VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

REST API PRO DOTAZY NAD SÍTÍ BITCOIN

REST API FOR BITCOIN NETWORK QUERYING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **TOMÁŠ ŽIGO**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JAN PLUSKAL**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Žigo Tomáš**
Program: Informační technologie
Název: **Rest API pro dotazy nad sítí Bitcoin**
Rest API for Bitcoin Network Querying
Kategorie: Počítačové sítě

Zadání:

1. Prostudujte způsob ukládání transakcí v blockchainu sítě Bitcoin. Neopomeňte evoluci blockchainu a různé typy informací, které jsou v něm uloženy.
2. Po konzultaci s vedoucím navrhněte efektivní způsob uložení transformovaných dat o blocích, transakcích a adresách uložených v blokchainu.
3. Implementujte aplikaci zpracovávající historická i živá data, která zpřístupní pomocí Rest API.
4. Otestujte Vaši aplikaci s využitím dostupných Bitcoin blockchain prohlížečů a změřte výkonnostní charakteristiky. Srovnejte s existujícími nástroji insight-api, blockbook.

Literatura:

- Crosby, M., Pattanayak, P., Verma, S., & Kalyanaraman, V. (2016). BlockChain Technology: Beyond Bitcoin. *Applied Innovation*, *2*(6-10), 71.
- Cachin, C. (2016, July). Architecture of the Hyperledger Blockchain Fabric. In *Workshop on distributed cryptocurrencies and consensus ledgers* (Vol. 310, p. 4).
- Zyskind, G., & Nathan, O. (2015, May). Decentralizing Privacy: Using Blockchain to Protect Personal Data. In *2015 IEEE Security and Privacy Workshops* (pp. 180-184). IEEE. Chicago

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Pluskal Jan, Ing.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 25. října 2019

Abstrakt

Táto bakalárska práca rieši problematiku spracovania transformovaných dát v sieti Bitcoin. Cieľom tejto práce je priblížiť fungovanie detailov v sieti kryptomeny Bitcoin a následne vytvoriť aplikáciu schopnú spracovávať a sprístupňovať historické, ako aj aktuálne dáta v sieti. Riešenie bolo zamerané na rýchlu synchronizáciu a schopnosť odpovedania na požiadavky v čo najkratšom čase. Problém bol implementovaný použitím NoSQL databázového systému Cassandra a stratégie importovania do databázy, kde niektoré údaje sú aktualizované až po vyžiadaní. Vytvorené riešenie je vo väčšine prípadov konkurencie schopné aktuálne dostupným nástrojom.

Abstract

This bachelor's thesis solves issues of Bitcoin network transformed data processing. The main aim of this thesis is to describe the details of Bitcoin network cryptocurrency principles and subsequently create an application which is able to process and allow access to historical data as well as actual data in the network. The solution was focused on fast synchronization and the ability to respond to requests in the shortest possible time. The problem was implemented using the NoSQL database system Cassandra with database import strategy where some data fields are updated only on request. Created solution is mostly able to compete with currently available tools.

Kľúčové slová

Bitcoin, blockchain, blok, transakcia, ťažba bitcoinu, sieť bitcoinu, Bitcoin Core, Cassandra, NoSQL, databázový systém

Keywords

Bitcoin, blockchain, block, transaction, bitcoin mining, bitcoin network, Bitcoin Core, Cassandra, NoSQL, database system

Citácia

ŽIGO, Tomáš. *Rest API pro dotazy nad sítí Bitcoin.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jan Pluskal

Rest API pro dotazy nad sítí Bitcoin

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Jana Pluskala. Uviedol som všetku použitú literatúru , všetky publikáce a ďalšie zdroje, z ktorých som čerpal.

.....
Tomáš Žigo
28. mája 2020

Poďakovanie

Ďakujem vedúcemu práce Ing. Janovi Pluskalovi za odborné vedenie a užitočné rady pre tvorbu tejto práce.

Obsah

Kapitola 1

Úvod

Automatizácia okolitého sveta prebieha v každej sfére nášho života. Inak tomu nie je ani v oblasti financií. Na trh prichádza mnoho nových myšlienok pre správu nášho finančného majetku, ktoré sa určitým spôsobom líšia od tradičných systémov bánk, na ktoré sme všetci zvyknutí. Jednou z inovácii je aj digitálna mena nazývaná *kryptomena*. Pojem kryptomena je stále čoraz viac a viac blízkym pojmom pre širokú verejnosť. Azda najpopulárnejšou zo všetkých je Bitcoin, ktorá bola prvou kryptomenou vo svete a jej popularita neustále narastá.

Cieľom tejto bakalárskej práce je poskytnúť užívateľovi teoretický prehľad v oblasti Bitcoinu a následne tieto poznatky použiť pre implementáciu databázy a samotnej aplikácie, ktorá bude s touto databázou komunikovať a odpovedať na požiadavky užívateľa.

Práca je tvorená štyrmi hlavnými kapitolami. Prvá kapitola poskytuje všeobecný prehľad v oblasti Bitcoinu. V úvode kapitoly je vysvetlený samotný pojem Bitcoin a spôsob, ktorým sa líši od ostatných klasických mien. Ďalej sú v tejto kapitole rozobrané všetky podstatné vlastnosti a charakteristiky danej meny, jej typy sietí, adresy, transakcie a samotná blockchain technológia. Obzvlášť blockchain predstavuje veľmi dôležitú časť tejto práce a pre správne porozumenie sú podrobne vysvetlené všetky podstatné informácie. Následne je opísaný proces ťažby Bitcoinu, spôsoby ťažby, odmeny a aj konflikty, ktoré pri ťažení môžu nastať.

V druhej kapitole, ktorá sa venuje návrhu riešenia, je na úvod opísaný zvolený databázový systém, ako aj spôsob fungovania použitej databázy. Kapitola taktiež obsahuje požiadavky, ktoré výsledná aplikácia podporuje, ako aj jednotlivé detaily databázy so zvolenými tabuľkami a rôznymi typmi obsiahnutých údajov. V záverečnej časti kapitoly je uvedený použitý typ uzla a nástroje použité v tejto práci.

Kapitola venovaná implementácii sa zaoberá implementačnými detailmi aplikácie. Úvod kapitoly tvorí stručný prehľad riešenia. Následne je podrobne rozvedené prvotné importovanie do databázy, ako aj stratégia použitá pri implementácii samotného importovania. Ďalej sú uvedené implementačné detaily spôsobu získavania aktuálnych dát z blockchainu, riešenie možných konfliktov a práca s adresami.

Posledná kapitola sa zaoberá analýzou výsledného riešenia. Kapitola sa zaoberá testovaním rýchlosti importovania fyzických dát, rôznymi štatistikami a validačnými testami plne synchronizovanej aplikácie. V závere sú stručne uvedené ďalšie nástroje na analýzu siete Bitcoin a ich porovnanie s implementovaným riešením.

Kapitola 2

Bitcoin a jeho Blockchain technológia

Úlohou kapitoly je čitateľovi priblížiť základnú tematiku Bitcoinu a oboznámiť ho s technológiou blockchainu. Najprv sú vysvetlené základné pojmy, ktoré súvisia so sieťou Bitcoin a následne aj všetky podstatné pojmy ako ťaženie Bitcoinov alebo technológia blockchainu. Celá kapitola tvorí teoretické minimum pre implementáciu v ďalších častiach práce.

Pojem Bitcoin označuje kryptomenu, ktorá sa začala používať v roku 2009, kedy bol vyťažený Genesis blok (prvý blok v sieti Bitcoin). Jedná sa o open source kryptomenu, ktorej dizajn a všetky údaje sú plne dostupné verejnosti [27].

Vďaka tomu, že sa jedná iba o virtuálnu menu, poskytuje mnoho odlišností oproti ostatným financiám s núteným obehom. Bitcoin nemá žiadneho vlastníka, čím sa líši od ostatných tradičných mien kontrolovaných bankou alebo inými štátno-správnymi orgánmi. Hlavným rozdielom oproti konvenčným menám je fakt, že Bitcoin funguje na princípe úplne decentralizovanej peer-to-peer siete (vizualizované v sekcii 2.1, obrázok 2.1), kde sa súčasťou siete môže stať každý jeden záujemca. Tento návrh zabraňuje rôznym manipuláciám s účtami používateľov a tokmi ich peňazí. Každý účastník tejto siete má kópiu jej celej hierarchie (blockchain), ktorú musí neustále obnovovať na aktuálnu verziu a prípadná snaha o nejakú manipuláciu s dátami by znamenala zmenu dát v obrovskom množstve zariadení, kde sa tieto kópie nachádzajú. Bližšie informácie o charakteristike siete sa nachádzajú v sekcii 2.1. Ďalším rozdielom kryptomien oproti tradičným menám je fakt, že kryptomeny ako aj Bitcoin majú definovaný rast hodnôt v obehu ako aj svoj koniec. Tento fakt umožňuje jednoduchšiu kontrolu ako aj predppokladanie ich počtu v budúcnosti, čo sa pri tradičných menách určiť nedá [18]. Jednotka Bitcoinu sa väčšinou v reálnom svete označuje ako `BTC` [18], táto skratka pre jednotku Bitcoinu však nie je pravidlom. Skratka `BTC` nevyhovuje oficiálnym štandardom `ISO 4217` [20] pre definíciu názvov mien a z tohto dôvodu je možné, že niekedy je Bitcoin označovaný burzovým symbolom `XBT` [17].

Adresy a kľúče

Pre zasielanie a prijímanie hodnôt sa používajú rôzne kľúče a adresy. Prvým a najdôležitejším prvkom v celej koncepcii adries a kľúčov je ***privátny kľúč*** (anglicky *private key*). Tento kľúč je náhodne vygenerované číslo o veľkosti 2^{256} [2]. Primárnou úlohou privátneho kľúča je vytváranie podpisov, ktoré sú potrebné pre potvrdenie vlastníctva hodnoty pri transakciách. Nakoľko pri transakciách vedomosť tohto podpisu dáva užívateľovi právo použiť danú sumu, je veľmi dôležité si tento kľúč uchovať v bezpečí.

Druhým používaným kľúčom je ***verejný kľúč*** (anglicky *public key*). Tento kľúč je voľne šíriteľný a aj napriek tomu, že vzniká z privátneho kľúča, spätná dedukcia nie je možná. Na zamedzenie spätného zistenia privátneho kľúča sa používa špeciálna matematická funkcia - eliptická krivka [2]. Pre zníženie veľkosti čísla, ktoré tento kľúč používa je zavedený pojem ***adresa*** (anglicky *address* alebo taktiež *Bitcoin Address*). Adresa je voľne šíriteľná haš[1] (anglicky *hash*) hodnota vytvorená z verejného kľúča, zahašovaním tohto kľúča pomocou hašovacieho algoritmu `SHA256`[2] a následným zahašovaním pomocou algoritmu `RIPEMD160`[3] [2]. Po vytvorení hašu je ešte na tento haš použité kódovanie [5]. Aktuálne sa v sieti Bitcoinu používajú dva hlavné typy kódovania adries. Tieto adresy ako aj práca s nimi, sú definované štandardmi, ktoré sa nazývajú `BIP`. Staršie štandardy definovali adresy pomocou kódovania *Base58*[4]. Od roku 2017 bol vytvorený štandard, konkrétne `BIP0173`, ktorý definuje adresy pomocou rýchlejšieho kódovania *Bech32*. Adresa je najčastejšie používaná ako definovanie užívateľa, ktorý bude príjemcom transakcie 2.2. Medzi aktuálne štandardné typy adries patria adries patria:

- **Pay-to-PubkeyHash (P2PKH)** - najbežnejší typ adries v sieti, kódované pomocou Base58. Jedná sa o haš hodnotu verejného kľúča [13].
- **Pay-to-Script Hash (P2SH)** - podporujú zasielanie transakcií na určitý skript 2.2, kde pre použitie dostupných zdrojov je nutné jeho odomknutie [12]. Je kódovaná pomocou Base58.
- **Bech32 kódované adresy** - najnovší typ používaných adries P2WPKH a P2WSH, taktiež označované *segwit* adresy [9].

Príklady jednotlivých adries sú dostupné v tabuľke 2.1.

Typ	Príklad
P2PKH	1J4m13hgTqNcxbBb5sNi8VKkdzabWVi4fx
P2SH	3QL47GNx6gtyp91bg4KRjWs8rs2qxdXLrK
P2WSH	bc1qrp33g0q5c5txsp9arysrx4k6zdkfs4nce4xj0gdcccefvpysxf3qccfmv3
P2WPKH	bc1q8h67dytup75uurg9jpj32mtudrauraf2fd7lnp

Tabuľka 2.1: Ukážka štandardných adriesv sieti Bitcoin.

2.1 Typy Sietí

Sieť Bitcoinu pracuje s nasledujúcimi typmi siete :

- **Mainnet** - originálna a hlavná sieť pracujúca s transakciami Bitcoinu, kde satoshi[5] má reálnu ekonomickú hodnotu [5]. Táto je sieť je bežnou používanou sieťou pri pojmoch spojenými s Bitcoinom, preto pri nešpecifikovaní typu siete, ide o tento typ siete.

[1] haš - výstup hašovacej funkcie. V prípade Bitcoinu sa jedná o funkciu prevádzajúcu vstupnú hodnotu na jeho stopu v podobe hexadecimálneho čísla.

[2] SHA256 - 256-bitový bezpečný hašovací algoritmus používaný na kryptografické zabezpečenie

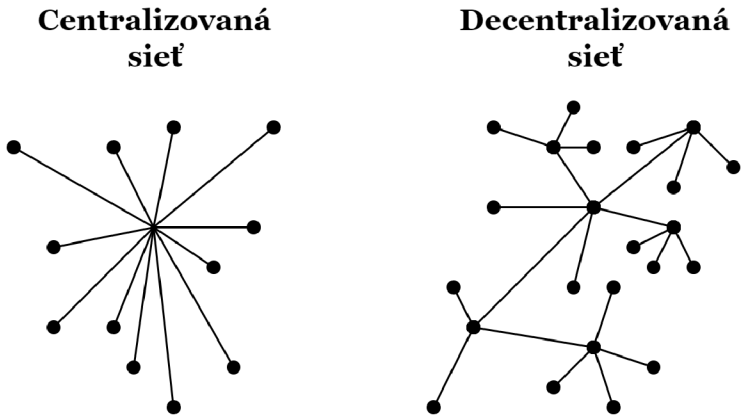
[3] RIPEMD160 - kryptografický hašovací algoritmus produkujúci 160 bitový výstup

[4] Base58 - sada znakov, ktorá je pre reálnych užívateľov jednoduchšia na porozumenie. Vynecháva znaky, ktoré je možné si pomýliť ako '0', 'O', 'I' a 'l'

[5] satoshi - najmenšia jednotka Bitcoinu ukladaná v blockchaine = 0.00000001 BTC

- **Testnet** - sieť používaná vývojármi. Táto sieť pracuje na podobnom princípe ako hlavná sieť, ale satoshi pri tomto type sieti nemajú žiadnu hodnotu. To znamená, že vývojári môžu testovať napríklad zasielanie transakcií bez dopadu na Mainnet.
- **Regtest** - lokálna sieť používaná vývojármi, kde takmer okamžite dokážu vytvárať bloky za účelom testovania a vytvárať privátne satoshi bez hodnoty v reálnom svete [5].

Používaná sieť pracuje na princípe decentralizovanej peer-to-peer sieti[6]. Jednotlivé zariadenia v sieti sa nazývajú *nodes* (anglicky *nodes*). Uzly pri tomto type sieti fungujú všetky na rovnakej priorite, bez nejakých špeciálnych uzlov [30]. Je použitá *zmiešaná* (anglicky *mesh*) topológia, ktorá tieto uzly po prepája, tak aby pri zlyhaní jedného uzlu dokázali ostatné uzly komunikovať bez problémov [2]. Na obrázku 2.1 je znázornený rozdiel medzi centralizovanou a decentralizovanou sieťou. V prípade centralizovanej siete je pri poškodení jedného uzla možné úplne znehodnotiť celú sieť. V prípade decentralizovaného modelu používaného aj pri sieti Bitcoin, je zlyhanie zariadenia s viacerými spojeniami v poriadku, nakoľko sa s daným uzlom dá komunikovať inou cestou v topológii.



Obr. 2.1: Ilustrácia centralizovanej a decentralizovanej siete [22].

Jednotlivé uzly Bitcoinu majú rôzne funkcie, medzi ktoré patrí napríklad funkcia *"baníka"*. Ďalšími funkciami, ktoré uzly môžu vykonávať je funkcia bitcoinovej peňaženky a funkcia dodržiavania plnej aktuálnej kópie blockchainu [2]. Všetky uzly v sieti majú smerovaciu (anglicky *routing*) funkciu aby dokázali plnohodnotne pracovať v sieti (t.j. pripájať sa na iné uzly, prehliadať a potvrdzovať bloky a transakcie) [21]. Okrem tejto funkcie sa však činnosti jednotlivých uzlov líšia vzhľadom na ich cieľ v sieti. V sieti Bitcoin sú podporované viaceré typy uzlov, tie hlavné z nich sú:

- **Uzly s kompletnou históriou siete** - (anglicky *full nodes*) uzly, ktoré udržiavajú plnú verziu blockchainu so všetkými transakciami. Pri prvej inicializáci nastáva proces *IBD*[7], čo zabezpečí synchronizáciu uzla od prvého bloku blockchainu [7]. Úplne uzly závisia na aktualizáciách zo siete pre udržanie aktuálneho stavu uzla [2]. Najbežnejším softvérom používaným pre beh úplného uzlu je ***Bitcoin Core*** (alebo *Satoshi klient*). Viac ako 75% uzlov v sieti používa práve rôzne verzie tohto softvéru [21].

[6]peer-to-peer sieť - sieť kde všetky uzly komunikujú rovnocenne bez použitia serveru a klientov
[7]IBD - Initial Block Download (slovensky stiahnutie počiatočného bloku) je proces používaný novými uzlami na stiahnutie veľkého množstva blokov

- **Zjednodušené SPV uzly** - (*Simplified Payment Verification* alebo slovensky *uzly so zjednodušeným potvrdzovaním platieb*) uzly, ktoré nepotrebujú udržiavať celú verziu blockchainu a potvrdzujú transakcie pomocou SPV metódy. Používateľ potrebuje iba uchovať kópiu záhlavia bloku najdlhšieho reťazca dôkazov o práci, ktorý môže získať požiadavkami na sieťové uzly, kým nie je presvedčený, že má najdlhší reťazec a získať vetvu hašového stromu, ktorá spája danú transakciu s blokom, v ktorom je časovo označená [24]. Bloky pridané po tejto transakcii ďalej potvrdzujú jej akceptáciu sieťou. Nevýhodou tohto typu uzlov je, že musia plne dôverovať uzlu na, ktorý sú pripojené. Tento uzol, ktorý má plnú verziu blockchainu síce nedokáže oklamať používateľa pridaním nereálnej transakcie, ale dokáže pre ostatné uzly skryť určitú transakciu. Následne nie je možné skontrolovať, či nenastala nejaká chyba, ako je napríklad znovupoužitie UTXO transakcie vysvetlenej v sekcii 2.2, nakoľko uzol nemá uloženú celú verziu blockchainu. Tento problém sa v reálnom svete rieši pomocou pripojenia na viacero uzlov [2].

- **Banícke uzly** - (anglicky *mining nodes*) uzly, ktoré zohrávajú banícku činnosť, vysvetlenej v sekcii 2.4. Tieto uzly môžu byť úplnými uzlami ale nie je to podmienkou, nakoľko sa môžu zúčastňovať ťaženia v bazénoch, vysvetlených v pod sekcii 2.4.4, kde ich úlohou nie je udržiavanie plnej verzie blockchainu. Pri skupinovom ťažení v bazénoch za udržiavanie aktuálnej úplnej verzie blockchainu zodpovedá administrátor.

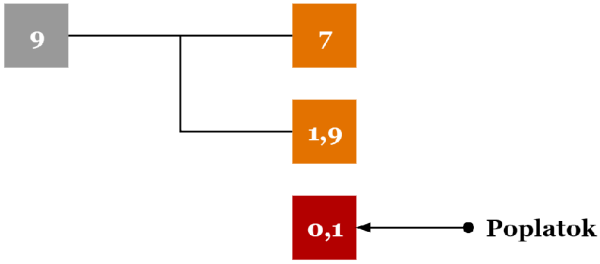
2.2 Transakcie v sieti Bitcoin

Jedná sa o verejný prenos hodnoty Bitcoinu z jednej adresy na druhú. Jednotlivé transakcie sú verejne dostupné v blokoch. Nasledujúce odseky predstavujú pojmy, ktoré sú pri transakciách dôležité.

- **UTXO** reprezentuje zatiaľ nepoužité výstupy transakcií v sieti Bitcoin [18]. Predstavuje základný stavebný prvok bloku transakcie. Vždy keď používateľ príjme nejakú čiastku na svoju adresu, táto čiastka je uložená ako UTXO. V jednotlivých blokoch sa nenachádza žiaden údaj o aktuálnej bilancii užívateľa a z tohto dôvodu sú pre získanie bilancie používané práve UTXO transakcie. Jednotlivé transakcie jedného užívateľa môžu byť v rôznych transakciách a blokoch. Výslednú bilanciu užívateľa je možné získať dohľadaním všetkých týchto UTXO transakcií v sieti pre užívateľovu adresu [2].

- **Coinbase transakcia** je jedinečná transakcia vytvorená baníkom, uložená na začiatku bloku. Tento typ transakcie nemá žiadne vstupy a je vytvorená pre každý novo vyťažený blok v sieti [23]. Baníci túto transakciu využívajú na vyzbieranie svojich odmien. Celková cena odmeny, ktorú baník obdrží je súčet odmeny za blok a poplatky za všetky transakcie, ktoré sú zahrnuté v bloku [3].

- **Poplatky transakcií**, alebo *Transaction Fees* značia poplatok predaný baníkovi (podobné prepitnému v reálnom svete), ktorý vyťažil blok obsahujúci túto transakciu. Baníci chcú na vyťaženom bloku získať čo najviac a preto priorizujú transakcie so zvolenými poplatkami. To znamená, že transakcia s väčšími poplatkami má vyššiu šancu sa dostať do najbližšie ťaženého bloku ako transakcie s veľmi nízkym, alebo až so žiadnym poplatkom [2].



Obr. 2.2: Na obrázku je znázornená ukážková transakcia s jedným vstupom a dvoma výstupmi. Po odčítaní oboch výstupov od vstupnej hodnoty zostala malá časť nezabratá. Táto časť prepadá baníkovi ako odmena za túto transakciu [30].

- **VarInt** - používa sa na reprezentáciu počtu nasledujúcich polí alebo dĺžky nasledujúceho pola. Väčšinou predstavuje 1 bajtové (2 znaky) hexadecimálne číslo. Veľkosť tohto čísla, ako napovedá jeho názov, je premenná v závislosti od hodnoty, ktorú prenáša [30].

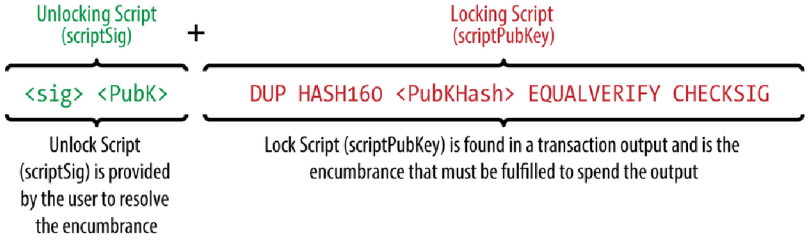
Hodnota	Použité Bajty	Popis
0xfc	1	
0xffff	3	0xfd nasledované 2 bajtmi
0xffffffff	5	0xfe nasledované 4 bajtmi
0xffffffffffffffff	9	0xff nasledované 8 bajtmi

Tabuľka 2.2: Zobrazenie veľkostí pri rôznych prenášaných hodnotách podľa [5]

- **Skript** - malý programovací jazyk na báze programovacieho jazyka *Forth*, používaný pre správu zámkov výstupov. Pracuje na princípe rôznych operačných inštrukcií (anglicky *opcodes*), ktoré sú volané tak aby vykonávali pokyny zľava do prava [30] (zásobníková metóda). Používajú sa dva typy týchto skriptov :
  - **scriptPubKey** - zamykací skript, vkladaný na výstup transakcie, pre zamedzenie použitia tohto výstupu inými používateľmi [30]. Hocijaký užívateľ, ktorý splní podmienky tohto skriptu môže použiť túto sumu[8].
  - **scriptSig** - skript, ktorý zabezpečí splnenie podmienok kladených uzamykacím skriptom a povolí tým použitie zvoleného výstupu [8]. Odomykanie z bezpečnostných dôvodov prebieha tak, že po úspešnom (bez chýb) odomknutí sa vykoná aj zamykací skript. Transakcia je validná a použiteľná až keď sa obe tieto procedúry previedli úspešne [2].

2.2.1 Raw transakcie

Transakcie sú po sieti zasielané v nespracovanom stave označovanom ako *raw* formát [5]. Má nasledujúci formát :



Obr. 2.3: Kombinovanie scriptSig a scriptPubKey pre stavbu transakčného skriptu (prevzaté z [2]).

Názov	**Veľkosť**
Version	4 bajty
Input Counter	1-9 bajtov (VarInt)
Inputs	Premenná
Output Counter	1-9 bajtov (VarInt)
Outputs	Premenná
Locktime	4 bajty

Tabuľka 2.3: Zobrazenie jednotlivých položiek transakcie a ich veľkosť [2].

- **Verzia** (anglicky *Version*) - momentálne používané verzie 1 a 2 [5]. Verzia transakcie napovedá jednotlivým uzlom siete a baníkom aké pravidlá použiť pre transakciu. Táto metodológia je nápomocná pre vývojárov, ktorý môžu vytvárať nové pravidlá bez zmeny už uložených transakcií v sieti [8].
- **Počet vstupov** (anglicky *Input Counter*) - definuje počet nasledujúcich vstupov [30]. Samotná transakcia môže mať viacero vstupov a výstupov, kde ich počet pred samotnou transakciou napomáho k ich rozboru.
- **Vstupy** (anglicky *Inputs*) - ide o míňanie predchádzajúcich výstupov (UTXO). Štruktúra vstupov v sieti Bitcoin je tvorená týmito položkami :
  1. **TXID** - jedinečný identifikátor transakcie v podobe hašu. Dá sa hovoriť o ukazovateli k transakcii obsahujúci UTXO na použitie [2].
  2. **VOUT** - index výstupu transakcie. Vďaka TXID je nájdená jedinečná transakcia v sieti bez špecifického výstupu. VOUT má za úlohu jednoznačný výber UTXO, kde indexovanie jednotlivých výstupov začína od 0 [2].
  3. **Veľkosť ScriptSig** - veľkosť nasledujúcej veľkosti odblokovacieho skriptu [30].
  4. **ScriptSig** - kód na odblokovanie zvoleného výstupu.
  5. **Sequence** - zablokovanie použitia locktime-u pri transakcii. Momentálne nepoužívaná zámerná vlastnosť a to aktualizovanie nepotvrdených časovo zamknutých tranzakcií pred finalizáciou [5]. Nastavuje sa pre väčšinu na maximálnu hodnotu `0xffffffff` [2].

Názov	Veľkosť
TXID	32 bajtov
VOUT	4 bajty
Veľkosť ScriptSig skriptu	1-9 bajtov (VarInt)
ScriptSig	Premenná
Sequence	4 bajty

Tabuľka 2.4: Položky vstupov transakcie a ich veľkosti [30].

- **Počet výstupov** (anglicky *Output Counter*) - definuje počet nasledujúcich výstupov [30].
- **Výstupy** (anglicky *Outputs*) položka transakcie, ktorá určuje akú hodnotu chce používateľ zaslať na danú adresu. Obsahuje dve hlavné časti, definovanie hodnoty a zámok na zabezpečenie.
  1. **Hodnota** (anglicky *Value*) - Bitcoin hodnota v jednotkách satoshi [2].
  2. **ScriptPubKey** - skript na uzamknutie výstupu.

Názov	Veľkosť
Hodnota	4 bajty
Veľkosť ScriptPubKey skriptu	1-9 bajtov (VarInt)
ScriptPubKey	Premenná

Tabuľka 2.5: Položky výstupov transakcie a ich veľkosti [30].

- **Locktime** - nastavuje čas kedy bude možné transakciu vložiť do bloku a ten následne potvrdiť [8]. Táto hodnota dokáže zablokovať transakciu do špecifickej výšky bloku (< `500000000`) alebo časový bod (>= `500000000`) vo formáte *Unix Epoch Time*[8] (slovensky *Unixový čas*). V prípade, že užívateľ nechce zablokovať transakciu, hodnota sa nastavuje na `0x00000000` [30]. Túto hodnotu si je možné predstaviť ako šek, ktorý sa má vyplatiť až o určitú dobu.

2.3 Blockchain

Sieť Bitcoinu funguje decentralizovaným peer to peer spôsobom, to znamená, že je potrebné uložiť na každý uzol v sieti rovnaké údaje o transakciách. Za týmto účelom Bitcoin používa Blockchain technológiu. Túto technológiu si je možné predstaviť ako jednú veľkú reťaz dát, ktorá je distribuovaná po celej sieti, pre každý jeden uzol. Dáta, ktoré sú ukladané do tohto zreťazenia sú transakcie, ktoré už jednotliví baníci potvrdili. Tým, že daný baník transakcie potvrdí je ich možno pridať na koniec Blockchainu a informovať všetky ostatné uzly na sieti, že je potrebné aby si aktualizovali kópiu Blockchainu. Vďaka jednej veľkej postupnosti transakcií je následne veľmi jednoduché potvrdzovať ďalšie transakcie, nakoľko každý jeden uzol na sieti si dokáže skontrolovať podľa doposiaľ vykonaných transakcií aktuálnu bilanciu adresy. Na to aby Bitcoin vedel maximálne využiť silu Blockchainu, má jasne definovanú štruktúru podľa ktorej sa ukladajú jednotlivé transakcie s ostatnými údajmi.

[8] Unix Epoch Time - počet sekúnd, ktoré uplynuli od 1970-01-01T00:00 UTC

2.3.1 Blok

Bloky (anglicky *Blocks*) v Blockchaine udržujú údaje o transakciách. Pri vytvorení nového bloku sa tento uloží za posledne vložený blok a vytvorí miesto pre ďalší blok do budúcna. Každý takýto blok uloží dáta nových transakcií a referenciu na blok za ktorý sa uložil [19]. Pod pojmom *Blok* si je možné v skratke teda predstaviť súbory, ktoré neustále ukladajú transakcie v sieti. Jednotlivé bloky nie sú vytvárané automaticky ale za ich tvorbu sú zodpovední baníci (princíp popísaný v sekcii 2.4). Podobne ako transakcie aj bloky majú definovanú štruktúru a tá sa skladá z dvoch hlavných častí, ktorými sú záhlavie bloku a transakcie.

Názov	Veľkosť	Vysvetlenie
Veľkosť bloku	4 bajty	Veľkosť nasledujúcej časti bloku v bajtoch
Záhlavie bloku	80 bajtov	Položky podľa pod-sekcie 2.3.1
Počet transakcií	1-9 bajtov (VarInt)	Koľko transakcií nasleduje
Transakcie	Premenná	Všetky transakcie v bloku

Tabuľka 2.6: Vizualizácia štruktúry bloku podľa [2]

- **Záhlavie bloku** - (anglicky *Block Header*) nachádza sa na vrchu bloku a ponúka súhrn všetkých dát obsiahnutých v bloku [30]. Záhlavie obsahuje nasledujúce položky:

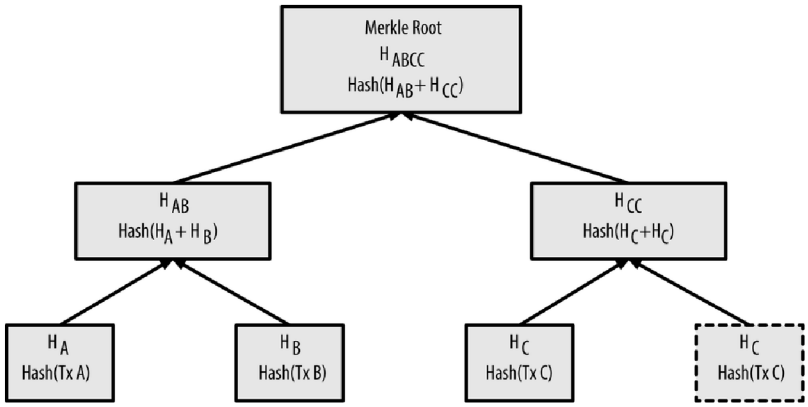
  1. **Verzia** — (anglicky *Version*) definuje verziu bloku, ktorá určuje pravidlá pre danú verziu protokolu (aktuálne verzie 1–4)[5].
  2. **Haš predchádzajúceho bloku** — (anglicky *previous Block Hash*) haš bloku na ktorom vrchu bol tento blok pridaný [30]. Tento prvok je kľúčový pre udržanie zreťazenia blokov. Haš bloku je jedinečný identifikátor bloku, ktorý je vytváraný dvojnásobným hašovaním celej hlavičky bloku pomocou funkcie algoritmu SHA256 [2]. Tento haš si však musí vypočítať uzol v sieti sám, nakoľko sa nenachádza na žiadnom mieste v bloku.
  3. **Koreň Hašového stromu** — (anglicky *Merkle root*)haš koreňového prvku *hašového stromu* bloku. Podrobnejší popis hašového stromu sa nachádza v podsekcii 2.3.2. Dovoľuje užívateľom skontrolovať dáta transakcií.
  4. **Časová značka** — (anglicky *Timestamp*) čas kedy baník daný blok vyťažil v Unixovom čase.
  5. **nBity** — (anglicky *nBits* alebo *Difficulty Target*) cieľový haš v kompaktnom formáte [30] používaný pri ťažení, bližšie popísaný v sekcii 2.4.
  6. **Nonce** — položka používaná baníkmi pre zmenu hašu bloku, podrobnejšie vysvetlená v sekcii 2.4.1.

Názov	Veľkosť
Verzia	4 bajty
Hash predchádzajúceho bloku	32 bajtov
Merkle Root	32 bajtov
Časová značka	4 bajty
nBity	4 bajty
Nonce	4 bajty

Tabuľka 2.7: Položky hlavičky bloku a ich veľkosti [2]

2.3.2 Hašový strom

Jednotlivé transakcie v bloku v ňom nie sú usporiadané podľa poradia ale v stromovej štruktúre, známej ako *hašový strom*. Ako je z názvu zrejme tento strom pracuje primárne s haš kódmi transakcií. Na najnižšej vrstve tohto stromu zvanej listy, sú iba haš kódy transakcií samostatne. Následne sú vytvárané haš kódy na vyššej úrovni stromu a to hašovaním párov hašových kódov nižšej vrstvy. V prípade že uzol nemá dvojicu s ktorou by vstupoval do hašovacej funkcie, je táto transakcia hašovaná samá so sebou. Takýmto spôsobom sa hašujú dvojice až pokým nevznikne jeden jediný haš na najvýššej koreňovej úrovni. Táto hodnota sa nazýva *koreň* hašového stromu a obsahuje teda informácie o každej jednej transakcii, ktorá je v danom bloku obsiahnutá [25]. Celý tento proces je znázornený na obrázku 2.4, kde pri zvolených transakciách bolo nutné jednu z nich duplikovať a hašovať samú so sebou. Pri hašovaní v hašovacích stromoch Bitcoinu sa na hašovanie využíva dvojité hašovanie pomocou `SHA256` (anglicky `double-SHA256`).



Obr. 2.4: Vytvorenie hašového stromu, kde A,B,C sú transakcie v prípade, že je potrebné jednu tranzakciu duplikovať (prevzaté z [2]).

2.4 Ťažba Bitcoinu

Na dodržanie bezpečnosti a validity celej siete sa používajú baníci (anglicky *miners*), ktorý spravujú hlavnú sieť namiesto inštitúcií ako sú napríklad banky. V skratke sa jedná o proces použitia počítačového hardvéru pre potvrdzovanie transakcií, zvýšenie bezpečnosti siete a tvorbu nových Bitcoinov v sieti.

2.4.1 Princíp

Každá transakcia, ktorá je pridaná do bloku musí byť potvrdená a to celkovo v jednom bloku. Do jedného bloku je možné uložiť 1MB transakcií, čo môže byť aj jedna tranzakcia ale zvyčajne sa jedná o tisíce transakcií. Proces potvrdzovania blokov transakcií, pre možnosť pridania ho do blockchainu, sa nazýva ťažba bitcoinu. Samotné ťaženie spočíva v tom, že baník musí uhádnuť riešenie matematického problému taktiež nazývaného *Proof of Work*. Systém Bitcoin využíva haš kódy na identifikáciu bloku. To znamená, že úlohou baníkov je uhádnuť 64 bitový hexadecimálny kód zvaný haš. Tento kód musí byť nižší ako cieľový haš poskytnutý v bloku. Zariadenia baníkov náhodne generujú tieto kódy rýchlosťami megahaš za sekundu (MH/s), gigahaš za sekundu (GH/s) alebo dokonca terahaš za sekundu

(TH/s) [18]. Tento haš sa ovplyvňuje pomocou inkrementácie hodnoty `nonce` (hodnota používaná pri ťažení na zmenu haš kódu bloku) [30]. V prípade, že viacero baníkov dosiahlo toto kritérium víťazom sa stáva baník, ktorému sa to podarilo skôr.

2.4.2 Náročnosť

Pri ťažení sa zohľadňuje ešte jeden faktor, náročnosť. Náročnosť má za úlohu udržiavať priemerné množstvo blokov, ktoré je vložené do blockchainu. Pridanie jedného bloku do blockchainu trvá v priemere 10 minút. Aby táto hodnota ostala konštantná, náročnosť je menená po 2016 blokoch čo predstavuje približne 2 týždne [11]. Samotná náročnosť je teda vypočítaná ako pomer cieľovej náročnosti, ktorá zabezpečí stabilné generovanie blokov každých 10 minút a aktuálnej náročnosti [29].

$$Náročnosť = cieľ\ náročnosti / aktuálna\ náročnosť \tag{2.1}$$

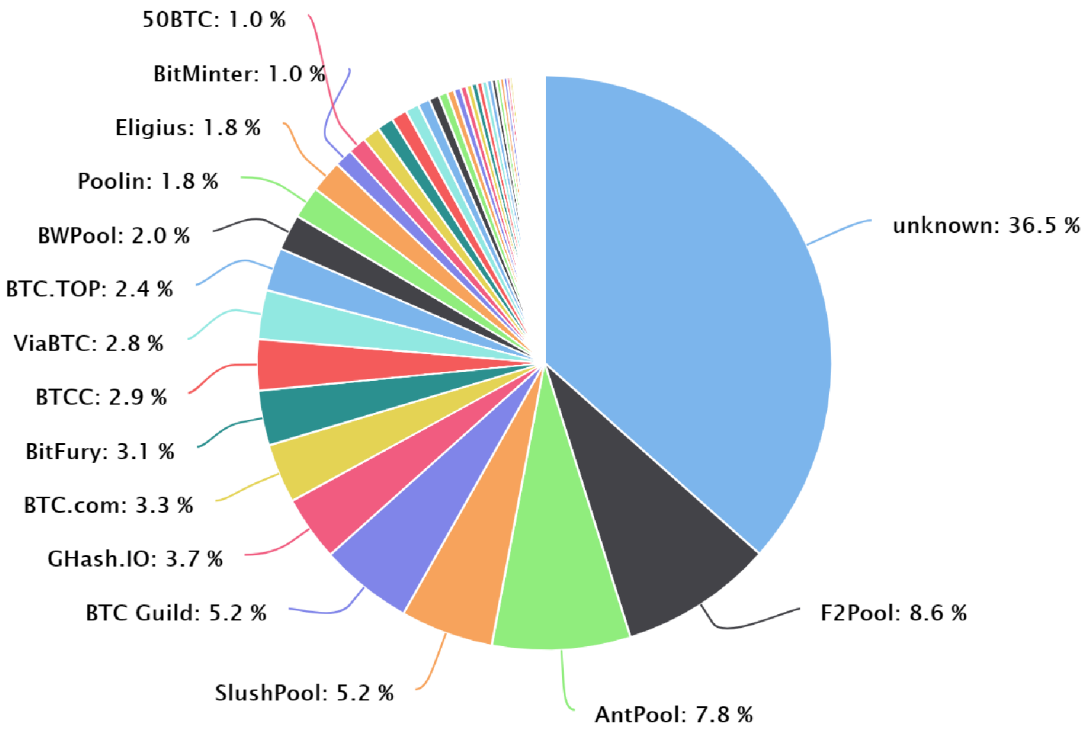
Pre vyššiu bezpečnosť siete je vhodné aby sa v nej nachádzalo čo najviac baníkov. Vysoký počet baníkov totiž znižuje riziko, že niekto dokáže nejakým spôsobom manipulovať so sieťou. Nevýhodou tohto princípu je zvýšenie náročnosti, ktorá sa zvyšuje podľa výpočtového výkonu na v sieti [28]. So stúpajúcou náročnosťou sa znižuje rozsah, ktorý bude vyhovovať ako výsledný haš bloku a tým sa predlžuje čas za ktorý sa približne nájde jeden blok [18].

2.4.3 Odmeny

Jednotliví baníci majú motiváciu ťažby v prijatí celkových poplatkov bloku a odmien za každý novo pridaný blok do blockchainu. Touto odmenou je nový Bitcoin do peňaženky baníka. Hodnota tejto odmeny neustále klesá po štyroch rokoch o polovicu. Dnes sa hodnota tejto odmeny pohybuje na hodnote 12.5 BTC (Bitcoin) a v roku 2020 sa hodnota odmeny zníži na 6.25 BTC [28]. Tým pádom, že odmenu získava vždy iba jeden baník, ktorý sa dostal k výsledku ako prvý, šanca, že práve daný účastník bude šťastlivcom je veľmi nízka. Celková šanca sa zvyšuje použitím kvalitnejšieho hardvéru, ktorý dokáže rýchlejšie generovať náhodné riešenia.

2.4.4 Mining Pools

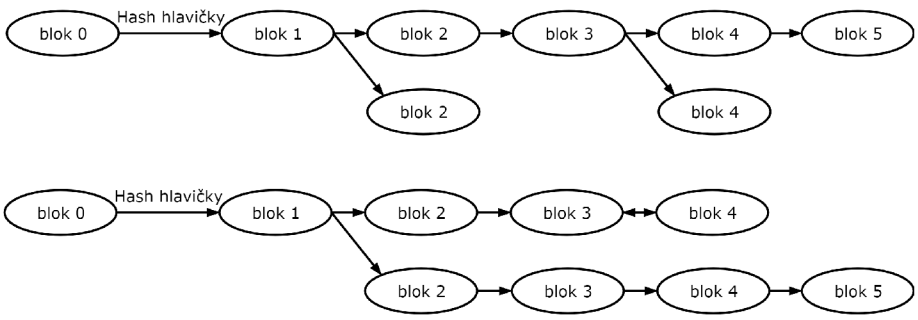
Nakoľko účastníci by nemali žiadnu šancu na víťazstvo oproti silnejšej konkurencii pri generovaní hašu, používajú často skupiny baníkov tzv. `Mining Pools`. Táto skupina im dovoľuje zdieľať si v rámci skupiny výpočtový hardvér a tým si zvýšiť šancu na úspech. Pri tejto stratégii sa možná odmena rozdelí medzi každého člena v skupine a to podľa percent práce, ktorú vykonal pri hľadaní riešenia [18].



Obr. 2.5: Viditeľné podiely trhu medzi Mining Pools k aktuálnemu dátumu, do ktorých sa je možné pridať (prevzaté z [14]).

2.4.5 Výška bloku pri konfliktoch

Každý blok v sieti má určitú výšku, táto výška značí koľko blokov predchádzalo danému bloku. Prvý Genesis blok má teda hodnotu výšky rovnú nule, nakoľko sa jednalo o prvý blok v sieti. Hodnota výšky sa následne pre každý nový blok lineárne zvyšuje [4]. Táto hodnota je v celej sieti jedinečná avšak iba teoreticky. Prakticky totiž pri ťažení Bitcoinu môže prísť aj k situácii, že viacero baníkov vyťaží blok rovnakej výšky v rovnakom čase. V takejto situácii sa obe bloky pridajú do blockchainu a vytvoria tým rozcestie (fork) v zreťazení blokov. Každý uzol si následne individuálne vyberá, ktorý blok bude akceptovať. Zvyčajne uzly použijú prvý viditeľný blok v sieti. Pri vytváraní ďalších blokov baníkmi sa bloky pridávajú iba na jednu stranu vetvy. Vetva, ktorá je následne dlhšia je považovaná uzlami za vhodnejšiu a prioritne si vyberajú dlhšiu stranu vetvy pred kratšou. Bloky, ktoré sa nachádzajú na ďalej nepoužívanej vetve blockchainu sa nazývajú `stale` bloky [6]. Z dôvodu, že v určitých momentoch nemusí byť výška bloku jedinečná hodnota, nedá sa považovať za jedinečný identifikátor.



Obr. 2.6: Ilustrácia vytvorenia viacerých blokov súčasne. Na vrchnej časti obrázku je vidieť častý jav vytvárania rozcestia a následné používanie jednej dlhšej vetvy. Na spodnej časti obrázku je vidieť zriedkavý jav, kedy uzly siete používajú viacero vetiev [6].

Každý uzol si z dôvodu nutnosti pracovania v sieti s momentálne najdlhšou vetvou, musí uchovávať určitú históriu predchádzajúcich blokov. Uzol následne pri každom prijatom bloku kontroluje jeho zaradenie do aktuálnej vetvy. V prípade že prijatý blok nevytvára jedno dlhé zreťazenie blokov v lokálnej databáze uzla, uzol musí prehodnotiť predchádzajúce prijaté bloky. Ak sa preukáže, že niektorý blok je ďalej nepoužívaný uzly musia zabezpečiť synchronizáciu na aktuálnu vetvu siete. Tento jav môže byť uskutočnený odstránením všetkých všetkých transakcií v bloku a následným nahradením prázdneho miesta blokom, z aktuálnej vetvy.

Kapitola 3

Návrh riešenia

Cieľom tejto kapitoly je priblížiť proces samotného návrhu implementácie riešenia.

Začiatok kapitoly umožňuje čitateľovi priblížiť detaily zvoleného databázového systému. Následne je podrobne popísaná architektúra výsledného databázového systému aplikácie. Pri návrhu architektúry aplikácie sú najprv uvedené jednotlivé požiadavky, ktoré má výsledná aplikácia podporovať. Po definovaní požiadaviek je popísaný samotný postup návrhu architektúry, pri snahe docieliť čo najrýchlejšiu funkčnosť aplikácie, ako aj jej výsledná architektúra.

Ďalej je v tejto kapitole priblížený softvér na synchronizáciu blockchainu, ktorý je potrebný pri implementácii výslednej aplikácie. Na konci kapitoly sú uvedené pomocné knižnice a nástroje, použité pri realizácii.

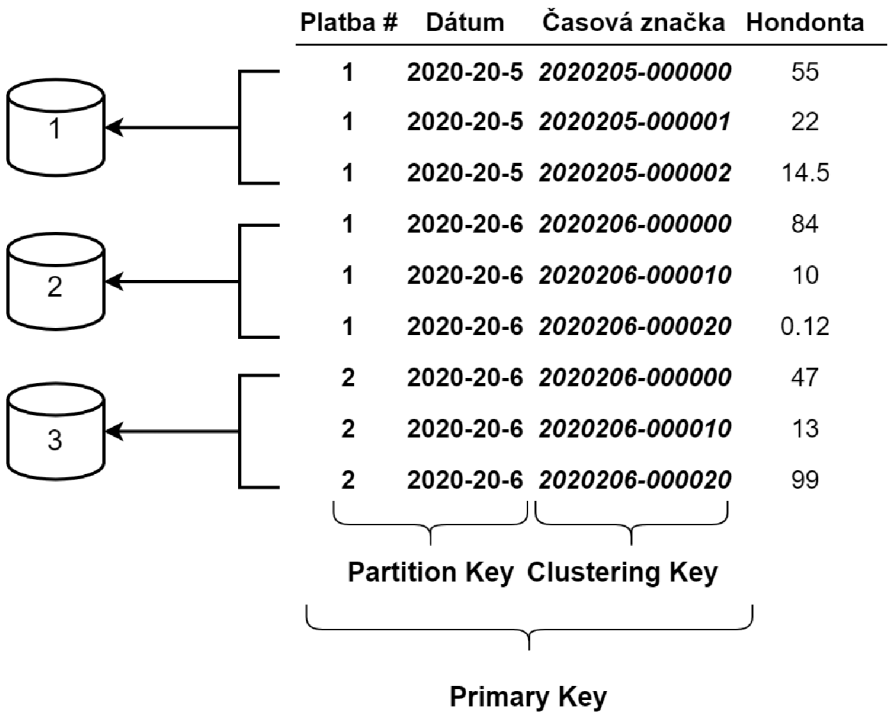
3.1 Zvolený databázový systém

Najdôležitejším prvkom pre správne a efektné fungovanie je vytvorenie vhodného návrhu databázy, ktorá dokáže efektívne plniť všetky požiadavky užívateľa. Pri výbere vhodného databázového systému je kľúčový výber správneho typu systému už na úplnom začiatku.Nakoľko úplná verzia blockchainu, popísaná v sekcii 2.3 má ku dnešnému dňu veľkosť vyše 301 GB a stále sa zväčšuje, je mimoriadny dôraz kladený na možnosť pracovania rýchlo s veľkým množstvom dát. Tradičné relačné databázy v tomto prípade teda už pri návrhu nedávajú veľký zmysel. Vhodným riešením je NoSQL typ databáz, ktorý bol vytvorený vyslovene pre rýchlu a jednoduchú prácu s veľkým množstvom dát.

Mnou zvoleným databázovým systémom je `Apache Cassandra` od firmy `Apache Software Foundtation`. Jedná sa o jednoduchý softvér, ktorého syntax (jazyk CQL) je podobná tradičným relačným databázam. Používa stĺpcovo orientovaný databázový systém, čiže ukladá dáta do jednotlivých stĺpcov a tie do rodín stĺpcov (podobné tabuľkám v relačných databázach). Jednotlivé dáta následne rovnomerne rozdelí na všetky zariadenia v sieti, tak aby sa zaťaženie na jednotlivé uzly rovnomerne rozložilo. Týmto prístupom sa teda vytráca stratégia známa ako *master / slave*, kde o operáciách rozhoduje hlavný uzol. Použitím stratégie rovnomerného zaťaženia každého uzla dokáže každý uzol reagovať na jemu zaslané požiadavky používateľa a tým značne zrýchľovať čas odozvy.

Tento databázový systém dokáže ukladať a následne vyhľadávať údaje, podobne, ako ostatné databázové systémy, pomocou primárneho kľúča. Zloženie primárneho kľúča v systémoch Cassandra je však od ostatných systémov trochu odlišné. Povinnou položkou primárneho kľúča je hodnota nazývaná *oddielový kľúč* (anglicky - partition key), ďalej označovaný

ako PK. Hodnota tohto kľúča určuje oddiel, na ktorý sa daný záznam uloží. V prípade, že je primárny kľúč zložený iba z oddielového kľúča, tento kľúč je aj primárnym kľúčom a teda sa v celej databáze smie vyskytnúť iba raz. Druhým atribútom používaným pri zostavovaní primárneho kľúča je *blokovací kľúč* (anglicky - clustering column), ďalej označovaný CK. Definuje rozdelenie jednotlivých záznamov v rámci už definovaných oddielov. Táto vlastnosť povoľuje vytváranie záznamov, pri ktorých sa výsledky môžu filtrovať aj podľa hodnoty tohto kľúča. Obe vyššie uvedené kľúče môžu byť zložené z viacerých hodnôt, pomocou ktorých vytvárajú už zmienený primárny kľúč. Haš hodnota primárneho kľúča je označovaná ako *token* a každý uzol v topológii uzlov ma pridelené určité rozmedzie token-ov, ktoré obsluhuje. Celý systém vyššie menovaných kľúčov je znázornený na obrázku 3.1.



Obr. 3.1: Zobrazenie významu jednotlivých kľúčov používaných v systémoch Cassandra a ich následné rozdelenie do príslušného uzla.

Databázový systém Cassandra [16] podporuje tvorbu materializovaných pohľadov, ktoré vytvárajú ďalšiu tabuľku v pamäti, bez nutnosti manuálnej správy údajov dedených tabuliek, nakoľko túto činnosť spravuje server Cassandra. Pri použití materializovaného pohľadu následne nie je potrebné denormalizovať[1] celú tabuľku za účelom zmeny primárneho kľúča pre inú obsluhovanú požiadavku z rovnakej tabuľky.

3.2 Požadovaná funkčnosť

Prvým krokom pri tvorbe určitej schémy NoSQL databáz je určenie požiadaviek, ktoré má aplikácia podporovať. Podľa týchto požiadaviek sa následne môže vytvoriť efektívna databáza. Podporované požiadavky na získanie údajov sú nasledovné:

[1]Denormalizácia - proces duplikovania dát v databázach pre zvýšenie rýchlosti čítania na úkor zápisu údajov.

- **Získanie bloku na základe výšky bloku v blockchaine**
- **Získanie bloku na základe haš kódu bloku**
- **Haš kód bloku na základe výšky bloku**
- **Transakcie na základe haš kódu transakcie**
- **Všetky transakcie v ktorých figuruje zadaná adresa**
- **Všetky doposiaľ neminuté UTXO transakcie pre danú adresu**
- **Aktuálnu bilanciu adresy ako aj hodnoty jej celkových výdajov a príjmov v transakciách**

Pre čo najrýchlejšie fungovanie databázového systému Cassandra, je vhodné pre každú požiadavku vytvoriť vlastnú tabuľku. Tento prístup je zaužívaný z toho dôvodu, že Cassandra nedokáže používať spojenia viacerých tabuliek, ako je zvykom u relačných databáz. Pre jednotlivé požiadavky užívateľa je následne nutné vo väčšine prípadov komunikovať s viacerými uzlami, čo značne spomaľuje celý chod aplikácie. Vytváranie viacerých tabuliek sa dokonca odporúča aj pri možnom duplicitnom definovaní niektorých dát, nakoľko základnou terminológiou databázového systému Cassandra, je prednostné rýchle spracovanie požiadavky aj na úkor zväčšeného ukladacieho priestoru databázy.

3.3 Architektúra databázy

Pri analýze blockchainu bolo potrebné prihliadať aj na veľkosť aktuálnych súborov blockchainu a množstvo dát, kde nie všetky hodnoty sú priamo prístupné v užívateľovi ľahko čitateľnom formáte. Veľké množstvo tabuliek a zlých typov atribútov, by mohlo celý proces importovania spomaliť o niekoľko dní až týždňov. Množstvo údajov, ktoré samotný blockchain ukladá sú taktiež iba priebežne kalkulované hodnoty, neprístupné priamo vo fyzickom súbore. Tieto údaje je potrebné získať až pri kompletnom spracovaní celého úložiska. Pri zohľadnení samotných pravidiel tvorby tabuliek v systémoch Cassandra ako aj rýchlosti získania prístupu k údajom.

Tabuľka na obrázku 3.1 vystihuje užívateľskú požiadavku na získanie jednotlivého bloku v prípade, že užívateľ pozná jeho haš kód. Zvolený oddielový kľúč samostatne zabezpečuje jedinečnosť každého jedného bloku, nakoľko vygenerovanie dvoch rovnakých haš kódov je takmer nemožné. Transakcie v tejto tabuľke sú uvedené pomocou ich haš kódov. V prípad, keď chce užívateľ získať blok na základe jeho výšky, sú použité materializované pohľady. Materializovaný pohľad zjednoduší správu všetkých tabuliek blokov nakoľko sú vytvárané, upravované aj vymazávané automaticky pri práci s rodičovskou tabuľkou bloku. Sú vytvorené dva pohľady (pre nájdenie bloku, prípadne iba jeho hašu), obe s výškou bloku zvolenou ako oddielovým kľúčom. Z tohto dôvodu je nutné v rodičovskej tabuľke mať hodnotu výšky definovanú ako blokovací kľúč.

Názov	Typ
«PK» Haš bloku	text
«CK» Výška bloku	bigint
Haš predchádzajúceho bloky	text
Čas vyťaženia bloku	timestamp
Merkle Root	text
Nonce	bigint
Bity	bigint
Verzia	int
Počet transakcií v bloku	int
Transakcie v bloku	list<text>

Tabuľka 3.1: Entita pre uloženie jednotlivých blokov (`block_by_hash`).

Všetky transakcie sú v databáze uložené a pripravené iba na hľadanie podľa hašu transakcie. Tabuľka 3.2 zobrazuje aj všetky ostatné položky, ktoré sú k dispozícii. Oddielovým kľúčom v tejto tabuľke je teda haš kód transakcie. Ako blokovací kľúč je zvolený haš kód bloku. Aj napriek tomu, že celý blockchain Bitcoinu je organizovaný tak, že jedna transakcia môže byť obsiahnutá iba v jednom bloku, niekedy toto pravidlo nemusí platiť. Pri konfliktoch opísaných v sekcii 2.4.5 môže nastať situácia, kde dva bloky s dvoma rôznymi haš kódmi budú obsahovať rovnakú transakciu. Táto situácia by mohla znamenať porušenie platnosti databázy v prípade označení doposiaľ platného bloku za blok, ktorý už ďalej nepatrí do hlavnej vetvy blockchainu. V prípade objavenia takéhoto bloku je nutné jeho odstránenie, ako aj všetkých v ňom obsiahnutých transakcií, pri čom však treba dbať na to, aby sa neodstránila aj rovnaká transakcia z nového bloku, ktorý už do hlavnej vetvy patrí.

Názov	Typ
«PK» Haš transakcie	text
«CK» Haš kód bloku	text
Číselná hodnota transakcie	decimal
Poplatok transakcie	decimal
Verzia	int

Tabuľka 3.2: Entita zobrazujúca informácie o transakciách (`transaction_by_hash`).

Vstupy transakcií v tabuľke 3.3 sú ukladané a pripravené pre prístup pri známom haši transakcie, prípadne aj samotného identifikátora vstupu. Tieto hľadania sa vykonávajú pri každom výbere transakcie z databázy alebo pri hľadaní vstupov transakcie. Vstupy nie sú ukladané priamo do tabuliek transakcií z toho dôvodu, že pri prechádzaní súborov obsahujúcich dáta blockchainu, nie sú pri vstupoch uvedené žiadne informácie okrem odkazu na predošlí výstup transakcie, ktorý je v transakcii použitý. Samotné neustále aktualizovanie stĺpca tabuľky dohľadaním predchádzajúceho výstupu pri vkladaní vstupov by bolo časovo náročné a celý chod importovania by sa značne spomalil. Napriek tomu, že systém Cassandra ponúka možnosť aktualizovania jednotlivých riadkov, pre prípad viacerých vstupov v jednej transakcii by bolo potrebné vytvoriť zoznam tzv. *frozen* (slovensky - zmrazených) elementov. Pri aktualizovaní takéhoto elementu je potrebné načítať celý jeho obsah a následne ho celý vložiť, čo predstavuje zbytočnú záťaž, ktorá je odstránená samostatnou tabuľkou pre vstupy a rovnako aj výstupy.

Pre tabuľku vstupov je taktiež vytvorený jeden materializovaný pohľad používaný pri kontrolovaní stavu výstupu. Nakoľko užívateľ musí vedieť vyhľadať všetky výstupy, ktoré doposiaľ neboli použité ako aj vstupy inej transakcie, sú označené ako UTXO. Tieto vstupy musia byť dostupné aj pri hľadaní pomocou výstupov. Pri hľadaní UTXO sa pre každý doposiaľ nepoužitý vstup vyhľadá informácia, či už nebol použitý, pomocou jeho hašu a identifikátora (v tabuľke 3.3 položky predchádzajúceho hašu a identifikátora výstupu).

Názov	Typ
«PK» Haš kód transakcie	text
«CK» Id	int
Hodnota	decimal
Haš predchádzajúcej transakcie	text
Id výstupu predchádzajúcej transakcie	int
Adresa odosielateľa	text
ScriptPubKey odosielateľa	text

Tabuľka 3.3: Entita znázorňujúca uloženie vstupov transakcie do databáze (`input_by_hash`).

Výstupy, ktoré je možné získať pre každú transakciu sú uvedené v tabuľke 3.4. Pre túto tabuľku nie je nutné vytvorenie materializovaného pohľadu nakoľko každé vyhľadanie sa dá realizovať pomocou haš kódu transakcie prípadne kombináciou tohto kódu s identifikátorom výstupu.

Názov	Typ
«PK» Haš kód transakcie	text
«CK» Id	int
«CK» Stav výstupu	int
Hodnota	decimal
Adresa prijímateľa	text
ScriptPubKey prijímateľa	text

Tabuľka 3.4: Entita pre uloženie všetkých výstupov požadovaného bloku. Stav výstupu označuje, či daný výstup už bol použitý alebo sa považuje za zatiaľ nepoužitý výstup (`output_by_hash`).

Poslednou tabuľkou, ktorá je vytvorená v databáze, je tabuľka pre prácu s adresami 3.5. Táto tabuľka má za úlohu čo najrýchlejšie sprístupniť všetky výstupy pre danú adresu, ktorá sa dekóduje na `ScriptPubKey`. Proces dekódovania je podrobne vysvetlený v sekcii 4.6. Následne je možné do-hľadať všetky výstupy kombináciou hašu transakcie a identifikátora výstupu. Pomocou tejto tabuľky sú vykonávané všetky požiadavky na správu adries. Týmito požiadavkami sú získanie všetky transakcie a UTXO transakcie, v ktorých adresa figuruje alebo výpočet bilancie adresy. Všetky menované požiadavky majú spoločnú vlastnosť, a to potrebu nájsť všetky výstupy. Z tohto dôvodu je použitá iba jedna tabuľka, ktorá spĺňa všetky požiadavky.

Názov	Typ
«PK»ScriptPubKey	text
«CK» Haš kód transakcie	text
«CK» Id výstupu	int
Hodnota	decimal

Tabuľka 3.5: Entita pre uloženie dát potrebných na splnenie požiadavkov pri hľadaní pomocou adresy (`output_by_address`).

3.4 Bitcoin Core - uzol s kompletnou históriou

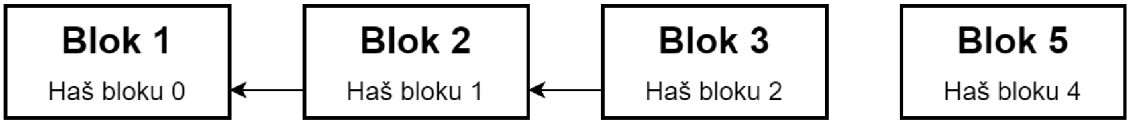
Ako typ uzla je zvolený uzol s kompletnou históriou siete, podľa sekcie 2.1. Pri tomto type uzla nie je potrebné dbať na dôveryhodnosť ostatných uzlov v sieti, aj za dôsledku stiahnutia veľkej štruktúry dát. Za synchronizačný softvér bol zvolený Bitcoin Core[2], ktorý je oficiálne voľne dostupným softvérom pre fungovanie uzla s kompletnou históriou siete, od samotných vývojárov Bitcoinu. Dodávaný softvér obsahuje verziu s grafickým užívateľským rozhraním *bitcoin-qt*, ktoré je väčšinou využívané bežnými používateľmi. Či už grafická verzia alebo verzia *bitcoind*, ktorá dokáže bežať ako *daemon*[3], poskytuje možnosť komunikácie s užívateľom pomocou RPC [4] serveru. Pre implementáciu tejto práce je použitá komunikácia pomocou RPC volaní ako aj priamy prístup k úložisku blokov na disku. Formát týchto súborov je štandardizovaný a je možné ich prehliadať napríklad pomocou rôznych knižníc, ktoré už sú implementované.

Bitcoin Core sa pri prvom spustení synchronizuje s ostatnými uzlami v sieti procesom *IBD*, ktorý trvá niekoľko hodín, v niektorých prípadoch aj niekoľko dní. Po stiahnutí tohto veľkého objemu dát (aktuálne 301 GB) sa jednotlivé bloky sťahujú postupne a to pomocou prednostného spracovania hlavičiek (anglicky - headers-first). Tento prístup je používaný od verzie aplikácie `0.10.0`. Pred touto verziou program pracoval na princípe prednostného spracovania blokov (anglicky - blocks-first), kde sa sťahovali celé bloky od ostatných uzlov v sieti. V tomto prístupe sa stávalo, že klient mohol obdržať bloky, ktoré sa označujú ako *orphan blocks* (slovensky - bloky označené ako siroty). Tieto bloky, znázornené na obrázku 3.2, nie sú takýmto uzlom ihneď vyhodnotené ako validné, ale na určenie validity je potrebná ďalšia komunikácia s uzlom, ktorý tento blok zaslal. Po prijatí všetkých chýbajúcich blokov od uzla, musí klient (v tomto prípade uzol, ktorý žiadal o zaslanie všetkých chýbajúcich blokov) sám vyhodnotiť validitu bloku. Všetky uzly, ktoré pracujú na princípe prednostného spracovania hlavičiek, už takéto bloky nemôžu prijať. Uzly, ktoré zasielajú svoje bloky ostatným po sieti, odošlú všetky hlavičky blokov, ktoré obsahujú. Uzol vyhodnotí či sa jedná o najdlhšiu možnú reťaz blokov a nové bloky sú stiahnuté len v tom prípade, že všetky bloky na seba nadväzujú. Týmto spôsobom sa zamedzí prijatiu blokov, ktoré nenadväzujú na aktuálnu hlavnú reťaz blokov a nie je potrebné daný problém riešiť v implementovanej aplikácii.

[2] https://bitcoin.org/en/bitcoin-core/

[3] Daemon - typ programu, ktorý beží v pozadí bez prevádzkovania užívateľom

[4] RPC - remote procedure call (slovensky - vzdialené volanie procedúr) je technológia, ktorá dovoľuje program vykonávať na inom mieste ako je umiestnený volajúci program



Obr. 3.2: Zobrazenie možného orphan bloku v sieti Bitcoinu. Blok 5 bez odkazu na blok v hlavnom blockchaine, označený ako orphan blok.

3.5 Použité knižnice

Programovacím jazykom tejto práce je `C#` na platforme `.NET`. Pre tento jazyk ako aj pre platformu `.NET` je knižnica *NBitcoin*[5] najvhodnejšou a najjednoduchšou alternatívou, ktorá ponúka širokú škálu funkcií pre prácu s technológiou Bitcoinu. Táto knižnica implementuje komunikáciu s uzlom Bitcoin Core pomocou RPC volaní ako aj samotnú analýzu súborov blockchainu. Definuje všetky potrebné kodéri a dekodéri, ktoré sú ďalej popísane v sekcii 4.6. Knižnica je pravideľne aktualizovaná, čo zabezpečuje implementáciu všetkých najnovších zmien v protokoloch Bitcoinu. Pri implementovaní boli vyskúšané viaceré knižnice, ktoré dokázali preskúmavať bloky priamo cez súbory. Za zmienku stojí knižnica *BlockchainParser*[6], ktorá ponúka zrovnateľné výsledky v syntaktickej analýze súborov blockchainu. Táto knižnica však nebola použitá z dôvodu nízkej miery aktualizácií a teda podpory nových štandardov.

Cieľom druhej časti tejto práce je vytvorenie Rest API. Za týmto účelom je použitá šablóna ASP.NET Core Web API [1]. Na komunikáciu s databázovým systémom Cassandra je použitý ovládač Datastax C# Driver [7].

[5] https://github.com/MetacoSA/NBitcoin
[6] https://github.com/ladimolnar/BitcoinBlockchain
[7] https://docs.datastax.com/en/developer/csharp-driver/3.15/

Kapitola 4

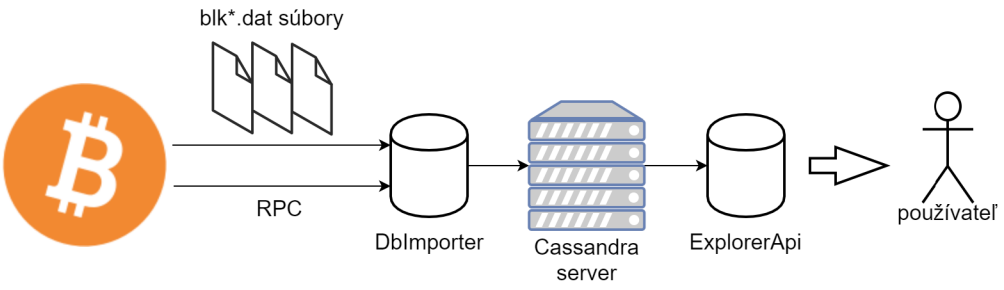
Implementácia

V tejto kapitole sú opísané implementačné detaily výslednej aplikácie. Jedná sa o kapitolu, ktorá reprezentuje najdôležitejšiu časť práce. Jej základným účelom je priblíženie spôsobu riešenia importovania transformovaných dát siete Bitcoin a ozrejmiť niektoré aspekty riešenia problémových situácií.

Zo začiatku čitateľovi približuje základnú charakteristiku implementovaného riešenia, teda proces inicializačného behu aplikácie, spôsob importovania údajov do databázy ako aj následného udržovania databázy v aktuálnom stave. Ďalej je vysvetlený proces, ktorým aplikácia udržiava svoje údaje validné a akým spôsobom dokáže riešiť prípadné konflikty v sieti. Na konci kapitoly je uvedený spôsob dekódovania adries.

4.1 Prehľad

Výsledná aplikácia má vykonávať počas svojho behu dva rozličné úkony, napĺňanie databázy údajmi zo siete a zároveň poskytovať užívateľovi možnosť komunikovať s touto databázou pomocou Rest API. Z dôvodu aby tieto dve činnosti mohli byť vykonávané súčasne sú implementované ako dve samostatné aplikácie `DbImporter` a `ExplorerApi`, kde každá plní svoju činnosť nezávisle od druhej aplikácie. Grafické znázornenie funkčnosti je znázornené na obrázku 4.1. Napriek tomu, že je možné komunikovať s Rest API aj počas importovania blokov Bitcoinu do databázy, neodporúča sa pracovať a vytvárať požiadavky počas inicializačného procesu importovania. V priebehu kedy sa importovací mechanizmus snaží čo najrýchlejšie importovať do databázy veľké množstvo historických údajov z blockchainu, pričom databáza môže byť úplne vyťažená a nemusí zvládať odpovedať na požiadavky užívateľov v maximálnom povolenom čase na odpoveď.

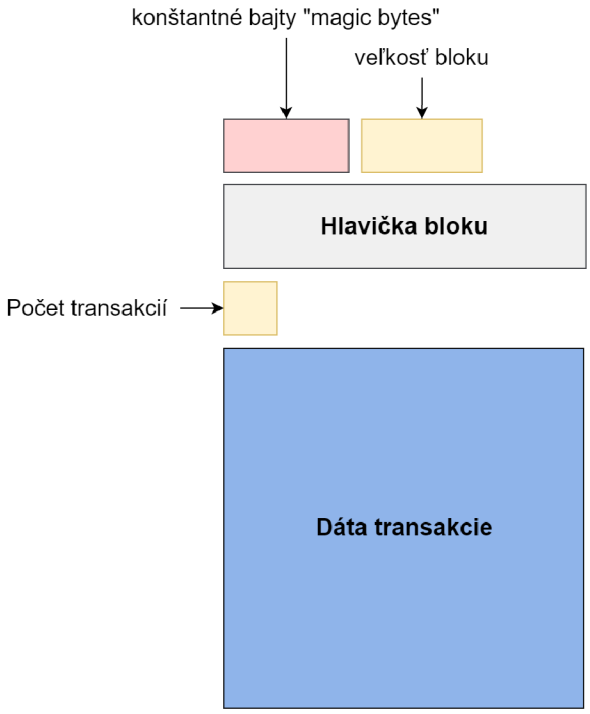


Obr. 4.1: Diagram funkčnosti implementovaného riešenia.

Samotné importovanie do databázy sa teda delí na dva procesy, ktoré sú rozdelené na dvoch po sebe nasledujúcich dejov. Importovanie historických údajov je prvým procesom, ktorý musí prebehnúť celý na to, aby mohla byť databáza úspešne zaplnená všetkými potrebnými údajmi, akými sú napríklad výšky blokov. Po tomto procese sa aplikácia periodicky aktualizuje novými údajmi v reálnom čase.

4.2 Prvotné importovanie

Pri prvom spustení aplikácie na importovanie údajov sa zaháji proces prvotného importovania blokov. Tento proces importuje všetky doteraz stiahnuté bloky do databázy. Kvôli zabezpečeniu čo najrýchlejšieho importovania do databázy nie je použité posielanie požiadavok pomocou RPC volaní ale priamy prístup k súborom uloženým na disku. Tieto súbory sú stiahnuté pomocou softvéru Bitcoin Core a je ich možné nájsť v priečinku *~/.bitcoin/blocks/*, pod názvom *blk*.dat*. Každý súbor má maximálnu veľkosť nastavenú na 128MiB a pri zaplnení kapacity jedného súboru je vytvorený nový, do ktorého sa postupne ukladajú údaje. Súbory majú štandardizovanú štruktúru, zobrazenú na obrázku 4.2. Pri analýze týchto súborov je potrebné dbať na poradie blokov v súboroch. Jednotlivé bloky pre zefektívnenie rýchlosti sťahovania `IBD` softvér Bitcoin Core sťahuje paralelne a nečaká na prijatie aktuálne požadovaného bloku (pri základom nastavení klienta je odchýlka 1024 blokov).



Obr. 4.2: Štrukúra uloženia blokov v súboroch blk*.dat.

Pre syntaktickú analýzu súborov je použitá trieda `BlockStore` z knižnice NBitcoin. Táto trieda obsahuje funkcionalitu, ktorá prechádza všetky bloky vo fyzicky uložených súboroch na disku. Syntaktická analýza jednotlivých blokov prebieha podľa ich štruktúry v súbore a to nasledovne:

```
1 public async Task DbImportAsync()
```

```
2 {
3     _blockStore = new BlockStore(Config.BlocksDir, Network.Main);
4     foreach (var block in _blockStore.Enumerate(false))
5     {
6         foreach (var tx in block.Item.Transactions)
7         {
8             foreach (var input in tx.Inputs)
9             {
10
11            }
12            foreach (var output in tx.Outputs)
13            {
14
15            }
16        }
17    }
18 }
```

Výpis 4.1: Ukážka zdrojového kódu na priechod jednotlivých blokov. Jednotlivé metódy na importovanie dát do databázy ako aj medzi-výpočty v cykloch sú pre jednoduchosť vynechané.

Samotné výšky blokov sú kvôli nezoradeniu blokov zapísané do tabuliek až po skončení prvotného importovania. Táto procedúra je zrealizovaná pomocou metódy `GetSlimChain()` implementovanej v knižnici NBitcoin. Pomocou nej sa do aplikácie stiahne aktuálna najdlhšia vetva blokov, ktorú má lokálny uzol k dispozícii. Všetky bloky, ktoré sú importované do databázy sú následne aktualizované hodnotou výšky bloku, ktorú má daný haš bloku pridelenú v stiahnutom blockchaine. Po aktualizovaní je proces prvotného importovania ukončený a aplikácia prechádza do stavu periodických aktualizácií.

4.3 Stratégia importovania

Pri vkladaní dát na server Cassandra, je možné použiť množstvo metód. Rozhodujúcim elementom tejto práce je však ich rýchlosť. Pre viacnásobné vykonávanie rovnakých operácií vkladania na server je použitá činnosť predpripravených príkazov (anglicky - prepared statements). Tieto príkazy sú definované na začiatku komunikácie so serverom, ktorý si jednotlivé príkazy ukladá do vyrovnávacej pamäte a pri opakovanom použití rovnakého príkazu nemusí analyzovať celý príkaz od klienta, ale použije len jeho identifikátor. V rámci predpripravených príkazov, je vo väčšine prípadov použitý príkaz `INSERT`, pričom sú k nemu pri exekúcii pripojené parametre, potrebné pre jednotlivé vkladania. Príkaz `UPDATE` je použitý iba pri aktualizovaní tabuliek blokov jeho výškou v aktuálnej vetve.

Pri projektoch, ktoré vyžadujú realizáciu rýchleho importovania údajov, sa odporúča vykonávať jednotlivé predpripravené príkazy asynchrónne. Tento projekt je však pri inicializačnom importovaní implementovaný použitím stratégie dávok príkazov. Všetky príručky pre používateľov databázových systémov navrhujú na zefektívnenie rýchlosti spracovanie údajov asynchrónne vykonávané predpripravené príkazy. Pri použití dávky sa jedna dávka týchto príkazov zašle na jeden uzol servera, nazývaného koordinátorom. Tento uzol ukladá iba určité rozmedzie primárnych kľúčov a ostatné príkazy odosiela ďalej na susedné uzly. Týmto spôsobom je možné koordinačný uzol úplne vyťažiť operáciami určenými pre ďalšie

uzly v sieti. V riešení tohto projektu je však použitý iba jeden uzol Cassandra, tým pádom je lokalita na jednom uzly 100% a všetky príkazy sú automaticky vykonávané uzlom, na ktorý sa dávka odošle. Z tohto dôvodu je pre vkladanie údajov použitá stratégia asynchrónneho odosielania dávok predpripravených príkazov na server. Celú koordináciu odosielania dávok realizuje trieda `BatchImporter`, ktorá zabezpečuje nepresiahnutie maximálneho počtu príkazov v jednej dávke a rovnako aj limit aktuálne rozpracovaných dávok. Aktualizácia výšky blokov je však vykonávaná použitím asynchrónnych predpripravených požiadaviek v maximálnom počte 100. Toto riešenie je implementované z dôvodu, že každý príkaz o aktualizáciu výšky bloku musí byť vykonaný v troch tabuľkách a v aplikácii sa dostavovali chybové hlásenia o vyťažení uzla použitím vyššie spomínaných várok alebo vyššieho limitu povolených operácií.

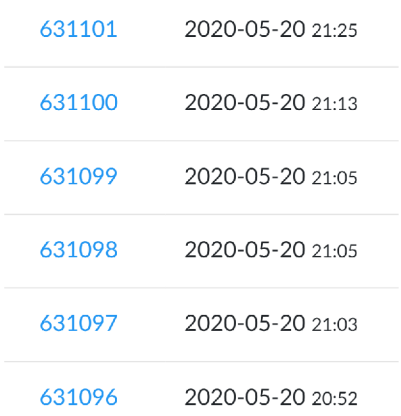
Pri vkladaní nových údajov do databázy je potrebné, aby aplikácia nejakým spôsobom ukladala svoj postup pre prípad reštartovania aplikácie. Najjednoduchším riešením je z databázy vybrať blok s najvyššou výškou a začať od daného bloku synchronizáciu. Táto metóda pri použití databázového systému Cassandra nie je realizovateľná, nakoľko prípadné agregačné funkcie na celú tabuľku, v tomto prípade tabuľku blokov, sú vykonávané čítaním každého jedného záznamu v tabuľke a následným vybraním maximálnej hodnoty, prípadne počtu záznamov. Tento proces je v rozumnom čase nerealizovateľný a z tohto dôvodu je do databázy pridaná tabuľka na uchovávanie stavu databázy. Do tejto tabuľky je vložení záznam pri každom novom bloku. V prípade bloku z inicializačnej fázy importovania, sa do tejto tabuľky vloží číslo súboru, pozícia v súbore, z ktorej je posledný blok a číslo bloku. Toto číslo nie je ekvivalentom výšky bloku (kvôli nezoradeniu blokov v súboroch), ale predstavuje informačnú hodnotu o počte vložených blokov. Počas vkladania nových blokov vo fáze udržiavania aktuálneho stavu, je do stavovej tabuľky vložená výška posledného bloku, bez lokalizačných informácií o poslednom súbore. Pri opätovnom spustení aplikácie je týmto spôsobom jednoduché odlíšiť v ktorom z dvoch možných stavov sa aplikácia nachádzala a získať poslednú výšku bloku, prípadne pozíciu súboru, v ktorom importovanie skončilo.

4.4 Získavanie aktuálnych blokov z blockchainu

Po úspešnom zrealizovaní prvotného importovania blockchainu, trieda `DbImporter` zabezpečuje automatickú správu nových blokov. V prípade, že aplikácia už mala prvotné importovanie zvládnuté a pri aktualizovaní bola užívateľom vypnutá, najskôr prechádza synchronizačným procesom. V tomto procese sa databáza zosynchronizuje s aktuálnym blockchainom dostupným na lokálnom uzle Bitcoin Core.

V stave aktuálnej verzie databázy bola prvotne aplikácia navrhnutá tak, že systematicky, každých 10 minút žiadala pomocou RPC požiadavkov lokálny Bitcoin Core server o najnovší dostupný blok. Časová jednotka aktualizovania bola zvolená podľa podsekcie 2.4.2, kde uvedená priemerná časová jednotka, počas ktorej je vyťažený nový blok, je práve 10 minút. Je teda možné očakávať, že nový blok bude pridaný do hlavnej vetvy blockchainu práve v tomto časovom intervale. Tento interval však nie je pravidlom, nakoľko sa jedná iba o priemernú hodnotu, kedy sa vyťaží nový blok. Za tento časový interval je možné prijať aj viac blokov ako práve jeden alebo dokonca niekedy žiaden. Túto skutočnosť je možné vidieť na obrázku 4.3, kde boli práve tri bloky vyťažené v skoro rovnaký čas. Z tohto dôvodu je prednastavený časový interval aktualizovania vo výslednej implementácii znížený na 1 minútu, aby koncový užívateľ nemusel čakať na niektorých prípadoch na určité bloky skoro celý interval v nečinnosti. Aj pri takto malom intervale však aplikácia môže prijať viacero validných blokov a práve preto aplikácia pri každom prijatom bloku skontroluje jeho zarade-

nie do blockchainu, ktorý je uložený v databáze. Pri nekonzistencii si vyžiada od lokálneho uzla ďalší blok, na ktorý najvyšší blok odkazoval, pomocou hašu predchádzajúceho bloku. Takýmto spôsobom sa postupuje až pokiaľ v prijatom bloku nebude odkaz na najlepší blok v uloženej databáze, teda blok s najvyššou hodnotou výšky.



Obr. 4.3: Jednotlivé časy a výšky novo vyťažených blokov, dostupné z [15].

4.4.1 Rest API aktualizácie

Svoju úlohu pri importovaní dát zohráva aj Rest API aplikácia. Popri vytváraní odpovedí na užívateľské požiadavky, automaticky aktualizuje niektoré stĺpce tabuliek. Pri vkladaní bloku do databázy počas inicializačného importovania alebo počas následného procesu automatického aktualizovania, sú niektoré stĺpce vkladaných záznamov vynechané. Za zapisovanie týchto údajov zodpovedá Rest API časť aplikácie. V nasledujúcom zozname sú uvedené bunky, ktoré sú aktualizované vyvolaním špecifických požiadaviek.

- Získanie špecifickej transakcie:
  - poplatky danej transakcie
  - hodnoty, ScriptPubKey a adresy vstupov
  - adresy a stav každého výstupu
  - zmenu stavu pre výstupy, ktoré boli v transakcii použité ako vstupy
- Získanie UTXO transakcií alebo bilancie adresy:
  - adresy výstupov
  - zmenu stavu výstupov v prípade, že výstup sa nachádza v niektorej transakcii ako vstup

Vytvorením požiadavky na zobrazenie všetkých transakcií adresy je v implementovanej aplikácii aktualizácia údajov zanedbaná, kvôli nadmernému spomaleniu odpovedí pri adresách s veľkým počtom transakcií. Aktualizácia jednotlivých položiek tabuliek spomalí odpoveď pri prvom vytvorení danej požiadavky. Tým sa však nahrajú do tabuliek chýbajúce údaje a každá ďalšia požiadavka, ktorá s týmito záznamami bude potrebovať pracovať, už tieto údaje nemusí získavať špeciálnymi hľadaniami, nakoľko chýbajúce údaje už budú v daných záznamoch obsiahnuté.

4.5 Riešenie konfliktov

Pri práci s blockchainom Bitcoinu sa skôr či neskôr objavia situácie, kedy sa aplikácia bude musieť vysporiadať s konfliktami, ktoré môžu nastať (viz. podsekcia 2.6). Softvér Bitcoin Core je momentálne implementovaný spôsobom, kde orphan bloky nemôže prijať 3.4. *Stale* bloky sa ale občas v sieti môžu vyskytnúť a je nutné s týmto javom počítať. Tieto bloky sú vyhodnotené ako validné, ale nie sú súčasťou najdlhšej vetvy blockchainu.

Trieda `DbImporter` si z tohto dôvodu uchováva históriu predošlých blokov, ktoré už v databáze uložené sú. Pre zachovanie čo najnižšej pamäťovej náročnosti v pamäti nie sú ukladané celé bloky, ktoré môžu dosahovať stále väčšie a väčšie veľkosti, ale iba ich haš kódy. Podľa odporúčania pri práci s Bitcoin technológiou, má aplikácia haš kódov v pamäti uložených práve 6 (v prípade potreby si užívateľ môže toto číslo ukladaných hašov zvýšiť). Uchovávanie dlhšej histórie uložených blokov je aktuálne nepraktické. Uzol Bitcoin Core totiž svojím prístupom efektívne filtruje bloky, ktoré nemajú šancu patriť do hlavného blockchainu. Stale bloky je možné prijať iba v čase, keď je uzol aktívny a príjme dva bloky v takmer rovnaký čas. Tieto bloky sa však prijímajú na jednom uzle s veľmi nízkou pravdepodobnosťou. Na mnou testovanom uzle, ktorý bol minimálne mesiac v aktualizovanom stave, bol počet prijatých týchto blokov rovný nule.

Napriek tomu, je pri každom prijatí nového bloku nutné skontrolovať jeho odkaz na predchádzajúci haš bloku. V prípade, že daný odkaz neodkazuje na posledný blok v zozname uložených blokov, je nutné celý blok z databázy ako aj všetky transakcie, ktoré v ňom boli vykonané odstrániť. Tým pádom je potrebné vrátiť výstupy týchto transakcií späť do nepoužitého stavu. Týmito operáciami sa zaoberá trieda `StaleBlockHandler`. Táto trieda je zodpovedná aj za skontrolovanie celej databázy po inicializačnom procese a v prípade, že objaví blok, ktorý je označený ako stale, údaje daného bloku sa odstránia.

Pri prvotnom importovaní do databázy je potrebné klásť vysoký dôraz na rýchlosť importovania všetkých dostupných dát na server. Nakoľko jednotlivé súbory nedodržujú postupnosť blokov akú majú v blockchaine, kontrola stale blokov sa vykonáva až po dokončení importovania. Pri vkladaní do databázy sa každý blok vloží s negatívnou výškou bloku. Následným aktualizovaním ich výšky sa vyselektujú bloky s negatívnou hodnotou výšky, ktoré už ďalej nepatria do hlavnej vetvy blockchainu. Pri odstraňovaní týchto blokov ako aj všetkých transakcií, je potrebné dávať pozor na to, aby neboli odstránené aj údaje, ktoré aktuálne validné sú. Môže totiž nastať situácia, kde pri prechádzaní súborov sa do databázy zapíše blok, ktorý bude neskôr označený ako stale. Následným prechádzaním ostatných súborov sa však mohli objaviť úplne rovnaké transakcie v bloku, ktorý už je aktívnym blokom. Z tohto dôvodu, je do tabuľky transakcií pridaný ako clustering key aj haš bloku. Týmto prídavným kľúčom v tabuľke dokážeme odstrániť transakcie a teda aj vstupy a výstupy iba pre transakcie, ktoré naozaj patria do odstraňovaného bloku.

4.6 Dekódovanie adries

Prvotným plánom pri implementovaní tabuliek pre adresy, bolo použiť adresy ako súčasť primárneho kľúča, pomocou ktorého sa bude môcť medzi adresami vyhľadávať. Pri evolúcii Bitcoinu sa však zvyšoval aj počet transakcií v blokoch a priamo úmerne aj počet výstupov transakcií. Vo väčšine novších blokoch sa už môže vyskytovať v jednom bloku približne 8 až 10 tisíc výstupov. To môže v najhoršom prípade znamenať aj približne rovnaký počet adries na jeden blok. Súbory s blokmi na disku ukladajú adresy iba vo formáte ScriptPub kľúčov, z ktorých je pri syntaktickej analýze nutné zistiť typ transakcie, následne z nej získať

verejný kľúč a v poslednom kroku ešte tento verejný kľúč zakódovať na požadovanú adresu. Z dôvodu zrýchlenia prvotného importovania sú v databáze adresy ukladané podobne ako v prípade fyzických súborov a to pomocou ScriptPubKey hodnôt. Jednotlivé adresy zadané užívateľom môžu byť kódované štandardizovaným kódovaním v sieti Bitcoin `Base58` a `Bech32` (kapitola 2).

Jednotlivé adresy sú rozlíšené prefixom, ktorý môže mať hodnoty *1, 3* alebo v prípade `Bech32` adresy *bc1*, znázornené v tabuľke 2.1. Každý typ transakcie má štandardizovanú syntax, podľa toho o akú transakciu sa jedná. Po zistení typu adresy, je ďalej skontrolovaná, či spĺňa požadované štandardné pravidlá. Adresy sú dekódované na verejný kľúč, z ktorého sa následne podľa typu transakcie vytvára ScriptPubKey. Pri adresách typu `Bech32`, kde P2WPKH a P2WSH adresy majú rovnaký prefix a ich dĺžka sa môže meniť, je typ transakcie získaný až po dekódovaní adresy na verejný kľúč. Dĺžka tohto kľúča je už pri oboch transakciách štandardizovaná (P2WPKH - 20 bajtov, P2WSH - 32 bajtov) a definuje presný typ adresy. Všetky tieto dekódovania sú realizované pomocou enkóderov implementovaných v knižnici NBitcoin. Pri importovaní výstupov sa pi adresách zisťuje, či sa nejedná o nepoužiteľný výstup, teda o transakciu typu OP RETURN. Nakoľko sú tieto výstupy automaticky vyhodnotené ako nepoužiteľné, nie je nutné ich ukladať do databázy pre požiadavky pomocou adries, a je možné čas importovanie znížiť ich vynechaním [10].

Kapitola 5

Analýza a testovanie

V tejto kapitole sa nachádza analýza dosiahnutých výsledkov. V jednotlivých sekciách sú demonštrované rôzne štatistiky, ktoré boli pri implementácii dosiahnuté. Následne je vyhodnotená správnosť dosiahnutých údajov a porovnanie implementovanej aplikácie s aktuálne existujúcimi nástrojmi.

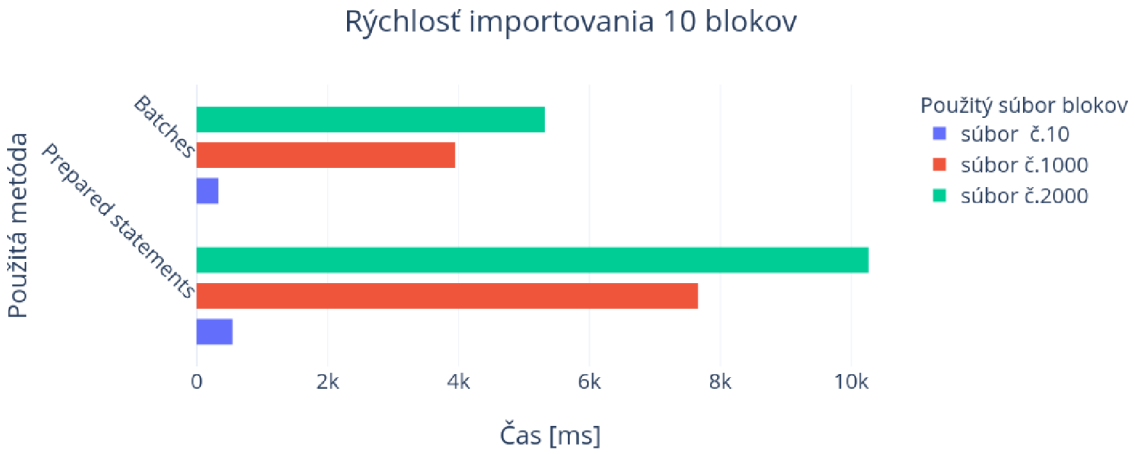
5.1 Výkonnostné testy importovania

Pre výkonnostné testy v tejto sekcii bol použitý nasledovný hardvér:

Komponent	Popis
Procesor	Intel Core i5-8250U 1.6 GHz x4
Pamäť	8 GB
Disk	1 TB HDD
OS	Windows 10

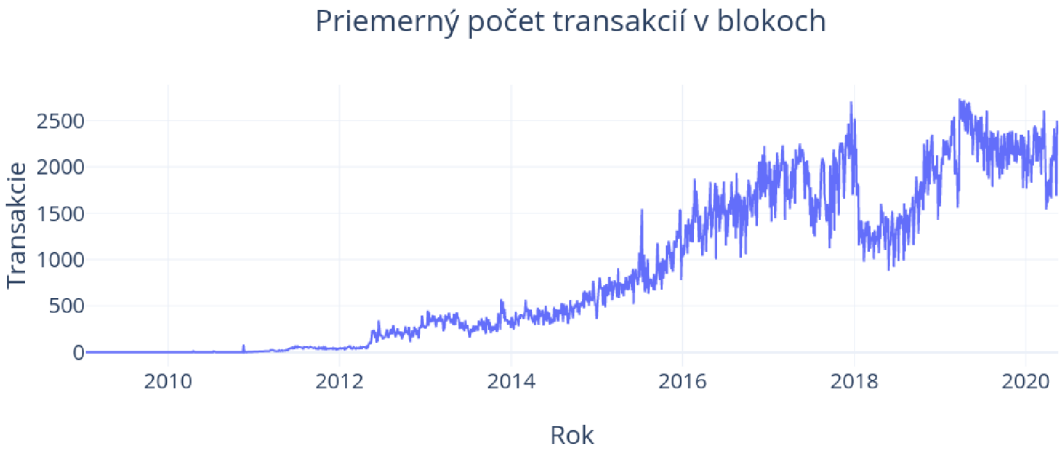
Tabuľka 5.1: Hardvér použitý pri meraní čiastkových výkonností aplikácie.

Ako už bolo spomenuté v predchádzajúcej kapitole, pri importovaní je použitá stratégia importovania pomocou várok príkazov, tzv. batchov. Porovnanie tohto riešenia s odporúčaným, teda pomocou jednotlivo zasielaných asynchrónnych príkazov, je zobrazené na obrázku 5.1. Pri tomto porovnaní, kde sa testovalo importovanie 10 blokov z rôznych súborov, je zjavné, že použitím metódy zasielania várok príkazov na jednom uzle Cassandra sa dá importovanie značne zrýchliť. Veľkosť jednej várky v tomto teste je 10 000. Tino limit je vo výslednej aplikácii nastaviteľný pre koncového užívateľa, avšak pri nesprávne zvolenej veľkosti sa server dokáže vyťažiť a zvýšenie limitu bude mať nežiadúci účinok. Pri asynchrónnych operáciách bol použitý limit 1 milión aktuálne vykonávaných operácií.



Obr. 5.1: Porovnanie rýchlostí importovania 10 blokov blockchainu.

Už na prvý pohľade je jasne viditeľná odlišnosť aj pri rovnakých metódach importovania dát rôznych súborov. Táto skutočnosť je daná neustálim zväčšovaním veľkosti blockchainu počtom transakcií, ktoré sa v bloku nachádzajú a zvyšujú náročnosť analýzy daného bloku. Jav zvyšovania počtu transakcií v blokoch je možné vidieť na obrázku 5.2.



Obr. 5.2: Priemerný počet transakcií uložených v blokoch, štatistika dostupná z [14].

5.2 Importovanie Blockchainu

Prvotné importovanie blockchainu dát s aktuálnou výškou blokov 631,035 trvalo pri použití hardvéru z tabuľky 5.2 66 hodín a 30 minút. Následná aktualizácia výšky blokov trvala približne 6 minút. Výsledky dosiahnuté pri importovaní vykazovali zaujímavú štatistiku. Importovanie ≈500,000 blokov dokázala aplikácia zvládnuť za približne 30 hodín, čo je porovnateľná časová hodnota importovania v bakalárskej práci D.Sommer, 2019 [26]. V tejto práci bolo použité exportovanie dát do csv súborov, nasledované úpravou týchto súborov a importovaním do databázy. Čas potrebný na vytvorenie csv súborov pri 508,000 blokoch bol 22 hodín a 58 minút. Pri práci s vtedy aktuálnou výškou blokov, teda 564,700 sa však

tento čas zvýšil na 30 hodín. Pri zohľadnení rozdielneho hardvéru (použitie SSD disku narozdiel HDD disku s rozdielnymi maximálnymi rýchlosťami), potreby ďalej upraviť vytvorené súbory a importovať ich do databázy, ukazujú obe riešenia podobnú charakteristiku zvyšovania časovej náročnosti pri novších blokoch.

Komponent	Popis
Procesor	Intel Xeon (Skylake) x8
Pamäť	30 GB
Disk	3 TB HDD
OS	Ubuntu 18.04 LTS

Tabuľka 5.2: Hardvér použitý pri synchronizácii celého blockchainu dát.

Výsledná veľkosť celej databázy je 938 GiB. Veľkosti jednotlivých tabuliek sú dostupné v tabuľke 5.3. Podľa očakávania najväčšiu kapacitu zaberajú tabuľky so vstupmi a výstupmi transakcií.

Tabuľka	Veľkosť
block_by_hash	33 GiB
block_by_height	65.86 GiB
hash_by_height	89.67 MiB
input_by_hash	147.68 GiB
input_by_previous	268.37 GiB
output_by_address	180.71 GiB
output_by_hash	144.05 GiB
transaction_by_hash	98.25 GiB

Tabuľka 5.3: Veľkosti jednotlivých tabuliek na serveri Cassandra.

5.2.1 Validita údajov

Účelom tejto podsekcie je zhodnotiť validitu údajov prijatých pomocou Rest API požiadaviek z implementovanej aplikácie. Spôsob testovania je porovnávaním výstupu aplikácie s hodnotami, ktoré sú uvedené v dostupných blockchain prehliadačoch.

Pri kontrolovaní validity dát v databáze, je možné okrem ponúkaných nátrojov opísaných v sekcii 5.3 použiť aj rôzne bežné webové prehliadače siete Bitcoin. Medzi najviac využívané, ktoré boli použité aj pri vyhodnocovaní validity, patria Blockchain Explorer[1] a Blockchair[2]. Jednotlivé výstupy, ktoré budú v tejto sekcii uvedené, sú zámerne zmenšené o niektoré údaje, nakoľko výstupy môžu obsahovať až príliš veľa hodnôt.

Prvou testovanou požiadavkou je požiadavka na získanie bloku, ktorá je realizovaná pomocou nasledujúceho príkazu:

```
curl https://localhost:5001/block/610000
```

Korektnosť výstupu tejto požiadavky je možné skontrolovať pomocou prehliadača blokov zadaním výšky tohto bloku alebo jeho hašu. Samotný výstup je v skrátenej forme s jednou ukážkou transakcie vo výpise 5.1.

[1] https://www.blockchain.com/explorer
[2] https://blockchair.com/

```
{
    "blockHash": "0000000000000000000
    a6f607f74db48dae0a94022c10354536394c17672b7f7",
    "previousBlockHash": "00000000000000000005
    d5d8ee9235d74f38445289f3420de6250eeffcff028f",
    "height": 610000,
    "merkleRoot": "2
    f9bbebc71d77f823042fc7d154bf902e024104d017d83f72bd78232c32ba221",
    "nonce": 4129488977,
    "bits": 387300560,
    "version": 536928256,
    "txCount": 1916,
    "time": "2019-12-27T11:53:47+00:00",
    "transactions": [
        "bdb73a95213c7fe3db646d22a67a7acb9aab33dd071cbb58c6ebc2e909683399",
        ...
        ]
}
```

Výpis 5.1: Výstup pri hľadaní bloku

Nasledujúcim príkladom je získanie transakcie, pomocou jej hašu. Použitý príkaz je nasledovný :

```
curl https://localhost:5001/transaction/
bdb73a95213c7fe3db646d22a67a7acb9aab33dd071cbb58c6ebc2e909683399
```

Výpis 5.2 ukazuje transakciu, ktorá je definovaná ako coinbase, čo je možné vidieť podľa prvého vstupu transakcie. Pre jednoduchosť výpisu je uvedený iba jeden výstup. Transakciu je možné porovnať aj s výstupmi v ľubovoľnom spomenutom prehliadači blokov po zadaní hašu transakcie
`bdb73a95213c7fe3db646d22a67a7acb9aab33dd071cbb58c6ebc2e909683399`.

```
{
    "transaction": {
        "transactionHash": "
    bdb73a95213c7fe3db646d22a67a7acb9aab33dd071cbb58c6ebc2e909683399",
        "value": 12.51969494,
        "version": 1,
        "fees": 0,
        "blockHash": "0000000000000000000
    a6f607f74db48dae0a94022c10354536394c17672b7f7"
    },
    "inputs": [
        {
            "inputId": 0,
            "previousTransactionHash":
    "0000000000000000000000000000000000000000000000000000000000000000",
            "previousOutputId": -1,
            "value": 0,
```

```
            "address": null,
            "scriptPubKey": null
        }
    ],
    "outputs": [
        {
            "outputId": 0,
            "address": "1MUz4VMYui5qY1mxUiG8BQ1Luv6tqkvaiL",
            "scriptPubKey": "OP_DUP OP_HASH160
    e0ad60c897901128662623c500a4a6079e99cd3e OP_EQUALVERIFY OP_CHECKSIG",
            "value": 12.51969494,
            "spent": true
        }
    ]
}
```

Výpis 5.2: Výstup pri hľadaní transakcie

Porovnaním týchto výstupov je možné konštatovať správnosť údajov vložených do databázy implementovanej aplikácie. Všetky výstupy podporované v tejto práci sú kvôli ich množstvu uvedené prílohe B.

5.3 Porovnanie s ostatnými nástrojmi

Medzi dve nástroje, ktoré podporujú podobnú funkcionalitu ako táto práca patria *Insight-API* a *Blockbook*. Obe nástroje podporujú analýzu a správu blockchain siete. Nakoľko cieľom práce je vytvorenie analyzačného nástroja, funkcie oboch nástrojov, ktoré povoľujú napríklad vytváranie transakcií nie sú v tejto sekcii brané do úvahy.

5.3.1 Insight API

Insight API[3] je nástroj vyvinutý spoločnosťou BitPay, ktorý je implementovaný v jazyku JavaScript použitím prostredia NodeJS. Nachádza sa v sade nástrojov Bitcore Node, ktorého činnosť riadi samotnú synchronizáciu ukladania a aktualizovania údajov blockchainu[4]. Pre svoj beh potrebuje mať jeden bežiaci Bitcoin Core uzol s ktorým dokáže komunikovať pomocou RPC alebo fyzicky uložených blokov. Projekt je voľne dostupný aj pomocou užívateľského rozhrania[5].

Databázovým systémom použitým na ukladanie dát v projekte od firmy BitPay je LevelDB, vyvíjaný spoločnosťou Google. LevelDB je ďalším typom NoSql databázového systému typu key-value (slovnesky - kľúč-hodnota), čo znamená, že databáza si ukladá údaje ako kolekcie párov týchto hodnôt, kde kľúč je jedinečným identifikátorom a hodnotu tvoria všetky ostatné údaje. Ako hodnota môžu figurovať napríklad všetky ostatné údaje obsiahnuté v blokoch. Tento typ databázy je podobne ako v mnou implementovanom riešení zvolený tak, aby bol schopný vykonávať rýchle hľadania bez rôznych zložitejších požiadavkách, ktoré je možné implementovať iba v SQL databázach.

[3] https://github.com/bitpay/insight-api
[4] https://github.com/bitpay/bitcore-node
[5] https://insight.bitpay.com/

Insight API dokáže reagovať na podobné požiadavky ako je to v prípade tejto práce. Všetky požiadavky sú rozdelené do troch hlavných kategórií, nakoľko dokážu prehľadávať údaje blokov, transakcií a adries. Nástroj dokáže vyhľadávať bloky na základe hašu bloku alebo jeho výšky. Podobne ponúka prehľadávanie transakcií a adries. Insight API však ponúka aj rozširujúce agregačné metódy, ktoré dokážu filtrovať bloky alebo transakcie iba v udanom časovom období.

5.3.2 Blockbook

Blockbook je back-end služba pre Trezor Wallet[6]. Pod pojmom Trezor Wallet sa rozumie rozhranie prehliadača, pre Trezor[7]. Blockbook implementuje prácu s viacerými typmi kryptomien. Pre túto analýzu sa však bude brať ohľad iba na prácu so sieťou Bitcoin.

Databázový systém použitý pri tomto nástroji je RocksDB. Rovnako ide o NoSQL key-value databázu, podobne ako pri nástroji Insight-API. Blockbook implementuje ukladanie dát do rôznych rodín stĺpcov (anglicky - column families). Jednotlivé rodiny má vytvorené napríklad pre výšku bloku, vďaka ktorej je možné mapovať výsledný haš bloku a jeho ďalšie údaje. Blockbook podporuje väčšinu implementovaných API požiadaviek ako táto práca. Na rozdiel od tejto práce dokáže blockbook pracovať s HD peňaženkami a teda rozšírenými verejnými kľúčmi adries tretej úrovne, pre ktoré derivovaním dokáže vygenerovať všetky použité adresy v sieti.

5.3.3 Porovnanie výkonnostných štatistík

Pri testovaní boli použité API požiadavky, ktoré sú implementované pri všetkých nástrojoch. Kvôli zachovaniu rovnakých podmienok pre každú testovanú aplikáciu nebola implementovaná aplikácia používaná lokálne ale cez externý server, nakoľko ani jeden z nástrojov nebol spúšťaný lokálne. Implementovaná aplikácia, ako aj všetky ostatné nástroje, mali pri nameraných hodnotách prístup k úplnej histórii údaj, čiže boli plne synchronizované. Každá vykonaná požiadavka bola realizovaná na rovnaký údaj, napríklad pri požiadavke na blok sa pri všetkých nástrojoch použil ten istý haš bloku. Pri testovaní boli pri každej požiadavke vykonané testy s 5 rôznymi argumentmi. Jednotlivé výsledky testov predstavujú priemerné hodnoty dosiahnuté pri uvedených požiadavkoch. Výsledky testov sú dostupné v tabuľke 5.4. Pri výsledkoch pomocou nástroja Blockbook je zdieľaný časový údaj bilancií a transakcií pre adresy z dôvodu, že Blockbook nepodporuje tieto dve požiadavky samostatne a výsledok je teda prijatý pri jednej požiadavke.

[6] https://github.com/trezor/blockbook

[7] Trezor - typ hardvérovej peňaženky, ktorá ponúka zvýšenú bezpečnosť pre prácu s rôznymi typmi kryptomien

Použitá požiadavka	Implementovaná aplikácia	Insight-API	Blockbook
Haš bloku	177 ms	592 ms	128 ms
Blok pomocou hašu	406 ms	831 ms	455 ms
Transakcia	501 ms	519 ms	216 ms
UTXO pre adresu	386 ms	518 ms	141 ms
Transakcie pre adresu	199 ms	899 ms	242 ms
Bilancia adresy	197 ms	492 ms	

Tabuľka 5.4: Porovnanie priemerných štatistík na čas odozvy jednotlivých požiadaviek, kde sú údaje uvedené pri implementovanej aplikácii vždy v prípade už aktualizovaných chýbajúcich dát.

Z dosiahnutých výsledkov je jasne viditeľná účinnosť aplikácie Blockbook. V skoro každom teste tento nástroj dosahoval najlepšie výsledky. Za zmienku tiež stojí, že tento nástroj pri každom výstupe preukazoval najväčšie množstvo údajov obsiahnutých vo výstupoch.

5.4 Vyhodnotenie

Implementovaná aplikácia musela byť navrhnutá tak, aby mohol byť vykonaný určitý kompromis medzi dĺžkou importovania všetkých historických dát siete a dĺžkou odozvy pri jednotlivých požiadavkách užívateľa. Z tohto dôvodu musela byť databázová schéma oproti prvotným plánom zmenená na aktuálny stav s tým, že niektoré položky budú vypočítavané a vyhľadávané až pri ich použití pomocou Rest API aplikácie. Táto myšlienka a realizácia aplikácie vykazuje konkurencie schopné výsledky pri všetkých požiadavkách ohľadom blokov ako aj transakciách. Čas odozvy je mierne dlhší pri prvých vyhľadávaniach danej transakcie ako dôsledok aktualizovania niektorých polí záznamov bližšie opísaných v sekcii 4.4.1. Pri ďalších hľadaniach je však už daná transakcia užívateľovi prístupná v oveľa kratšom čase. V prípade prvého hľadania bola transakcia v prípadoch veľkého výskytu vstupov a výstupov nameraná hodnota približne 2-3 sekundy. Po aktualizácii potrebných údajov však hodnota hľadania klesla na priemernú hodnotu 501 ms, ktorá je uvedená v tabuľke 5.4. Takýto prístup je možné považovať za efektívny.

Negatívne výsledky však mala aplikácia pri práci s užívateľskými adresami. Jednotlivé adresy, ktoré sú použité pri priemernom množstve transakcií dokazovali v každom teste podobné výsledky ako pri práci s transakciami alebo ako konkurenčné nástroje pri rovnakých požiadavkách. Lenže prácou s adresami, ktoré sa nachádzajú v niekoľkých tisícoch až v desaťtisícoch transakciách, je nutné prehľadávať všetky výstupy alebo aj vstupy použité pri danej adrese. Toto hľadanie je v prípadoch takejto kvantity adries časovo náročné a značne ovplyvňuje odozvu aplikácie. Hľadanie všetkých transakcií takejto adresy, aplikácii zaberie v niektorých prípadoch aj viacero sekúnd až minút, čo je značný rozdiel oproti priemerným hodnotám dosiahnutým v testoch alebo pri konkurenčných nástrojoch. V krajných prípadoch, kde je adresa súčasťou stoviek tisíc adries dokonca aplikácia požiadavku nebola schopná vykonať pri prednastavenom počte asynchrónnych operácií, kvôli vyťaženiu uzla. V tomto prípade je nutné užívateľom znížiť maximálny počet vykonávaných operácií, čo však zapríčiní spomalenie získavania výsledkov. Riešením, ktoré by mohlo túto situáciu vyriešiť je použitie viacerých uzlov serveru Cassandra za účelom zníženia záťaže, ktorá je vytváraná pri použití iba jedného uzla v testoch tejto práce. Databázový systém Cassandra je navrhnutý hlavne pre rýchle vykonávanie požiadaviek pomocou rozloženia práce na rôzne uzly v sieti, čo bolo aj touto prácou dokázané. Implementácia databázovej schémy v tejto práci

má teda negatívny vplyv pri práci s adresami, ktorých počet výstupov a vstupov dosahuje príliš vysoké hodnoty a je použitý iba jeden uzol serveru.

Kapitola 6

Záver

Cieľom tejto práce bolo implementovanie aplikácie schopnej odpovedať na Rest API požiadavky užívateľa pri práci s historickými a aktuálnymi dátami v sieti Bitcoin.

Výsledkom tejto práce sú dve funkčné samostatné aplikácie, kde jedna aplikácia spravuje synchronizáciu lokálnej databázy so sieťou a druhá reaguje na požiadavky užívateľa. Aplikácie podporujú aktuálne platné štandardy v sieti Bitcoin a všetky súčasti výsledných aplikácií sú implementované podľa zákonitostí a pravidiel pri práci so sieťou Bitcoin.

Práca pre mňa predstavuje prínos v oblasti funkcionality detailov systému siete Bitcoin, nakoľko pre tvorbu práce sa vyžadovalo značné množstvo teoretických vedomostí na priblíženie spôsobu, akým by mali byť jednotlivé operácie implementované. Ďalším prínosom bolo zoznámenie sa s NoSQL databázovými systémami. Prácou s týmto typom databázového systému som si uvedomil rozdiely pri procese návrhu databázovej schémy a rozdielny spôsob prístupu k jednotlivým položkám oproti tradičným relačným databázovým systémom, s ktorými som mal doteraz skúsenosti.

Výsledná aplikácia má veľmi široké spektrum možných vylepšení, ktoré môžu byť veľmi užitočnými. Pri implementovaní tejto práce bol pri návrhu riešenia kladený vysoký dôraz na rýchlu inicializačnú synchronizáciu. Po výslednom testovaní však bolo zistené, že implementovaný dizajn databázy by mohol byť upravený viacerými tabuľkami pre prácu s užívateľskými adresami. V tomto projekte je teda ďalej možné skúmať rôzne návrhy databáz, ktoré by mohli byť efektívnejšie ako dosiahnuté výsledky v tejto práci. Zaujímavým skúmaním do budúcna by mohlo byť aj použitie výkonnejšieho hardvéru pri aplikácii ako aj pri použitom servery, kde tieto vlastnosti môžu značne prispieť k efektivite riešenia. Ďalším navrhovaným vylepšením by mohlo byť vytvorenie grafického užívateľského rozhrania, ktoré by dokázalo konkurovať aktuálne funkčným webovým prehliadačom siete Bitcoin.

Literatúra

[1] Anderson, R., Larkin, K. a Wasson, M. *Tutorial: Create a web API with ASP.NET Core*. 2020 [cit. 2020-5-21]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-3.1&tabs=visual-studio.

[2] Antonopoulos, A. M. *Mastering Bitcoin*. 1. vyd. O'Reilly Media, Inc., 2014. ISBN 9781491902639.

[3] Asolo, B. *Coinbase Transaction Explained* [online]. Október 2018 [cit. 2019-12-25]. Dostupné z: https://www.mycryptopedia.com/coinbase-transaction-explained/.

[4] Asolo, B. *Block Height Explained* [online]. Október 2018 [cit. 2019-12-25]. Dostupné z: https://www.mycryptopedia.com/block-height-explained/.

[5] Bitcoin. *Bitcoin Developer Reference* [online]. 2019 [cit. 2019-12-24]. Dostupné z: https://bitcoin.org/en/developer-reference.

[6] Bitcoin. *Blockchain* [online]. 2019 [cit. 2019-12-24]. Dostupné z: https://bitcoin.org/en/blockchain-guide.

[7] Bitcoin. *P2P Network* [online]. 2019 [cit. 2019-12-29]. Dostupné z: https://bitcoin.org/en/p2p-network-guide.

[8] Bitcoin. *Transactions* [online]. 2019 [cit. 2019-12-25]. Dostupné z: https://bitcoin.org/en/transactions-guide.

[9] Bitcoin.it. *Bech32*. 2018 [cit. 2020-1-14]. Dostupné z: https://en.bitcoin.it/wiki/Bech32.

[10] Bitcoin.it. *OP_RETURN*. 2020 [cit. 2020-5-20]. Dostupné z: https://en.bitcoin.it/wiki/OP_RETURN.

[11] BitcoinMining.com. *What is Bitcoin Mining Difficulty* [online]. Jún 2015 [cit. 2019-12-25]. Dostupné z: https://www.bitcoinmining.com/what-is-bitcoin-mining-difficulty/.

[12] BitcoinWiki. *Pay-to-Script Hash*. 2018 [cit. 2020-1-14]. Dostupné z: https://en.bitcoinwiki.org/wiki/Pay-to-Script_Hash.

[13] BitcoinWiki. *Pay-to-Pubkey Hash*. 2020 [cit. 2020-2-18]. Dostupné z: https://en.bitcoinwiki.org/wiki/Pay-to-Pubkey_Hash.

[14] BLOCKCHAIN. *Blockchain Charts* [online]. 2020 [cit. 2020-5-20]. Dostupné z: https://www.blockchain.com/charts.

[15] BLOCKCHAIR. *Universal blockchain explorer and search engine.* 2020 [cit. 2020-05-22]. Dostupné z: https://blockchair.com/.

[16] BORSOS, D. *Everything you need to know about Cassandra Materialized Views* [online]. Február 2017 [cit. 2019-2-19]. Dostupné z: https://opencredo.com/blogs/everything-need-know-cassandra-materialized-views/.

[17] DOVBNYA, A. *BTC vs. XBT: What's the Difference Between Bitcoin Symbols?* [online]. U.Today, november 2018 [cit. 2019-1-18]. Dostupné z: https://u.today/guides/blockchain/btc-vs-xbt-whats-the-difference-between-bitcoin-symbols.

[18] FRANKENFIELD, J. *Bitcoin* [online]. Investopedia, október 2019 [cit. 2019-12-24]. Dostupné z: https://www.investopedia.com/terms/b/bitcoin.asp.

[19] FRANKENFIELD, J. *Block (Bitcoin Block)* [online]. Investopedia, august 2019 [cit. 2019-12-27]. Dostupné z: https://www.investopedia.com/terms/b/block-bitcoin-block.asp.

[20] ISO. *ISO 4217:2015 Codes for the representation of currencies.* standard 8. International Organization for Standardization, 2015.

[21] JNUVRENI. *The Bitcoin Network* [online]. August 2019 [cit. 2019-12-28]. Dostupné z: https://sheinix.com/the-bitcoin-network/.

[22] LASTOVETSKA, A. *Blockchain Architecture Basics: Components, Structure, Benefits and Creation* [online]. Január 2019 [cit. 2019-12-28]. Dostupné z: https://mlsdev.com/blog/156-how-to-build-your-own-blockchain-architecture.

[23] MORROW, J. *What is a Coinbase Transaction?* [online]. Október 2014 [cit. 2019-12-25]. Dostupné z: https://blog.cex.io/bitcoin-dictionary/coinbase-transaction-12088.

[24] NAKAMOTO, S. *Bitcoin: A Peer-to-Peer Electronic Cash System* [online]. Október 2008 [cit. 2019-12-20]. Dostupné z: https://bitcoin.org/bitcoin.pdf.

[25] RAY, S. *Merkle Trees* [online]. December 2017 [cit. 2019-12-27]. Dostupné z: https://hackernoon.com/merkle-trees-181cb4bc30b4.

[26] SOMMER, D. *Processing Bitcoin Blockchain Data using a Big Data-specific Framework.* Zürich, Switzerland, 2019. [cit. 2020-5-24]. Bakalárska práca. University of Zurich, Department of Informatics (IFI). Dostupné z: https://files.ifi.uzh.ch/CSG/staff/scheid/extern/theses/BA-D-Sommer.pdf.

[27] VOIGT, K. *What Is Bitcoin, and How Does It Work?* [online]. nerdwallet, jún 2019 [cit. 2019-12-24]. Dostupné z: https://www.nerdwallet.com/blog/investing/what-is-bitcoin/.

[28] VRABCOVA, Z. *Čo je ťažba kryptomien a ako ťažiť Bitcoin?* [online]. Marec 2019 [cit. 2019-12-25]. Dostupné z: https://kriptomat.io/sk/blockchain/co-je-tazba-kryptomien-a-ako-tazit-bitcoin/.

[29] Walker, G. *Difficulty, A mechanism for regulating the time it takes to mine a block.* [online]. Marec 2015 [cit. 2019-12-24]. Dostupné z: https://learnmeabitcoin.com/beginners/difficulty.

[30] Walker, G. *Technical Guide, An explanation of each part of bitcoin.* [online]. 2019. 2019-12-23 [cit. 2019-12-24]. Dostupné z: https://learnmeabitcoin.com/guide/.

Príloha A

Obsah CD

Priložené CD obsahuje nasledujúce súbory:

- src/ - zdrojové súbory implementovaných aplikácií
  - README.md - užívateľský manuál
  - LICENSE.txt - licencie
  - App.config - konfiguračný súbor projektu
  - Benchmarks/ - výkonnostné testy importovania
  - DbImporter/ - implementovaná aplikácia pre importovanie údajov
  - ExplorerApi/ - aplikácia podporujúca Rest API požiadavky
- doc/ - zdrojové súbory tejto práce
- xzigot00.pdf - PDF súbor tejto práce

Príloha B

Príklady získaných údajov

B.1 Získanie informácií o bloku

Získanie informácií o bloku(v ukážke sú záznamy transakcií zámerne skrátené z dôvodu zachovania jednoduchosti výstupu v prílohe) je možné získať nasledovnými príkazmi:

```
curl https://localhost:5001/block/601321
curl https://localhost:5001/block/000000000000000000094e2205cc2dcd657
8154ac9efabeeff4230d4bfc88de1
```

Zobrazený výstup aplikáciou:

```
{
    "blockHash": "000000000000000000094
   e2205cc2dcd6578154ac9efabeeff4230d4bfc88de1",
    "previousBlockHash": "000000000000000000003
   ccb83a5f42fe4b47ae031dc0082f1fe72c398f7337ca",
    "height": 601321,
    "merkleRoot": "83
   aedac951e5bf72fdfb8d69704b294320384ef7ecb926a4716a83c2cba74898",
    "nonce": 2954788617,
    "bits": 387223263,
    "version": 545259520,
    "txCount": 2614,
    "time": "2019-10-28T03:05:19+00:00",
    "transactions": [
        "035f392bf6d97e39d3df1dca3256e6d45500701be56a1a1669f13151bf79885a",
        "bd2012e1bc6fc589fff5e4ff57669b04d9b9afe2e7b3c41ff159224096c4467e",
        "19d509390d00117373dbcdb31f959ebaa5af8dd074ecf0e08c8f5fbf056e9749",
        ...]
}
```

Výpis B.1: Výstup pri hľadaní bloku

```
curl https://localhost:5001/blockhash/601321
```

Zobrazený výstup aplikáciou:

```
{
    "height": 601321,
    "blockHash": "00000000000000000094
   e2205cc2dcd6578154ac9efabeeff4230d4bfc88de1"
}
```

Výpis B.2: Výstup pri hľadaní hašu bloku

B.2 Získanie transakcie

Druhým príkladom sú informácie o špecifikovanej transakcii, vyhľadanej týmto príkazom:

```
curl https://localhost:5001/transaction/2e4f5d7e7f5c23b2e4ab7c2793e94ce
c5020dc0b3896e3b5685b967567d4a255
```

Zobrazený výstup aplikáciou:

```
{
    "transaction": {
        "transactionHash": "2
   e4f5d7e7f5c23b2e4ab7c2793e94cec5020dc0b3896e3b5685b967567d4a255",
        "value": 0.10679111,
        "version": 2,
        "fees": 0.00104520,
        "blockHash": "00000000000000000188252
   ee9277e8f60482a91b7f3cc9a4a7fb75ded482a8"
    },
    "inputs": [
        {
            "inputId": 0,
            "previousTransactionHash": "4655
   a367ab353cf5792b227000fd715b238ee8ec52691f95804095bd2a99c0de",
            "previousOutputId": 1,
            "value": 0.0059804,
            "address": "12gxGMnUeLNdSieGfneCX78bKsh3scBQZT",
            "scriptPubKey": "OP_DUP OP_HASH160 12867
   b246d8f864458a0e71d6c6d976b60e2c3ea OP_EQUALVERIFY OP_CHECKSIG"
        },
        {
            "inputId": 1,
            "previousTransactionHash": "51
   e0c342e339f63fa660cb6ddc662261231b37f3409d7bc1bac1d86ec039e8a4",
            "previousOutputId": 8,
            "value": 0.10033854,
            "address": "15j7QA2aPpDM9ngmKpFmwujyADZxVa85Zr",
            "scriptPubKey": "OP_DUP OP_HASH160 33
   d750ba4df714e42afea41a65e2d968b4ca0364 OP_EQUALVERIFY OP_CHECKSIG"
        },
```

```
        {
            "inputId": 2,
            "previousTransactionHash": "
    c5809a8b4b0c24eb7b5258b2b0775855348b1b1f13300ad233e02a2056772b8e",
            "previousOutputId": 2019,
            "value": 0.00151737,
            "address": "1PU8gBx6XsgpguxhNPJhD5sAUQxxJmTtE7",
            "scriptPubKey": "OP_DUP OP_HASH160
    f67467d14732a42439e44a0313fab4ab78350eb2 OP_EQUALVERIFY OP_CHECKSIG"
        }
    ],
    "outputs": [
        {
            "outputId": 0,
            "address": "1GSaAi1yS8UjYDNym5FGw9t8bZ1HqV17xL",
            "scriptPubKey": "OP_DUP OP_HASH160
    a95ff8f74f08b0da7b09ed0a6279446468880fdb OP_EQUALVERIFY OP_CHECKSIG",
            "value": 0.02011457,
            "spent": true
        },
        {
            "outputId": 1,
            "address": "33JTp9wkkNdTzV8GcDXEijZ6L6Qq5b66Wo",
            "scriptPubKey": "OP_HASH160 11
    ac4bd40d2206245871 48e3f2625dac2b0da4a3 OP_EQUAL",
            "value": 0.08667654,
            "spent": true
        }
    ]
}
```


Výpis B.3: Výstup pri hľadaní transakcie

B.3 Získanie údajov pre adresy

Nasledujúce ukážky sú ukážky pre výstupy pri práci s adresami.

Zobrazenie UTXO transakcií adresy, realizované pomocou príkazu:

```
curl https://localhost:5001/wallet/utxos/11647cCig8jWPgRMh1ApoojqE9bkrWU1G9
```

Zobrazený výstup:


```
[
    {
        "transactionHash": "6630
    a42da4df9003097b0c3432a6eb11b4051ae6338b9fd8b955f418bc7967bb",
        "outputId": 1,
        "value": 0.0041
    }
```

```
]
```

Výpis B.4: Výstup pri zobrazení UTXO transakcií adresy

Požiadavka na zobrazenie všetkých transakcií adresy, limitovaná v tomto prípade na zobrazenie jedného vstupu a výstupu, pomocu príkazu :

```
curl https://localhost:5001/wallet/transactions/1BFmKCjkGuQVaJ9xTNear2zyG45
r461qtx?outCount=1&inCount=1
```

```
{
    "address": "1BFmKCjkGuQVaJ9xTNear2zyG45r461qtx",
    "format": "P2PKH",
    "page": 1,
    "totalPages": 3,
    "outputs": [
        {
            "transactionHash": "
    cc0ff3a325ac399ed8a4292c316bbc179a1155a8bd6d0db40ef4a9c6af4229ba",
            "outputId": 30
        }
    ],
    "inputs": [
        {
            "inputId": 246,
            "transactionHash": "58
    dd53a771b4ba523869c8e4e8509b51ad457452ea451e6d4e06fb1bf591489e"
        }
    ]
}
```

Výpis B.5: Výstup pri zobrazení všetkých transakcií

Zobrazenie aktuálnej bilancie adresy:

```
curl https://localhost:5001/wallet/balance/1BFmKCjkGuQVaJ9xTNear2zyG45r461q
tx
```

Zobrazený výstup:

```
{
    "address": "1BFmKCjkGuQVaJ9xTNear2zyG45r461qtx",
    "scriptPubKey": "OP_DUP OP_HASH160 707
    c1297c6f7e930005e5baeb9a292ed7ddd8518 OP_EQUALVERIFY OP_CHECKSIG",
    "format": "P2PKH",
    "totalRecieved": 0.06019502,
    "totalSend": 0.06019502,
    "balance": 0.00000000
}
```

Výpis B.6: Výstup bilancie adresy