VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

Blockchain technologijos pritaikymas Mikroservisų architektūroje

Blockchain technology application in Microservices architecture

Bakalauro darbas

Atliko:	Marius Alchimavičius	(parašas)
Darbo vadovas:	Liutauras Ričkus	(parašas)
Darbo recenzentas:	Viktoras Golubevas	(parašas)

Vilnius – 2018

Santrauka

Daug dėmesio susilaukusi blockchain technologija rodo potencialą išspręsti aibę problemų kaip internetiniai pinigai, duomenų skaidrumas, pašalinti centralizuotus tarpininkus, kurti skaitmeninį patikimumą. Tačiau šios technologijos pritaikymas turi savo apribojimus, todėl norint pritaikyti blockchain sprendimus sistemoje reikalingas tikslinga strategija, gairės. Taip pat šiuo metu sparčiai auga aplikacijų skaičius naudojantis mikroservisų architektūrą, tačiau nėra daug informacijos kaip šioje architektūroje galima pritaikyti/naudoti blockchain technologiją.

Šio darbo tikslas yra ištirti blockchain technologijos teikiamą nauda ir apribojimus ir pagal tai suteikti gaires kaip blockchain technologija gali būti pritaikyta mikroservisų architektūroje. Šiame darbe bus pateikiami aplikacijų pavyzdžiai naudojantys NodeJS ir Ethereum technologijas tam kad būtų galima palyginti skirtingus blockchain technologijos pritaikymo būdus.

Raktiniai žodžiai: blockchain, mikroservisai, integracija, pritaikymas

Summary

A lot of attention blockchain technology shows potential to solve a number of problems like internet money, data transparency, removing centralized third parties, create digital trust. But this technology application has its own limitations, that is why there is a need for guidelines and direct strategies in order for people to integrate blockchain solutions to their applications. Also there is increasing number of applications that are adopting microservices architecture, but there is not a lot of information how people can implement blockchain solutions into this style of computer architecture.

The goal of this work is to research the benefits and limitations given by blockchain technology and to create guidelines how blockchain technology could be used in microservices architecture. In this work there will be multiple examples using NodeJS and Ethereum technologies to illustrate and compare different approaches how one could implement blockchain solutions.

Keywords: blockchain, microservices, integration, application

TURINYS

Įvadas

Daug dėmesio pastaraisiais metais sulaukusi blockchain technologija pirmą kartą buvo panaudota sukuriant internetinę valiutą Bitcoin. Tai yra pirmas pavykęs bandymas sukurti necentralizuotą pinigų sistemą nepriklausančią nuo pasaulio politinių jėgų. Kad tai būtų pasiekta Bitcoin kriptovaliuta pristatė anksčiau nenaudotą blockchain technologiją, būtent dėl šios priežasties Bitcoin gali užtikrinti internetinių pinigų saugumą, bei apsisaugoti nuo įvairių su vertės perdavimu susijusių problemų. Laikui bėgant buvo pastebėta, kad blockchain technologija gali būti pritaikyta ne tik internetiniams pinigams, bet ir aibei kitų problemų kurios reikalauja duomenų integralumo, decentralizuotumo.

Dėl šios priežasties atsirado daug projektų susijusių su duomenų perdavimu, saugojimu, pasitikėjimo kūrimu, atsiskaitymo, skolinimo sistemomis. Atsirado poreikis daryti blockchain technologijos pritaikymą standartinėse aplikacijose dėl teikiamų privalumų.

Paraleliai greitai plintanti mikroservisų architektūra taip pat patraukė žmonių dėmesį kaip alternatyva standartinei monolitinei architektūrai. Ši architektūra moduliaresnė ir dėl to įgauna savybių, kurios padeda prisitaikyti prie didėjančio vartotojų skaičiaus, didėjančio duomenų srauto bei visumoje didėjančio kuriamų programų kompleksiškumo.

Šio darbo tikslas yra ištirti kodėl ir kaip galima pritaikyti blockchain technologiją mikroservisų architektūroje. Šio darbo išvadose bus stengiamasi pateikti gaires kada verta naudoti ir įtraukti blockchain technologiją mikroservisų architektūroje.

Tam, kad pasiekti užsibrėžtą tikslą išskiriami šia uždaviniai.

- Apibendrinti blockchain technologiją, teikiamą naudą, apribojimus, veikimo principą, esamus panaudojimus.
- Apibendrinti mikroservisų architektūrą, teikiamą naudą, apribojimus, palyginti su standartinėmis praktikomis.
- Nustatyti blockchain taikymo galimybes tradicinėse programų sistemose.
- Išbandyti įvairias taikymo galimybes pavyzdiniuose prototipuose, juos aprašyti.
- Palyginti skirtingus taikymo būdus. Rasti pranašumus, trūkumus, bendrąsias savybes.
- Palyginti su centralizuotos sistemos veikimu.

1. Blockchain technologija

1.1. Atsiradimas

2008 metais įvykus finansiniai krizei Satoshi Nakamoto sukūrė mokslinį darbą kaip sukurti patikimą skaitmeninę valiutą. Pagrindiniai šios valiutos tikslai buvo sukurti valiutą, kuri nėra priklausoma nuo išorinių politinių jėgų, valiutos valdymas niekam nepriklausytų, būtų neįmanoma pakeisti transakcijų istorijos, transakcijų istorija būtų vieša ir pati valiuta būtų atspari žinomoms atakoms susijusioms su vertės apsikeitimu. Tai buvo pirmasis projektas panaudojęs blockchain technologiją tam, kad laikyti visas įvykusias transakcijas necentralizuotu būdu ir apsisaugoti nuo "dvigubo pinigų išleidimo" problemos.[min]

1.2. Komponentai

- **Piniginė** - Programa, kuri atlieka šį funkcionalumą:
  - Kuria vartotojui privatų ir viešą raktus vėliau naudojamus kurti transakcijas ir naujus piniginės adresus.
  - Saugo vartotojo pinigus, autentifikuojant vartotoją.
  - Tikrina sąskaitos likutį.
  - Kuria bei pasirašo transakcijas.
- **Adresas** Primityvus identifikatorius, kurį sukuria piniginė. Skirtas identifikuoti kitą šalį pinigų siuntimui. Piniginė dažniausiai turi žymiai daugiau nei vieną adresą pinigų siuntimui ("Trezor" piniginė kiekvienam vartotojui sukuria 100 adresų).
- **Transakcija** Istorijos įrašas, skirtas vertės perdavimui užregistruoti. Skirtingos blockchain sistemos saugo skirtingą informaciją, bet bendru atveju yra saugomi šie laukai.
  - Gavėjas
  - Siuntėjas
  - Pinigų/Vertės kiekis
- **Transakcijų maišos kodas** "Merkle tree" maišos algoritmu didelis skaičius transakcijų sąrašas yra sutrumpinamas į vieną simbolių eilutę, kuri yra naudojamu transakcijų validumui užtikrinti.
- **Blokas** Duomenų struktūra naudojama išsaugoti transakcijų sąrašą kas tam tikrą laiko intervalą. Bitcoin atveju naujas blokas kuriamas maždaug kas 5 - 15 min. Jame saugoma informacija, skiriasi skirtinguose blockchain tinkluose, bet bendru atveju turi:
  - Praeito bloko maišos kodas
  - Transakcijų maišos kodas

  - Einamojo (savo) bloko maišos kodas

- **Blokų grandinė** Duomenų struktūra panaši į vienos krypties sąrašą (angl. "Singly linked list"), tačiau vietoj saugomos nuorodos į praeitą mazgą (šiuo atveju bloką) yra saugomas bloko maišos kodas. Taip yra sudaroma duomenų struktūra kur kiekvienas blokas turi "nuorodą" į praeitą bloką ir tai vadiname blokų grandine.

- **Bloko maišos kodas** Specifiniu maišos algoritmu (skirtingi blockchain tinklai - skirtingi maišos algoritmai) bloko informacija sutrumpinta į simbolių eilutę. Į maišos algoritmo įvestį bendru atveju paduodami šie duomenys:

  - Transakcijų maišos kodas
  - Praeito bloko maišos kodas
  - Kūrimo laikas
  - Nereikšminis duomuo

- **Nereikšminis duomuo** (angl. Nonce) Nieko nenusakantis duomuo bloke skirtas įgalinti kriptografinės užduoties sprendimą tam, kad nauji blokai galėtų būti pridėti į blockchain tinklą.

- **Tinklo mazgas** (angl. "Network node") Prie sistemos prijungtas kompiuteris, kuris atlieka blockchain tinklo reikalaujamas užduotis už tam tikrą finansinę naudą. Bendru atveju atliekami šie darbai:

  - Naujo bloko maišos kodo paieška.
  - Tinklo validumo tikrinimas

[gee]

1.3. Tinklo inicijavimas

Pagrindinis blockchain technologijos tikslas - sukurti decentralizuotą, nekeičiamą duomenų bazę. Bitcoin atveju tam, kad saugoti visas pinigų transakcijas.

- Nekeičiamumas reikalingas, kad nebūtų galima, niekaip pakeisti jau įvykusios transakcijos, tai yra jei vartotojas gavo pinigų į savo sąskaitą, tai jokiu būdu nebus galima atšaukti, nepaisant politinių ar kitų išorinių veiksnių.

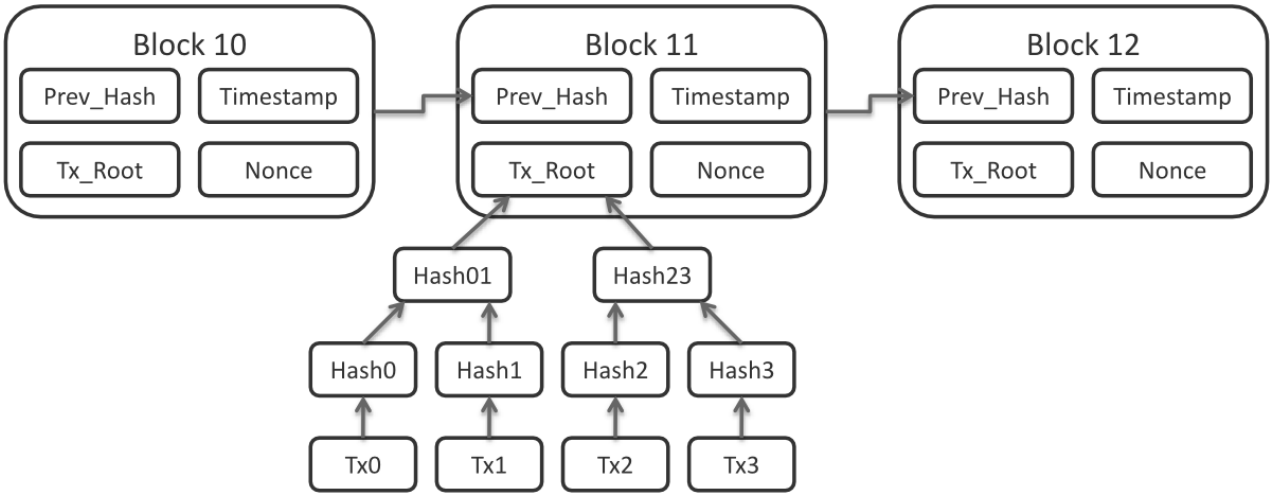
- Decentralizacija reikalingą, kad nebūtų vieno taško, kurio pagalba būtų galima sugadinti visą sistemos darbą. Tai apsaugo nuo kenkėjiškų atakų ir tinklo mazgų atsijungimo nuo blockchain tinklo, tai yra sistema gali veikti tol yra bent vienas tinklo mazgas (šiuo metu Bitcoin tinklą sudaro 10000 tinklo mazgų [bit]).

Sukuriant blockchain tinklą, viskas prasideda nuo pirmojo bloko (angl. ”Genesis block”), kuris yra sukuriamas ”rankomis”, tai yra programoje nurodant pradinę transakcijų istoriją, kas tai pat reiškia su kurias adresais bus susieti pirmieji blockchain pinigai.

Sekantis žingsnis pridėti į blockchain tinklą tinklo mazgų. Tinklo mazgai keleto rūšių

- ”Pilnieji” (angl. ”Full node”) - turi visą transakcijų istoriją ir jų pareiga tikrinti sukurtų naujų blokų validumą.
- ”Sutrumpinti” (angl. ”Pruned node”) - turi transakcijų istoriją iki tam tikro atminties kiekio limito, jų pareiga taip pat tikrinti naujų blokų validumą, tačiau reikalauja mažiau resursų.
- ”Kasėjas” (angl. ”Miner”) - savyje laiko einamųjų transakcijų istoriją ir jų tikslas spręsti kriptografines užduotis tam, kad sukurti naują bloką.

1.4. Naujų blokų kūrimas



https://commons.wikimedia.org/wiki/File:Bitcoin_Block_Data.png

1 pav. Grandies blokų jungimas

Nauji tinklo blokai yra kuriami tinklo kasėjų. Naujo bloko sukūrimui reikalinga validi einamųjų transakcijų istorija, praeito bloko maišos kodas (1.pav.). Kai kurių einamųjų transakcijų kasėjas gali pasirinkti neįtraukti, pavyzdžiui jei transakcija turi mažą ar išvis neturi jokio kasimo mokesčio. Kad blokas būtų patvirtintas sistemos kaip tinkamas prijungti jis turi būti validus ir turėti maišos kodą (bloko identifikatorių), kuris yra gaunamas išsprendus kriptografinę užduotį. Toks procesas vadinamas - įrodymas darbu (angl. ”Proof of work”) (daugiau informacijos sekančioje skiltyje). Sukurus bloką atitinkantį visas sąlygas bloko informacija transliuojama į tinklą. Blokas yra patikrinamas kitų tinklo mazgų (Bitcoin atveju ”Pilnųjų” ir ”Sutrumpintų” tinklo mazgų) ir yra prijungiamas į blokų grandinę. Prijungimas transliuojamas visiems tinklo mazgams. Po prijungimo tinklo mazgai atnaujina einamųjų transakcijų istoriją ir vėl pradeda ieškoti tinkamo bloko identifikatoriaus, naujo bloko sukūrimui. [gee]

1.5. Konsensuso radimas

Dažniausiai sutinkamas konsensuso radimo algoritmas vadinamas - įrodymas darbu (angl. ”Proof of work”), jį naudoja Bitcoin. Yra ir alternatyvių algoritmų (daugiau informacijos skiltyje ”Alternatyvos”), tačiau aptarsime šį, nes jis šiuo metu yra populiariausias.

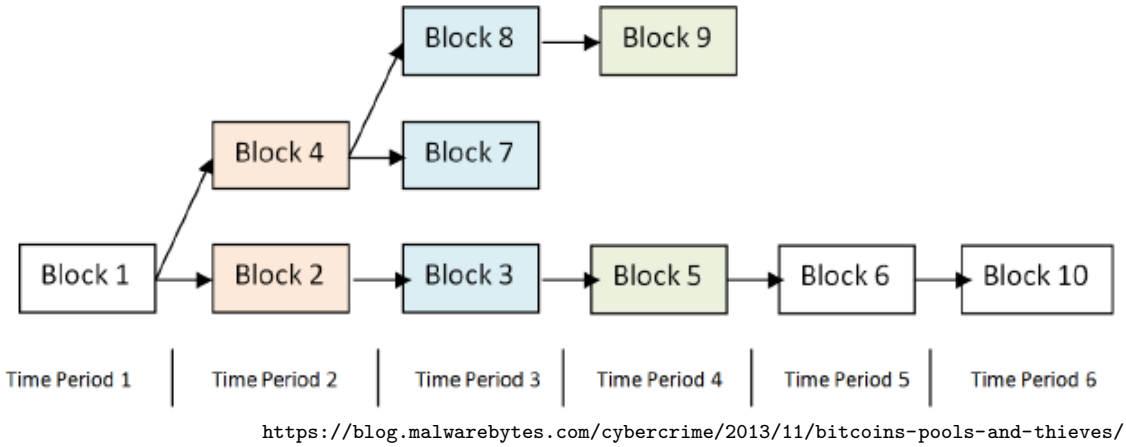
1.5.1. Veikimo principas

Kuriant bloko maišos kodą naudojama blockchain tinklui specifinė maišos funkcija. Į maišos funkciją kaip įvestis perduodamas laikas, transakcijų maišos kodas, praeito bloko maišos kodas ir nereikšminis duomuo, kuris parenkamas atsitiktiniu būdu. Bloko maišos kodas turi prasidėti tam tikru kiekiu pradžioje esančių ”0” simbolių. Reikalaujamas ”0” simbolių skaičius yra vadinamas maišos kodo sudėtingumu ir jis nusakomas tinklo tam, kad nebūtų per greit kuriami blokai (2.pav.). Jei pirmas maišos funkcijos rezultatas netenkina sudėtingumo sąlygos nereikšminis duomuo yra pakeičiamas ir vėl yra ieškomas naujas maišos kodas. Tokį darbą atlieka visi tinklo kasėjai.

Kodėl naudojamas kintantis sudėtingumas? Kuo daugiau kasėjų tuo daugiau kompiuterinio pajėgumo turi tinklas ir dėl to maišos kodas turėtų būti greičiau randamas. Tam kad kontroliuoti naujų blokų radimą (10-15 min) naudojamas kintantis sudėtingumas. Laikas reguliuojamas tam, kad piniginis atlygis už blokų radimą nebūtų per greit dalinamas. Piniginis atlygis bėgant laikui mažėja ir galiausiai jo nebebus ir kasėjai gaus atlygį tik iš mokesčio už transakciją, kurią sumoka, vartotojas kuriantis transakciją. Stabilus atlygio mažėjimas skatina naujų mazgų prisijungimą kol nėra didelio transakcijų skaičiaus, kuris atlygintų kasėjų darbą. Bitcoin atveju paskutinis bloko atlygis planuojamas apie 2140 m. [gee]

2 pav. 2018-04-12 13:57:20 Bitcoin bloko informacija

1.5.2. Grandinės išsišakojimas



3 pav. Blokų grandinės išsišakojimas

Labai retais atvejais gali atsitikti taip, kad maždaug tuo pačiu metu yra randami du validūs blokai (du tinkami maišos kodai) (3.pav.). Tokiu atveju blokų grandinė išsišakoja į dvi taip pat validžias grandis. Taip grandinėje atsiranda du galimi praeito bloko maišos kodai. Tam, kad suprastume kaip randama pagrindinė šaka paanalizuokime pavyzdį. Tarkime einamasis blokas yra 100-asis vienu metu yra sukuriami nauji blokai 101a ir 101b, kuriuos atitinkamai sukūrė adresas A ir adresas B. Tarkime sekančiam blokui kurti 50% tinklo dalyvių pasirenka šaką 101a kita 50% dalis pasirenka šaką 101b. Pagal tai, kuri dalis pirma suras naująjį bloką bus nuspręsta kuri šaka bus pagrindinė. Jei randamas blokas 102b visi tinklo dalyviai "atsisako" šakos 101a ir naudoja 102b šaką naujam blokui kurti. Yra labai maža tikimybė, kad vėl tuo pačiu metu bus sukuriami du blokai 102a ir 102b arba naujas išsišakojimas 102aa 102ab arba abu variantai kartu, naujos šakos: 102aa 102ab ir 102b. Tokiais atvejais yra tiesiog vėl atliekamas tas pats procesas pagrindiniai šakai atpažinti su keliomis senojo bloko galimybėmis, tačiau toks variantas yra ypatingai retas. [Ros]

Kas atsitinka šakai, kurios atsisakoma? Tarkime tokią situaciją:

```
a: 100 -> 101a -> 102a -> 103 -> 104 ...
b: 100 -> 101b -> 102b
```

Kadangi "a" išsišakojimas tapo ilgesnis pagrindinė šaka tapo "a" šaka, o "b" šakos buvo atsisakyta. Tokiu atveju atsitinka keletas dalykų.

- Atlygis - tiek 101b tiek 102b kasėjai praranda savo atlygį kadangi grandinė, kurioje egzistuoja jų prijungi blokai ir už tai sumokėtas atlygis nepripažįstama kaip pagrindinės grandinės dalis. Dėl šios priežasties yra rekomenduoja gavus mokėjimą, kuriame yra naudojamas nesenai rasto bloko atlygis, palaukti keleto blokų atradimų tam, kad įsitikinti, jog atlygis pateko į pagrindinę blokų grandinę ir mokėjimas nebus atsisakytas.

- Transakcijos - galimas atvejis kad išsišakojusios grandys įtraukia nevienodą einamųjų transakcijų skaičių (dėl per mažo kasimo mokesčio) pavyzdžiui 101a bloko kasėjas įtraukia 50

transakcijų o 101b įtraukia 52 transakcijas. Kadangi dviejų transakcijų buvo atsisakyta, jos nebeegzistuoja grandinėje. Dėl to yra rekomenduojama taip pat palaukti bent 6 naujų blokų sukūrimo, kad įsitikinti, jog mokėjimas liks grandies istorijoje ir gavėjas nepraras pinigų po mokėjimo patvirtinimo.

1.5.3. Alternatyvos

Pagrindinis įrodymo darbu trūkumas, kad norint kasėjui padidinti šansą rasti naują bloką, jam reikia turėti daugiau kompiuterinės galios. Tai sukuria kompiuterinės galios lenktynes tarp visų tinklo kasėjų. Toks metodas sunaudoja labai daug nereikalingos elektros energijos ir laikomas neekonomišku ir neekologišku, todėl atsirado papildomų konsensuso radimo alternatyvų.

- Įrodymas įtaka (angl. "Proof of stake") - antras pagal populiarumą konsensuso radimo algoritmas. Jis suteikia kasėjui didesnį šansą surasti naują bloką, kuo daugiau kasėjas turi tinklo pinigų. Vietoje to, kad kasėjas didintų šansą pirkdamas papildomą kompiuterinę galią. Kasėjas gali tiesiog daugiau investuoti į tinklą, taip taupydamas elektrą ir kaštus kompiuterijos priežiūrai. Taip pat yra logiška manyti, kad kuo daugiau pinigų investavęs kasėjas tuo mažiau turės noro kenkti sistemai.

- Įrodymas deleguota įtaka (angl. "Proof of delegated stake") - Šis algoritmas praplečia įrodymo įtaka algoritmą. Tinklo kasėjai turi balsą, kurį skiria tam, kad išrinktų, kurie iš daugiau pinigų turinčių kasėjų turėtų padidintą šansą rasti naują bloką. Tai padeda išvengti tokios situacijos, kad daugiausiai pinigų turintis kasėjas, gali turėti didžiausią įtaką tinkle.

- Įrodymas pinigų deginimu - (angl. "Proof of burn") Šis algoritmas suteikia didesnį šansą tiems kasėjams, kurie sumoka didesnę kainą už tai, kad galėtų kasti naujus blokus.

- Įrodymas talpa - (angl. "Proof of capacity") algoritmas, kuris pakeičia papildomos kompiuterijos pirkimą už padidintą bloko radimo galimybę į papildomos skaitmeninės talpos pirkimą. Kuo daugiau talpos paskiria kasėjas, tuo didesnį šansą turi atrasti naują bloką.

[Sha18]

1.6. Pagrindinės blockchain technologijos savybės

1.6.1. Duomenų integralumas

- Naujų blokų integralumas - Sukūrus naują bloką, kuris tenkina sudėtingumo sąlygą, galima lengvai patikrinti ar jis yra korektiškas. Tai atlieka kiti tinklo dalyviai tikrindami nevalidžių transakcijų (pakeistų arba naujai sukurtų), kadangi apie naujas transakcijas yra informuojami visi tinklo dalyviai. Tai įgalina transakcijų apsaugą einamajame bloke.

- Transakcijų istorijos integralumas - Panašiai kaip ir su einamosiomis transakcijomis nauji blokai priklauso nuo praeito bloko maišos kodo. Įsivaizduokime situaciją, kad sukuriant naują bloką kasėjas suskaičiuoja naują maišos su teisingomis einamosiomis transakcijomis,

tačiau pakeičia buvusio bloko transakcijų istorija, randą naują praeito bloko maišos kodą su suklastotomis transakcijomis ir tą maišos kodą panaudoja naujo bloko radimui. Tinklo vartotojai lengvai gali pastebėti tokia klaidą, nes blokų grandinė yra vieša ir tinklo mazgai gali patikrinti ar įmanoma gauti tokią suklastotą maišos reikšmę su nepakeista blokų istorija. Tai užtikrina istorijos integralumą naujų blokų kūrime. [gee]

1.6.2. Saugumas

- Vartotojo duomenų modifikacija - duomenų modifikacija tinkle yra uždrausta, todėl yra neįmanoma, kenkėjui pakeisti vartotojo pinigus (transakcijų istoriją).

- Dvigubo pinigų išleidimo problema - Bitcoin pirmajam pavyko išspręsti dvigubo išleidimo problemą pasitelkus blockchain technologiją.

  Įsivaizduokime tokią situaciją vartotojas A turi 100 pinigų ir sukuria transakciją su kuria perveda 80 pinigų iš A į B, ir šia transakciją transliuoja du kartus vienu metu. Kaip atpažinti kad įvyko klaida? Abi transakcijos validžios ir vartotojui pinigų netrūksta.

  Blockchain tinkle naujų blokų kūrime naujas transakcijos pridedamos paeiliui, taip kad pirmoji transakcija, kuri būna prijungta vadinama validžia, o sekanti būna atmetama dėl vartotojo pinigų trūkumo.

- Tinklo suklastojimas - Blockchain tinkle problemų sprendimai yra priimami, tie kurie surenka daugiau nei 51% dalyvių balsų. Jeigu kenkėjas nori priimti nevalidų sprendimą tinkle (pvz. patvirtinti klastota transakcijų istoriją), jam reikėtų turėti didesnę kompiuterinę galią nei pusė blockchain tinklo, kas yra praktiškai neįmanoma.

  Dėl šių priežasčių blockchain tinklui dar niekada nebuvo sėkmingai pakenkta. Sėkmingos atakos yra pavykusios tik piniginių pasisavinimui už kurių saugumą yra atsakingos kompanijos, kuriančios piniginių klientines aplikacijas. [IBM]

1.6.3. Skaidrumas

Visa bloku grandinės istorija yra viešai pasiekiama tai užtikrina žmonių pasitikėjimą. Lyginant su tradicinėmis pinigų sistemomis paprastam vartotojui yra labai sunku arba neįmanoma atsekti iš kur pinigai buvo sukurti ir ar jo mokėjimas buvo užregistruotas, patvirtintas ir taip toliau. pavyzdžiui Bitcoin transakcijų/blokų istoriją galima patikrinti nuėjus į **https://blockchain.info**.

1.7. Ethereum

Antras pagal populiarumą blockchain tinklas yra vadinamas Ethereum. Jis buvo sukurtas praplėsti Bitcoin tinklo galimybes tam, kad tinkle įgalinti sudėtingesnius veiksmus pavyzdžiui periodinės išmokos, skaitmeniniai aukcionai, teisių į pinigus keitimas, projektų finansavimas ir taip toliau. Tokiam funkcionalumui įgalinti tinkle galima sukurti programas vadinamus išmaniaisiais kontraktais (angl. "Smart Contracts"). Šie kontraktai patalpinti į tinklą sukompiliuojami į mašininį kodą ir naujam blokui generuoti tinklo dalyviai turi saugoti ne tik transakcijų istoriją bet ir

galutines į tinklą patalpintų kontraktų būsenas. Transakcijos šiame tinkle gali tiek pervesti pinigus, tiek būti naudojamos kaip būdas iškviesti programos metodus. Taip įgalinama interakciją su tinklo programomis. Kontraktai kaip ir tinklo vartotojai turi adresus į kurios galima pervesti tinklo pinigus, taip kontraktas įgauna galimybę automatizuoti pinigų procesus. Gilesniam supratimui paanalizuokime konkretų kontraktą.

```
contract MoneyDoubler {

    function receiveFunds() payable public
    {
      // metodas nieko nedaro, tik priima pinigus
    }

    function doubleMoney() payable public
    {
        uint amount = msg.value * 2;
        msg.sender.transfer(amount);
    }

}
```

Kontrakto idėja labai paprasta. Vartotojas sukuria kontraktą, kuris turi du metodus, vienas pinigams įdėti į kontraktą, kitas vartotojui pasidvigubinti pinigus. Jei vartotojas iškviesdamas kontrakto metodą ”doubleMoney” kontraktui nusiunčia 1 vertės vienetą, tai kontraktas į jo adresą perveda 2 piniginius vienetus, su sąlyga kad kontraktas turi pakankamą kiekį pinigų ”nuostoliui” padengti. pavyzdys:

	Sąskaitų įverčiai		
Operacija	Kontraktas	Vartotojas A	Vartotojas B
Sukuriamas kontraktas	0	100	30
Vartotojas A iškviečia kontrakto metodą "receiveFunds" su 80 pinigų transakcijoje	80	20	0
Vartotojas B iškviečia kontrakto metodą "doubleMoney" su 30 pinigų transakcijoje	50	20	60
Vartotojas B iškviečia kontrakto metodą "doubleMoney" su 50 pinigų transakcijoje	0	20	110

4 pav. Operacijų scenarijus

pavyzdys (4.pav.) iliustruoja kaip ethereum kontraktai įgalina programuojamas pinigų operacijas patikimoje blockchain tinklo aplinkoje. Didžiausias Ethereum tinklo privalumas tai, kad yra užtikrinamas ne tik duomenų integralumas (kaip Bitcoin tinkle) bet ir programuojamų piniginių operacijų integralumas. Jei kontraktas buvo sukurtas blockchain tinkle tai reiškia, kad jo veikimas bus amžinas ir nekeičiamas. Tai leidžia programuotojams ir įmonėms kurti pinigines operacijas, kuriomis žmonės gali pasitikėti dėl Ethereum tinklo integralumo.

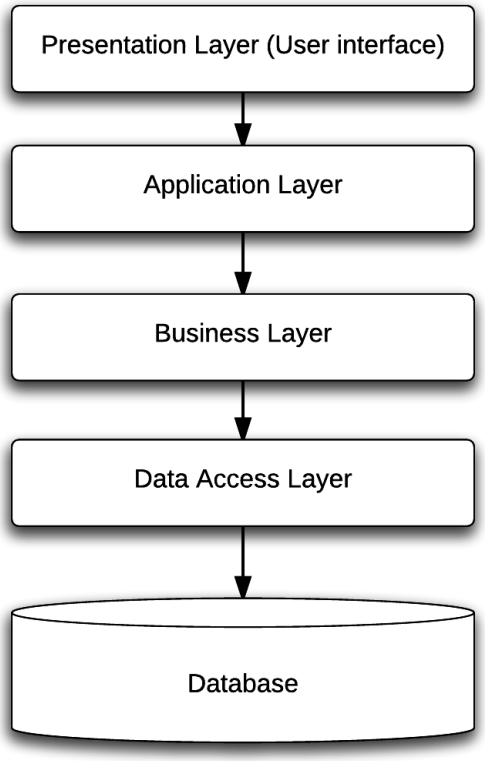
Šis darbas sekančiuose skyriuose naudos Ethereum tinklą integracijų ir pritaikymo pavyzdžiams iliustruoti, dėl jo populiarumo ir lankstumo programuojant išmaniuosius kontraktus. [Mar18]

2. Mikroservisų architektūra

2.1. Atsiradimas

Mikroservisų architektūra natūraliai atsirado iš poreikio labiau skaldyti augančias didelio masto aplikacijas. Manoma, kad pirmą kartą terminas ”mikroservisai” buvo paminėtas Venecijoje gegužės mėnesį, per tada vykusį programų architektų praktinių darbų užsiėmimą. Taip buvo pavadintas programų architektūros stilius, kurį tuo metų, daug kas nagrinėjo, tačiau tai neturėjo tikro pavadinimo. Toliau architektūros stilius buvo analizuojamas moksliniuose darbuose kaip ”Microservices - Java, the Unix Way”, parašyto James Lewis, ”Fine grained SOA” parašyto Adrian Cockcroft ir t.t. [Bea]

2.2. Monolitinė architektūra



https://dzone.com/articles/splitting-a-monolith-application-into-services

5 pav. Monolitinė architektūra

Monolitinės architektūros stilius (5.pav.), kuris šiuo metu yra populiariausias programų kūrimo stilius, tačiau turi nemažai ribojimų, kurie augant programos mastui ir sudėtingumui, lėtina programos veikimą bei jos programavimo ir priežiūros procesą.

Pagrindinė monolitinės architektūros savybė yra tai, kad visas programos funkcionalumas yra sudedamas į vieną aplikaciją. Programoje atskiri sistemos moduliai gali būti skaidomi į atskirus komponentus, tačiau visi komponentai yra laikomi vienoje programoje. Tai turi savo pliusų.

- **Programos diegimas** (angl. ”deployment”). Užtenka paleisti vieną aplikaciją ir visos sistemos funkcionalumas yra pasiekiamas.

- **Paprastumas**. Aplikacijai esant mažo ir vidutinio dydžio, kadangi visas funkcionalumas yra pasiekiamas toje pačioje aplikacijoje nereikia rūpintis tinklo problemomis kaip, užklausos/atsako išsiuntimo greitis, pastovaus ryšio palaikymas su vidiniais komponentais ir t.t.

- **Populiarumas**. Yra lengva rasti efektyvių programuotojų, todėl kad tai yra populiariausias programavimo stilius, su kurio yra susipažinę beveik visi dirbantys programuotojai.

Šis stilius yra geras, kuriant nedideles aplikacijas, bet aplikacijai išaugus atsiranda tam tikrų ribojimų.

- **Plečiamumas**. Kadangi vienintelis būdas praplėsti monolitinę aplikaciją yra diegti daugiau monolitinės aplikacijos vienetų, plečiamumas tampa brangus ir neefektyvus. Dažniausiai sistema yra plečiama tam, kad patenkinti padidėjusį užklausų skaičių, kurios būna skirtos vienam sistemos moduliui, t.y. sistemos plėtimas reikalingas dėl mažos sistemos dalies padidėjusio naudojimo. Kadangi nėra būdo kaip praplėsti vieną sistemos modulį yra plečiama visa sistema (padidinamas diegtų aplikacijų skaičius skaičius). Tai padidina ir tų sistemos modulių skaičių, kurių naudojimas nepakito, sumažinant plėtimo efektyvumą.

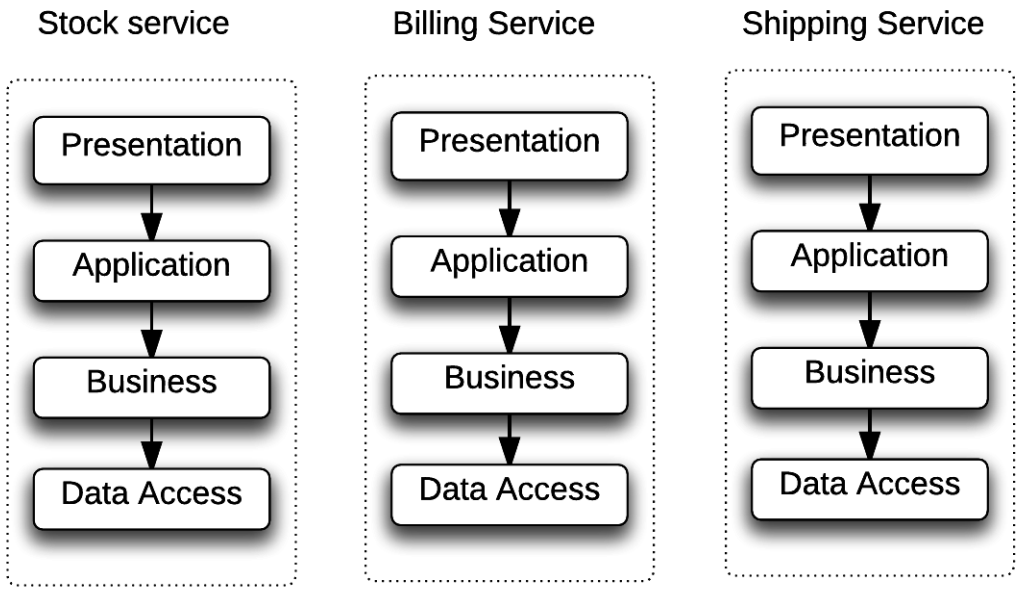
- **Technologinis keičiamumas**. Sukuriant monolitinę aplikaciją yra ”prisirišama” prie tam tikrų technologijų su kuriomis aplikacija buvo parašyta. Aplikacijos technologijas atnaujinti arba naudoti kitas technologijas tampa sudėtinga, nes tai dažniausiai reikalauja pakeitimų visoje aplikacijoje, kas dažniausiai būna labai daug darbo ir aplikacijos dažnai lieka neatnaujintos ir technologijos niekada nekeičiamos nors tai galimai duotų daug naudos.

- **Lėtas diegimas**. Šiuo metu populiarus pastovus diegimas (angl. ”Continuous deployment”) tampa sudėtingas, nes aplikacijos inicijavimas trunka ne mažą laiką (dažnai apie 5 min). Norint diegti aplikaciją pavyzdžiui kelis kartus per valandą yra prarandamas aplikacijos veikimo laikas arba reikia naudoti sudėtingesnes diegimo strategijas‘’ strategijas, kas reikalauja papildomų resursų.

- **Lėtas programavimo ciklas**. Didelio masto programos yra per sudėtingos pilnai suprasti ir kurti naujus ir teisingus pakeitimus. Taip pat programavimo metu pakeitimų peržiūra užtrunka ne mažą laiką, nes reikia laukti kol iš naujo inicijuojasi aplikacija.

[18]

2.3. Mikroservisų architektūros pagrindinės savybės



https://dzone.com/articles/splitting-a-monolith-application-into-services

6 pav. Mikroservisų architektūra

Kadangi mikroservisų architektūros stilius (6.pav.) atsirado iš pragmatiškų paskatų, griežto mikroservisų architektūros apibrėžimo nėra, bet architektūrą galimą laikyti į servisus orientuotos architektūros (angl. "Service oriented architecture") poaibiu. Mikroservisų architektūra modeliuoja sistemą kaip daug mažų, išskaidytų, vienas nuo kito nepriklausomai diegiamų servisų (savarankiškų programų). Servisai yra sistemos komponentai turintys konkretų funkcionalumo įgyvendinimą, pavyzdžiui atsiskaitymų servisas, inventoriaus servisas, statistikos rinkimo servisas ir t.t. Servisai dažniausiai komunikuoja tarpusavyje HTTP protokolu. Pagrindinis skirtumas nuo į servisus orientuotos architektūros yra tai, kad sistemos servisai turi galėti būti diegiamas atskirai, skirtingai nuo į servisus orientuotos architektūros, kur sistemos servisai gali būti ir vieno aplikacijoje, ir diegiami atskirai ar grupėmis.

Kaip buvo minėta mikroservisų architektūra natūraliai atsirado tam, kad išspręstų problemas, kurios atsiranda augant tradicinės monolitinės architektūros stiliaus aplikacijai. Palyginsime kaip monolitinės architektūros ribojimai yra išsprendžiami mikroservisų architektūroje. [Bea]

2.3.1. Plečiamumas

Kadangi sistema yra išskaidyta į izoliuotus funkcionalumo vienetus yra lengva praplėsti tą sistemos vietą (funkcionalumą), kurį reikia, nepadidinant kitų servisų skaičiaus. Paprasčiausiai yra padidinamas serviso diegtų aplikacijų skaičius, kuris gali būti viename serveryje arba išskaidytas į nutolusius serverius. [aca]

2.3.2. Technologinis keičiamumas

Dėl sistemos išskaidymo yra lengva įtraukti naujas technologijas į sistemą, nes vienintelis reikalavimas naujam servisui yra tai, kad jis turi komunikuoti sistemos protokolu (dažniausiai HTTP). Sistemos atnaujinimas (naudojamų technologijų versijos padidinimas) yra taip pat nesudėtingas nes jis gali būti iteratyvus, t.y. atnaujinant sistemą servisas po serviso. [aca]

2.3.3. Lėtas diegimas

Kadangi sistemos servisai gali būti diegiami atskirai pakeitus sistemos servisą, jį galima diegti nepriklausomai nuo likusios sistemos. Tai įgalina populiarią pastovią diegimo strategiją, kur sistemos servisas yra diegiamas kai naujas sistemos pakeitimas yra prijungiamas prie sistemos kodo bazės. [aca]

2.3.4. Lėtas programavimo ciklas

Kadangi yra dirbama su mažai izoliuotais sistemos vienetais programavimo ciklas tampa greitesnis ir paprastesnis. Servisas dažniausiai nėra didelės apimties, todėl jį pilnai suprasti yra paprasta ir padaryti reikalinga sistemos pakeitimą yra greičiau. Taip pat programavimo metu patikrinti pakeitimus užtrunka neilgai nes iš naujo startuojamo serviso inicijavimo laikas yra žymiai mažesnis palyginus su monolitine aplikacija. [aca]

3. Blockchain technologija mikroservisų architektūroje

Blockchain integracija su mikroservisų architektūros sistema yra panaši į integraciją su monolitine aplikacija. Pagrindiniai skirtumai, kuriuos turime paisyti yra tai, kad turime išlaikyti mikroserviso paprastumą (kitaip tariant mažą funkcionalumo sritį) bei išlaikyti mikroserviso nepriklausomo diegimo galimybę. Reikia išskirti kriterijus, kurie taps minimaliais sistemos reikalavimais tam, kad integracija nepakenktų tiek mikroservisų architektūros tiek blockchain technologijos teikiamiems privalumams.

- Blockchain technologijos naudai palaikyti reikalingi kriterijai:
  - Viešas duomenų prieinamumas
  - Duomenų nekintamumas
- Mikroservisų architektūros naudai palaikyti reikalingi kriterijai:
  - Nepriklausomas diegimas
  - Siaura funkcionalumo sritis
  - Komunikacijos su kitais sistemos mikroservisais užtikrinimas

Kriterijai sekančiuose skyriuose bus naudojami taikymo pavyzdžių trūkumams ir privalumams vertinti. [Dra]

4. Taikymo pavyzdžiai

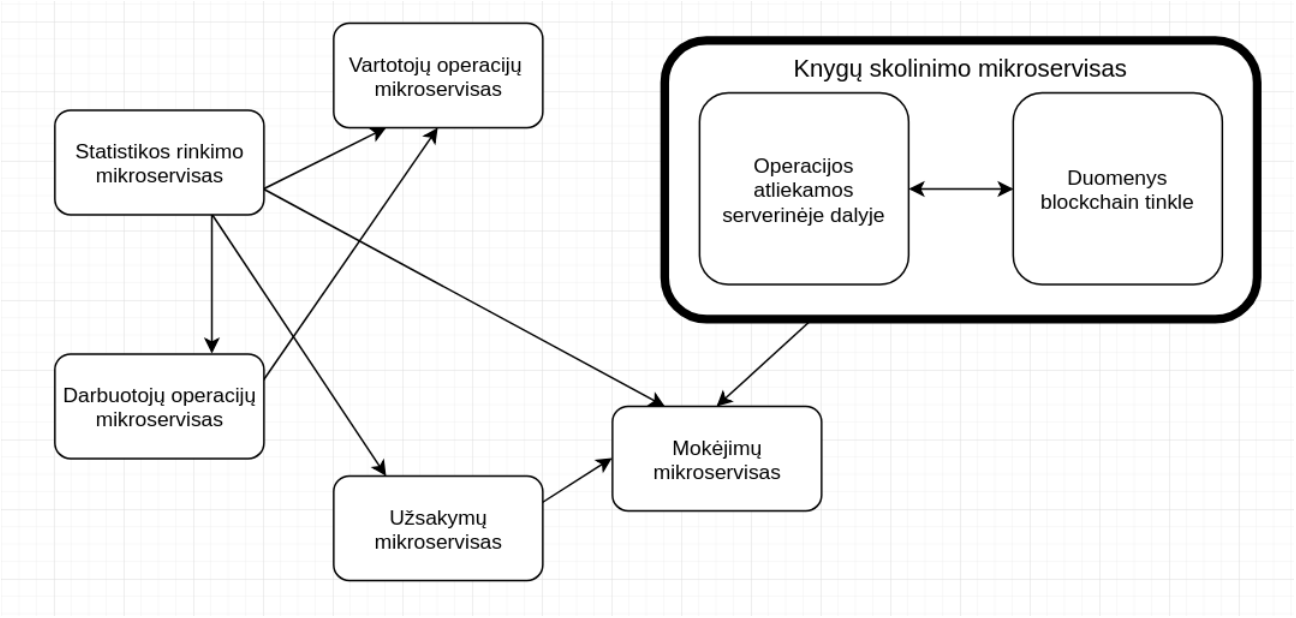
Iliustruoti pavyzdžiams buvo pasirinkta įgyvendinti bibliotekos registro teikiamą funkcionalumą panaudojus blockchain technologijos galimybes. Pagrindinis registro funkcionalumas:

- Paskolinti knygą.
- Grąžinti knygą.
- Patikrinti paskolintos knygos būseną.

4.1. Duomenų saugojimas blockchain tinkle

Pirma bus analizuojamas blockchain technologijos pritaikymas kaip duomenų saugyklos (7.pav.). Tam, kad būtų lengviau iliustruoti teikiamą naudą buvo parašytas išmanusis kontraktas realizuojantis saugojimo galimybę Ethereum blockchain tinkle (Priedas nr. 1). Parašytas kontraktas turi 3 funkcijas, kurios realizuoja primityvaus registro funkcionalumą. Visos šios funkcijos operuoja ”LendingRecord” duomenų struktūra, kuri yra taip pat sukurta ir saugojama ethereum tinkle. Funkcijos:

- ”addRecord” (pridėti knygos paskolinimo įrašą) - funkcijai yra pateikiama operuojamos duomenų struktūros ”LendingRecord” laukai, tam kad būtu sukurtas paskolinimo įrašas ir jis išsaugotas į blockchain tinklo atmintį. Reikėtų paminėti, kad ši operacija yra mokama nes yra keičiama kontrakto būsena, todėl reikia transliuoti kontrakto pakeitimus visam tinklui. Kaip buvo minėta ankstesniuose skyriuose tokie tinklo atnaujinimai kainuoja tam, kad tinklo kasėjai turėtų iniciatyvą palaikyti stabilų sistemos darbą.
- ”removeRecord” (panaikinti knygos paskolinimo įrašą) - funkcijai yra paduodamas paskolintos knygos įrašo indeksas ir pagal tą indeksą yra panaikinamas įrašas esantis blockchain tinklo atmintyje. Taip pat operacija yra mokama dėl priežasčių išvardintų anksčiau.
- ”getRecord” (grąžinti knygos paskolinimo įrašą) - funkcija pagal pateiktą indeksą grąžina pasiskolinimo įrašą, jei įrašo su pateiktu indeksu nėra vartotojui arba sistemai yra grąžinama klaida.



7 pav. Verslo logika atliekama serverinėje dalyje

4.1.1. Naudojimas

Kadangi blockchain tinkle yra aprašomos tik primityvios duomenų valdymo funkcijos sistemos modulio logika yra laikoma serverinėje dalyje. Serveris su blockchain tinkle patalpintais duomenimis gali bendrauti su pagalba įvairių bibliotekų, kurios palengvina Ethereum JSON-RPC protokolo naudojimą.

Sistemos būsena yra palaikoma taip. Įvykus operacijai, kuri keičia sistemos būsena serverinė dalis informuoja apie pokyti blockchain tinkle esantį kontraktą ir kontraktas atnaujina savo būseną. Serverinė dalis yra atsakinga už korektiškų duomenų palaikymą patalpintame kontrakte.

4.1.2. Privalumai ir Trūkumai

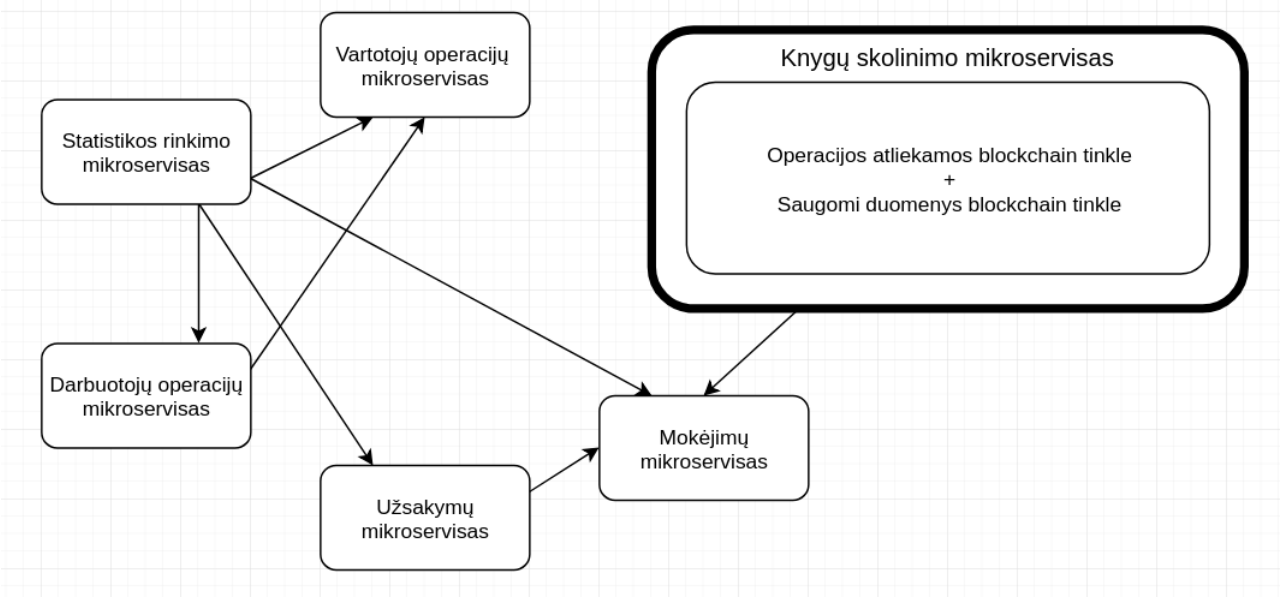
- Saugumas - Duomenis esančius blockchain tinkle suklastoti yra praktiškai neįmanoma.
- Viešas duomenų prieinamumas - Sistemos būsenų pokyčiai yra viešai prieinami, kas leidžia vartotojams labiau pasitikėti sistema.
- Komponentų paskirties aiškumas - Mikroservisas yra sudarytas iš dviejų komponentų, kurie atlieka aiškiai apibrėžtas užduotis.
- Kaina - Dalis operacijų mokamos, todėl būsenos keitimo operacijas reikia atlikti kuo rečiau. Taip pat kuo daugiau duomenų yra saugojama blockchain tinkle tuo kaina didesnė, todėl kiekvienas kontrakto praplėtimas padidina operacijų išlaidas.
- Sudėtingas programos koregavimas - Kadangi kontraktai yra nekintami yra sudėtinga padaryti kodo pakeitimus, todėl reikia sukurti strategiją kaip bus koreguojama programa, klaidai pastebėjus.

4.2. Operacijų perkėlimas į blockchain tinklą

Sekančiu pavyzdžiu (8.pav.) analizuosime mikroserviso operacijų iškėlimą į blockchain aplinką tam, kad dar labiau padidinti vartotojų pasitikėjimą sistema. pavyzdžio iliustracijai parašytas mikroserviso pavyzdys (Priedas nr 2). Šiame pavyzdyje yra saugomi tiek modulio duomenys tiek funkcionalumo operacijos blockchain tinkle.

Parašytas kontraktas yra praplėstas pirmojo pavyzdžio kontraktas (Priedas Nr. 1), jis yra praplėstas su funkcijomis, kurios yra atsakingos už verslo logikos įgyvendinimą. Papildomos funkcijos:

- ”lendBook” (funkcija kviečiama skolinant knygą vartotojui) - Skolinant knygą funkcija yra iškviečiama tam, kad nustatytų knygos gražinimo datą ir įtraukų skolinimo įrašą į blockchain tinklo atmintį panaudojus funkciją ”addRecord”, kuri buvo aptarta ankstesniame pavyzdyje.
- ”returnBook” (funkcija kviečiama grąžinant knygą atgal į registrą) - Grąžinant knygą funkcija yra iškviečiama tam, kad patikrintų knygos gražinimo datą ir pagal tai nuspręstų ar reikia sumokėti baudą. Jei knyga yra grąžinama nelaiku kviečiant grąžinimo operaciją reikia sumokėti baudą su knygos grąžinimo funkcijos kvietimu ir tik tada yra panaikinamas knygos skolinimo įrašas.



8 pav. Verslo logika atliekama blockchain tinkle

4.2.1. Naudojimas

Patalpintas kontraktas turi pilną verslo logikos įgyvendinimą, todėl tokią blockchain technologijos pritaikymo strategiją galima naudoti dvejopai:

1. **Su serverine dalimi** - Mikroservisas gali turėti tradicinę serverinę dalį, tam kad pertransliuotų įvykusias operacijas į blockchain esantį kontraktą. Šiuo būdu mes yra sutaupomą infrastruktūros išlaidos, kadangi kiekviena operacija Ethereum tinkle yra mokama, todėl

yra naudinga iškelti dalį operacijų į serverinę dalį taip sumažinant tiek Ethereum kontrakto naudojimą, tiek operacijų kompiuterinės galios naudojimą. Papildomai tai leidžia išvengti komunikacijos skirtingais protokolais tarp egzistuojančių mikroservisų, kadangi standartiškai yra naudojama komunikacija HTTP JSON protokolu, o su komunikacijai su kontraktais esančiais Ethereum tinkle reikia naudoti JSON-RPC protokolą.

2. **Be serverinės dalies** - Kaip minėta Ethereum kontraktas gali turėti visą verslo logikos įgyvendinimą, todėl galima pilnai atsisakyti serverinės dalies ir interpretuoti kontraktą esantį Ethereum tinkle kaip savarankišką mikroservisą. Tai supaprastina mikroserviso architektūrą paliekant tik vieną komponentą. Papildomas tokio naudojimo privalumas yra tai, kad kontraktas yra viešas todėl pasirinktinai galima leisti ne sistemos vartotojams naudotis kontrakto teikiamu funkcionalumu.

4.2.2. Privalumai ir Trūkumai

Pagrindiniai privalumai ir trukumai išlieka tokie patys kaip ir ankstesniame pavyzdyje, tačiau yra keletas skirtumų, kuriuos reikia paminėti.

- Viešumas - Mikroserviso viešumas yra dar labiau padidinamas naudojant tokia taikymo strategiją, nes vartotojas gali matyti veikiančios programos ne tik duomenų istoriją, bet ir labiau pasitikėti sistema, kuri yra įkelta į nekeičiamą aplinką.

- Kaina - Kadangi yra iškeliama daugiau funkcionalumo į Ethereum tinklą, tai padidina kainą dėl padidėjusios kompiuterinės galios naudojimo blockchain tinkle.

- Sudėtingas programos koregavimas - Dėl padidėjusios funkcionalumo dalies iškėlimo padidėja ir poreikis keisti sistemą aptikus klaidą, nes didesnėje kodo bazėje yra dažniau aptinkamos programavimo klaidos.

Rezultatai ir išvados

Šis darbas parodo, kad blockchain technologija gali būti pritaikyta mikroservisų architektūroje nepažeidžiant pagrindinių mikroservisų architektūros idėjų. Tačiau toks taikymas reikalauja pasirinkti tinkamą taikymo strategiją pagal keliamus funkcionalumo reikalavimus, nes norint užtikrinti vartotojų pasitikėjimą ir sistemos komponento saugumą, yra aukojamas programos atnaujinimų lengvumas bei sistemos palaikymo kaina dėl būtinų išlaidų naudojant blockchain tinklą. Tikslui pasiekti buvo atliktos šios užduotys.

1. Atlikta blockchain technologijos analizė. Įvardintas veikimo principas, pagrindiniai komponentai, išanalizuotas bendro sprendimo priėmimo (konsensuso radimo) algoritmas bei jo alternatyvos, pateikiami pagrindiniai blockchain technologijos taikymo metodai, projektai.

2. Atlikta mikroservisų architektūros analizė pagrindinėms savybės apibrėžti, palyginta su monolitine serverio architektūra.

3. Pateiktas blockchain technologijos taikymo pavyzdys kai jautrus duomenys yra iškeliami į blockchain tinklą, nurodytos naudojimo rekomendacijos, taikymo trukumai ir privalumai.

4. Pateiktas blockchain technologijos taikymo pavyzdys kai verslo logika ir sistemos duomenys iškeliami į blockchain tinklą, nurodytos naudojimo rekomendacijos, taikymo trukumai ir privalumai.

Literatūra

[18] Clickbait: Microservices are bad! So are monoliths. 2018-01-02. URL: https://oprea.rocks/blog/microservices-are-bad-so-are-monoliths (tikrinta 2018-05-03).

[aca] Cloud academy. Microservices architecture: advantages and drawbacks. URL: https://cloudacademy.com/blog/microservices-architecture-challenge-advantage-drawback/ (tikrinta 2018-05-03).

[Bea] Smart Bear. What is Microservices Architecture? URL: https://smartbear.com/learn/api-design/what-are-microservices/ (visited on 2018-05-03).

[bit] bitnodes. Global Bitcoin nodes distribution. URL: https://bitnodes.earn.com/ (tikrinta 2018-05-26).

[Dra] Lautaro Dragan. Microservices and the Blockchain. URL: https://blog.po.et/microservices-and-the-blockchain-68413a208b90 (tikrinta 2018-05-03).

[gee] Block geeks. What Is Hashing? Under The Hood Of Blockchain. URL: https://blockgeeks.com/guides/what-is-hashing/.

[IBM] IBM. Blockchain security: What keeps your transaction data safe? - Blockchain Unleashed: IBM Blockchain Blog. URL: https://www.ibm.com/blogs/blockchain/2017/12/blockchain-security-what-keeps-your-transaction-data-safe/ (tikrinta 2018-05-03).

[Mar18] Jon Martindale. Everything you need to know about one of the most exciting cryptocurrencies. 2018-01-31. URL: https://www.digitaltrends.com/computing/what-is-ethereum-blockchain-cryptocurrency/ (visited on 2018-05-03).

[min] Genesis mining. The history of bitcoin & how bitcoin is used | genesis mining. URL: https://www.genesis-mining.com/the-history-of-bitcoin.

[Ros] Meni Rosenfeld. Analysis of hashrate-based double-spending.

[Sha18] Toshendra Kumar Sharma. What are the alternative strategies for proof-of-work? 2018-01-25. URL: https://www.blockchain-council.org/blockchain/what-are-the-alternative-strategies-for-proof-of-work/ (tikrinta 2018-04-13).

Priedas nr. 1

Solidity programavimo kalba parašytas duomenų saugojimo blockchain tinkle kontraktas

```
pragma solidity ^0.4.18;

contract LibraryRegistrar {

    struct LendingRecord {
        string fullName;
        uint bookISBN;
        uint returnDate;
    }

    LendingRecord[] public lendedBooks;

    function addRecord (string fullName, uint bookISBN, uint returnDate) public {
        lendedBooks.push(LendingRecord({
            fullName: fullName,
            bookISBN: bookISBN,
            returnDate: returnDate
        }));
    }

    function removeRecord (uint index) public {
        if (index >= lendedBooks.length) return;

        for (uint i = index; i<lendedBooks.length-1; i++){
            lendedBooks[i] = lendedBooks[i+1];
        }
        lendedBooks.length--;
    }

    function getRecord (uint index) view public returns (
        string fullName,
        uint bookISBN,
        uint returnDate
    ) {
        return (
          lendedBooks[index].fullName,
          lendedBooks[index].bookISBN,
          lendedBooks[index].returnDate
        );
    }
}
```

Priedas nr. 2

Solidity programavimo kalba parašytas bibliotekos registro funkcionalumo įgyvendinimo blockchain tinkle kontraktas

```
pragma solidity ^0.4.18;

contract LibraryRegistrarV2 {

    struct LendingRecord {
        string fullName;
        int bookISBN;
        int returnDate;
    }

    int FINE_PER_DAY = 1000;
    int DAY = 60 * 60 * 24;

    LendingRecord[] public lendedBooks;

    function addRecord (string fullName, int bookISBN, int returnDate) public {
        lendedBooks.push(LendingRecord({
            fullName: fullName,
            bookISBN: bookISBN,
            returnDate: returnDate
        }));
    }

    function removeRecord (uint index) public {
        if (index >= lendedBooks.length) return;

        for (uint i = index; i<lendedBooks.length-1; i++){
            lendedBooks[i] = lendedBooks[i+1];
        }
        lendedBooks.length--;
    }

    function getRecord (uint index) view public returns (
        string fullName,
        int bookISBN,
        int returnDate
    ) {
     return ( lendedBooks[index].fullName, lendedBooks[index].bookISBN, lendedBooks[index].returnD
    }

    function getFine(int returnDate) view public returns (int) {
        int daysLate = (int(block.timestamp) - int(returnDate)) / DAY;
```

```
        if(daysLate <= 0) {
            return 0;
        } else {
            return int(daysLate * FINE_PER_DAY);
        }
    }

    function returnBook(int bookISBN) public payable returns (string message, int fine){
        for(uint i = 0; i < lendedBooks.length; i++) {
            LendingRecord storage record = lendedBooks[i];
            if(record.bookISBN == bookISBN) {
                fine = getFine(record.returnDate);
                if(int(msg.value) == fine) {
                    removeRecord(i);
                    return ("book returned", 0);
                } else {
                    return ("you must pay a fine", fine);
                }
            }
        }
        return ("book was not lended", 0);
    }

    function lendBook(string fullName, int bookISBN) public {
        int returnDate = int(block.timestamp) + 30 * DAY;
        addRecord(fullName, bookISBN, returnDate);
    }
}
```