FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

| | |
|---|---|
| **Název:** | FITCOIN: webový náhled na blockchain |
| **Student:** | Rostislav Kaufman |
| **Vedoucí:** | Mgr. Jan Starý, Ph.D. |
| **Studijní program:** | Informatika |
| **Studijní obor:** | Webové a softwarové inženýrství |
| **Katedra:** | Katedra softwarového inženýrství |
| **Platnost zadání:** | Do konce letního semestru 2018/19 |

## Pokyny pro vypracování

1. Seznamte se s problematikou virtuálních měn obecně a technologie blockchainu zvláště.
2. Nastudujte strukturu blockchainu měny FitCoin, tak jak vzniká v paralelních BP.
3. Vytvořte webovou aplikaci, která umožní podrobný náhled na obsah existujícího FitCoin blockchainu, podobně jako např. blockchain.info umožňuje náhled na blockchain BitCoinu.
- umožněte náhled na jednotlivé bloky,
- umožněte náhled na jednotlivé transakce,
- umožněte náhled na jednotlivé adresy,
- zpřístupněte jednoduché statistiky: objem denního provozu apod,
- aplikace by měla reagovat v reálném (nebo téměř reálném) čase na dění ve FitCoin síti.
4. Aplikaci řádně zdokumentujte z pohledu uživatele i z pohledu programátora.
5. Aplikaci řádně otestujte.

## Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 18. prosince 2017

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

# FITCOIN: webový náhled na blockchain

***Rostislav Kaufman***

Katedra softwarového inženýrství
Vedoucí práce: Mgr. Jan Starý, Ph.D.

15. května 2018

# Poděkování

Děkuji vedoucímu práce Mgr. Janu Starému, Ph.D. za povzbuzení a dohled nad mou prací, dále Ing. Petře Pavlíčkové, Ph.D. za několik cenných rad. Děkuji svým rodičům za tichou podporu a děkuji Karolíně a Minnie za společnost.

# Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 15. května 2018 ……

### Odkaz na tuto práci

Kaufman, Rostislav. *FITCOIN: webový náhled na blockchain.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

# Abstrakt

Práce se zabývá technologií blockchainu, jejím využitím při implementaci kryptoměny a možnostmi takový blockchain za běhu kryptoměny sledovat a procházet. Součástí práce je soupis možných řešení a přístupů použitelných pro zisk dat z blockchainu, zmínky o jejich výhodách a nevýhodách, a nakonec volba přístupu vhodného pro nasazení na kryptoměnu Fitcoin, vznikající souběžně.

Výstupem práce je návrh v tuto chvíli použitelného systému a jeho částečná implementace ve formě dvou aplikací v jazyce Ruby, webové aplikace postavené na frameworku Rails a běžné Ruby aplikace pro komunikaci se sítí Fitcoin.

**Klíčová slova** kryptoměna, Fitcoin, Ruby, Rails, blockchain, náhled, node, Bitcoin

# Abstract

The thesis deals with the blockchain technology, the ways it can be used to implement a cryptocurrency and the possibilities of browsing through the cryptocurrency's blockchain during its operation. It includes a summary of options and approaches that are useful for the acquisition of data from a blockchain, and then a choice of the most suitable approach to be used with the Fitcoin cryptocurrency that is being developed concurrently.

The main output of this thesis is a design of a viable system and its partial implementation in the form of two applications written in Ruby, a web app created using the Rails framework and a plain Ruby app for the communication with the Fitcoin network.

**Keywords** cryptocurrency, Fitcoin, Ruby, Rails, blockchain, explorer, node, Bitcoin

# Obsah

# Seznam obrázků

# Úvod

IT svět zachvátila na přelomu let 2017–2018 horečka kryptoměn. Hodnota Bitcoinu — první a nejznámější kryptoměny — se v průběhu roku 2017 zhruba zdvacetinásobila. Kryptoměny se dostaly na titulní stránky novin, do diskusních pořadů i hlavních televizních zpráv. Každý den se objevují nové a nové deriváty původního konceptu Bitcoinu i zcela nové návrhy využívající odlišné technologie.

Kryptoměnou nazýváme elektronický systém využívající internet a moderní kryptografii pro bezpečné a trvalé uchování záznamů o velikosti držených finančních prostředků. Velmi podobné systémy je možné vytvořit a využívat pro ukládání mnoha druhů informací, např. i katastru nemovitostí, obvykle pak ale nejsou nazývány kryptoměnou. O všech těchto systémech lze ale s jistotou říci, že přinášejí funkcionalitu, kterou lidstvo nikdy v minulosti nemělo k dispozici — spolehlivou, decentralizovanou, perzistentní databázi, která může existovat zcela legálně i být upravena k utajenému použití. Jediným požadavkem pro provoz většiny kryptoměn je dostupné připojení k internetu a alespoň desítky uživatelů.

Tato práce je jednou ze skupiny FITCOIN:

- Blockchain pro FITCOIN
- FITCOIN: peer-to-peer komunikace
- FITCOIN: webový náhled na blockchain
- FITCOIN: peněženka pro Android
- FITCOIN: transakce

Společným cílem těchto prací je vytvořit jednoduchou modelovou kryptoměnu inspirovanou ranými verzemi Bitcoinu, nazvanou pracovně Fitcoin (FITcoin, FITCOIN, fItCOIn, …— budu používat tvar „Fitcoin“). Fitcoin by měl být dostatečně jednoduchý, aby bylo možné ho případně využívat jako učební pomůcku.

Každá z prací se pak zabývá specifickým problémem, který je třeba při návrhu a implementaci takové kryptoměny vyřešit. Tato práce se zabývá problematikou monitorování.

Většinu vznikajících kryptoměn s Bitcoinem spojuje důraz na decentralizaci a nezávislost. Zatímco provozovatel centralizované kryptoměny vždy zná do posledního bitu stav celého systému, což mu dává možnost libovolně provoz v síti monitorovat, zpracovávat statistiky a v případě veřejného systému transakcí i sledovat a/nebo zveřejňovat obsah každé z nich, v decentralizované kryptoměně podobný všemocný subjekt zpravidla neexistuje. Případný zájemce o náhled na fungování takové kryptoměny je tudíž odkázán na řešení provozovaná třetí stranou.

Téma jsem si zvolil, protože slibovalo možnost začít pracovat „na zelené louce“ a vymyslet si řešení, které, jak jsem později potvrdil, by bylo vlastní pouze Fitcoinu.

V kapitole **Blockchain a kryptoměny** popisuji datovou strukturu jménem *blockchain*, její užitečné i problematické vlastnosti a její využití při implementaci digitální měny.

Kapitola **Implementace a sledování** dále přibližuje implementaci kryptoměny Bitcoin, která blockchain využívá. Popisuji v ní architekturu jednoho „účastníka“ v síti kryptoměny, tzv. nodu, a způsoby, jakými nody komunikují mezi sebou nebo s jejich uživatelským rozhraním v podobě konzolové aplikace. Druhá část kapitoly obsahuje zamyšlení nad různými cestami, kterými se lze vydat za cílem monitorovat provoz kryptoměny, jsou-li nebo nejsou-li k dispozici popsané mechanismy komunikace. Nakonec ve třetí části uvádím příklady několika existujících systémů, tzv. blockchain explorerů, a snažím se odhadnout, na jakém principu obecně explorery fungují.

Kapitola **Fitcoin** se zabývá specifiky vyvíjené kryptoměny, především jejími odlišnostmi od Bitcoinu a chybějící funkcionalitou, která mou práci nakonec výrazně ztěžuje.

V kapitole **Návrh systému pro sledování** se zamýšlím nad parametry řešení vhodného pro Fitcoin a následně volím to nejméně problematické, implementované pomocí jazyka Ruby a frameworku Rails.

Kapitola **Realizace** obsahuje stručný popis toho, proč bylo zvolené řešení stále ještě příliš problematické.

**Poznámka k pořadí kapitol, kultuře práce a splnění zadání:** Přestože jsou kapitoly v práci seřazené a formálně oddělené a například návrh

tak zdánlivě chronologicky předcházel realizaci (a oboje přišlo až po důkladném studiu dokumentace Fitcoinu a zavedených kryptoměn), ve skutečnosti probíhal vývoj Fitcoinu zcela souběžně a mé aplikace tak vznikaly poněkud „agilnějším“ způsobem.

Téma kryptoměn pro mě bylo z technického pohledu zcela nové a trvalo mi poměrně dlouho se v něm zorientovat a začít nacházet paralely mezi tím, o čem hovořili autoři Fitcoinu a tím, co jsem viděl v dokumentaci Bitcoinu.

Až pozdě jsem pochopil, že původní zadání bylo příliš optimistické a nepočítalo ani s tím, že se nemusí podařit do Fitcoinu vůbec implementovat funkcionalitu potřebnou pro efektivní sledování, natož abychom počítali s tím, že se kryptoměnu nemusí podařit vůbec spustit ani v základní podobě — což se nakonec stalo.

Zadání práce proto zůstalo původní, přestože se z požadavku na kvalitní finální aplikaci postupně stal požadavek na jakýsi MVP (minimální životaschopný produkt), a nakonec jen požadavek na kostru aplikace, kterou nebude možné ani otestovat.

Ve své práci se pokouším tento nedostatek odčinit důkladným popisem toho, co jsem se snažil implementovat, jaké problémy se vyskytly a jak je bude možné vyřešit po spuštění Fitcoinu a jeho vybavení další funkcionalitou.

Kapitola 1

# Cíle práce

Hlavním cílem práce je pochopit specifika různých způsobů řešení problému monitoringu kryptoměny založené na technologii blockchain, a následně tyto poznatky aplikovat při návrhu konkrétního řešení pro Fitcoin. Výstupem práce by měl být systém vzdáleně podobný například rané podobě *blockchain.info*.

K dosažení tohoto cíle je nutné se nejdříve důkladně seznámit s vnitřním fungováním blockchainu a kryptoměny ho využívající, tedy například skrze dokumentaci Bitcoinu. Poté je třeba shromáždit maximum dostupných informací z jakýchkoliv zdrojů (nejen oficiálních/vědeckých, kterých je nedostatek) o principech využívaných již existujícími systémy. Tyto informace by bylo vhodné nějak formálně ucelit, roztřídit a podrobně popsat.

Dále bude potřeba zvolit nejvhodnější přístup ke sledování Fitcoinu a navrhnout aplikaci, která ho bude využívat. Na závěr je nutné tuto aplikaci implementovat alespoň v jednoduché podobě a otestovat spojení se sítí kryptoměny.

Kapitola 2

# Blockchain a kryptoměny

V roce 2008 se na doméně *bitcoin.org* objevil dokument s názvem *bitcoin.pdf* [1]. Ve stejné podobě je k dispozici dodnes (2018). Jedná se o tzv. whitepaper (ustavující návrh) nového systému digitální měny. Autor (autoři) tohoto dokumentu vystupuje pod pseudonymem *Satoshi Nakamoto*.

Měna popsaná ve whitepaperu je nazvána *Bitcoin* a dnes je považována za první moderní kryptoměnu, která — jak dnes již víme — s odstupem času spustila lavinu zájmu o kryptoměny a vytvořila tak celé nové odvětví IT. Nakamoto na krátkém rozsahu whitepaperu přehledně a srozumitelně popisuje možné řešení hned několika kritických problémů digitálních měn — především důvěryhodnosti a dvojitého utrácení — s využitím kryptografie a systému ukládání dat do navazujících bloků.

Zajímavostí bezesporu je, že whitepaper neobsahuje [1] ani jednou slovo „kryptoměna“ nebo „blockchain“.

## 2.1 Blockchain

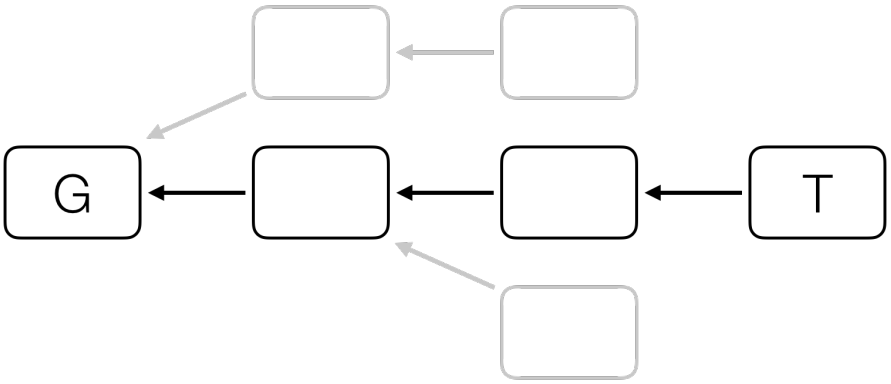
Situace okolo pojmu „blockchain“ je podobná situaci samotných kryptoměn. Tento pojem totiž nikdy nebyl formálně definován, dokonce nebyl svým tvůrcem ani použit — slovo „blockchain“ vzniklo postupným spojením dvouslovného výrazu „block chain“, kterým byla nazývána struktura popsaná ve whitepaperu. „Block chain“ je doslova „řetězem bloků“, a tak ho v zásadě je možné chápat.

*Následující popis je zjednodušený pro účely této práce. Jejím cílem není popisovat kryptografické principy ani analyzovat datové struktury. Původní koncept bitcoinového blockchainu popsaný ve whitepaperu byl v referenční implementaci Bitcoinu dodržen, obsahuje ale řadu nuancí, jejichž pochopení a napodobení při implementaci Fitcoinu je součástí jiných BP uvedených v úvodu.*

### 2.1.1 Struktura

Základním stavebním kamenem blockchainu je blok [1][2, kap. 9]. Jeho hlavní součást (tedy tzv. payload) tvoří (v kryptoměně) data transakcí — těch je v případě Bitcoinu proměnlivý počet, což si ale blockchain (coby technologie) nijak nevynucuje. Blok si dále nese malé množství metadat a — co je nejdůležitější — odkaz na svého bezprostředního předka (otce, předchůdce). Forma tohoto odkazu bude vysvětlena dále.

Tvorba blockchainu začíná speciálním, počátečním, tzv. „genesis“ blokem. Tento blok celý blockchain v podstatě identifikuje — je a musí být předkem všech ostatních bloků. Každý další blok do blockchainu přidávaný se poté odkáže (navěsí) na blok, který už součástí blockchainu je. Blockchain tak v kterýkoliv okamžik své existence vypadá jako orientovaný strom. Pro strom platí, že mezi dvěma vrcholy existuje právě jedna cesta. Pro blockchain, jehož bloky mají k dispozici vždy jen odkaz na předchůdce, potom platí, že z každého bloku existuje právě jedna cesta do genesis bloku. Pro každý blok tak dokážeme jednoznačně určit jeho vzdálenost.




Obrázek 2.1: Jednoduchý blockchain. **G** představuje genesis blok, **T** je vrcholem nejdelší větve.

Blok, který je v danou chvíli nejvzdálenější od genesis bloku (tedy pomyslný vrchol stromu, rostl-li by od kořene nahoru), je považován za validní, společně se všemi bloky na cestě do genesis. Transakce obsažené ve všech validních blocích jsou v tu chvíli brány v potaz a jejich zpětný průchod (směrem od genesis do vrcholu) tvoří závazný obsah „účetní knihy“ kryptoměny. Strukturu jednoduchého blockchainu představuje obrázek 2.1.

### 2.1.2 Proměnlivost

Velice důležité je uvědomit si, že se blockchain s časem mění. Validní a závazná jsou vždy jen data v nejdelším možném řetězu bloků. Vyroste-li z blockchainu jiná větev a objeví-li se na ní vzdálenější blok, začne tato nová větev před-

stavovat závazné informace a z „účetní knihy“ tak mohou zmizet data, která o chvíli dříve byla její součástí.

Ještě důležitější je chápat, že tato situace není chybou nebo selháním, jde o regulérní způsob fungování blockchainu. Pravděpodobnost „vypadnutí“ daného bloku z nejdelší větve samozřejmě prudce klesá s jeho vzdáleností od vrcholu takové větve (tedy s množstvím bloků nad ním), ale nikdy není nulová. Jistotu přítomnosti v aktuálně validním řetězu má jen genesis.

#### 2.1.2.1 Odkaz na předchůdce

Nic z výše uvedeného by nebylo užitečné, kdyby neexistoval robustní princip, jak vynutit „zavěšení“ jednoho konkrétního bloku pod jiný konkrétní blok (nebo „postavení“ na něj, podle úhlu pohledu).

Kvůli odkazování je vhodné každý blok v blockchainu jednoznačně identifikovat. Použití rostoucího indexu by intuitivně stačilo. Použití nějakého hashe, například z metadat bloku, by usnadnilo generování bloků několika zapisujícími aplikacemi současně. Blockchain jde ale ještě o kus dál — hash, kterým je blok identifikován (a na který se bude další blok odkazovat), se počítá z celého obsahu bloku, **včetně odkazu na předchůdce** a **včetně všech dat**!

### 2.1.3 Funkce

V předchozí části popsaná struktura blockchainu je na první pohled jednoduchá, zajišťuje [2, kap. 12] ale některé důležité vlastnosti:

**neměnnost bloků:** Blok je identifikován hashem veškerého svého obsahu. Změní-li se jakkoliv obsah bloku, změní se hash a tento nově vzniklý objekt není další (kolidující) instancí téhož, ale zcela odlišným blokem.

**neměnnost odkazů:** Stejným způsobem — blok, ve kterém by se změnil odkaz na předchůdce, by se stal odlišným blokem s odlišným hashem.

**chronologická návaznost:** Každý blok může být navázán pouze na blok, který již existuje, protože odkaz má formu hashe, který musí být již známý (tedy první blok musí již existovat). To zaručuje, že jakákoliv větev blockchainu, přečtená od genesis směrem k vrcholu, obsahuje chronologickou posloupnost bloků.

**nezmenšující se blockchain:** Bloky nelze z blockchainu nijak odebírat. Regulérní přidání nového bloku, stejně jako pokus pozměnit data již existujícího bloku (nelze od sebe odlišit), blockchain zvětší o jeden blok.

O tyto vlastnosti se pak opírá celá funkcionalita blockchainu a jeho využití v kryptoměnách.

#### 2.1.3.1 Obtížné pozměňování dat

Blok, který je součástí validní větve blockchainu, je dobře chráněný proti pozměnění, je-li na něj navěšeno už několik dalších bloků. Potenciální útočník se zájmem pozměnit data v takovém bloku a přesvědčit o své pravdě další uživatele téhož blockchainu by musel do blockchainu přidat vlastní verzi daného bloku s pozměněnými daty. Vznikl by nový blok s odlišným hashem, který by ale nikdo jiný nepovažoval za validní (tvořil by vrchol kratší větve).

Aby se validním stal, musel by útočník pokračovat v přidávání dalších bloků tak dlouho, dokud by jeho větev nepředstihla v tu chvíli validní větev — teprve poté by byla považována sama za validní, se všemi bloky, které obsahuje na cestě od vrcholu do genesis. Následuje vysvětlení, proč toto není obvykle možné.

#### 2.1.3.2 Ochrana proti spamování bloků

Protože běžný provoz blockchainu, coby úložiště dat, má předem odhadnutelný objem, je možné o každém výrazně rychlejším přibývání bloků rozhodnout, že se jedná o útok na blockchain a snahu předběhnout legitimní nejdelší větev. Proto se blockchain opatřuje nějakým druhem ochrany. Bitcoin a jeho deriváty používají tzv. proof-of-work — nutnost „zaplatit“ za přidání bloku výpočetním výkonem.

Tato ochrana opět chytře využívá hash představující identifikátor bloku. Blok kromě všech běžných dat obsahuje navíc jednu položku, tzv. *nonce*. Ta nemá žádný význam a představuje jen jakousi volnou proměnnou pro proces hashování. Změnou nonce se mění hash výsledného bloku, přestože data v něm obsažená zůstávají stejná. Zájemce o přidání bloku do blockchainu tak musí zkoušet další a další hodnoty nonce a porovnávat vypočítané hashe s nějakým kritériem (toto kritérium může být pro blockchain pevné, případně se může dynamicky měnit). Celá operace může být při přísně nastaveném kritériu výpočetně extrémně náročná — toho se dá využít (a využívá) pro regulaci rychlosti přidávání bloků.

## 2.2 Kryptoměna založená na blockchainu

### 2.2.1 Obecně

Jak bylo zmíněno v úvodu, kryptoměna je systém určený pro bezpečné a trvalé uchování záznamů o velikosti držených finančních prostředků, který umožňuje jejich vlastnictví převádět, a který jejich pravost a vlastnictví zajišťuje pomocí kryptografie [3]. Popis blockchainu z předchozí sekce dává tušit, proč se k takovým účelům jeho použití nabízí (a proč k těmto účelům také vznikl).

Bloky blockchainu, tvořícího součást kryptoměny, obsahují jako data seznamy transakcí. Transakce pak vyjadřují přesun financí mezi jednotlivými entitami.

Na tomto místě je vhodné důrazně připomenout, že kryptoměnu nelze 1:1 srovnávat s tradičními bankami — nejen funkcionalitou, ale ani strukturou (transakcí, účtů). Následující popis platí pro Bitcoin a jeho deriváty.

#### 2.2.1.1 Adresa

Adresu si lze intuitivně představit nikoliv jako účet v bance, ale jako anonymní schránku na nádraží. Samotný řetězec adresy je zároveň identifikátorem (číslem na schránce), a v kryptografickém smyslu také veřejným klíčem (zjednodušeně).

Odpovídající privátní klíč potom představuje klíček, kterým se tato schránka odemyká.

#### 2.2.1.2 UTXO

*Unspent transaction output* navzdory doslovnému překladu svého názvu odpovídá intuitivně jedné minci (která je tvořena takovým množstvím drahého kovu, jaká je její hodnota). Jeho hlavní informací je právě tato hodnota.

#### 2.2.1.3 Transakce

Transakci tvoří [2][kap. 2, 6] dva seznamy UTXO — vstupy a výstupy. Intuitivně si lze transakci představit jako nádražního zřízence a kováře v jednom.

Seznam vstupů obsahuje podepsané UTXO, které se mají sečíst a stát vstupem do transakce. Seznam výstupů poté obsahuje UTXO, které mají vzniknout. Každé UTXO na výstupu je přiřazeno k jedné adrese. Tento proces je v Bitcoinu výrazně zkomplikován existencí skriptovacího systému pro určování, kdo je oprávněn s výstupem transakce manipulovat, proto bude lepší se držet intuitivního popisu:

Pomyslný nádražní zřízenec/kovář dostane k dispozici seznam mincí (UTXO) a klíče ke skřínkám (tady porovnání trochu skřípe, má jen podpisy), ve kterých jsou mince uloženy. Všechny je projde, skřínky si díky vlastnictví klíče otevře a mince vybere. Následně mince sesype na hromádku (sečte hodnoty). Poté mince roztaví a podle výstupního seznamu postupně odlévá nové mince o požadovaných hodnotách. Obvykle mu zbyde malá část drahého kovu neodlitá, ta se stává poplatkem transakce. Nově odlité mince jsou pak vloženy do jiných schránek (bez potřeby je otevírat, informace o adrese-schránce je ve skutečnosti uložena přímo UTXO-minci), kde čekají na to, aby se samy staly vstupem další transakce.

#### 2.2.1.4 Poznámka — důsledky

Z výše uvedeného je patrné, že představa existence jakési „banky Bitcoin“ a obecně přirovnávání kryptoměn ke klasickým platebním systémům jsou myšlenky zcela liché. Zjistit například i tak jednoduchou věc, jako „od koho mi

přišly peníze?“ je díky struktuře transakce prakticky nemožné — transakce mohla na jednu mnou používanou adresu přiřadit jeden UTXO slitý z dvaceti různých vstupů, nebo mohla na deset mnou používaných adres přiřadit deset odlišných UTXO slitých ze dvou různých vstupů. Dokonce i idea „používané adresy“ je poněkud pomýlená — adresa by se neměla [4] používat opakovaně.

### 2.2.2 Běh

Síť kryptoměny tvoří [2, kap. 8] velké množství tzv. nodů (uzlů, používat budu anglický výraz), které implementují protokol komunikace o dění v síti, a které si každý udržují vlastní blockchain. Každý z nodů je spojen s několika ostatními a informace se pak po síti šíří formou tzv. gossipu.

Nody si mezi sebou předávají informace o transakcích, které se někdo snaží provést. Každý node se pak snaží vybrat si transakce, které se mu hodí (připadnou mu z nich nejvyšší poplatky), z těchto transakcí sestavit blok a pro tento blok najít vhodnou hodnotu nonce tak, aby blok prošel kritériem pro obtížnost a byl tedy zařazen do blockchainu. V okamžiku, kdy nějaký node takový blok nalezne, svůj nález oznámí svým sousedům. Díky gossipu se tato informace rychle rozšíří po celé síti. Node, který se dozví o existenci nového objektu, o něj může informující node obratem požádat.

Proces ověřování transakcí a tvorby nového bloku se obecně nazývá [2, kap. 2, 10] *těžba*, přestože generování nových mincí (uvolněných sítí) není jeho primárním účelem (tím je ověřování) a přestože náročnost na výpočetní výkon je zcela umělá (je to reakce ochrany proti útoku na blockchain).

Kapitola 3

# Implementace a sledování

*Kapitola se opět týká primárně Bitcoinu a jeho derivátů, konkrétní příklady se týkají referenční (nejpoužívanější) implementace Bitcoin Core [2, kap. 3].*

## 3.1 Implementace

V předchozí části bylo nastíněno fungování kryptoměny založené na blockchainu tak, jak ho popsal Satoshi Nakamoto ve whitepaperu Bitcoinu. Stejný autor nebo skupina autorů byl také prvním, kdo myšlenky whitepaperu uvedl do praxe, v podobě referenční implementace Bitcoinu. Deriváty Bitcoinu se architektury referenční implementace převážně držely, díky čemuž lze i Fitcoin s něčím kriticky srovnávat.

### 3.1.1 Architektura nodu

V případě Bitcoinu a jeho derivátů je node představován běžícím démonem (*bitcoind* v případě Bitcoinu, *litecoind* v případě Litecoinu), který obstarává provoz podle protokolu dané kryptoměny. Aplikace je ale skutečným démonem, běží tedy tzv. headless a sama zpravidla neimplementuje žádné uživatelské rozhraní.

*Referenční bitcoinový klient obsahuje kromě démona bitcoind i aplikaci s uživatelským rozhraním. Její existence nebo bližší zkoumání pro tuto práci nejsou důležité.*

Z popisu blockchainu v kapitole č. 2 vyplývá, že pro správné fungování blockchainu je nezbytně nutné [2, kap. 3] mít ho k dispozici celý (zjednodušeně). Jakákoliv chybějící data by způsobila roztržení pomyslného řetězu tvořeného aktuálně validní větví, nebo větví, která by validní být mohla.

Velikost blockchainu Bitcoinu se dnes již počítá na stovky [5] gigabytů, dávno již tak není možné (a nikdy se s tím ani reálně nepočítalo), že by si jej node držel jen v paměti a při spuštění o něj vždy požádal ostatní. *bitcoind* si tak musí [6] obsah blockchainu udržovat hned třemi způsoby:

**adresář `blocks`:** Obsahuje kompletní surová data o všech blocích, tak, jak byla přijata ze sítě.

**podadresář `blocks/index`:** Obsahuje LevelDB databázi s metadaty bloků a jejich umístěním na disku. Výrazně zvyšuje rychlost vyhledávání.

**adresář `chainstate`:** Obsahuje LevelDB databázi s údaji o všech dostupných UTXO a metadaty transakcí. Umožňuje rychlou validaci transakcí bez průchodu celým blockchainem.

Po spuštění nového nodu tento požádá okolní známé nody o obsah blockchainu a stráví i desítky minut „synchronizací“, tedy stahováním a ukládáním dat.

### 3.1.2 Komunikace

#### 3.1.2.1 Mezi nody

Pro komunikaci mezi jednodlivými běžícími instancemi *bitcoind* se využívá úsporný protokol přenášený přes běžné TCP spojení. Podporováno je i IPv6.

Nody si mezi sebou vyměňují [2, kap. 8] informace kritické pro provoz kryptoměny (viz kapitola 2) a informace týkající se samotné sítě (handshake, dotazy na možné sousedy…) pomocí asi třiceti různých zpráv [7], které jsou identifikovány dvanáctimístnými řetězci s jednoduchými názvy zpráv doplněnými nulami.

Nejdůležitější zprávami protokolu jsou:

**`version`, `verack`:** Používají se jako handshake před zahájením komunikace.

**`getaddr`:** Žádost o adresy dalších nodů, posílá nový node po zahájení provozu.

**`inv`:** Obsahuje seznam objektů, jejichž vlastnictví chce node ohlásit svému okolí, používá se především po nalezení (vytěžení) nového bloku.

**`getblocks`:** Žádost o seznam bloků (`inv`) v daném rozsahu.

**`getdata`:** Žádost o kompletní obsah objektu s daným hashem.

**`block`, `tx`:** Odpověď na `getdata`, obsahuje kompletní objekt s požadovaným hashem.

Podobně jako tyto příklady, i zbylé neuvedené zprávy jsou zprávami interními, důležitými pro fungování kryptoměny. Nody si je posílají mezi sebou a přijímají je jen od jiných nodů.

#### 3.1.2.2 Ovládání, API

Síťový protokol pro komunikaci s ostatními nody ale není jedinou řečí, kterou referenční implementace ovládá. *bitcoind* totiž není jen správcem blockchainu, ale také implementací tzv. peněženky — aplikace, která dokáže generovat adresy, ukládat jim náležející privátní klíče, vytvářet transakce, vyhledávat v blockchainu transakce ve prospěch jí vygenerovaných adres, tedy zkrátka fungovat jako fyzická peněženka a platební brána zároveň.

Tato funkcionalita se samozřejmě neobejde bez nějaké formy uživatelského rozhraní a, jak bylo již uvedeno, *bitcoind* běží v *headless* režimu. Musí tak existovat způsob, jak může *bitcoind* komunikovat s jinými aplikacemi nebo uživatelem.

Pro tyto účely poskytuje *bitcoind* ještě API formou RPC rozhraní dostupného defaultně na portu 8332. To očekává HTTP POST požadavky. Data požadavku musí tvořit JSON objekt odpovídající specifikaci JSON-RPC. Odpověď na takový požadavek obsahuje také JSON objekt s informacemi o provedení/neprovedení příkazu, nebo odpovědí na dotaz.

Zpráva pro toto API pak může vypadat například následovně:

```
{
    "method": "getblockhash",
    "params": [0],
    "id": "foo"
}
```

Tato zpráva je příkladem volání GetBlockHash, představuje žádost o vrácení hashe genesis bloku (jak naznačuje parametr 0).

Na rozdíl od utilitárního komunikačního protokolu samotné kryptoměny, toto API poskytuje mnohem bohatší funkcionalitu, včetně funkcí potenciálně užitečných pro sledování stavu blockchainu a celé sítě:

**`GetBestBlockHash`:** Vrací hash (tedy identifikaci) posledního bloku na nejdelším řetězu. Tato zpráva jednoznačně identifikuje nejdelší řetěz a tedy i celý stav „účetní knihy“.

**`GetDifficulty`:** Vrací aktuální hodnotu obtížnosti nastavené v síti.

**`GetMemPoolInfo`:** Vrací informaci o aktuálním stavu mempoolu (známé transakce, které jsou kandidáty na vložení do bloku) daného nodu.

**`GetNetTotals`:** Vrací informace o stavu sítě, o množství přenesených dat a například i datovém limitu nodu.

**`GetNetworkHashPS`:** Vrací odhad průměrné hashrate celé sítě během n posledních bloků, nebo od poslední změny obtížnosti.

**`GetBlockChainInfo`:** Vrací 30 různých údajů o stavu blockchainu, včetně obtížnosti, mediánu času mezi jednotlivými bloky, stavu synchronizace nebo informací o prostříhávání bloků.

**`GetChainTips`:** Vrací informace o vrcholech jednotlivých větví tvořících blockchain, včetně informací o jejich délce (množství bloků mimo nejdelší řetěz). Dává jednoznačnou informaci o struktuře celého blockchainu.

Součástí referenčního řešení je i RPC klient *bitcoin-cli*, který umožňuje z prostředí příkazové řádky toto API jednoduše využívat. Jeho použití vypadá takto:

```
bitcoin-cli [options] <method name> <param1> <param2> ...
```

tedy například

```
bitcoin-cli addwitnessaddress 1BRo7qrYHMPrzdBDzfjmzteBdYAyTMXW75
```

## 3.2 Přístupy ke sledování

V současné době na internetu běží téměř 1600 [8] kryptoměn. Drtivou většinu z tohoto počtu tvoří decentralizované systémy, pro které přirozeně neexistuje konkrétní autorita, která by umožnila snadný náhled do „účetní knihy“. Aplikace, která takovou funkcionalitu nabízí, se obecně [2, kap. 2] nazývá „blockchain explorer“. Pro nejpoužívanější decentralizované kryptoměny proto již vznikla celá řada řešení, založená na různých principech. Nadále budu uvažovat jen kryptoměnu podobnou Bitcoinu, decentralizovanou a založenou na blockchainu, případně samotný Bitcoin.

Komunikační protokol samotné kryptoměny, API, které poskytují její nody, i data, která si nody ukládají pro svou vlastní potřebu, nabízejí několik možností, jak lze shromažďovat data o blockchainu za účelem přehledného zobrazování uživateli, především pomocí webové prezentace.

*Následující část kapitoly, přestože teoretická, je tvořena z velké části již jen mnou. Žádné podobné shrnutí ani rozdělení jsem nikde (na webu) nedokázal najít, v literatuře pak tím spíše vůbec neexistuje. Kusé informace, které jsem v následující části formalizoval, se nacházely na naprosto necitovatelných místech — jako off topic v diskuzních příspěvcích, na Redditu, jako zmínky v jiných článcích…*

### 3.2.1 Aplikace „parazitující“ na nodu

Základní ideou tohoto principu je využití dat, která si node potřebuje ukládat pro svůj vlastní běh (jak bylo popsáno v první části kapitoly). O těchto datech víme, že (nevyužívá-li node pokročilé techniky prořezávání blockchainu) musí obsahovat veškerý obsah všech bloků i všech transakcí. Aplikace tohoto typu postavená nad Bitcoinem by tak využívala již běžící *bitcoind*, k jehož souborům by měla přístup, případně si sama spouštěla vlastní instanci. *bitcoind* si ukládá surová data bloků, metadata pro snadné vyhledávání a rychlý seznam dostupných UTXO. Tato data by stačilo běžícímu *bitcoind* doslova „číst pod rukama“. Takto fungující aplikaci by navíc bylo možné implementovat dvěma/třemi způsoby:

**Čtení tzv. „on the fly“:** Aplikace by při každém požadavku na zobrazení určitého objektu (bloku, transakce), tedy například po kliknutí uživatele na tlačítko Detail ve webové prezentaci, v metadatech vyhledala polohu dat v souboru a data objektu jednorázově načetla, transformovala a zobrazila.

**Čtení z vlastní databáze:** Aplikace by při požadavku na zobrazení objektu sáhla do vlastní databáze, kterou by sama (nebo pomocným démonem) periodicky doplňovala o data přečtená z pracovních souborů nodu.

**Líné načítání:** Aplikace by spojovala oba přístupy a transformovaná data si držela ve vlastní databázi, ovšem přidávala je do ní až na popud uživatele při prvním zobrazení.

Tento přístup je zřejmě nejjednodušší na implementaci, dokud se jedná o jednoduchou kryptoměnu. Ve zbytku této kapitoly bude k nalezení příklad, proč se nejedná o nikterak robustní řešení. Zároveň tento přístup vyžaduje velmi dobrou znalost využívaného nodu a jasně zadokumentované nakládání s daty.

Obrovskou nevýhodou může být nutnost řešit redundantní logiku. Z pracovních souborů se surovými objekty nemusí být patrná celková struktura blockchainu, tedy především informace o tom, který řetěz je aktuálně nejdelší a které bloky tvoří vrcholy jednotlivých větví. Vyžádá-li si to formát prezentace, je nutné veškerou potřebnou funkcionalitu replikovat uvnitř monitorovací aplikace, což v případě složitějších operací vede k rozsáhlé redundanci kódu. V extrémním případě pak může dojít k tomu, že se aplikace stane téměř novou implementací nodu, jen rozšířenou o prezentaci dat a ochuzenou o protokol kryptoměny.

### 3.2.2 Aplikace využívající API nodu

V případě, že je k dispozici chytrý node, jakým je například referenční implementace Bitcoinu [2, kap. 3], není nic jednoduššího, než intenzivně využívat bohaté API [9], které node sám nabízí. V minulé části této kapitoly bylo uvedeno několik příkladů funkcí, které node poskytuje a které by mohly skvěle posloužit monitorovací aplikaci. Takovou aplikaci by bylo možné zpracovat libovolným způsobem mezi dvěma extrémy:

**„Překladatel“:** Celý blockchain explorer by představovala jediná webová aplikace. Její zodpovědností by byl především frontend, tedy přehledná prezentace dat. Místo ORM a databázového stroje by ovšem zdrojem dat byl jakýsi modul, která by překládal dotazy z webové prezentace na dotazy pro API nodu a odpovědi vracel ve vhodném formátu. Například dotaz na všechny bloky v nejdelším řetězu by tak webová aplikace přijala, vygenerovala odpovídající dotaz (pro Bitcoin) `GetBestBlockHash` a navazující sérii `GetBlock` pro API, počkala na odpovědi, data deserializovala a prezentovala. Toto řešení by bylo velmi jednoduché na implementaci a díky využití oficiálního API zároveň robustní, i když v této formě také neefektivní (a citlivé na stabilitu nodu — jeho vypnutí nebo pád by znamenaly ztrátu zdroje dat).

**Plnohodnotná webová aplikace:** Klasická webová aplikace využívající jako zdroj dat oddělený databázový stroj. Ukládání dat do databáze by měla na starost jiná aplikace, periodicky se dotazující běžícího nodu skrze API na možné změny v blockchainu, a případně transformující nová data podle použitého databázového schématu.

Využití chytrého API některé z implementací je zjevně nejvhodnějším přístupem při návrhu blockchain exploreru, protože nabízí největší flexibilitu. Navíc jde o jediný možný „technicky čistý“ přístup, protože při něm nedochází k duplikaci žádné funkcionality.

### 3.2.3 Aplikace využívající protokol kryptoměny

Třetí přístup představuje jedinou použitelnou možnost v situaci, kdy nejsou k dispozici pracovní soubory nodu a zároveň nejsou nody samy tak chytré, aby dokázaly data poskytnout ve vhodnější podobě. Spočívá ve využití samotného komunikačního protokolu kryptoměny pro sběr dat. Protože nody nemají možnost zjistit, kdo se s nimi pomocí protokolu baví (dokud protokol dodržuje, samozřejmě), je možné napsat „aplikaci-prostředníka“, který se připojí do sítě kryptoměny, běžným způsobem se synchronizuje s ostatními a poslouchá gossip v síti stejně jako ostatní nody — s tím rozdílem, že sám nemá žádný zájem chovat se jako node a vynucovat pravidla kryptoměny.

Tento prostředník může získaná data vhodně transformovat a přepisovat do databáze. Ta je — na rozdíl od dvou předchozích principů — v tomto případě nezbytná. Prostředník sbírá data nepřetržitě tak, jak se o nich dozví. V případě, že by měl v krátkém časovém intervalu odpovědět obsahem určitého objektu, nemohl by tuto odpověď garantovat, protože sám by o ni musel pomocí síťového protokolu teprve požádat jiný node. Odpověď od tohoto nodu by nemusela vůbec přijít, nebo by přišla pozdě.

Pokud bychom se tomuto problému snažili vyhnout použitím nějaké formy cachování nebo využitím například lehké LevelDB databáze, v určitém okamžiku bychom zjistili, že implementujeme vlastní node, obohacený o sledování blockchainu.

Navíc, podobně jako aplikace parazitující na interních souborech nodu, takovýto pseudo-node nemá k dispozici žádné nástroje týkající se logiky. Strukturu blockchainu si musí domyslet sám, nejdelší možný řetěz musí rozeznat sám, možnou změnu validní větve na jinou musí kontrolovat sám. Tento systém tak trápí stejné problémy s redundancí většiny kódu.

### 3.2.4 Node s integrovaným explorerem

Poslední — poněkud exotickou — možností by bylo funkcionalitu webového blockchain exploreru integrovat do nodu. Například Bitcoin Core pro své nody nabízí kromě *bitcoind* i verzi s grafickým rozhraním, které umožňuje ovládání běžným způsobem, nebo například procházení transakcí konkrétní peněženky.

Takové grafické rozhraní je samozřejmě možné implementovat místo nativních knihoven pomocí webových technologií a podle potřeby zpřístupnit jen lokálně pro ovládání, nebo i přes internet pro funkcionalitu blockchain exploreru.

Výhodou takového přístupu je, že práci s daty zajišťuje samotný node — data ho v žádném okamžiku neopustí. Není tak třeba duplikovat logiku ani kód pro čtení dat ze souborů nodu.

Nevýhodou je nutnost při návrhu takového systému úzce kooperovat s vývojáři nodu, nebo jedním z nich být. Dalším problémem může být větší velikost i výpočetní náročnost takového nodu, je-li cílem efektivní implementace.

## 3.3 Existující řešení

Od počátku existence Bitcoinu vzniklo hned několik projektů zaměřených na sledování blockchainu a umožnění náhledu na něj. O složitosti celého problému nejlépe svědčí fakt, že většina takových systémů za své služby postupně začala vybírat peníze, a to od uživatelů i od programátorů, kteří se tak mohou soustředit na business logiku a další aspekty svých aplikací, a pro práci s blockchainem využít ověřené a stabilní nástroje.

### 3.3.1 blockchain.info

Jedním z nejznámějších explorerů je *blockchain.info*. Vznikl v srpnu 2011 a od té doby svou působnost rozšířil daleko za hranice prostého prohlížeče obsahu blockchainu — dnes nabízí například nákup nebo prodej kryptoměn a je přítomen i na síti Tor. Jedná se o komerční službu, která vývojářům nabízí bezplatné API dostupné bez registrace a omezené počtem požadavků, případně další placené služby. Architektura celé aplikace ovšem není veřejně známá a na jejím hladkém fungování stojí úspěch celé služby.

Inspirací pro mou práci může být UI samotného webového rozhraní.

### 3.3.2 BlockCypher

Další službou podobného ražení je *BlockCypher*, který, podobně jako *blockchain.info*, poskytuje vývojářům blockchainových aplikací pohodlné API implementované proprietární aplikací.

Na rozdíl od *blockchain.info* ale *BlockCypher* uvolňuje alespoň referenční webový explorer (`live.blockcypher.com`) jako open-source příklad využití BlockCypher API.

BlockCypher spolupracuje se sítí Bitcoin, deriváty Litecoin a Dogecoin a s kryptoměnou Dash. Architektura aplikace je taktéž neznámá.

Má aplikace se může inspirovat UI i architekturou referenční webové aplikace.

### 3.3.3 Abe

*Abe* je open-source [10] explorer pro Bitcoin. Ke svému fungování vyžaduje existenci nodu Bitcoinu (*bitcoind*), jehož soubory využívá, jedná se tedy o druh aplikace popsaný v sekci 3.2.1.

Do verze *bitcoind* 0.8 bylo možné takto využívat aktuálně běžící node, protože se dalo očekávat, že bude *bitcoind* zapisovat pouze na konec souborů se surovými daty bloků. V pozdějších verzích se na toto nelze spolehnout a Abe tak může přeskakovat [10] bloky.

Abe je napsaný v Pythonu, pro ukládání dat používá vlastní databázi (PostgreSQL nebo SQLite 3) a zároveň provozuje jednoduchou webovou prezentaci načtených dat. Načtení blockchainu mu trvá i několik dní a výměna souborů s bloky si může vynutit nové vybudování databáze. Jeho vývoj se v poslední době v podstatě zastavil, navíc z výše uvedených důvodů již nefunguje správně nad aktuálně běžícím full nodem.

Inspirací pro můj explorer může být jen webová část aplikace.

### 3.3.4 insight

open-source blockchain explorer poskytující vlastní API (WebSocket a REST) i webové rozhraní. Jeho API umožňuje využívat i složitější dotazy, než běžný

*bitcoind*. insight sám využívá API poskytované nodem, ovšem místo referenční implementace Bitcoin Core (tedy *bitcoind*) používá [11] alternativní implementaci jménem *bitcore*.

### 3.3.5 Další

Dalšími příklady explorerů jsou:

- Block Explorer (blockexplorer.com)
- block.io
- SoChain (chain.so)

Block Explorer využívá *insight* API, zbylé dvě služby jsou komerční, a tudíž proprietární.

### 3.3.6 Závěr

Aniž bych toto tvrzení mohl věrohodně ozdrojovat, obecně mám ze zkoumání problematiky pocit, že většina takovýchto systémů — ať už je obalena kolika chce vrstvami API různých společností — využívá nakonec funkcionalitu samotných nodů, ať už referenčních, nebo speciálně upravených.

Kapitola 4

# Fitcoin

Fitcoin je kryptoměna, která vzniká souběžně s touto prací (tedy souběžně se svým vlastním blockchain explorerem). Jedná se o spolupráci pěti studentů (M. Dvořák, R. Kaufman, K. Pajskr, J. Tománek, M. Trieu) v rámci pěti bakalářských prací.

Fitcoin vzniká na popud vedoucího všech prací (Mgr. Jan Starý, Ph.D.). Cílem projektu Fitcoin je navrhnout a implementovat napodobeninu raných verzí Bitcoinu, tedy kryptoměnu řešenou pokud možno elegantně a chytře, ovšem pochopitelnou i pro člověka bez podrobnějších znalostí problematiky (typicky studenty).

## 4.1 Parametry

Fitcoin byl proti svému vzoru, Bitcoinu, v několika ohledech zjednodušen, a to nejen pro snadnější implementaci, ale především pro snadnější pochopení. Některé důležité parametry Fitcoinu vypadají [12] v době psaní této práce takto:

**Kryptografie:** Fitcoin používá asymetrickou kryptografii nad eliptickými křivkami, stejně jako Bitcoin. Použitou křivkou je *secp256k1*. Pro hashování se používají funkce *SHA-256* a *RIPEMD-160*

**Mince:** Základní jednotkou měny je jeden fitcoin (FTC), používat se dají násobky podle soustavy SI (kFTC, MFTC, GFTC, …)

**Adresa:** Je vyjádřena vždy jako 40znaková hexadecimální podoba hashe veřejného klíče. Na rozdíl od bitcoinové adresy neobsahuje checksum ani verzi

**Transakce:** Cílem outputů transakcí jsou přímo adresy, Fitcoin nepoužívá žádný skriptovací systém.

**Blok:** Hodnota `bits` v hlavičce bloku regulující obtížnost těžby má rozsah 8 bitů oproti 32b hodnotě v Bitcoinu.

**Seznamy:** Delší datové typy se chovají jako pole bytů a používají *big-endian* kódování. To je důležité brát v úvahu při deserializaci.

## 4.2 Komunikace

### 4.2.1 Síťový protokol

Nody Fitcoinu spolu komunikují [12] téměř identickým způsobem, jako nody Bitcoinu — přes TCP spojení si vyměňují zprávy v následujícím úsporném formátu:

1. hlavička `[20 b]`
   a) magická hodnota `[4 b]`
   b) zpráva `[12 b]`
   c) velikost dat *(X)* `[4 b]`
2. data `[X b]`

Zprávy, které si nody vyměňují, jsou jednoduššími verzemi zpráv používaných Bitcoinem. Zprávy podporované Fitcoinem jsou [12]:

**`version`:** Je posílán stranou, která otevírá spojení.

**`getaddr`:** Žádost o adresy dalších nodů.

**`addr`:** Obsahuje seznam IP adres dalších nodů.

**`inv`:** Obsahuje seznam objektů, které má node k dispozici. Může být odpovědí na `getblocks`.

**`getdata`:** Žádost o objekty se zadanými hashi.

**`notfound`:** Informace, že objekty nejsou k dispozici.

**`tx`:** Obsahuje data jediné transakce.

Dalšími zprávami, které by protokol měl podporovat, jsou zprávy `block` a `getblocks`. *(Tyto zprávy nejsou v okamžiku psaní práce zadokumentované ani funkční.)*

### 4.2.2 Lokální zprávy, API

Podobně jako v referenční implementaci Bitcoinu, i ve Fitcoinu představuje nejjednodušší formu nodu démon *fitcoind*. Stejně jako v Bitcoinu je třeba *fitcoind* nějak řídit, minimálně mu oznámit, že má skončit. K těmto účelům Fitcoin používá zvláštní sadu zpráv předávanou po běžném síťovém protokolu (např. pomocnou aplikací *fitcoin-cli*), tzv. lokální zprávy. Mezi lokální zprávy patří:

**`quit`:** Oznámí nodu, aby se ukončil.

**`getinfo`:** Vrátí informace o nodu a jeho sousedech.

**`addnode`:** Informuje node o jiném nodu, ke kterému by se měl připojit.

Jako další by měl `fitcoind` rozumět zprávám pro ovládání peněženky nebo tvorbu transakcí a dalším. *(Tyto zprávy nejsou v okamžiku psaní práce zadokumentované ani funkční.)*

#### 4.2.2.1 Chytrá funkcionalita

Projekt Fitcoin ve své první verzi nenabízí žádnou chytrou funkcionalitu. Není schopen na požádání poskytnout údaje o blockchainu ani parametrech sítě. Kromě lokálních zpráv neposkytuje žádné jiné API. Případná chytrá funkcionalita (která, jak dává tušit kapitola 3, je kritická pro provoz blockchain exploreru) může být implementována později jak formou lokálních zpráv, tak separátního API, například formou JSON-RPC, kterou používá Bitcoin Core.

Kapitola 5

# Návrh systému pro sledování

## 5.1 Úvaha

Při návrhu možného řešení systému pro monitorování Fitcoinu je třeba brát v potaz několik faktorů:

- zaměření Fitcoinu, coby jednoduché modelové kryptoměny
- minimální požadavky na rychlost/škálování aplikace
- nutnost se přizpůsobovat vývoji Fitcoinu v rámci paralelních BP
- webový charakter aplikace
- omezení daná absencí API

Tyto faktory se odráží v následující úvaze:

Zaměření Fitcoinu vyžaduje přístupnost všech použitých součástí a snadnou čitelnost zdrojového kódu. Protože se neočekává masivní rozšíření, je zřejmé, že nebude třeba zpracovávat vysoká množství transakcí, (de)serializovat mnohagigabytový blockchain a pravděpodobně zatím ani při monitorování spolupracovat vícevláknově s mnoha nody. Protože celý Fitcoin, tedy struktura bloků a transakcí i samotný protokol komunikace, je vyvíjen souběžně s monitorovacím systémem, je vhodné pracovat s přehledným kódem, který je možné snadno přepracovávat podle poslední dokumentace Fitcoinu. Webový charakter aplikace jen dává další důvody, proč se vyhnout vývoji na příliš nízké úrovni.

Výsledná aplikace — coby možná učební pomůcka — by měla být snadno pochopitelná, její funkcionalita by měla být přímočará, nekomplikovaná a zod-

povědnost každé součásti jasně definovaná. Vhodná by také byla modularita — možnost jednotlivé součásti aplikace dle potřeb měnit nebo vylepšovat, aniž by tím byla narušena funkčnost řešení.

### 5.1.1 Architektura

#### 5.1.1.1 Forma

Absence jakéhokoliv chytrého API neumožňuje využít nejčistější přístup explorerů — spolupráci s „ochočeným“ nodem. V úvahu by připadalo použití parazitní aplikace, jak byla popsaná v části 3.2.1, toto řešení ale obecně vyžaduje podrobnou znalost struktury dat, která node ukládá, a ideálně také dostupnost hotové, stabilní, referenční implementace. Druhým použitelným řešením je řešení popsané v části 3.2.3 — aplikace maskující se jako node a využívající síťový protokol pro sběr dat. V každém případě bude explorer muset provozovat vlastní databázi.

#### 5.1.1.2 Monolitická aplikace

Nabízí se možnost aplikaci koncipovat monoliticky — pak by obstarávala jak webovou část práce, tedy práci s protokolem HTTP a databází, tak i zisk dat.

Odlišný formát dat a nutnost je serializovat pro uložení do databáze nepředstavují výrazný problém, nestandardní by pro (primárně) webovou aplikaci ale byla práce se stále otevřeným spojením (v případě použití falešného nodu). Monitorovací systém dále musí celou řadu operací provádět autonomně, například iniciovat komunikaci. V případě dalšího vývoje Fitcoinu, a tedy nutnosti aplikaci učinit robustnější a bezpečnější, by pak vznikla potřeba autonomně sledovat integritu databáze, vyhodnocovat komunikaci s několika „přátelskými“ nody najednou a provádět další akce typické pro démony. Webové frameworky tento způsob fungování obvykle [13] nijak nepodporují.

#### 5.1.1.3 Modulární aplikace

Zcela opačnou možností je rozdělení aplikace na tři součásti:

- webovou prezentaci
- databázi
- prostředníka pro zisk dat

Webová prezentace a prostředník jsou aplikace zcela odlišného typu, proto je intuitivním řešením je také pojmout zcela odlišně.

Z webové prezentace by se tak stal jednoduchý náhled na data z databáze, využívající vhodné ORM pro spojení s ní. Změnou konfigurace ORM by

bylo snadné celou aplikaci nasměrovat na odlišnou databázi, například využívající jiný zdroj dat, případně zobrazující jinou kryptoměnu (nebo hard fork Fitcoinu), za podmínky, že by bylo dodrženo schéma.

Prostředníka by bylo možné implementovat jako libovolnou aplikaci v libovolném jazyce, využívající libovolný způsob sběru dat — za předpokladu, že by tato aplikace byla schopna data zpracovat a předat do databáze s korektním schématem.

Volba databázového stroje by v tomto případě byla omezena jen průnikem technologií, se kterými dokáží pracovat webová část i prostředník.

### 5.1.2 Technologie

Úvaha z úvodu kapitoly se promítá i do volby vhodných technologií. Vývoj monitorovacího systému souběžně s vývojem (i návrhem!) sledované kryptoměny vyžaduje velmi agilní přístup, kterému nejlépe vyhoví moderní, přehledný jazyk a intuitivní, snadno uchopitelné technologie (např. webový framework). Pro využití Fitcoinu při výuce nebo zájmové činnosti je také vhodné zvolit technologie dostatečně rozšířené a dokumentované.

## 5.2 Návrh

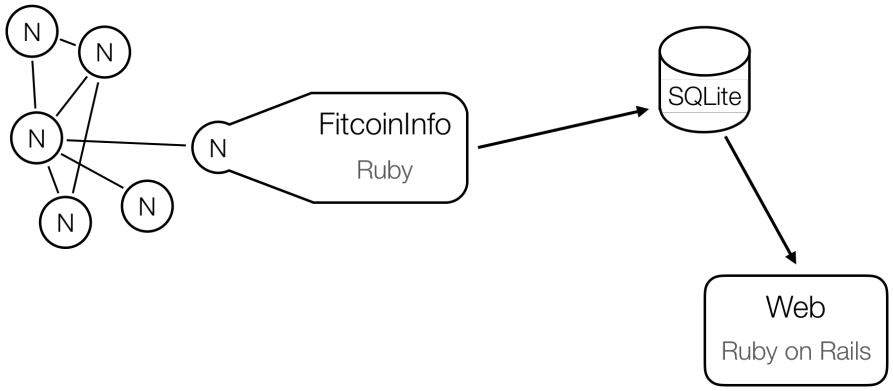
Z řešení nastíněných v předchozí sekci jsem zvolil aplikaci maskující se jako node a využívající síťový protokol. Donutila mě k tomu nedostupnost referenčního nodu již od začátku návrhu a neexistence API, které bych mohl použít.

### 5.2.1 Technologie

Vhodnou volbou pro podobný úkol by se mohl stát jazyk PHP a některý z prověřených webových frameworků. Nutnost programovat zvlášť webovou část aplikace a „síťovou“ část (bude vysvětleno dále) ale favorizuje použití „nativnějšího“ jazyka, například v podobě populárního tandemu Python + Django. Rozhodl jsem se ale použít z mého pohledu elegantnější jazyk Ruby v kombinaci se známým frameworkem *Ruby on Rails*. Pravděpodobně kvůli tomuto názvu je někdy jazyk Ruby k *Rails* přiřazován těsněji, než je ve skutečnosti potřeba — je možné ho využívat k mnoha různým účelům a je v něm napsána celá řada aplikací zcela mimo svět webu, například propracovaný balíčkovací systém *Homebrew* pro macOS.

### 5.2.2 Architektura

Systém bude rozdělen na tři části, jak ukazuje obrázek 5.1.




Obrázek 5.1: Schéma navrhované aplikace. *FitcoinInfo* se bude chovat jako node.

#### 5.2.2.1 Webová aplikace

Webovou část blockchain vieweru bude tvořit aplikace postavená na frameworku *Ruby on Rails*. Ta bude zodpovědná za zobrazování dat uživateli. Aplikace bude sloužit pouze pro zobrazování dat z databáze — nebude schopna v databázi provádět žádné změny. Frontend bude generovat také Rails s využitím vestavěného systému šablon založených na Embedded Ruby a CSS frameworku Bootstrap. Komunikaci s databází bude zajišťovat vestavěné ORM *ActiveRecord*.

#### 5.2.2.2 Prostředník

Data do databáze bude zapisovat oddělená aplikace, jejímž jediným úkolem bude komunikace s nody Fitcoinu. Tato aplikace bude také napsaná v Ruby a využívat *ActiveRecord*, ovšem bez použití Rails.

Aplikace se spustí se známou adresou běžícího nodu. Naváže s ním komunikaci a ohlásí se mu stejným způsobem, jako by sama byla node a měla zájem účastnit se těžby. Následně bude poslouchat a čekat na informace, které se po síti nodů budou všesměrově šířit. Změny bude převádět do formátu používaného v databázi a zapisovat je pomocí ORM.

#### 5.2.2.3 Databáze

Protože není potřeba používat výkonný nebo prostorově úsporný systém, data budou ukládána do databáze SQLite, kterou Rails defaultně používá (primárně pro vývoj).

Kapitola 6

# Realizace

Samotná realizace začala až poměrně pozdě, v okamžiku, kdy bylo alespoň částečně patrné, co vůbec bude potřeba (a možné) vyrobit. Jako první bylo možné začít pracovat na webové prezentaci.

## 6.1 Webová část

Webovou aplikaci jsem se rozhodl implementovat pomocí frameworku Rails, přestože jsem s ním neměl žádné zkušenosti. Zvolil jsem ho, protože programovací jazyk Ruby již nějakou dobu používám jako hlavní skriptovací jazyk a protože se jedná o známý, stabilní framework chlubící se vysokou úrovní pohodlí (*convention over configuration*) a, mimo jiné, šablonovacím systémem Embedded Ruby, který sliboval možnost psát logiku, konfiguraci i renderování dat ve stejném jazyce.

### 6.1.1 Příprava

Rails jsem nainstaloval ve formě gemu (balíčku pro Ruby) z RubyGems. Tento postup mi zajistil automatickou instalaci závislostí bez nutnosti se o cokoliv starat. Pro vývoj jsem zvolil IDE RubyMine od JetBrains, o kterém jsem byl přesvědčený, že mi práci s frameworkem může ulehčit.

### 6.1.2 Kostra projektu

Začal jsem pracovat s prázdným projektem a díky tutorialům postupně zjistil, jakou má aplikace v Rails strukturu, jaké soubory a jaké postupy využívá. Jedná se o klasický webový framework využívající achitekturu MVC. Hlavní slovo tak mají soubory s modelem (`app/models`), pohledem (`app/views`) a controllerem (`app/controllers`).

Ukázalo se, že Rails nezapře své hluboké zakořenění v Ruby světě a celou řadu věcí řeší typickým Ruby způsobem — úplně jinak než ostatní. Například

konfigurační soubory. Část z nich používá — podobně jako jiné frameworky — formát YAML, ale další část konfigurací se píše přímo v Ruby, s využitím oblíbené schopnosti Ruby ohnout se do podoby DSL pro konkrétní úkol. Konfigurace routování pak vypadá třeba takto:

```
Rails.application.routes.draw do
  resources :utxos
  resources :transactions
  resources :blocks
  root 'blocks#index'
end
```

`…draw`, `resources` a `root` jsou v tomto případě překvapivě volání metod, dvojtečky pak neoznačují žádnou příslušnost, ale patří k následujícím slovům — označují symboly (zvláštní datový typ v Ruby hojně využívaný).

Databázové schéma se také nikde nepopisuje v konečné podobě, ale je vytvořené formou migrací (změn) ze zcela prázdné databáze. Migrace jsou opět napsané v Ruby formou DSL, zde například tvorba nové tabulky pro UTXO:

```
class CreateUtxos < ActiveRecord::Migration[5.1]
  def change
    create_table :utxos do |t|
      t.integer :value
      t.string :txhash
      t.integer :txindex
      t.references :created_at
      t.references :spent_at
      t.timestamps
    end
  end
end
```

Název migrace je názvem třídy (`CreateUtxos`), dědící od třídy `Migration`. Tato třída definuje metodu `change` (nevolá, jen definuje), uvnitř které je zavolána metoda `create_table` s názvem tabulky a je jí předán blok se změnami. Sloupce tabulky vznikají voláním metod předaného objektu `t` s názvy odpovídajícími požadovaným datovým typům (`integer`), formou symbolů jsou předávány názvy sloupců ( `:value`). Tato sekvence nemá v jazyce Ruby žádný význam, přijmeme-li ji ale jako DSL, může být poměrně přehledná.

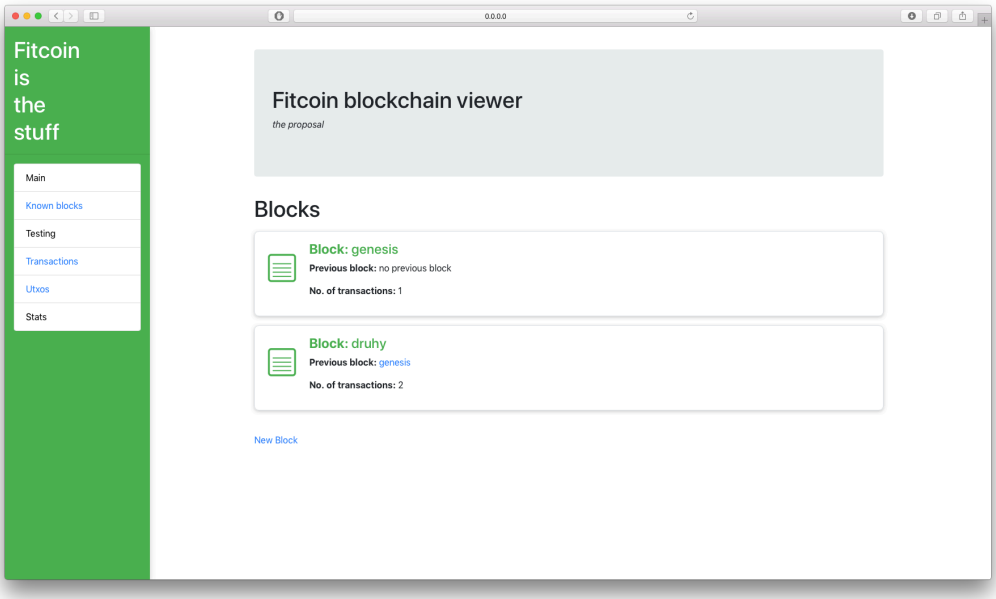
Další věcí, která mě překvapila, byl rozsah zmíněného *convention over configuration*. Například model entity Block vypadá (úplně celý) takto:

```
class Block < ApplicationRecord
  belongs_to :block, optional: true
  has_many :blocks
  has_many :transactions
end
```

Nic dalšího není potřeba. Vazby modelu `Block` jsou popsány formou Ruby metod `belongs_to` a `has_many`, jako symboly jsou předána jména protějšků. Strukturu tabulky má ORM k dispozici v podobě migrace a do modelu ji automaticky doplní formou metod pro přístup ke všem položkám (sloupcům). V kódu je tak možné bez problému zavolat například `block.hashname` (sloupec uchovává hash bloku, `hash` je ale již názvem metody pro hash objektu).

### 6.1.3 Uživatelské rozhraní

Pro zobrazení UI (obr. 6.1) jsem se rozhodl použít framework *Bootstrap*, protože obecně s frontendem nemám téměř žádné zkušenosti a nechtěl jsem mít více starostí, než je nezbytně potřeba.




Obrázek 6.1: Vzhled exploreru

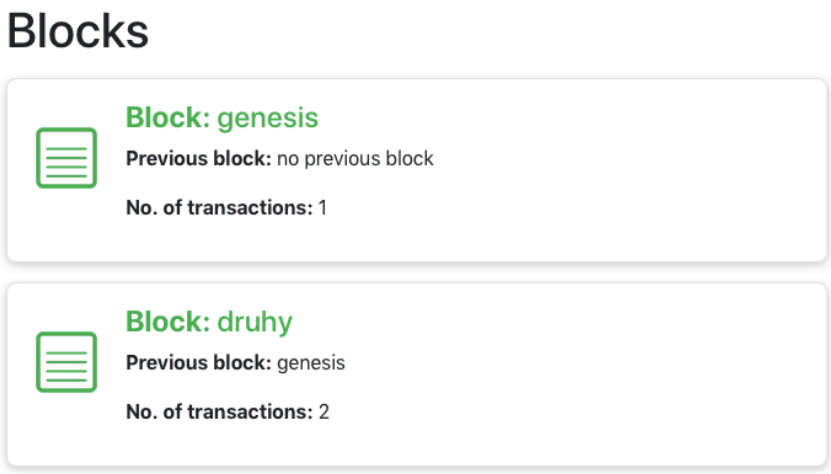
Bootstrap je pro použití v Rails k dispozici ve formě gemu, jehož název tak stačí přidat do `Gemfile` a spustit Bundler příkazem `bundle install` — gem se automaticky nainstaluje pro použití v aplikaci.

Gem jsem nainstaloval a importoval bootstrap přidáním

```
@import "bootstrap";
```

do souboru `app/assets/stylesheets/appliaction.scss`. Rails (i Bootstrap) defaultně využívá preprocesor *Sass*, proto přípona `scss`.

Pro zobrazení bloků, transakcí a vstupů/výstupů jsem si nakreslil jednoduché ikony a používám je jako obrázek v `media` objektu, pomocí kterého je zobrazuji (viz. obrázek 6.2).




Obrázek 6.2: `media` objekty použité pro zobrazení bloku

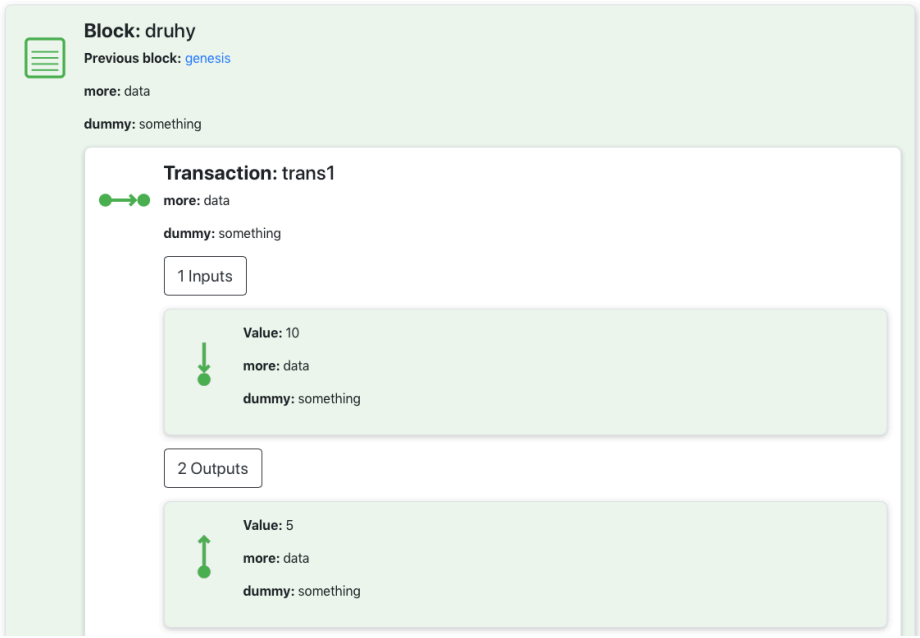
### 6.1.4 Funkcionalita

Nejprve jsem připravil pohledy pro zobrazení bloků, transakcí, UTXO i adres a pro tyto entity měl také připravené modely. Postupně, jak dostával i Fitcoin své obrysy a začalo být patrné, že se v žádném případě nepovede implementovat žádnou pokročilou funkcionalitu, jsem postupně úplně odebral sledování adres (které není technologicky možné), samostatný pohled na UTXO (který nemá sám valný smysl) a po dohodě s vedoucím i pohled na transakce. Aplikace tak zobrazuje jen seznam známých bloků, po rozkliknutí detail bloku včetně vnořeného seznamu transakcí a v nich dále vnořených seznamů vstupů a výstupů, jak ukazuje obrázek 6.3.

Webové rozhraní také v současnosti neumí zobrazit tvar blockchainu, dokonce ani rozlišit mezi bloky na nejdelším řetězu a mimo něj. Proč tomu tak je, bylo vysvětleno v části 3.2.3. Proč to nevadí, bude vysvětleno v části 6.2.4.

### 6.1.5 Databáze

Aplikace čte data z vestavěné databáze SQLite, umístěné v adresáři `db`. Bloky, transakce i UTXO jsou zhruba namodelované a mají definované vazby. Pro-




Obrázek 6.3: Pohled na obsah bloku a jeho první transakce

tože zdroj dat není k dispozici, v souboru `db/seeds.rb` je popsáno několik testovacích objektů — tři bloky, čtyři transakce a sedm UTXO. Příkazem

```
rails db:drop db:create db:migrate db:seed
```

je možné v kterémkoliv okamžiku resetovat databázi a načíst výchozí data.

Aplikace dále obsahuje pohledy i na transakce a UTXO, včetně jejich tvorby a úpravy. Tato funkcionalita samozřejmě v blockchain exploreru nemá co dělat, ale ponechávám ji dostupnou pro testování, než bude možné explorer napojit na Fitcoin.

### 6.1.6 Spuštění

Příkaz

```
bundle install
```

v kořeni aplikace nainstaluje závislosti podle gemfile,

```
rails server
```

potom spustí zabudovaný server. V případě spouštění verze stažené z GitLabu je třeba ještě použít příkaz uvedený v sekci 6.1.5.

## 6.2 Prostředník pro komunikaci

Práci na aplikaci plnící úlohu „prostředníka“ jsem zahájil v okamžiku, kdy bylo patrné, že budu muset sledovat kryptoměnu tvořenou poměrně „hloupými“ nody a využívat tedy postup, který jsem dodatečně poněkud formalizoval v části 3.2.3. Tuto aplikaci jsem si pracovně nazval *FitcoinInfo*.

***Poznámka:*** *FitcoinInfo v této formě nemá budoucnost, jak bude ukázáno později. Některé části (způsob registrace podporovaných zpráv, strukturu tříd, způsob spojení s databází) z něj ale bude možné zachovat a využít.*

### 6.2.1 Forma

*FitcoinInfo* jsem začal psát coby aplikaci v jazyce Ruby, dodržující formu tzv. gemu — balíčku používaného systémem RubyGems, standardním balíčkovacím systémem pro Ruby. Přestože dodržuje správnou strukturu a dá se do formy gemu i zabalit a tímto způsobem distribuovat, rozhodl jsem se *FitcoinInfo* do repozitáře RubyGems nenahrávat. Po instalaci gemu se *FitcoinInfo* začne chovat jako běžná nainstalovaná aplikace, je tedy možné ho spustit příkazem `fitcoininfo`.

### 6.2.2 Struktura

Celý obsah *FitcoinInfo* je zabalený do stejnojmenného modulu. Jeho součástí jsou tři třídy:

**`InfoNode`:** Obsahuje logiku a představuje samotný falešný node.

**`FitcoinConnection`:** Představuje spojení se sítí Fitcoin. Poskytuje následující metody:

- **`initialize`:** Otevře spojení.
- **`get_next_message`:** Vrátí typ přijaté zprávy a data.
- **`dispatch_message`:** Odešle zprávu s daným typem a daty.
- **`close_connection`:** Uzavře spojení.

**`Database`:** Představuje úložiště dat. Poskytuje metody:

- **`initialize`:** Spojí se s databází.
- **`save`:** Uloží objekt.

#### 6.2.2.1 `FitcoinConnection`

Stará se o drobné úkony týkající se síťového protokolu — přijímá a parsuje hlavičky zpráv, validuje je, následně přijímá správnou délku dat a spárovaný typ zprávy a data vrací, nebo naopak zprávu sestavuje a odesílá.

#### 6.2.2.2 `Database`

Metody `initialize` (konstruktor) a `save` jsou jen naznačeny a neobsahují užitečný kód. V metodě `save` bude pravděpodobně probíhat logika týkající se transformace přijatého objektu na odpovídající model. Konstruktor při vytvoření objektu `Database` vypisuje, kolik bloků, transakcí a UTXO v připojené databázi nalezl.

Tuto funkcionalitu lze otestovat — je třeba jen uvnitř souboru `database.rb` přepsat cestu k souboru `.sqlite3` s databází, využívanou webovou aplikací. Zobrazení korektních dat z databáze webové aplikace pak dokazuje, že je ORM *ActiveRecord* správně nakonfigurované a může být tímto způsobem využíváno.

Popis schématu a modely jsou do *FitcoinInfo* jen zkopírované, při finální integraci obou aplikací nebude obtížné zajistit sdílení stejných souborů a konfigurace.

#### 6.2.2.3 `InfoNode`

Představuje centrální bod aplikace a má na starost nejvyšší úroveň logiky — vyřizování zpráv. Objekt `InfoNode` má k dispozici jednu instanci `FitcoinConnection` a jednu instanci `Database`. V okamžiku, kdy celá věc vypadala nejambiciózněji, jsem uvnitř `InfoNode` dokonce udržoval seznam připojení, z nichž každé bylo obsluhováno vlastním vláknem. Později jsem tuto část zase smazal, protože by pro účely Fitcoinu byla příliš komplikovaná.

Snažil jsem se najít vhodný způsob, jak vyřizování zpráv elegantně nakonfigurovat, který by umožnil snadno přidávat nebo odebírat podporované zprávy a dal se například psát do konfiguračního souboru. Způsob podobný řešení, které používá Rails — využití Ruby jako DSL pro konkrétní úlohu.

Vymyslel jsem následující způsob:

```
@supported_incoming_messages = {}

def receive_type(message_type, &handling_block)
  unless block_given?
    raise  'A block has to be given to decide how to handle the message.'
  end
  @supported_incoming_messages[message_type] = handling_block
end
```

`supported_incoming_messages` je objekt typu `Hash` (slovník), který udržuje seznam podporovaných zpráv. Metoda `receive_type` přijme jako argu-

ment typ zprávy a blok, kterým se tato zpráva má zpracovat. Při přijetí zprávy je pak zavolána metoda:

```
def handle_message(message)
  unless handler = @supported_incoming_messages[message[:type]]
    raise "Unsupported message type: #{message[:type]}"
  end

  . . .

  handler.call message[:data], @database
end
```

`handle_message` se pokusí pro zadaný typ zprávy vyhledat odpovídající blok, který by ji zpracoval. Není-li takový blok nalezen, je vyvolána výjimka. Je-li nalezen, metoda ho zavolá a předá mu data přijaté zprávy a přístup do databáze. Tímto je zajištěno, že se kód zpracovávající zprávu dostane jen k datům, která potřebuje.

Registrace nového druhu zprávy a návod k jejímu zpracování pak může vypadat například takto:

```
receive_type :logthensave do |data, database|
  log_or_something data
  database.save data
end
```

Dle mého názoru se jedná o řešení přesně v duchu nejlepších tradic Ruby.

### 6.2.3 Funkcionalita

*FitcoinInfo* nyní obsahuje jen následující testovací výměnu zpráv:

1. *FitcoinInfo* přijme zprávu `getaddr` (neodpovídá)

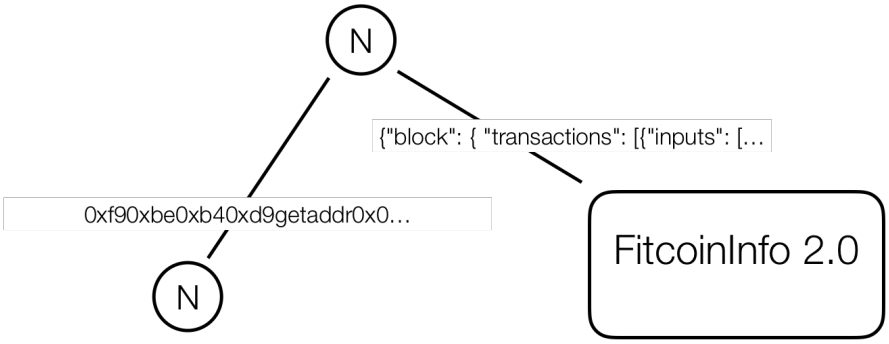
2. *FitcoinInfo* odešle zprávu `getaddr`

3. *FitcoinInfo* přijme odpověď: `addr`

Tuto sekvenci jsem otestoval v krátké době, kdy pracovní verze *fitcoind* běžela. Příjem, parsování, zpracování, sestavení i odeslání zprávy pracovaly zcela korektně.

### 6.2.4 Chybějící logika

V části 6.1.4 jsem uvedl, že exploreru v současné době chybí důležitá funkcionalita — rozeznání validních bloků od nevalidních (bloků na jiné než nejdelší větvi) a obecně znalost tvaru blockchainu. Implementace takové funkcionality by si při zachování této formy řešení vyžádala duplikaci velké části logiky z *fitcoind* uvnitř *FitcoinInfo*.




Obrázek 6.4: Schéma vylepšeného nodu komunikujícího s budoucí verzí *FitcoinInfo*

Po konzultaci s vedoucím a autory Fitcoinu jsem se rozhodl touto cestou nejít a *FitcoinInfo* už nerozvíjet, protože by to představovalo investici spousty času do práce, kterou by nyní nebylo možné spustit ani otestovat, neboť samotná kryptoměna ještě neběží. Autoři Fitcoinu navíc vyjádřili pochopení pro situaci a přislíbili, že Fitcoin nakonec skutečně dostane API, které by umožnilo data pro explorer získávat podobně snadno, jako od nodů Bitcoinu, tedy způsobem, který ukazuje obrázek 6.4. To by znamenalo, že by jakoukoliv další práci v tomto směru bylo brzy možné zahodit.

### 6.2.5 Spuštění

Aplikaci lze zabalit do podoby gemu a následně nainstalovat (není potřeba), případně ihned spustit pomocí souboru v adresáři `exe`. Příkaz

```
bundle install
```

nainstaluje závislosti podle gemspec souboru.

# Závěr

Cíle práce — jak byly původně zadány — se podařilo splnit jen částečně. V průběhu práce se vyskytlo několik komplikací souvisejících se souběžným vývojem Fitcoinu. Výstupem jsou tak — místo dvou spolupracujících aplikací — jedna aplikace s minimální funkcionalitou (webová část) a jedna v současnosti nefunkční aplikace (prostředník).

Naproti tomu v teoretické části se podařilo dostupné informace překvapivě přehledně formalizovat a popsat způsoby, jakými mohou blockchain explorery interagovat s kryptoměnou. Podobný souhrn se mi při psaní práce a shromažďování informací nikde nepodařilo najít, přestože by každému zájemci o implementaci vlastní jednoduché kryptoměny (včetně sledování) mohl ušetřit spoustu času a rozmýšlení.

Ani výstupy praktické části nejsou zcela bezcenné. Webovou aplikaci je možné, kromě náhledu na bloky (který dnes nabízí jako jediný), rozšířit i o náhled na tvar a strukturu blockchainu, budou-li taková data časem k dispozici. Další součástí zadání bylo zobrazování statistik — ani to není možné v tento okamžik implementovat kvůli chybějícím datům z kryptoměny (chybí údaje o validitě bloků), ovšem jeho případné doplnění by bylo jen otázkou zvolení vhodného pohledu na dostupná data.

Prostředníka je také možné zachovat v této podobě. Co je třeba změnit, je způsob získávání dat — místo zneužívání síťového protokolu a implementace vlastní logiky by bylo daleko vhodnější upravit přímo *fitcoind* a využívat jeho API. Vhodné je i zachovat způsob registrace přijímaných i odesílaných zpráv v podobě DSL — umožní to aplikaci snadno upravovat bez zásahů do vnitřního kódu.

Důrazným doporučením do budoucna je pokusit se vyhnout situaci, do které se tato práce dostala — tedy nepokoušet se implementovat zjevně suboptimální řešení jen kvůli minimalizaci zásahů do kryptoměny.

Doporučením pro Fitcoin je pak rozhodně implementace API pro komunikaci s blockchain explorerem. Jakmile bude takové API připravené, bude

dokončení práce započaté touto BP záležitostí pouhého přepsání aplikace pro zisk dat (prostředníka) a zvolení vhodnějšího databázového systému pro uchovávání dat o tvaru a stavu blockchainu. Další alternativou je přechod na některý z databázových systémů orientovaných přímo na grafy, nebo například úplné vyřazení databáze a používání pouze API tak, jak bylo popsáno v části 3.2.2 — „Překladatel“.

Příloha **A**

# Seznam použitých zkratek

**BP:** bakalářská práce

**DSL:** doménově specifický jazyk

**IT:** informační technologie

**MVP:** minimální životaschopný produkt

**ORM:** objektově relační mapování

**RPC:** vzdálené volání procedur

**YAML:** YAML Ain't Markup Language

Příloha B

# Seznam zdrojů

***Poznámka ke zdrojům:*** *Téma kryptoměn bylo ještě nedávno (před obrovským nárůstem hodnoty Bitcoinu na konci roku 2017) okrajovou záležitostí, navíc v komunitě, která si na oficiální publikace příliš nepotrpí. Vycházet tedy mohu především z online zdrojů, především ve formě referenční dokumentace nebo poznámek na online repozitářích.*


1. NAKAMOTO, Satoshi. *Bitcoin: a peer-to-peer electronic cash system*. 2008. Dostupné také z: `http://www.bitcoin.org/bitcoin.pdf`.
2. ANTONOPOULOS, Andreas M. *Mastering Bitcoin: unlocking digital cryptocurrencies*. O'Reilly Media, Inc., 2014.
3. LANSKY, Jan. Possible State Approaches to Cryptocurrencies. *Journal of Systems Integration*. 2018, roč. 9, č. 1, s. 19.
4. BITCOINWIKI. *Address reuse* [online] [cit. 2018-04-18]. Dostupné z: `https://en.bitcoin.it/wiki/Address_reuse`.
5. BITCOIN.COM. *Blockchain size* [online] [cit. 2018-05-10]. Dostupné z: `https://charts.bitcoin.com/chart/blockchain-size`.
6. BITCOINWIKI. *Data directory* [online] [cit. 2018-04-18]. Dostupné z: `https://en.bitcoin.it/wiki/Data_directory`.
7. BITCOINWIKI. *Protocol documentation* [online] [cit. 2018-04-18]. Dostupné z: `https://en.bitcoin.it/wiki/Protocol_documentation`.
8. COINMARKETCAP.COM. *All Cryptocurrencies* [online] [cit. 2018-04-18]. Dostupné z: `https://coinmarketcap.com/all/views/all/`.
9. BITCOIN.ORG. *Bitcoin Developer Reference* [online] [cit. 2018-05-13]. Dostupné z: `https://bitcoin.org/en/developer-reference`.
10. ABE. *README.md* [online] [cit. 2018-04-18]. Dostupné z: `https://github.com/bitcoin-abe/bitcoin-abe`.

11. INSIGHT.IS. *Insight* [online] [cit. 2018-05-11]. Dostupné z: `https://insight.is`.

12. FITCOIN. *Dokumentace protokolu* [online] [cit. 2018-05-13]. Dostupné z: `https://gitlab.fit.cvut.cz/staryja2/fitcoin/wikis/home`.

13. ORMAN, Michał. *Daemons In Rails Environment* [online] [cit. 2018-05-13]. Dostupné z: `http://michalorman.com/2015/03/daemons-in-rails-environment/`.

Příloha C

# Obsah přiložené paměťové karty

***Pozn.:*** *zdrojové kódy aplikací i práce jsou k dispozici na školním GitLabu:*

- https://gitlab.fit.cvut.cz/kaufmros/BP-webovy-nahled-na-blockchain
- https://gitlab.fit.cvut.cz/kaufmros/FITCOIN-FitcoinInfo
- https://gitlab.fit.cvut.cz/kaufmros/FITCOIN-viewer

```
readme.txt ..... popis obsahu paměťové karty
FITCOIN ..... webová aplikace
FitcoinInfo ..... aplikace-prostředník
src ..... adresář s elektronickou verzí BP
    FITthesis.cls ..... použitá LaTeX šablona
    BP_Kaufman_Rostislav_2018.tex ..... text práce ve formátu LaTeX
    BP_Kaufman_Rostislav_2018.pdf ..... text práce ve formátu PDF
    img ..... složka s obrázky
```