekti koji su klasi potrebni za obavljanje svoje funkcije. Injekcija ovisnosti (eng. *dependency injection*) je oblikovni obrazac u kojem klasa zahtijeva ovisnosti iz vanjskih izvora umjesto da ih stvara sama. Angularov okvir za injekciju ovisnosti klasi pruža potrebne ovisnosti nakon instanciranja te klase.

```
@Injectable({
  providedIn: 'root'
})
export class AppService {

  getBlockchain(): Observable<Block[]> {
    return this.http.get<Block[]>( url: this.url + '/blocks');
  }
```

Slika 2.10: Injekcija ovisnosti: servis koji se injektira

Dekorator `@Injectable()` omogućava Angularu da tu klasu može injektirati u drugu komponentu kao ovisnost. Metapodatak `providedIn: 'root'` označava da je ta klasa vidljiva cijeloj aplikaciji.

```
export class BlockchainComponent implements OnInit, OnDestroy {

  constructor(private service: AppService) {}

  ngOnInit(): void {
    this.subscription.add(
      this.service.getBlockchain().subscribe( next: blockchain => {
        this.blockchain = blockchain;
      })
    );
  }
```

Slika 2.11: Injekcija ovisnosti: komponenta koja zathijeva servis u konstruktoru

Usmjeravanje

U jednostraničnim aplikacijama, umjesto da od servera zahtijevamo novu stranicu, ono što korisnik vidi mijenja se prikazujući ili skrivajući dijelove zaslona koji pripadaju određenim komponentama. Korisnik se krećući po aplikaciji zapravo kreće po različitim pogledima definiranim u komponentama. Za rukovanje navigacijom od jednog pogleda do drugog, koristi se usmjerivač. On omogućuje navigaciju tako da interpretira URL preglednika kao uputu za prikaz odabranog pogleda. Na taj način se može značajno poboljšati korisničko iskustvo aplikacije.
Da bi se definiralo kako se korisnici trebaju kretati po aplikaciji, potrebno je definirati rute od jednog pogleda do drugog. Također je moguće kofigurirati rute da štite od neočekivanog ili neovlaštenog pristupa nekom pogledu. Tada se na neki element u pogledu može postaviti direktiva `routerLink` koja taj element učini vezom koja će onda inicirati navigaciju do dane rute.

```
const routes: Routes = [
  { path: 'wallet', component: WalletComponent },
  { path: 'blockchain', component: BlockchainComponent },
  { path: 'pool', component: TransactionPoolComponent },
  { path: '', component: HomeComponent },
  { path: '**', redirectTo: '' }
];
```

Slika 2.12: Definirane rute u aplikaciji

```
<nz-header>
  <div class="logo"></div>
  <ul nz-menu nzTheme="dark" nzMode="horizontal">
    <li nz-menu-item  [routerLink]="'/wallet'">Wallet</li>
    <li nz-menu-item [routerLink]="'/blockchain'">Blockchain</li>
    <li nz-menu-item [routerLink]="'/pool'">Pool</li>

    <li [style.float]="'right'">{{ peerName }}</li>
  </ul>
</nz-header>
```

Slika 2.13: `routerLink` koji stvara veze na rute u predlošku

Poglavlje 3

Opis i funkcionalnost aplikacije

3.1 Opis aplikacije

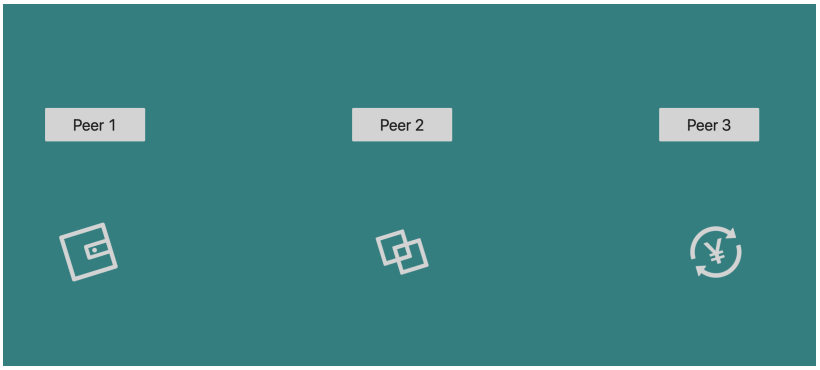
Kao što je rečeno i u uvodu, aplikacija razvijena u ovom radu jest demonstrativna aplikacija koja prikazuje kako funkcionira jednostavni blockchain sustav. U aplikaciji sudjeluju tri korisnika koji međusobno komuniciraju preko websocketa. Time se uspostavlja decentralizirana mreža, odnosno korisnici razmjenjuju informacije o blokovima i transakcijama bez potrebe centralne baze podataka.

Korisnik aplikacije ima vlastiti jednostavni novčanik u kojem je vidljivo njegovo trenutno stanje računa, može pregledavati stanje blockchaina te detalje blokova u njemu. Također, može pregledavati sve transakcije koje čekaju na potvrdu. Kao sudionik mreže, ima mogućnost stvaranja novih transakcija, te validacije nepotvrđenih transakcija, pri čemu se stvara novi blok u lancu blokova.

Tijekom stvaranja nove transakcije, u pozadini se izvršava stvaranje digitalnog potpisa kojeg svi ostali sudionici verificiraju pomoću ranije objašnjenih kriptografskih metoda. Također, tijekom dodavanja novog bloka, svi sudionici mreže potvrđuju da je blok pravedno i propisno stvoren, tj. da su sve informacije na njemu točne te da je učinjen algoritam dokaza o radu.

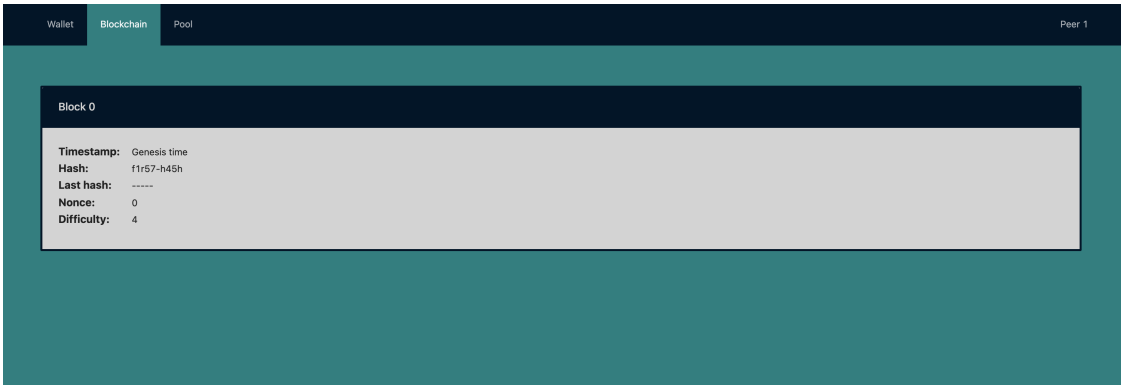
3.2 Klijentska strana aplikacije

Na početnom zaslonu ponuđen je odabir sudionika u mreži kojim želimo upravljati. Nakon odabira pojavljuje se zaglavlje koje nam nudi pregled novčanika, lanca ili nepotvrđanih transakcija.



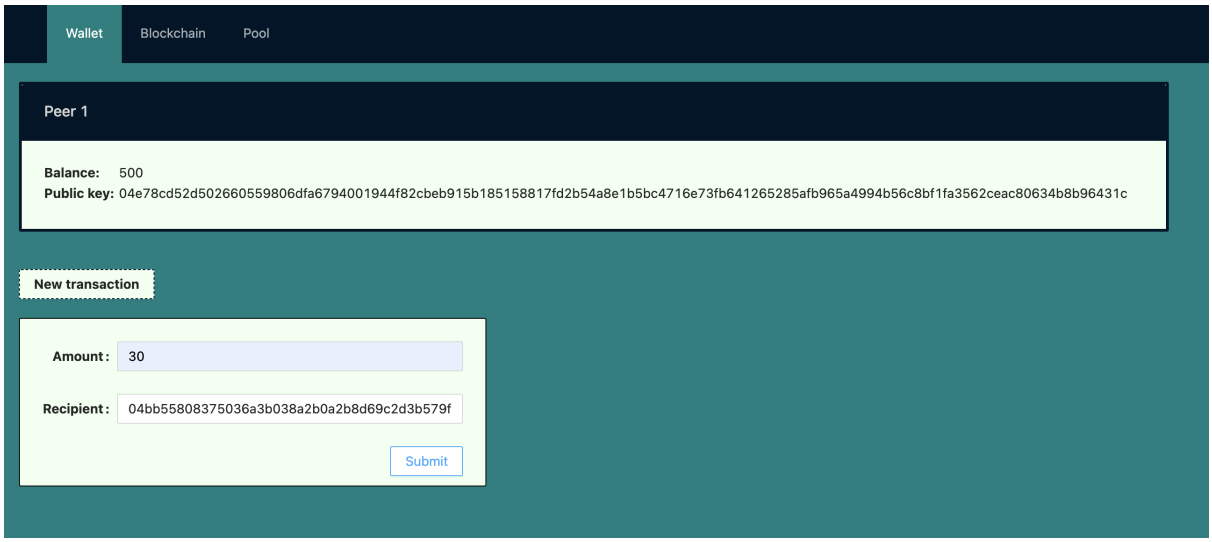
Slika 3.1: Prikaz početnog zaslona

Na početku, lanac blokova je prazan, odnosno postoji samo izvorni blok. Vidljivo je da ima neodređenu vremensku oznaku, hash koji označava da je to prvi blok, te prazan hash prethodnog bloka. Vrijednost nonce je postavljena na nulu zato što se za njegovu kreaciju nije izvršavao algoritam dokaza o radu, a težina algoritma postavljena je na početnu vrijednost.



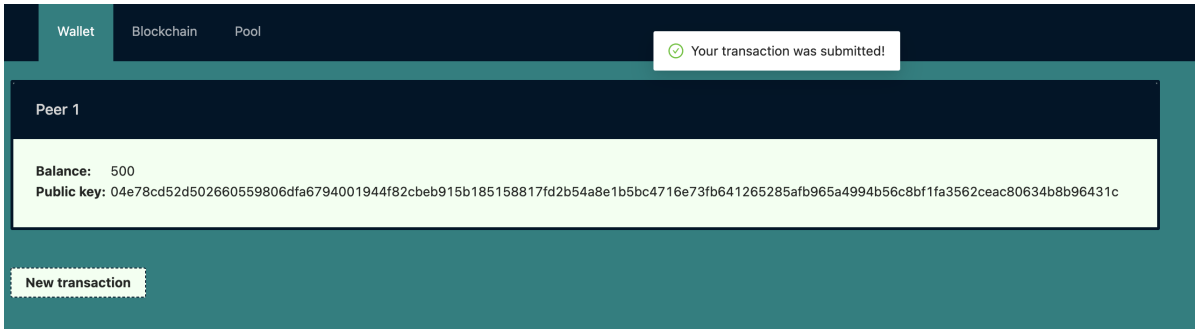
Slika 3.2: Prikaz praznog lanca bloka

Na zaslonu novčanika korisnik može vidjeti iznos svog računa, te svoj javni ključ koji služi drugim sudionicima kao adresa na koju mogu poslati novac. Ispod toga postoji gumb za stvaranje nove transakcije koji otvara formu s iznosom i adresom primatelja.



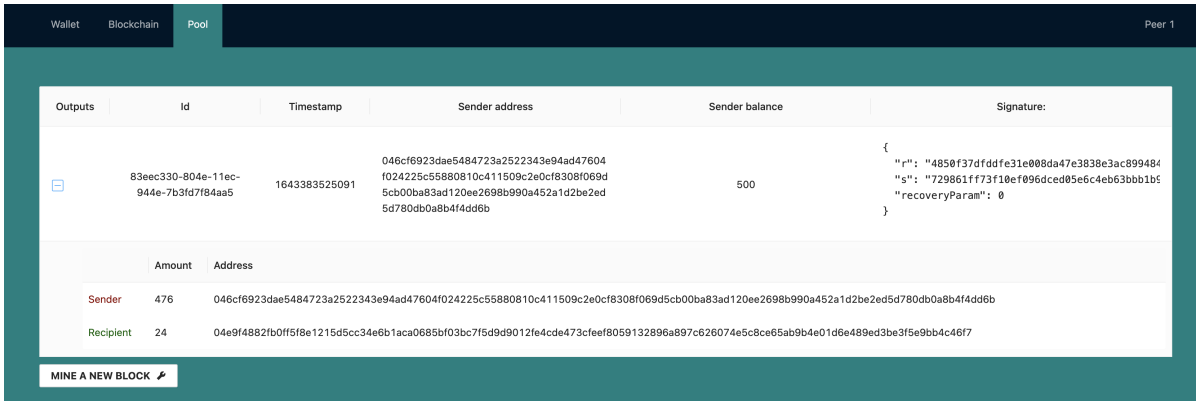
Slika 3.3: Prikaz stvaranja nove transakcije

Nakon klika na gumb *Submit*, forma se zatvara te se transakcija šalje u listu nepotvrđenih transakcija.



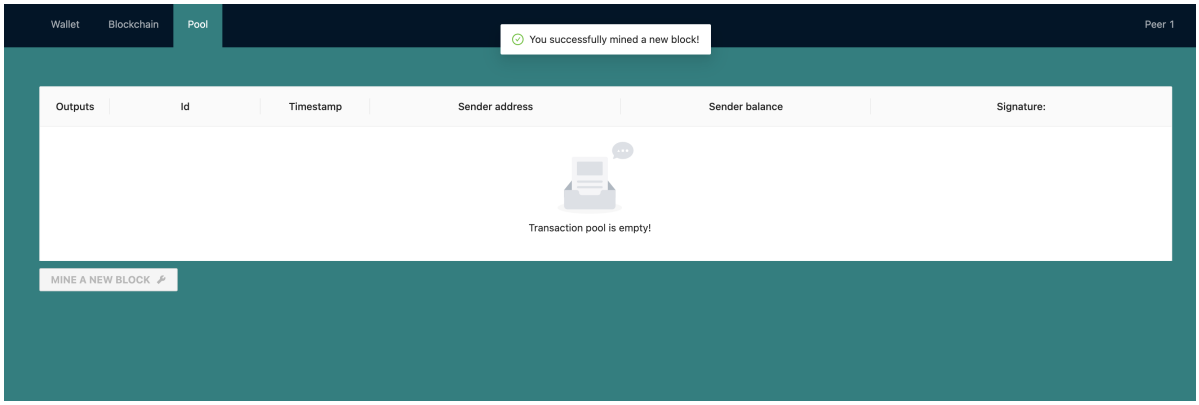
Slika 3.4: Prikaz uspješno stvorene transakcije

Na zaslonu transakcija tada možemo vidjeti poslanu transakciju u listi. Ispod liste postoji gumb *Mine a new block* s kojim korisnik može nepotvrđene transakcije verificirati te s njima stvoriti novi blok.



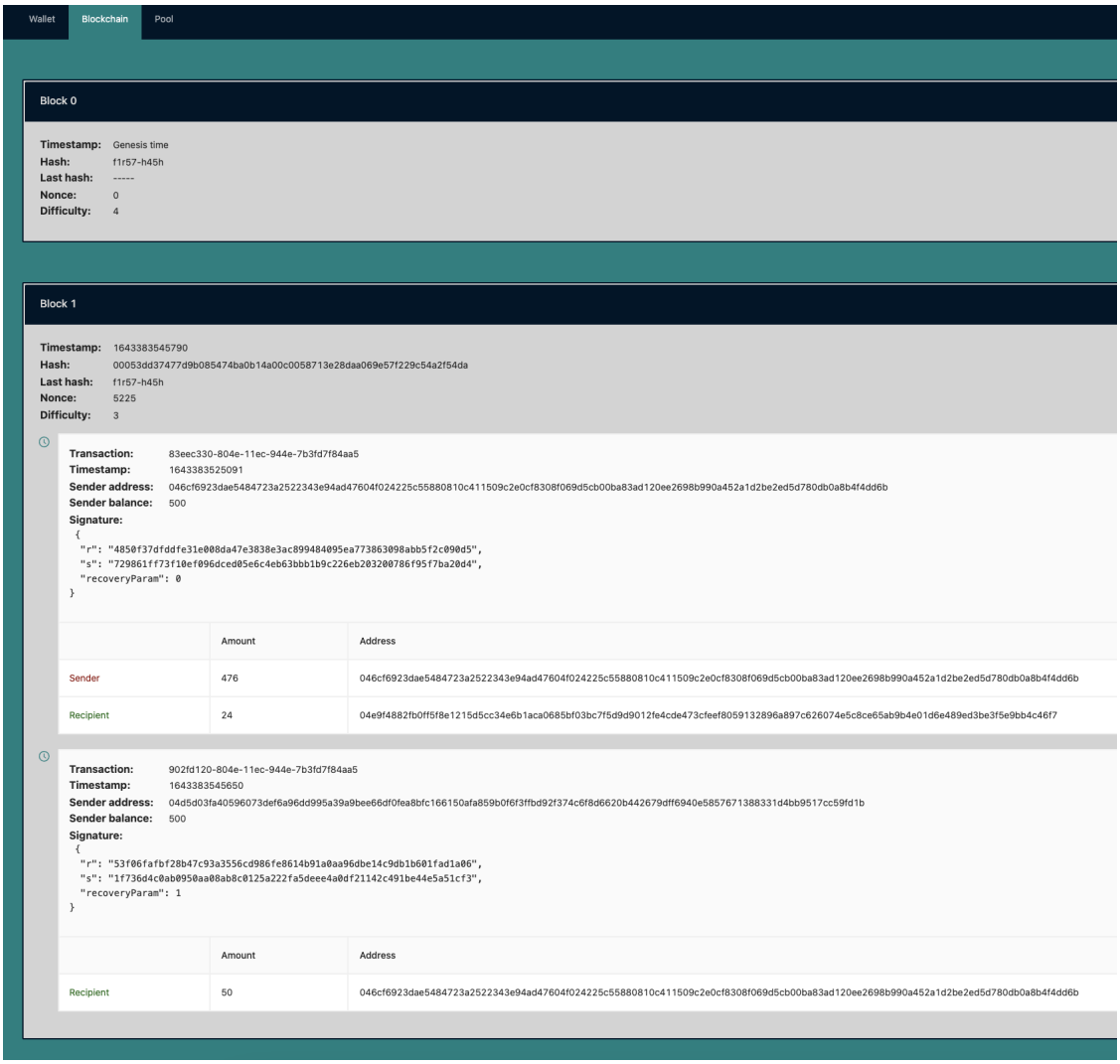
Slika 3.5: Prikaz tranakcije u listi nepotvrđenih transakcija

Nakon uspješnog rudarenja bloka, korisnik dobije potvrdu o stvorenom bloku te lista nepotvrđenih transakcija ostaje prazna.



Slika 3.6: Prikaz uspješno rudarenog bloka

Tada, na zaslonu blockchaina vidimo novo stvoreni blok u lancu. Osim izvršene transakcije koju je korisnik verificirao, također postoji i transakcija čiji je primatelj upravo rudar koji je stvorio novi blok. Ta trasnsakcija predstavlja nagradu od sustava za rudarenje bloka.



Slika 3.7: Prikaz blockchaina s novim blokom

3.3 Serverska strana aplikacije

Serverski dio aplikacije napisan je u programskom jeziku Javascript koristeći Node.js (*Javascript runtime environment*). U ovom poglavlju proći ćemo kroz najbitnije dijelove implementacije samih principa i strukture blockchaina.

Već je rečeno da sudionici međusobno komuniciraju preko socketa. Nakon uspostavljanja konekcije među svim korisnicima, funkcija `messageHandler` brine o razmjeni informacija. Ovisno o vrsti poruke koja se pošalje s nekog socketa, funkcija poziva odabrane metode. Poruke se mogu slati nakon dodavanja novog bloka u svrhu sinkroniziranja lanaca svih sudionika, potrebe da se isprazni lista nepotvrđenih transakcija te izračunavanja novih iznosa na računu. Također, nakon dodavanja nove transakcije potrebno je sinkronizirati liste nepotvrđenih transakcija.

```
messageHandler(socket) {
  socket.on('message', message => {
    const data = JSON.parse(message);

    switch (data.type) {
      case MESSAGE_TYPES.chain:
        this.blockchain.replaceChain(data.chain);
        break;
      case MESSAGE_TYPES.transaction:
        this.transactionPool.updateOrAddTransaction(data.transaction);
        break;
      case MESSAGE_TYPES.clear_transactions:
        this.transactionPool.clear();
        break;
      case MESSAGE_TYPES.calculate_balance:
        this.wallet.calculateBalance(data.block);
        break;
    }
  });
}
```

Slika 3.8: Metoda `messageHandler` - komunikacija unutar mreže

Primjerice, nakon dodavanja nove transakcije, potrebno ju je *broadcastati* putem cijele mreže, te pozivanjem metode `broadcastTransaction`, svaki sudionik postaje svjestan nove transakcije u listi. Slično, nakon dodavanja novog bloka, kada je potrebno isprazniti listu, poziva se metoda `broadcastClearTransactions`.

```
broadcastTransaction(transaction) {
  this.sockets.forEach(socket => {
    this.sendTransaction(socket, transaction);
  })
}

sendTransaction(socket, transaction) {
  socket.send(JSON.stringify( value: {
    type: MESSAGE_TYPES.transaction,
    transaction: transaction
  }));
}

broadcastClearTransactions() {
  this.sockets.forEach(socket => socket.send(JSON.stringify( value: {
    type: MESSAGE_TYPES.clear_transactions
  })));
}
```

Slika 3.9: Metoda `broadcastTransaction` - slanje transakcija unutar mreže

Kreiranje nove transakcije je moguće samo ako pošiljatelj ima dovoljno sredstava na računu. U slučaju da ima, poziva se metoda koja stvara novi objekt transakcije koji sadrži iznos koji će pošiljatelj imati na računu nakon što se transakcija izvrši te njegovu adresu, uz iznos koji želi poslati i adresu primatelja.

```
static newTransaction(senderWallet, recipient, amount) {
  if (amount > senderWallet.balance) {
    console.log(`Amount: ${amount} exceeds balance.`);
    return;
  }
  return Transaction.transactionsWithOutputs(senderWallet, outputs: [
    { amount: senderWallet.balance - amount, address: senderWallet.publicKey },
    { amount: amount, address: recipient }
  ]);
}
```

Slika 3.10: Metoda `newTransaction` - stvaranje nove transakcije

Unutar te metode, pozvat će se i funkcija `signTransaction` koja kreira digitalni potpis pomoću kojeg pošiljatelj dokazuje da je zaista on poslao novac. Taj digitalni potpis će kasnije verificirati ostali sudionici nakon *broadcasta* te transakcije. Digitalni potpis se stvara uz pomoć biblioteke `elliptic` i metode `sign` unutar te biblioteke.

```
static signTransaction(transaction, senderWallet) {
  transaction.input = {
    timestamp:Date.now(),
    amount: senderWallet.balance,
    address: senderWallet.publicKey,
    signature: senderWallet.sign(ChainUtil.hash(transaction.outputs))
  };
}
```

Slika 3.11: Metoda `signTransaction` - stvaranje digitalnog potpisa

Sljedeća bitna stavka u implementaciji blockchaina jest rudarenje novog bloka. Metoda `mine` se poziva kada sudionik mreže želi dodati nepotvrđene transakcije u lanac. Najprije se transakcije validiraju jedna po jedna, provjeravajući digitalne potpise na njima te ima li pošiljatelj dovoljno sredstava na računu da ih izvrši. Nakon validacije, u listu transakcija koja se stavlja u lanac dodaje se još jedna transakcija, a to je nagrada za rudara koji dodaje novi blok. Tada se poziva metoda `addBlock` u kojoj se izvršava algoritam dokaza o radu te se u novi blok dodaje dobivena vrijednost nonce.

```
mine() {
  if (this.transactionPool.empty()) {
    console.log('Cannot mine a block with no transactions.');
    return false;
  }

  const validTransactions = this.transactionPool.validTransactions();
  if (validTransactions.length !== 0) {
    validTransactions.push(Transaction.rewardTransaction(this.wallet, Wallet.blockchainWallet()));

    const block = this.blockchain.addBlock(validTransactions);

    this.wallet.calculateBalance(block);

    this.p2pServer.syncChains();
    this.transactionPool.clear();
    this.p2pServer.broadcastClearTransactions();
    this.p2pServer.broadcastCalculateBalance(block);

    return block;
  }
}
```

Slika 3.12: Metoda `mine` - rudarenje bloka

Nakon što je cijeli proces izvršen, pozivaju se ranije spomenute metode kojima se sinkronizira stanje lanca, lista transakcija i iznosa na računu u cijeloj mreži.

```
static mineBlock(lastBlock, data) {
  let hash, timestamp;
  const lastHash = lastBlock.hash;
  let { difficulty } = lastBlock;
  let nonce = 0;

  do {
    nonce++;
    timestamp = Date.now();
    difficulty = Block.adjustDifficulty(lastBlock, timestamp);
    hash = Block.hash(timestamp, lastHash, data, nonce, difficulty);
  } while (hash.substring(0, difficulty) !== '0'.repeat(difficulty));

  return new this(timestamp, lastHash, hash, data, nonce, difficulty);
}
```

Slika 3.13: Metoda `mineBlock` - algoritam Proof of Work

Zatim, kada svi sudionici dobiju informaciju o novom bloku, na njima je da provjere je li validan prije nego što se blok doda u lanac. Tada se poziva metoda `isChainValid` gdje se svaki blok provjerava počevši od izvornog bloka.

```
isValidChain(chain) {
  if (JSON.stringify(chain[0]) !== JSON.stringify(Block.genesis())) {
    return false;
  }
  for (let i = 1; i < chain.length; i++) {
    const block = chain[i];
    const lastBlock = chain[i - 1];

    if (block.lastHash !== lastBlock.hash || block.hash !== Block.blockHash(block)) {
      return false;
    }
  }
  return true;
}
```

Slika 3.14: Metoda `isChainValid` - validacija lanca

Bibliografija

[1] Satoshi Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2013., https://bitcoin.org/bitcoin.pdf

[2] Jake Frankenfield, *Merkle tree*, 2021., https://www.investopedia.com/terms/m/merkle-tree.asp

[3] Hasib Anwar, *Consensus Algorithms: The Root Of Blockchain Technology*, 2018., https://101blockchains.com/consensus-algorithms-blockchain/#1

[4] Souptik Dutta Gupta, *Implementing Transaction Management System Based on Blockchain Architecture*, 2020., https://www.researchgate.net/publication/343006174_Implementing_Transaction_Management_System_Based_on_Blockchain_Architecture

[5] SuperDataScience Team, *Blockchain A-Z™: Learn to Build Your Own Blockchain - Additional Resources*, 2018. https://www.superdatascience.com/pages/blockchain

[6] Yakov Fain, Anton Moiseev, Manning, *TypeScript Quickly*, 2020.

[7] *Angular documentation*, https://angular.io/

[8] *Typescript documentation*, https://www.typescriptlang.org/

Sažetak

Ovaj rad započinje uvodom u kojem se definira cilj rada. Zatim slijedi poglavlje koje detaljno opisuje principe blockchain tehnologije. Poglavlje započinje poviješću blockchaina i nastavlja se kratkim pregledom kriptografskih metoda korištenih u blockchain tehnologiji. Iduće potpoglavlje navodi vrste blockchaina, a nakon toga se objašnjavaju struktura jednog bloka, decentralizirani sustav ravnopravnih partnera, način na koji funkcionira rudarenje blokova, te algoritmi konsenzusa korišteni u validaciji transakcija i blokova.

Drugo poglavlje stavlja naglasak na samu primjenu blockchain tehnologije, te se opisuje njena uloga u kriptovalutama i pametnim ugovorima.

Treće poglavlje se usredotočuje na razvojni okvir Angular pomoću kojeg je napravljen klijentski dio aplikacije. Nakon uvoda, opisuje se arhitektura razvojnog okvira navodeći sve osnovne dijelove nužne za izradu aplikacije. Zatim se tumače osnovni koncepti na kojima se zasniva povezivanje i interakcija svih elemenata u jednoj Angular aplikaciji.

Zadnje poglavlje opisuje izgled i značajke demonstracijske blockchain aplikacije napravljene za potrebe ovog diplomskog rada. Objašnjavaju se osnovne ideje iza najbitnijih funkcionalnosti, a zatim se prikazuje i sam rad aplikacije kroz nekoliko snimaka zaslona te njihovih kratkih opisa.

Summary

This thesis begins with an introduction that defines the purpose of the thesis. The first chapter provides a detailed explanation of principles upon which blockchain technology is based on. The chapter begins with the history of blockchain and continues with a brief overview of cryptographic methods used in blockchain technology. The next subsection lists the types of blockchains, followed by an explanation of the structure of a single block, the decentralized peer-to-peer network, the way block mining works, and the consensus algorithms used to validate transactions and blocks.

The second chapter highlights the usage of blockchain technology and describes its role in cryptocurrencies and smart contracts.

The third chapter focuses on the Angular framework used to create the frontend part of the application. After the introduction, the architecture of the framework is described, listing all the basic parts necessary for the development of an application. Then, the basic concepts upon which the connection and interaction of all elements in one Angular application are based on are explained.

The last chapter describes the appearance and features of the demo blockchain application made for the purposes of this thesis. The basic ideas behind the most important functionalities are explained, and then the work of the application itself is shown through several screenshots and their short descriptions.

Životopis

Tihana Oremuš rođena je 1. siječnja 1997. godine u Zagrebu. Školovanje je započela godine 2003. u osnovnoj školi Marina Držića. Nakon završetka, 2011. upisuje prirodoslovno-matematičku gimnaziju, XV. gimnaziju. 2015. godine upisuje preddiplomski sveučilišni studij Matematika na Prirodoslovno-matematičkom fakultetu u Zagrebu. Nakon tri godine završava navedeni studij čime stječe akademski naziv Baccalaurea matematike (sveučilišna prvostupnica matematike). Po završetku preddiplomskog studija, godine 2018. upisuje diplomski studij Računarstvo i matematika na istom fakultetu u Zagrebu. Tijekom posljednje godine studija radi u tvrtci AG04 Innovative solutions na poziciji softverskog injženjera.