Implementacija blockchain sustava u Pythonu

Šimunjak, Ivana

Undergraduate thesis / Završni rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:871908

Rights / Prava: Attribution-NonCommercial 3.0 Unported/Imenovanje-Nekomercijalno 3.0

Download date / Datum preuzimanja: **2024-04-24**

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

zir.nsk.hr

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

VARAŽDIN

Ivana Šimunjak

IMPLEMENTACIJA BLOCKCHAIN SUSTAVA U PYTHONU

ZAVRŠNI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Ivana Šimunjak

Matični broj: 0016137527

Studij: Informacijski sustavi

IMPLEMENTACIJA BLOCKCHAIN SUSTAVA U PYTHONU

ZAVRŠNI RAD

Mentor/Mentorica:

Doc. dr. sc. Maretić Marcel

Varaždin, rujan 2021.

Ivana Šimunjak

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

---

Sažetak

U ovom završnom radu prikazana je implementacija *blockchain* sustava u programskom jeziku *Python-u*. Objašnjena je teorijska pozadina i navedene su najvažnije prednosti i nedostaci *blockchain-a*. Kako bi se demonstrirao način rada *blockchain* sustava, u ovom radu prikazane su funkcionalnosti od kojih se sastoji, a dvije glavne su rudarenje bloka i dodavanje transakcije. Kako bi se *blockchain* zaštitio od napada koristi više vrsta provjera i specifičnu bazu podataka, koja je objašnjena u nastavku. Ostale funkcionalnosti prikazuju predradnje koje se odvijaju kako bi se točni podaci zapisali u blok i zatim isti dodali u *blockchain*. Na kraju opisuje se jedna od raširenih kriptovaluta diljem svijeta *Bitcoin* i primjeri korištenja *blockchain-a* na više različitih načina.

Ključne riječi: blockchain; blok; kriptovaluta; implementacija; baza podataka; glavna knjiga; sigurnost

Sadržaj

1. Uvod 1

2. Blockchain 2

3. Prednosti i nedostaci 4

4. Implementacija blockchain sustava u Pythonu 6

4.1. Prvi primjer: Colab 6

4.1.1. Klasa Block 6

4.1.2. Klasa Blockchain 7

4.1.3. Poziv funkcije i ispis 9

4.2. Drugi primjer: Visual studio 9

4.2.1. Funkcije 10

4.2.2. Provjera korisničkog unosa 17

5. Bitcoin 20

6. Primjeri korištenja 22

7. Zaključak 23

Popis literature 25

1. Uvod

Blockchain tehnologija omogućava informaciji da se u digitalnom obliku distribuira između svih čvorova koji sudjeluju u sustavu. Svaki čvor ima svoju kopiju glavne knjige, zbog toga nema potrebe za trećom stranom koja će vršiti kontrolu nad distribucijom informacija jer je sustav autonoman. Kako bi to bilo moguće, vrši se veliki broj provjera radi istinitosti i autentičnosti informacija koje se zapisuju u blok.

U današnje vrijeme putem interneta razmjenjuje se velika količina informacija, no nekim korisnicima je bitna diskretnost i tajnost. *Blockchain* je prva tehnologija koja nudi takvu sigurnost jer u svakom svom bloku sadrži vremensku oznaku i hash koji se ne mogu mijenjati u trenutku kada se blok nalazi u *blockchain-u*. Svaki pokušaj mijenjanja informacija u bloku zahtjeva veliku količinu resura jer se mora promijeniti više od 51 posto blokova koji se nalaze *blockchain-u*.

Cilj ovog završnog rada je prikaz općeniti primjer *blockchain-a*. Programskim dijelom izrađene su funkcionalnosti koje demonstriraju rad sustava. One se sastoje od primarnih radnji koje korisnik *blockchain-a* mora izvršiti kako bi uspješno proveo transakciju. Svaka radnja praćena je teorijskim opisom funkcija i varijabli radi boljeg razumijevanja programskog koda i načina na koji se informacije unutar *blockchain-a* spremaju i rudare.

2. Blockchain

Blockchain tehnologiju prvi put su predstavili *Stuart Haber* i W. *Scott Stornetta* 1991. godine. Njihov cilj je bio napraviti implementaciju sustava gdje se vremenske oznake dokumenata ne mogu mijenjati. Tek dvadeset godina kasnije kada se prvi puta 2009. godine lansirao *Bitcoin*, taj je *blockchain* dobio svoju prvu aplikaciju primjenu. *Bitcoin* protokol izgrađen je na *blockchain-u*.[1] [2]

Blockchain je sustav bilježenja informacija na način da oteža ili onemogući promjenu, hakiranje ili prijevaru sustav u kojemu radi. *Blockchain* je zapravo digitalna knjiga, drugim riječima jedan oblik baze podataka svih transakcija koje se dupliciraju i distribuiraju po cijeloj mreži računalnih sustava na *blockchain-u*. Svaki blok sadrži niz transakcija, a svaki put kada dođe do nove transakcije, zapis te transakcije dodaje se u knjigu svakog sudionika.

Blockchain je decentralizirana baza podataka poznata po nazivu *Distributed Ledger Technology (DLT)*. Takva baza podataka je ravnopravna omogućava da više članova zadrže vlastitu kopiju dijeljene knjige. Također DLT svim svojim članovima omogućava pristup podacima kao što su sigurnosne provjere, izvršavanje i snimanje vlastite transakcije bez oslanjanja na posrednika ili zahtijevanja tih informacija od središnjeg tijela kao što su banke.*Blockchain* je vrsta DLT-a u kojoj se transakcije bilježe s nepromjenjivim kriptografskim potpisom koji se naziva hash. [2]

Razlika između *blockchain* baze podataka i tipične *SQL* ili *NOSQL* baze je način na koji se podaci u njoj strukturiraju. *Blockchain* prikuplja podatke u grupe koje se nazivaju blokovi koji sadrže informacije. Svaki blok ima određeni kapacitet za skladištenje informacija, a kada se napuni onda se veže za blok koji je prethodni te zbog decentralizirane baze podataka svaki blok kada se spoji u lanac dobiva svoju vremensku oznaku. Nakon što se blok napunio i povezao svaki sljedeći blok koristi istu logiku punjenja i povezivanja. Dok tipična baza podataka strukturira svoje podatke u tablice koje u sebi sadrže određene atribute za željeni entitet koji se opisuje. [3] [2]

Sigurnost *blockchain-a* se omogućuje tako da se novi napravljeni blok uvijek linearno i kronološki pohranjuju. Nakon što je blok dodan u lanac vrlo je teško mijenjati njegov sadržaj a razlog tome je što svaki blok ima svoj hash zajedno s hash-om bloka prije njega kao i vremensku oznaku. Hash kod je izrađen matematičkom funkcijom koja digitalne informacije pretvara u niz brojeva i slova, te ako se te informacije na neki način preurede mijenja se i hash kod. [2]

Kada bi se dogodio hakerski napad gdje određeni haker želi promijeniti Blokchain i

ukrasti valutu od svih, te ako bi promijenio vlastitu pojedinačnu kopiju digitalne knjige koju dobiva kao i svi drugi sudionici, ona se više ne bi usklađivala s kopijom digitalne knjige od ostalih. U tom slučaju kada bi svi drugi išli uspoređivati svoje kopije s tom od hakera i primijetili razlike ujedno bi bio i izbačen iz lanca blokova. Pokušaj takvog napada zahtijevalo bi od hakera da promjeni više od pedeset posto kopija drugih sudionika na taj način da se sve kopije međusobno podudaraju. Takav napad zahtjeva veliku količinu resursa i financiranja.[2]

Trenutno postoji veliki broj projekata temeljenih na blokchain-u na načine koji bi mogli pomoći svijetu, a ne samo za bilježenje transakcija. Jedan od tih primjera bio bi način na koji bi se moglo izvršiti glasovanje na demokratskim izborima. Pošto je blokchain-ova priroda da se teško može promijeniti sadržaj bloka, lažiranje glasa kao takvog bi bilo puno teže nego u današnje vrijeme.[2]

3. Prednosti i nedostaci

Svaka tehnologija ima svoje pozitivne i negativne strane, tako isto i *blockchain*. Najvažnije prednosti su:

1. Dobivanje distribuiranog sustava koji nema treće strane u provođenju usluga koje *blockchain* nudi. To je prednost jer neke treće strane dobivaju mali udio kao proviziju za odrađenu uslugu te se na taj način rješava problem dodatnih troškova i povjerenja prema trećoj strani. [4]

2. *Blockchain* tehnologije da nudi veliku kvalitetu podataka. Kako je i prije navedeno, to je sustav distribuirane glavne knjige u kojem se pohranjuju podaci o transakcijama. Ne mogu se dodati bilo koji podaci u glavnu knjigu zbog dodatnih provjera koje se vrše nad njom. To korisnicima nudi sigurnost i istinitost podataka.[4]

3. U usporedbi s drugim sustavima, da nudi najvišu razinu integriteta do sada. U stvarnosti to znaci da će svi podaci uvijek biti ispravni i da ih nitko ne može promijeniti kada se nađu u glavnoj knjizi. Održivost ovog svojstva ovisi o hashu koji se nalazi u svakom bloku i ulozi koju ima.[4]

4. Također, da nudi brže transakcije u odnosu na tradicionalna sredstva. Obično centraliziranim bankama može trebati dulji period u obradi transakcije. Međutim, s blockchain-om se transakcija može izvršit u roku od nekoliko sekundi. [4]

Nedostaci koje je važno istaknuti su:

1. Prvi nedostatak ove tehnologije je veliki broj ponavljanja ažuriranja knjiga i svih čvorova kako bi svaki korisnik dobio svoju verziju. To je zato što priroda distribuirane knjige zahtjeva da svaki čvor mora imati svoju kopiju glavne knjige. [4]

2. Svaka *blockchain* lista ima postupak provjere potpisa. Zato za svaku transakciju u sustav treba privatno-javna provjera kriptografskog potpisa. Odnosno, koristi *ECDSA (Elliptic Curve Digital Signature Algorithm)* kako bi osigurao da transakcije koje se odvijaju budu između točna dva čvora. Taj cijeli proces provjere potpisa je težak i složen. [4]

3. Jedna od velikih mana *blockchain-a* je što ne dolazi s skupom propisa na mreži. Zbog toga može doći do koncepta *ICO (initial coin offerings)* prijevare. To je način na koji poduzetnik, koji želi lansirati novu kriptovalutu za koju treba resurse, nudi ICO početni prozivod koji investitori mogu kupiti te preko nje mogu dobiti nekakve pogodnosti. [2] [5]

4. Također, jedan od velikih nedostataka je pitanje privatnosti korisnika. Poduzećima je potrebna njihova privatnost te ne žele svoje osjetljive podatke otkrivati drugim korisnicima. Stoga mnoge organizacije ne žele koristiti *blockchain* u poslovne svrhe. [2]

4. Implementacija blockchain sustava u Pythonu

4.1. Prvi primjer: Colab

Za izradu prvog primjera koristio se Colaboratory ili skraćeno Colab koji je Googleov proizvod te dopušta korisniku pisanje i izvršavanje Python koda putem web preglednika **colab**. Kod izrade koda koriste se dvije biblioteke datetime i hashlib.

```
import datetime
import hashlib
```

4.1.1. Klasa Block

Prva klasa koja se definira je *Block*. Ona je bitna za kasniji razvoj klase *Blockchain*. U njoj se nalazi šest bitnih atributa pomoću kojih se opisuje taj blok. [6]

```
class Block:
  BlockNumber=0
  BlockData=None
  BlockPointer=None
  BlockHash=None
  BlockNonce = 0
  PreviousHash= 0x0
  Timestamp= datetime.datetime.now()
```

Prva varijabla *BlockNumber* predstavlja broj bloka, koji služi za kasnije indeksiranje, te je zato na početku postavljen na nulu. Sljedeća varijabla je *BlockData* i služi za spremanje podataka. Kriptovaluta *Bitcoin*, napravljena pomoću *blockchain-a*, navedenu varijablu koristi za spremanje transakcija, no u ovom primjeru koristit će se za spremanje stringova. Varijabla *BlockPointer* služi kako bi se mogli pozicionirati na sljedeću hash tablicu u lancu blokova, odnosno kao pointer na sljedeći blok u nizu. Nadalje, varijabla *BlockHash* koristi se kao jedinstveni ID za taj blok kojem je dodjeljena i provjerava njegov integritet, a na početku postavljena je na *none*. Kako bi dobili ID koristi se funkcija za konvertiranje podatak koji se nalazi u određenom intervalu.*BlockNonce* varijabla je broj koji se u bloku pojavljuje samo jednom, no kasnije se koristi za izradu izračuna jedinstvenog hash-a za blok koji se izrađuje. Predzadnja varijabla

PreviousHash pohranjuje prošli hash od prošlog bloka u lancu blokova, dok u posljednjoj varijabli *Timestamp* definira se datum i vrijeme nastajanja bloka pomoću funkcije *now* koja u tu varijablu sprema datum i vrijeme u sadašnjem trenutku. [6]

```
def __init__(self, BlockData):
    self.BlockData=BlockData
```

Unutar klase definirane su dvije funkcije. U ovom primjeru koristi se već postojeća funckija *init* koja kao argumente prima *self* i *BlockData* [7]. U funkciji pomoću *self* varijable poziva se *BlockData* iz klase u kojoj se nalazi. Funkcija *init* definira inicijalizaciju bloka koristeći pružene podatke kao jedini parametar. [6]

```
def hash(self):
    hash=hashlib.sha256()
    hash.update(
        str(self.BlockNonce).encode('utf-8') +
        str(self.BlockData).encode('utf-8') +
        str(self.BlockNumber).encode('utf-8') +
        str(self.PreviousHash).encode('utf-8') +
        str(self.Timestamp).encode('utf-8')
    )
    return hash.hexdigest()
```

Sljedeća funkcija koja se koristi u primjeru je *hash* koja služi za izračunavanje hash-a određenog bloka. Kako bi to bilo moguće, za definiranje navedene funkcije koristi se *sha256* algoritam koji generira jedinstveni potpis za taj *hash*. Nadalje, kako bi potpis bio jedinstven za svaki hash koristi se funkcija *update*. U toj se funkciji poziva funkcija *str* koja služi za vraćanje objekta, koji joj je zadan, u obliku stringa. Na kraju se dobivena vrijednost vrati u heksadekatskom formatu. [6] [8]

4.1.2. Klasa Blockchain

Sada kada je definirana klasa za jedan blok, sljedeća klasa je za *blockchain* pod nazivom *Blockchain*. [6]

```
class Blockchain:
  diff = 20
```

```
maxNonce = 2**32
target = 2 ** (256-diff)
block = Block("Genesis")
head = block
```

U njoj se nalazi sve skupa pet različitih ključnih varijabli. Prva varijabla je *maxNonce* veličine 2^{32}, što označava broj koji se može pohraniti u 32 bitnom broju. Sljedeća po redu je *diff* koja predstavlja vrijednost teškoće rudarenja *(mining difficulty value)*, a njena vrijednost se koristi kako bi se izračunao ciljani hash. Nadalje, varijabla *target* se izračuna po formuli prikazanoj u primjeru iznad teksta. Također, u varijabli *block* zapisana je vrijednost *Genesis* bloka. Na kraju je potrebno odrediti glavu bloka, koja joj je najnoviji dio. Kako se blokovi dodaju u lanac tako se glava ažurira. [6]

```
def add(self, block):
    block.PreviousHash=self.block.hash()
    block.BlockNumber=self.block.BlockNumber + 1

    self.block.BlockPointer = block
    self.block = self.block.BlockPointer
```

Nakon određivanja varijabli treba napraviti funkciju *add* koja sadrži dvije vrijednosti: *self* i *block*. Njena svrha je dodavanje bloka u *blockchain-a*. To se odrađuje tako da se za zadani blok postavi u varijablu *PreviousHash* vrijednost hash-a od bloka koji je u nizu iza njega te nakon čega se postavlja novi broj bloka, tako da se broj iz *BlockNumber* prethodnog bloka zbroji s jedan i zapiše u istu. Zatim se postavlja pokazivač od bloka na samog sebe i postaje nova glava *blockchain-a*.[6]

```
def mine(self, block):
   for n in range(self.maxNonce):
     if int(block.hash(), 16) <= self.target:
       self.add(block)
       print(block)
       break
     else:
       block.BlockNonce += 1
```

Zadnja funkcija u blokchainu, koja se nalazi u ovom primjeru, je *mine* funkcija. Ona omogućuje čvorovima u mreži koji se ne moraju nužno poznavati da dođu do saznanja o tome kako izgleda pravi *blockchain-a*. Ta funkcionalnost je zapravo obrambeni mehanizam *blockchain-a*. [6]

4.1.3. Poziv funkcije i ispis

Prije ispisa *blockchain-a* poziva se klasa *Blockchain*. Sljedeće što treba napraviti je for petlja koja će rudariti dvadeset blokova u *blockchain-u*. U for petlji se poziva funkcija *mine*. Svaki od tih dvadeset blokova biti će inicializirani jedinsvenim nizom.[6]

Na kraju se printa svaki blok koji se nalazi u *blockchain-u*. U ovom primjeru se to radi pomoću *while* petlje. U toj petlji se nalazi funkcija *print* u kojoj ispisujemo glavu svakog *blockchain-a*.[6]

```
blockchain = Blockchain()

for n in range(20):
    blockchain.mine(Block("Block " + str(n+1)))

while blockchain.head != None:
  print(blockchain.head)
  blockchain.head = blockchain.head.BlockPointer
```

4.2. Drugi primjer: Visual studio

Za izradu drugog primjera koristio se *Visual studio 2019* i njegov paket za rad u pythonu *PyCharm*. U tom paketu uključeno je uređivanje koda, ispravljanje pogrešaka, interaktivni razvoj za *Python* aplikacije i razne biblioteke koje se mogu koristiti pri izradi koda [9]. Biblioteke koje se koriste u ovom primjeru su *functools, hashlib, json i cillections*. U ovom slučaju iz biblioteke *collections* koristi se samo *OrderedDict* pri izradi koda. [10]

```
import functools
import hashlib
import json
from collections import OrderedDict
```

Nakon uključivanja biblioteka, nalazi se varijabla *MINING REWARD* kojoj je vrijednost 10 i ona služi kao nagrada koju će korisnik primiti pri rudarenju bloka. Nadalje, ovaj primjer *blockchain-a* izvršava se lokalno na našem računalu pa se unaprijed određuje varijabla *owner*, s imenom određene osobe, kako bi indentificirali osobu koja vrši transakcije. Ime osobe koja se nalazi u varijabli *owner*, nalazi se kao prva dodana osoba u rječniku *participants*. Sljedeća varijabla je *otvorene transakcije* koja zadržava informacije o transakciji *owner-a*, dok se one ne prihvate ili odbiju. Nakon toga, ovisno o krajnjem ishodu, određena se transakcija rudari u *blockchain*. Također, da bi *blockchain* mogao raditi na početku se nalazi *genesis block* koji se odmah stavlja kao prvi blok u *blockchain-u* kako bi se sljedeći rudareni blok mogao vezati za njega. [10]

```
MINING_REWARD = 10

genesis_block={
        'prosli_hash': '',
        'index': 0,
        'transakcija': [],
        'dokaz': 100
        }

blockchain = [genesis_block]
otvorene_transakcije=[]
owner = 'Ivana'
participants = {'Ivana'}
```

4.2.1. Funkcije

Prva funkcija koja je potrebna za rad zove se *valid dokaz*. Ona služi kako bi se provjerio hash koji se nalazi u bloku. U prvoj liniji, varijabla *pogodak* sprema string od varijabli *transakcija, zadnji hash i dokaz* te se enkodira u *UTF-8*. Nakon toga, varijabla *pogodak hash* pomoću biblioteke *hashlib* i njezine funkcije *sha256* tu varijablu pretvara u kodirani heksadekatski hash. Na kraju, funkcija vraća vrijednost ako nađe hash kojoj su prva dva broja *00*. [10]

```
def valid_dokaz (transakcija, zadnji_hash, dokaz):
        pogodak = (str(transakcija) + str(zadnji_hash) + str(dokaz)).encode()
```

```
        pogodak_hash = hashlib.sha256(pogodak).hexdigest()
        #print(pogodak_hash)
        return pogodak_hash[0:2] == '00'
```

Sljedeća funkcija je *dokaz rada* koja prvo pronalazi zadnji blok u *blockchain-u*. Zatim od pronađenog bloka napravi hash pomoću funkcije *hash block* te se dobivena vrijednost pohranjuje u varijablu *zadnji hash*. Nadalje, kako bi se dokazalo koja je valjanja transakcija treba napraviti *while* petlju u kojoj se pomoću funkcije *valid dokaz* prolazi kroz listu *otvorene transakcije* i varijablu dokaz uvećava za jedan, ako funkcija vrati pronađeni hash. Na kraju, funkcija *dokaz rada* vraća broj *dokaza* koji se zapisuje u taj isti blok. [10]

```
def dokaz_rada():
        zadnji_blok= blockchain[-1]
        zadnji_hash = hash_block(zadnji_blok)
        dokaz=0
        while valid_dokaz(otvorene_transakcije, zadnji_hash, dokaz):
                dokaz+=1
        return dokaz
```

Jedna od najbitnijih funkcija za normalan rad *blockchain-a* zove se *get balance*. U toj se funkciji sakupljaju podaci o pošiljatelju, primatelju, količini novčića koja je poslana i zbraja se sveukupni saldo primatelja i pošiljatelja.[10]

Lista *tx sender* traži količinu za transakciju u bloku pod imenom *transakcija* te ju zapisuje samo ako je pošiljatelj isti kao i osoba koja se nalazi u rječniku *participants* za određeni blok u *blockchain-u*. Za listu *open tx sender* vrijedi isto samo ne traži količinu koju je poslao pošiljatelj u bloku nego u listi *otvorene transakcije* jer se u njoj nalaze transakcije koje još nisu obrađene do kraja i dodane u *blockchain*. Nakon toga se iz varijable *open tx sender* dodaje ta pronađena količina u listu *tx sender* kako bi se sve provedene transakcije, otvorene i već rudarene, nalazile na jednom mjestu. [10]

Kada se svi podaci transakcija nalaze u jednoj listi, treba se izračunati ukupan iznos svih količina iz liste *otvorene transakcije* i rudarenih iznosa za tog pošiljatelja, koji se zatim zapisuje u varijablu *amount sent*. Pomoću funkcije *reduce* poziva se funkcija *lambda* pomoću koje se izračunava suma. Nakon poziva određuju se varijable *tx sum* kao trenutni zbroj i *tx amt* kao trenutna količina. Zatim, navedena funkcija prolazi kroz sve brojeve koji se nalaze u listi *tx sender* i prilikom prolaska kroz listu jedan po jedan se zapisuju u *tx amt* te se njihov zbroj

sprema u varijablu *tx sum*. Na kraju se stavlja uvjet ako je dužina broja veća od 0 da dođe do međusobnog zbrajanja brojeva, a ako nije onda se zbraja s nulo. Zadnji uvjet nula predstavlja index elementa od kojeg želimo započeti zbrajanje.[10]

Isti postupak ponavlja se i za primatelja, nakon čega funkcija *get balance* vraća razliku između primljene i poslane varijable te tako dobijemo pravo stanje korisnika. [10]

```
def get_balance(participant):
        tx_sender=[[transakcija['Kolicina']
            for transakcija in block['transakcija']
            if transakcija['Posiljatelj'] == participant]
            for block in blockchain]

        open_tx_sender =[ transakcija['Kolicina']
                    for transakcija in otvorene_transakcije
                    if transakcija['Posiljatelj'] == participant ]

        tx_sender.append(open_tx_sender)

        amount_sent = functools.reduce(lambda tx_sum, tx_amt:
                                tx_sum + sum(tx_amt)
                                if len(tx_amt)>0
                                else tx_sum+0, tx_sender, 0)

        tx_recipient=[[transakcija['Kolicina']
                    for transakcija in block['transakcija']
                    if transakcija['Primatelj'] == participant]
                    for block in blockchain]
        amount_primljena = functools.reduce(lambda tx_sum, tx_amt:
                                    tx_sum + sum(tx_amt)
                                    if len(tx_amt)>0
                                    else tx_sum+0, tx_recipient, 0)


        return amount_primljena-amount_sent
```

Hash block funkcija služi kako bi se vratila vrijednost hash-a za blok koji želimo rudariti u *blockchain-u*. Ona koristi *sha256*, koja od stringa dobivenog pomoću funkcije *dumps* i enkodiranog preko funkcije *encode*, pretvara dobiveni string u heksadekatski hash. [10]

```
def hash_block(block):
        return hashlib.sha256(json.dumps(block,
                    sort_keys=True).encode()).hexdigest()
```

Funkcija bez koje *blockchain* ne bi mogao raditi je *mine block*. Pomoću nje se rudare blokovi u *blockchain*, odnosno zbog nje imamo blokove koji se spajaju u lanac blokova. Prva varijabla *zadnji block* traži zadnji rudareni blok u *blockchain* te se onda ta varijabla koristi za izračun hash-a pomoću funkcije *hash block* i zapisuje u varijablu *hashed block*. U varijabli dokaz pohranjuje se dokaz rada *blockchain-a*. [10]

Nadalje, kako bi se nagrada mogla dodjeliti svakom korisniku koji je rudario blok, postoji sortirani rječnik *transakcija nagrade* koji sadrži pošiljatelja, primatelja i količinu nagrade koja je prethodno određena. Sve primljene nagrade korisnika spremaju se u listu *kopirane transakcije*, koja je kopija liste *otvorene transakcije*. [10]

Nakon što je stvorena lista sa svim bilježenim transakcijama potrebno je napraviti rječnik *block* koji predstavlja jedan u nizu blokva u *blockchain-u*. Svaki blok sadrži podatke o hash-u prošlog bloka, njegovom index-u, transakciji i dokazu. Kada se u njega pohrane sve informacije zapisuje se pomoću funkcije *append* u *blockchain*. Na kraju u slučaju pravilnog rudarenja blokom navedena funkcija vraća vrijednost *true*. [10]

```
def mine_block():

        zadnji_block=blockchain[-1]
        hashed_block = hash_block(zadnji_block)

        dokaz=dokaz_rada()

        transakcija_nagrade = OrderedDict(
                [('Posiljatelj', 'MINING'), ('Primatelj',owner),('Kolicina',
                MINING_REWARD)]
                )
```

```
        kopirane_transakcije = otvorene_transakcije[:]
        kopirane_transakcije.append(transakcija_nagrade)

        block = {
                'prosli_hash': hashed_block,
                'index': len(blockchain),
                'transakcija': kopirane_transakcije,
                'dokaz': dokaz
                }
        blockchain.append(block)
        return True
```

Get zadnju blockchain vrijednost je funkcija koja vraća zadnju vrijednost u *blockchain-u*. Prvo se traži veličina *blockchain-a*, ako je ona manja od 1 vrati *None*, a ako je veća od 1 onda vrati zadnji element u blockchain-u. [10]

Nadalje, funkcija pod nazivom *provjera transakcij* ima parametar *transaction* i služi kako bi se napravila provjera nad izdanom transakcijom prije nego se ona izvrši. Pomoću nje se provjerava ima li korisnik dovoljno sredstava na računu da bi mogao izvršiti transakciju s količinom koju želi poslati. Na početku u varijabli *sender balance* zapisuje se zadnji saldo na računu korisnika koji želi napraviti transakciju te se provjerava je li veličina salda veća ili jednaka količini transakcije, ako je vrati *true* u suprotnom vrati *false*. [10]

```
def get_zadnju_blockchain_vrijednost():

        if len(blockchain) < 1:
                return None

        return blockchain[-1]

def provjera_transakcij(transaction):
        sender_balance=get_balance(transaction['Posiljatelj'])
        if sender_balance >= transaction['Kolicina']:
                return True
        else:
                return False
```

Dodaj transakciju funkcija služi kako bi dodali transakciju u listu *otvorene transakcije*. Funkcija prima tri parametra, a to su *posiljatelj* koji je *owner*, *primatelj i kolicina* koje su unešene od strane korisnika. Na početku funkcije potrebno je odrediti rječnik *transakcija* koji je sortiran. U njemu se pohranjuju svi parametri koji se unose pri pozovu funkcije. Prije samog spremanja transakcija u listu vrši se provjera koja je objašnjena u tekstu iznad. Ako je provjera vratila *true*, transakcija se sprema u listu *otvorene transakcije* te se pošiljatelj i primatelj spremaju u listu *participants*. U suprotnom ništa se ne dodaje i funkcija vraća *false*.[10]

```
def dodaj_transakciju(posiljatelj, primatelj, kolicina=1.0):
        posiljatelj=owner

        transakcija = OrderedDict(
                [('Posiljatelj',posiljatelj), ('Primatelj',primatelj),
                ('Kolicina', kolicina)]
                )

        if provjera_transakcij(transakcija):
                otvorene_transakcije.append(transakcija)
                participants.add(posiljatelj)
                participants.add(primatelj)
                return True
        return False
```

Funkcija *get vrijednost transakcije* služi kako bi dobili informaciju količine transakcije koju korisnik želi poslati i primatelja transakcije. Svaka vrijednost koju korisnik zapiše pretvara se u *float* broj radi daljnjeg izračuna.[10]

```
def get_vrijednost_transakcije():
        Primatelj_osoba= input('Unesite primatelja transakcije:')
        vrijednost_korisnika= float(input('Vrijednost vaše transakcije: '))
        return Primatelj_osoba, vrijednost_korisnika


def get_izbor_korisnika():
        vrijednost_korisnika=input('Vaš izbor je:')
```

```
        return vrijednost_korisnika


def print_blockchain_elemente():
        for block in blockchain:
                print('Outputting Block')
                print(block)
```

Prvojeri lanac funkcija služi za provjeru manipulacije *blockchain-a*. Pomoću nje želimo usporediti hash od prethodnog i trenutnog bloka. Prvo se napravi *for* petlja s kojom prolazimo kroz *blockchain*. Zatim se koristi funkcija *enumerate* koja vraća listu i tako se može saznati točan index za određeni blok. [10]

Pomoću prvog uvjeta preskaće se prvi blok jer je to *genesis* blok. Svi ostali slučajevi provjeravaju se tako da prošli hash, koji je zapisan u rječniku *block* mora biti jednak kao i hash od zadnjeg bloka u *blockchain-u*. *Hash* od zadnjeg bloka dobije se pomoću funkcije *hash block*. [10]

Na kraju se vrši provjera pomoću funkcije *valid dokaz* koja vraća *false* i odgovarajuću poruku korisniku, ako nije pronašla hash sa početne dvije vrijedosti nula. Navedena funkcija služi izbjegavanju manipulacije blokom. [10]

```
def provjeri_lanac():
        for (index, block) in enumerate(blockchain):
                if index == 0:
                        continue
                if block['prosli_hash'] != hash_block(blockchain[index-1]):
                        return False
                if not valid_dokaz(block['transakcija'][:-1],
                                        block['prosli_hash'], block['dokaz']):
                        print('Dokaz nije valjan!')
                        return False
        return True
```

Funkcija *provjera transakcija* provjerava jesu li sve transakcije valjane. Zato se na početku koristi funkcija *all* kako bi se provjerile sve vrijednosti koje se nalaze u listi *otvorene transakcije*. Prvo se izvrši funkcija pod nazivom *provjera transakcij* za svaku transakciju u

navedenoj listi.[10]

```
def provjera_transakcija():
        return all([provjera_transakcij(transak)
                for transak in otvorene_transakcije])
```

4.2.2. Provjera korisničkog unosa

Prije nego korisnik može raditi s programom treba se odrediti koje funkcionalnosti ima program. U ovom slučaju to se nalazi u *while* petlji gdje se provjerava ako varijabla *bool provjera*. Unos korisnika očita se pomoću funkcije *get izbor korisnika* te se odabrani broj zapisuje u varijablu *izbor korisnika*.[10]

```
while bool_provjera:
        print('Molim vas odaberite(1/2/3/...):')
        print('1: Dodaj novu vrijednost transakcije!')
        print('2: Ispis svih vrijednosti! ')
        print('3: Ispis participants! ')
        print('4: Mine a block! ')
        print('5: Provjera transakcije! ')
        print('h: Manipulacija! ')
        print('q: Izadi! ')

        izbor_korisnika=get_izbor_korisnika()
```

Prvu opciju koju korsnik može koristiti je unos nove transakcije. Za početak varijabla *K transakcije* poprima vrijednosti iz funkcije *get vrijednost transakcije* koju je korisnik upisao. Nakon toga zapisuje se vrijednost imena primatelja transakcije u varijablu *primatelj v* i vrijednost količine transakcije u *kolicina v* iz varijable *K transakcije*. Na kraju se provjerava ako je dodana transakcija pomoću funkcije *dodaj transakciju*.[10]

```
if izbor_korisnika == '1':
        K_transakcije = get_vrijednost_transakcije()
        primatelj_v, kolicina_v=K_transakcije

        if dodaj_transakciju(owner, primatelj_v,
```

```
                kolicina=kolicina_v ):
                print('Dodana transakcija')
        else:
                print('Transakcija nije bila uspjesna!')
```

Drugi izbor za korisnika je ispis svih vrijednosti koje je korisnik rudario u *blockchain*. U slučaju odabira funkcionalnosti tri, dobiva popis svih korisnika iz liste *participants*.

```
elif izbor_korisnika == '2':
        print_blockchain_elemente()

elif izbor_korisnika == '3':
        print(participants)
```

Četvrti izbor je mogućnost rudarenja provedene transakcije u *blockchain*. Ovdje se vrši provjera koja u slučaju ako funkcija *mine block* vrati *true*, prazni se lista *otvorene transakcije*. Funkcionalnost pod rednim brojem pet, provjerava sve transakcije koje se nalaze u listi *otvorene transakcije*. Ovisno o ishodu provjere, program izbaci odgovarajuću poruku.[10] [10]

```
elif izbor_korisnika == '4':
    if mine_block():
                otvorene_transakcije=[]

elif izbor_korisnika == '5':
    if provjera_transakcija():
                print('Sve transakcije su dobre!')
        else:
                print('Nisu dobre transakcije!')
```

Sljedeće, ako se korisnik odluči za opciju *h* može manipulirati blokom. U slučaju manipulacije blokom program izbacuje korisnika, provjerom koja se izvodi nakon svake radnje.

Za izlaz iz programa služi slovo *q*, ako se ništa ne odabere ispisuje se odgovarajuća poruka. Na samome kraju nalazi se ispis stanja korisnika koji se prikazuje nakon svake odrađene funkcionalnosti osim pri manipulaciji i izlazu iz programa.

```
elif izbor_korisnika == 'h':
        if len(blockchain)>=1:
```

```
                blockchain[0] = {
                        'prosli_hash': '',
                        'index': 0,
                        'transakcija': [{'Posiljatelj':'Chris',
                             'Primatelj': 'Max', 'Kolicina':100.0}]
                        }
elif izbor_korisnika == 'q':
        break
else:
        print('Niste dobro izabrali, molim vas ponovite!')

if not provjeri_lanac():
        print_blockchain_elemente()
        print('Izlaz, manipulacija')
        break

print('Stanje racuna od {}: {:6.2f}'.format('Ivana',
                                                get_balance('Ivana')))
else:
        print('Hvala što ste koristili našu aplikaciju!')
```

5. Bitcoin

Bitcoin je jedna od prvih digitalnih valuta koje koriste *peer-to-peer* tehnologiju za rad bez središnjeg tijela ili banaka. Upravljanje transakcijama i izdavanje bitcoina provodi mreža. To je otvoreni kod i njegov dizajn je javan, odnosno nitko ne posjeduje ili kontrolira *Bitcoin*, svi mogu sudjelovati. Zahvaljujući mnogim jedinstvenim svojstvima, *Bitcoin* omogućuje upotrebu koju prije nije mogao pokriti niti jedan prethodni sustav plaćanja. [11]

Bitcoin je napravljen tako da koristi implementaciju Blockchain-a kod evidentiranja transakcija u digitalnoj knjizi plaćanja. U istraživačkom radu koji uvodi digitalnu valutu, tvorac Satoshi Nakamoto, nazvao ga je "novim elektroničkim gotovinskim sustavom koji je u potpunosti peer-to-peer, bez trećih strana od povjerenja"[1]. To je vrsta kripto valute, ne postoji fizičko izdavanje *Bitcoin-a* nego se samo saldo vodi u javnoj knjizi kojoj svi imaju pristup. Nije podržan od strane nikakvih banaka ili vlade. No, unatoč tome što nije podržan od strane zakona, postao je tijekom godina vrlo popularan i pokrenuo je pokretanje stotinu drugih kripto valuta koje imaju zajednički naziv *Altcoins*.

Moglo bi se reci da je *Bitcoin* grupa računala da će svi pokrenuti njihov kod i pohraniti njegov blokchain. Stanje tokena bitcoina su javni i privatni ključevi koji su dugi niz brojeva i slova povezanih matematičkim algoritmom šifriranja koji se koristi za njihovo stvaranje. Javni ključ služi kao adresa svijetu i na koju drugi mogu slati *Bitcoin*. Moglo bi se reći da je javni ključ nešto slično kao i broj bankovnog računa s kartice. Dok je privatni ključ je čuvana tajna˘ i koristi se samo za autorizaciju prijenosa bitcoina, a što bi onda bilo slično kao pinu od bankovne kartice.[1]

Pojedinci i tvrtke koji sudjeluju u *Bitcoin* mreži zaduženi su za obradu transakcija na *blockchain-u* i motivirani su nagradama i naknadama za transakcije koje se plaćaju u bitconima. Oni se mogu smatrati decentraliziranim tijelom koje provodi vjerodostojnost bitcoin mreže. Novi bitcoin-i se po rudarenju puštaju po fiksnoj, ali periodičnoj padajućoj stopi. Postoji samo 21 milijun bitcoin-a koji se mogu sve ukupno rudariti . Od lipnja postoji 18 milijuna *Bitcoin-a*, a ostalo je za rudarenje još samo 3 milijuna bitcoina. [1]

Bitcoin rudarenje je proces puštanja *Bitcoin-a* u opticaj. Taj način puštanja zahtjeva rješava teške računske zagonetke za otkrivanje novog bloka koji se dodaje na kraj *blokchain-a*. Taj proces dodaje i provjerava zapise transakcija na čitavoj mreži. Na taj način *Bitcoin* i druge kripto valute posluju drugačije nego centralizirane bankarske valute koje se oslobađaju po tečaju koji odgovara rastu. Ovaj sustav je decentraliziran i ima u cilju održavanje stabilnosti

cijene, odnosno da ne dođe do naglog rasta ili pada *Bitcoin-a*. Takvi sustavi unaprijed prema preciznom algoritmu određuju brzinu oslobađanja.

6. Primjeri korištenja

U današnje vrijeme raširenost *blokchain-a* je sve veća. Osim za Bitcon koristi se i kod drugih aplikacija kao što je *Spotify*, jedna od raširenih pružatelja audio i medijskih usluga za svoje korisnike. *Spotify* je 2017. godine počeo koristiti decentraliziranu bazu podataka za povezivanje umjetnika s licenciranim ugovorima i pjesmama.[12]

Nakon *Bitcoin-a* i *Spotify-a* koji su u svijetu poznati počeli su dolazi na "Matchpool" platformu koja potiče povezivanje članova zajednica kao što su *Airbnb*, *Uber* ili usluge za upoznavanje. Ova aplikacija koristi plaćanje u kripto valutama, odnosno u *Guppy* žetonima, koji su poznati pod imenom *GUP*. *Guppy* žetoni se mogu kupiti i zaraditi u aplikaciji ili trgovati s njima na burzi.[12]

Još jedan od primjera bi bio *SimplyVital Health* koja koristi *blokchain* kako bi pacijentima i pružateljima usluga omogućio pristup dijeljenja i cak mogućnost premještanja njihovih zdravstvenih podataka. Aplikacija osigurava sigurnu i zaštićenu koordinaciju i komunikaciju s drugim pružateljima usluga te stvara pouzdanost svojim korisnicima. Razlog zašto se u toj aplikaciji koristi *blokchain* je problem razmijene podatka na siguran i privatan način u zdravstvenoj industriji.[12]

7. Zaključak

Blockchain tehnologija se u današnje vrijeme sve češće koristi kod kriptovaluta zbog svojih dodatnih provjera kroz koje informacije prolaze da bi se zapisale u blok. To ju čini jednom od sigurnijih metoda za transakcije. Također se koristi decentraliziranom bazom podataka, gdje svaki korisnik dobije svoju kopiju glavne knjige, te su se zbog toga brzine izvršenja transakcija znatno ubrzale i izbacile treću stranu, kao što je na primjer banka, iz postupka obrade transakcija.

Sigurnost *blockchain-a* ne ovisi samo o provjerama koje se vrše u postupku obrade podataka, već i o hash-u koji se nalazi unutar bloka. Zbog tog hash-a svaki rudareni blok se više ne može mijenjati te je trajno zapisan u *blockchain*. Zbog svoje sigurnosti i brzine obrade podataka može se koristiti i u druge svrhe kao što su izbori.

Sve u svemu, *blockchain* tehnologija je još u razvoju i iskorišteno je samo dio njezinog potencijala za daljnje razvijanje i upotrebu. Može se mijenjati i prilagođavati ovisno o potrebama kranjeg korisnika.

Literatura

[1] J. Frankenfield, *Bitcoin*, link: https://www.investopedia.com/terms/b/bitcoin.asp, preuzeto: 21.7.2021.

[2] L. Conway, *Blockchain Explained*, link: https://www.investopedia.com/terms/b/blockchain.asp, preuzeto: 21.7.2021.

[3] H. Hashgraph, *What are distributed ledger technologies (DLTs)?* link: https://hedera.com/learning/what-are-distributed-ledger-technologies-dlts?gclid=CjwKCAjw9aiIBhA1EiwAJ_GTSqKrDuVzGB2X6D4wWjcfBjhaXjfHyfwP9dtgddk9copAMm_6VIXGthoCzXEQAvD_BwE, preuzeto: 25.7.2021.

[4] G. Iredale, *Ultimate Guide To Pros And Cons Of Blockchain*, link: https://101blockchains.com/pros-and-cons-of-blockchain/,preuzeto: 2.2.2021.

[5] B. SHERRY, *What Is an ICO?* link: https://www.investopedia.com/news/what-ico/, preuzeto: 25.8.2021.

[6] S. Raval, *Simple Blockchain in 5 Minutes*, link: https://www.youtube.com/watch?v=MViBvQXQ3mM, preuzeto: 2.8.2021.

[7] edureka, *Init In Python: Everything You Need To Know*, link: https://www.edureka.co/blog/init-in-python/, preuzeto: 5.8.2021.

[8] S. Khan, *What is the str method in Python?* link: https://www.educative.io/edpresso/what-is-the-str-method-in-python, preuzeto: 2.8.2021.

[9] M. Barlow, *Revamped Project Properties UI*, Link: https://devblogs.microsoft.com/visualstudio/revamped-project-properties-ui/, preuzeto: 18.8.2021.

[10] M. S., *Python*, video, preuzeto: 18.8.2021.

[11] W. Binns, *Principles of Data Mining*, link: https://bitcoin.org/en/posts/content-now-available-in-over-25-languages, preuzeto: 25.8.2021.

[12] P. Surani, *Blockchain in Action – 16 Inspirational Examples*, link: https://computerrock.com/en/blog/blockchain-in-action-16-inspirational-examples, preuzeto: 18.8.2021.