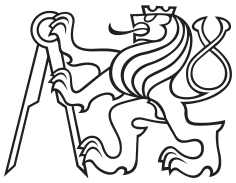
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

RPC Node s implicitní platbou při provádění transakce

Adam Zelfel

Školitel: Ing. Matěj Klíma, Ph.D.
Květen 2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Zelfel** Jméno: **Adam** Osobní číslo: **474545**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

RPC Node s implicitní platbou při provádění transakce

Název bakalářské práce anglicky:

RPC Node with implicit payment during transactions

Pokyny pro vypracování:

Cílem bakalářské práce je vytvořit decentralizovanou službu blockchainového RPC node providera a webovou aplikaci pro správu této služby. RPC Node umožní uživatelům implicitní platbu při provádění transakcí bez nutnosti vytváření speciálního uživatelského účtu a registrace platební karty. RPC Node bude mít standardní dostupnost služby (99,9%). Vzniklá webová aplikace uživatelům umožní
management účtu, možnost předplacení služby a sledování útraty. Bude navržena s využitím programovacího jazyka Solidity pro tvorbu smart contractu. Dále webová aplikace bude vytvořena s použitím frameworku React.js. Webová aplikace bude nasazena na AWS.
V rámci testování systému bude použit Hardhat pro unit testy smart contractu v Solidity. Pro webového rozhraní bude použito Selenium na e2e automatizované testy.

Seznam doporučené literatury:

Rajasekaran, Arun Sekar, Maria Azees, and Fadi Al-Turjman. "A comprehensive survey on blockchain technology." Sustainable Energy Technologies and Assessments 52 (2022): 102039.
Wohrer, Maximilian, and Uwe Zdun. "Smart contracts: security patterns in the ethereum ecosystem and solidity." 2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE). IEEE, 2018.
Ammann, Paul, and Jeff Offutt. Introduction to software testing. Cambridge University Press, 2016.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Matěj Klíma, Ph.D. laboratoř inteligentního testování systémů FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **29.02.2024** Termín odevzdání bakalářské práce: **24.05.2024**

Platnost zadání bakalářské práce: **15.02.2026**

Ing. Matěj Klíma, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. | Datum převzetí zadání

Podpis studenta

Poděkování

Chtěl bych poděkovat panu doktoru Matěji Klímovi za vedení této bakalářské práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Taktéž použití generativních technologií umělé inteligence (dále jen UI) bylo v souladu s metodickým pokynem Rámcová pravidla používání umělé inteligence na ČVUT pro studijní a pedagogické účely v Bc a NM studiu. UI bylo využito ke kontrole gramatiky.

V Praze, 24. ledna 2024

.....
podpis autora práce

Abstrakt

Blockchain a decentralizované aplikace zažívají v poslední době ve společnosti narůstající pozornost. Využití můžeme sledovat napříč mnohými průmyslovými sektory a vznikají i nová odvětví, která by bez blockchainu existovat nemohla. Při budování každé decentralizované nebo hybridní aplikace je zapotřebí mít zajištěnou konektivitu na externí výpočetní technologie, tzv. uzly blockchainu. Zajištění maximální možné konektivity a jednoduchost jejího získání je klíčové pro každou takovou aplikaci.

Cílem práce je popsat aktuální řešení, jeho výhody a nevýhody, a navrhnout alternativní optimalizované řešení. Takové řešení práce detailně popíše z hlediska funkčních a nefunkčních požadavků, návrhu architektury a konečně i implementace. Práce zhodnotí výsledné řešení a poskytne náměty na další optimalizace, která nebudou předmětem jejího rozsahu.

Klíčová slova: RPC Node, blockchain, web3.0, transakce

Školitel: Ing. Matěj Klíma, Ph.D.

Abstract

Blockchain and decentralized applications are receiving increased attention recently. We can track usage of this technology across multiple industries. New industries are emerging that could not exist without blockchain. When building decentralised and hybrid applications, it is necessary to ensure connectivity to external computational units, so-called blockchain nodes. Ensuring the maximum possible connectivity and the ease of obtaining it is the key aspect for every such application.

This thesis aims to describe the existing solution, its advantages and disadvantages and suggest an alternative optimized solution. The thesis will describe this solution in detail from the perspective of functional and non-functional requirements, design of architecture and finally the implementation. Thesis will offer the evaluation of proposed solution and provide ideas for further optimization, which will not be a subject of its topic.

Keywords: RPC Node, blockchain, web3.0, transactions

Title translation: PRC Node with implicit payment

Obsah

Obrázky

Kapitola 1
Úvod

Blockchainový systém je komplexní prostředí a může pro uživatele znamenat četné úskalí při jeho využívání. Je to uzavřený systém mnoha uzlů, které spolu komunikují. Uzly, které tvoří blockchain, jsou systém vzájemně komunikujících výpočetních jednotek. Jejich úkolem je udržovat chod blockchainu a zpracovávat požadavky uživatelů. Aby měl bežný uživatel možnost na blockchainu vykonávat transakce, musí se do tohoto systému připojit. K připojení do systému však nemusí provozovat vlastní uzel. Současně s narůstajícím povědomím a zájmem o blockchain vznikly na trhu služby poskytovatelů připojení k takovému uzlu skrze veřejné rozhraní. Uživateli, který chce tvořit transakce, tak stačí znát URL adresu veřejného rozhraní uzlu. Takový uzel je nazýván Remote Procedure Call uzlem.

Tato práce se zabývá implementací konceptu takové služby s řadou změn, které zlepší uživatelskou zkušenost a zjednodušší využití takové služby. Cílem práce je navrhnout alternativní optimalizované řešení, které zlepší uživatelskou zkušenost a zjednodušší využití služby. Práce se zaměří na návrh a implementaci konceptu takového řešení. Z hlediska metodik návrhu softwarových systémů se zaměří na identifikaci současného problému dostupných variant služby RPC uzlu. Práce dále popíše požadavky na software, který tyto problémy adresuje a popíše implementaci takového softwaru. Výsledným produktem práce je webová služba, která umožní uživatelům jednoduché připojení na blockchainový RPC uzel. Mezi hlavní výhody a inovace služby patří implicitní platba. Ta uživateli zajistí jednodušší práci s RPC uzlem bez nutnosti využívat kreditní kartu a procházet registračním procesem u existujících služeb. Součástí práce je popis architektury software, návod k inicializaci vývojového prostředí a soubor testů pro zajištění kvality a rozšiřitelnosti. V zadání této práce jsou uvedeny technologie, kterými bude cíle práce dosaženo. Vzhledem k technickým omezením Selenia, knihovny pro tvorbu testů, a specifičnosti tvořené decentralizované aplikace, byl po poradě s vedoucím práce zvolen pro automatizované testy nástroj Cypress. Nástroj Cypress plnohodnotně nahrazuje funkci Selenia. Z důvodu omezeného zbývajícího času do odevzdání práce již nebylo možné oficiálně provést úpravu v zadání.

Práce je dále členěna:

Druhá kapitola slouží jako teoretický podklad pro dále probíraná témata. Věnuje se teorii blockchainu a jeho komponent, decentralizovaných aplikací a přiblíží úlohu uzlu v blockchainovém systému.

Třetí kapitola popisuje aktuální varianty obdobných služeb zajišťujících konektivitu na blockchainové nody (RPC Node). Rozbor takových služeb zajistí popis současného stavu a navrhne body, kde může být současný stav vylepšen, které adresují další kapitoly.

Čtvrtá kapitola navrhuje řešení problémů vytyčených v třetí kapitole. Dále poskytuje analýzu a rozbor řešení. V rámci analýzy a rozboru jsou vydefinovány funkční požadavky, nefunkční požadavky a případy užití. V kapitole se nachází návrh architektury systému a je zvolena technologie pro implementační část.

Pátá kapitola se věnuje implementační části práce. Nachází se v ní popis jednotlivých komponent a prostředí, ve kterém byly komponenty tvořeny. Dále lze v páté kapitole nalézt návrh pro další možná rozšíření služby.

Šestá kapitola popisuje implementaci automatizovaných a jednotkových testů systému.

Sedmá, poslední kapitola, shrnuje práci a zhodnocuje splnění požadavků dle zadání.

Kapitola 2
Blockchain a decentralizované aplikace

Kapitola má za účel přiblížit čtenáři problematiku blockchainových technologií a decentralizovaných aplikací. Blockchain je termín, který zahrnuje velikou množinu technologií a jejich variant. V této části práce je představen blockchain v obecném, teoretickém pojetí a některé varianty a komponenty blockchainu pouze obecně představeny či úplně vynechány z důvodu omezeného rozsahu.

2.1 Blockchain

Blockchain je unikátní technologie, která byla poprvé spuštěna v roce 2009. Tento nový koncept byl poprvé zhmotněn osobou zvanou Satoshi Nakamoto. Blockchain se skládá z bloků s narůstajícím počtem záznamů, které jsou spojeny kryptografií, aby odolaly změnám dat. Každý blok v blockchainu obsahuje kryptografický hash předchozího bloku, nový údaj o aktuálním čase a data záznamů. Protože všechny bloky v blockchainu jsou decentralizované a distribuované, jakýkoliv z těchto bloků nemůže být zpětně pozměněn bez vlivu na všechny následující bloky. [1]

Protože se jedná o distribuovaný systém s incentivizací pro zapojení co nejvíce uzlů do sítě, má Bitcoin otevřený a transparentní kód. Toto je důvodem, proč je blockchainová technologie převážně budována komunitou. Aby takový systém velkého počtu zapojených uzlů mohl fungovat, musela vzniknout množina základních pravidel pro obsluhu blockchainu. Tato pravidla jsou v různých variantách blockchainových technologií sdílena a přijímána.

2.1.1 Základní vlastnosti blockchainu

Členové skupiny validátorů, jenž provozují jeho uzly, takto činí s ekonomickými úmysly. Provozování uzlu, též *mining* bloků či validace blockchainu, je odměňováno. Aby uzel obdržel odměnu, musí splňovat určitá kritéria, která jsou rozepsána v této kapitole. Varianta blockchainu bývá navrhována tak, aby zajišťovala bezpečnost a transparentnost. Nové varianty implemenetace převážně vznikají za účelem optimalizace nedokonalostí současných variant či zaměřením se na specifický případ použití této technologie.

2.1.2 Distribuovatelnost

Distribuovaný systém je soubor autonomních výpočetních jednotek, jež se uživateli jeví jako jednotný koherentní systém. [2] V případě blockchainu hraje distribuovatelnost klíčovou roli v zajištění dostatečného počtu kopií informace, aby taková informace byla přijímana jako pravdivá. Jednotliví účastníci (uzly), provádějí revalidaci informace a uchovávají její záznam.

2.1.3 Kategorizace dle přístupu a oprávnění

V každé množine uzlů provozujících jednotný systém je potřeba jednomyslný souhlas na pravidlech provozování sítě. Dle přístupu a oprávnění můžeme základně dělit blockchainy do několika skupin se stejnými rysy.

Veřejný blockchain

Typickým rysem veřejného blockchainu je otevřenost systému. Do veřejného blockchainu se může zapojit téměř každý uzel s rovnocennými právy. Validace či těžba takové sítě býva zpravidla ohodnocena odměnou. Výše odměny je samoregulující a tato odměna je motivací pro potenciální provozovatele uzlu. Faktorů, které ovliňují počet zapojených uzlů, je vícero a liší se dle algoritmu konsensu.

Privátní blockchain

Privátní blockchainy slouží jako interní systémy spolupracujících organizací. Každá organizace v privátním blockchainu musí obdržet oprávnění se do systému připojit. Tyto systémy jsou uzavřené a zpravidla obsahují důvěryhodné informace. [3]

2.1.4 Obecné rozdělení algoritmů konsensu

Konsensy a algoritmy se dělí na kategorie dle mechanismu zabezpečení sítě. Zásadními aspektem bezpečnosti blockchainu je způsob a podmínky, za kterých se může uzel zapojit do sítě. Po zapojení do sítě jsou uzly testovány, zda se v síti chovají dle pravidel. V různých typech algoritmu konsensu se taková situace řeší různými způsoby.

Proof of Work

Proof of Work (PoW) je algoritmus, který byl původně vytvořen již v roce 1993. Jeho cílem bylo bránit útokům denial-of-service. To je útok, kdy se útočník snaží přetížit službu dotazy, aby nebyla schopna dalšího provozu s běžnými uživateli. PoW se stal známý primárně až s vznikem sítě Bitcoin. Zde hraje roli pro uživatele při vytváření nových bloků v blockchainu. [4]

Proof of Stake

Proof of Stake byl představen jako optimalizace Proof of Work algoritmu zejména z důvodu snížení ekonomických a ekologických dopadů těžby. Snížení ekonomických a ekologických dopadů je způsobeno změnou způsobu distribuce odměny za tvorbu bloku. Proof of Work motivuje uzly k rychlé tvorbě bloku. Uzel, kterému se to podaří jako prvnímu, dostává odměnu. To způsobuje zvýšené nároky na kvalitu a výkon výpočetní technologie, která spotřebovává nezanedbatelné množství elektrické energie. V algoritmu Proof of Stake jsou nárok na tvorbu bloku a odměna určeny počtem uzamčených jednotek dané kryptoměny. V případě Proof of Stake musí validátor uzamknout kryptoměnu do tzv. *Staking poolu*. V případě sítě Ethereum se jedná o 32 jednotek etheru. Pokud validátor nedodržuje pravidla pro validaci a je vyřazen z množiny validátorů, toto množství kryptoměny je mu odebráno. [5]

Proof of Authority

Algoritmus Proof of Authority se používá převážně pro uzavřené systémy, kde je účelně omezený vnější přístup k informacím. Sítě s tímto algoritmem konsensu může validovat pouze uzel, který obdržel oprávnění. V takových sítích se obvykle nezavádí odměňovací mechanismus a počet validátorů je velmi omezený. [6]

Ostatní algoritmy konsensu

Vzhledem k neustálým snahám vytvořit optimalizované verze blockchainů, vzniká řada nových či hybridních algoritmů. V současné době největší světová burza Binance zpřístupnila vlastní verzi blockchainu s algoritmem, který nazvala PoSA neboli Proof of Staked Authority. V principu tento algoritmus naplňuje definice PoS i PoA. Validátoři v rámci sítě musí prokázat svout identitu, prokazát se jako důvěryhodná osoba a zároveň uzamknout určené množství kryptoměny do *Staking poolu*. [7]

2.1.5 Veřejné blockchainy a dělení dle vlastností

Tato práce se zaměřuje na práci s komponentou blockchainového systému, jenž se nazývá RPC uzel, a to konkrétně v případu použití pro veřejné blockchainy. Proto se práce zabývá popisem a fungováním systémů veřejných blockchainů.

EVM

V současné době je druhá nejrozšířenější síť Ethereum. Tato síť funguje na základě Ethereum Virtual Machine. Tato technologie je dále v práci referována svou zkratkou EVM. EVM je exekuční model jednoduché zásobníkové architektury, který umožňuje ukládání dat a EVM bajtkódu. EVM bajtkód je sada instrukcí pro EVM. EVM je schopno číst a spouštět bajtkód, který na rozdíl od toho, jak je tomu u bežných exekučních modelů, není ukládán v jednoduše

přístupné paměti, ale separátně v paměti, která umožňuje pouze čtení. EVM kompatibilní sítí je označována síť, která je postavena na architektuře, jenž umožňuje spouštění bajtkódu, který je spustitelný v prostředí EVM. [8]

Sítě první a druhé vrstvy

Sítě první vrsty, tzv. Layer 1 či L1 blockchainy, jsou takové sítě, které nemají tzv. root chain. Root chain je blockchain, do kterého jsou z bezpečnostních důvodů zálohována data jiné sítě. Sítě první vrstvy jsou nezávislé na kterémkoliv jiném blockchainu. Jejich funkčnost závisí pouze na jejích vlastních uzlech. Typickým příkladem takových sítí jsou Bitcoin a Ethereum.

Sítě druhé vstvy, tzv. Layer 2 či L2 blockchainy, vznikají za účelem vyřešení problému přetížení L1 blockchainů. L2 blockchain typicky shlukuje větší počet transakcí, vytvoří jejich zjednodušený zápis a následně v jedné transakci zanese otisk či hash této informace do root chain, což je typicky L1 síť. L2 síť tímto odlehčuje vytíženost L1 sítě. Typickým příkladem L2 sítě jsou Polygon či ZK-Sync blockchainy.

Sítě dle podpory uložení spustitelného kódu

Sítě se dělí na dvě skupiny dle podpory uložení spustitelného kódu. Spustitelný kód, neboli chytrý kontrakt (z anglického výrazu smart contract), je kód vykonávající operace nad informacemi uloženými v blockchainu. Základním využitím chytrého kontraktu je například tvorba vlastního tokenu. Token se využívá pro tvorbu ekonomických systémů, ve kterých lze tokenem například platit za služby, které uživatel nabízí. V takovém systému lze vytvořit další chytré kontrakty, které například přijímají a zpracovávají platbu ve zmíněném tokenu. Nejznámějším příkladem sítě, která nepodporuje chytrý kontrakt, je Bitcoin či jeho L2 síť Lightning Network. Z uživatelského pohledu tyto sítě podporují pouze transakce převodu obnosu kryptoměny z jedné peněženky na druhou.

Dále existují sítě, které uložení a spuštění chytrého kontraktu podporují. Neznámnějším takových příkladem jsou blockchainy Ethereum, Cosmos či Solana. Dále poté tuto funkcionalitu umožňuje celá řada L2 řešení jako například Polygon, ZK-Sync a Arbitrum.

Sítě dle způsobu uložení spustitelného kódu

Síť Bitcoin vznikla za účelem zpřístupnit peer-to-peer obdobu peněz. Pro jiná použití tato síť nemá explicitní podporu. Opakem toho je síť Ethereum, jež byla od prvopočátku stavěna tak, aby podporovala uložení spustitelného kódu.

Sítě, které podporují spustitelný kód, podporují blockchainové transakce,

při kterých se nahrává spustitelný kód, a následně distribuuje do všech uzlů. Takové sítě lze dělit do dvou hlavních skupin.

EVM kompatibilní sítě jsou označovány sítě, jejichž proces předání spustitelného kódu a podpora pro spustitelný kód jsou totožné. Spustitelný kód, v tomto případě chytrý kontrakt, napsaný v kódu Solidity, lze totožným způsobem předat všem EVM kompatibilním sítím stejným způsobem. Solidity je programovací jazyk, který umožňuje kompilaci do EVM bajtkódu. Kód, programovaný v Solidity, se na lokálním zařízení zkompiluje do bajtkódu a spolu se svým aplikačním binárním rozhraním, je nahrán do EVM. Aplikační binární rozhraní (dále ABI) předává uzlu informaci o deklarovaných funkcích a jejich vstupních a výstupních hodnotách.

EVM nekompatibilní sítě jsou označovány všechny sítě, které nejsou EVM kompatibilní. U těchto sítí není zajištěna podpora totožné implementace chytrého kontraktu, protože síť neumí spustit EVM bajtkód. Typickým znakem EVM kompatibilní sítě je, že stačí pouze v konfiguraci systému změnit síť na jinou EVM kompatibilní síť a systém bude fungovat. Dvě různé sítě však nemají rovnocennou historii, takže data a chytré kontrakty je třeba v určitých případech znovu nasadit či přemirovat.

2.1.6 Principy veřejných blockchainů

Transakce a její morfologie

V blockchainových systémech lze nazývat transakci jako uživatelskou operaci. Uživatelskými operacemi je například volání funkce chytrého kontraktu či převod kryptoměny Ether z jednoho účtu na druhý. Po vytvoření transakce uživatelem, je tato transakce poslána do všech uzlů peer-to-peer sítě a vložena do následujícího bloku. Blockchainy, jako jsou Bitcoin a Ethereum, můžeme obecně kategorizovat jako transakční modely zaměřené na fungování s účty. [9] Všechny transakce jsou vykonávany tzv. RPC uzlem. RPC uzel je uzel, jenž má veřejné rozhraní pro přijímání dat transakcí. Základním rozdělením transakcí jsou transakce čtení a transakce zápisu.

Transakce čtení

Transakce čtení je taková transakce, která nemění stav blockchainu, pouze vrací jeho část jako odpověd na dotaz. Aktuální stav blockchainu je vždy konstruován z jeho historických změn zaznamenaných v blocích. Transakci čtení lze vykonat přímo pomocí dotazu na RPC uzel blockchainu. V běžné praxi je získávání stavu blockchainu přes RPC uzel pomalé a využívají se služby, jež indexují blockchain a data ukládají v systémech s rychlejší odezvou. Transakce čtení jsou však z hlediska uzlu nejjednodušší na vykonání, protože nespouští chytré kontrakty a nemění stav. Proto tyto transakce nejsou zpoplatněny.

Transakce zápisu

Na rozdíl od transakce čtení transakce zápisu mění stav blockchainu a mohou spouštět chytré kontrakty. Tyto transakce již vyžadují přístup k RPC uzlu. Bez RPC uzlu není možné transakci propagovat do blockchainu. Životní cyklus transakce zápisu je zásadní proces pro zachování bezpečnosti blockchainu. V první fázi je třeba transakci specifikovat. Specifikace transakce je formulace metadat, která jsou předávána RPC uzlu. Metadata jsou prostředek k předání kontextu transakce. Příkladem jsou informace o tom, jaká funkce chytrého kontraktu je volána a s jakými argumenty. RPC uzel dále zanese transakci do transaction poolu. V této fázi je transakce též prvotně zvalidována a následně propagována dalším uzlům. Transkace je finálně potvrzena a zanesena do bloku, jakmile určitý počet uzlů provede validační proces.

Transakce zápisu má své náležitosti, bez kterých nemůže být přijata. Kromě přesné specifikace ve formátu, který je uveden v následujícím seznamu, vyžaduje zprocesování transakce poplatek (anglicky gas).

- **Nonce** — Číselná hodnota typu uint64, používaná k přidělení číselného pořadí transakce. Toto číslo musí být vždy o jedna větší, než u předchozí transakce.
- **Gas price** — Využívá se k výpočtu omezení poplatku za transakci. Hodnota gas price vynásobená hodnotou gas je maximální poplatek za transakci, kterou je uživatel ochoten zaplatit. Je určena v jednotkách Wei, což je zlomek etheru určen jednoduchým vztahem.

  $$ETH = 10^{18} WEI$$

  [10]
- **Gas** — Počet jednotek gas, který je konzumován uzlem.
- **To** — Účet příjemce transakce. Pokud je uživatelem volán chytrý kontrakt, tato položka může zůstat prázdná.
- **Value** — Počet etheru převáděných na jiný účet v etheru.
- **Input** — Obsahuje data předávaná chytrému kontraktu a jeho adresu v bajtkódu. [11]
- **R, S, V** — Kontrolní hodnoty ECDSA algoritmu pro účely ověření podpisu transakce.

Transakční poplatek

Transakční poplatek (z angl. gas fee) vznikl za účelem kompenzace nákladů výpočetního výkonu uzlu. Transakce čtení ze své povahy nejsou v porovnání s transakcemi zápisu zatěžující pro blockchainovou síť, protože se nepropagují

do transaction poolu, a proto zpoplatněny nejsou. U transakcí zápisu dochází k výrazně vetší konzumaci výpočetního výkonu na všech uzlech. Transakční poplatek vznikl i za účelem obrany proti nebezpečnému přetěžování sítě či DoS útokům.

Výpočet hodnoty gas fee je závislý na využitých funkcích a jejich konzumace výpočetního výkonu a samotným aktuálním vytížením sítě. Každá funkce, v závislosti na tom, zda pracuje s pamětí či využívá nejednotně složité operace, bude konzumovat různé množství poplatku. Nejjednodušší způsob zjištění ceny za spuštění funkcí je empirická analýza jednotlivých průchodů transakce, například použitím veřejně dostupného nástroje VMTrace. Výsledky dvou průchodů podobných transakcí se však budou měnit s aktuálním vytížením sítě. [12]

EVM účet

Účet na Etherum blockchainu je pár veřejného a privátního klíče. Veřejný klíč je ekvivalentem adresy Ethereového účtu. Privátní klíč je využíván k podpisu transakcí.

EVM účty lze rozdělit do dvou kategorií. Externě vlastněný účet, neboli EOA, je účet, který vznikl činem uživatele a má svůj privátní a veřejný klíč. Takový účet je využíván k tvorbě transakcí uživatelem. Druhým typem účtu je účet pro chytré kontrakty. Na rozdíl od EOA účtu, k tomuto účtu nevzniká privátní klíč a je určen k identifikaci daného chytrého kontraktu. [11] Pro zajištění bezpečnosti práce s párem klíčů vznikly služby kryptoměnových peněženek. Jejich účelem je vést uživatele procesem tvorby, zálohování a sdílení zásadních informací, které jsou potřeba k obnovení ztraceného přístupu k účtu a vykonávání transakcí. Příkladem takové peněženky je služba MetaMask. [13]

RPC uzel

RPC (z angl. remote procedure call), neboli vzdálené volání procedur, je metoda, která umožňuje spouštění kódu, umístěném na jiném zařízení, než je volající program. V kontextu blockchainových technologií slouží RPC uzel jako brána pro zpracování transakcí. Další kapitoly práce tuto komponentu popisují ve větším detailu.

2.2 Decentralizované aplikace

Decentralizované aplikace jsou systémy, které využívají chytré kontrakty na blockchainu. Neznámějším příkladem je decentralizovaná směnárna, tzv. DEX (z angl. decentralized exchange). Na nich je umožněno provádět směny kryptoměn dle trhem určeného kurzu. Význačnou vlastností decentralizovaných směnáren je, že transakce jsou zprocesovány v blockchainu bez zásahu člověka

či externí výpočetní technologie. Zároveň v ověřené směnárně není možné, aby uživatel v rámci směny poskytl kryptoměnu, která je předmětem směny, a neobržel ekvivalent její hodnoty v požadované měně. Taková situace by mohla nastat, pokud by nesplnil základní bezpečnostní doporučení, jako je například kontrola logiky kódu veřejně dostupných chytrých kontraktů užívaných v procesu směny. Chytrý kontrakt v decentralizovaných systémech zajišťuje bezpečnost, transparentnost informací, procedur a funkcí obsluhujících procesy.

2.2.1 Architektura decentralizovaných aplikací

Popis architektury decentralizovaných aplikací spočívá v jejím rozdělení do několika vrstev. Tyto vrstvy jsou rozděleny podlé svého účelu.

Procesní vrstva

V decentralizovaných systémech hraje významnou roli v uchovávání dat. Tato data jsou ve veřejných blockchainech dostupná veřejnosti a lze tak kontrolovat jejich změny. Decentralizované systémy také využívají přenesení části logiky systému do blockchainu pomocí implementace chytrých kontraktů. Je dobrým zvykem tvůrců chytrých kontraktů tyto kontrakty zvěřejňovat a zajistit tak transparenci celého kódu. Lze tedy říci, že uživatel má jistotu, že logika systému a pravidla určená tvůrcem chytrého kontraktu se v průběhu času nemohou změnit.

Aplikační vrstva

Tato vrstva se týká uživatelského rozhraní, které uživatel používá k interakci s blockchainem. Dále vrsta umožňuje abstrahovat pro uživatele složité koncepty kryptografie, například práci s privátním klíčem. Toto je realizováno pomocí knihoven, jako jsou například knihovny Wagmi či Ethers.js. Jsou to obalovače API rozhraní RPC uzlů a mají za úkol předat RPC uzlu potřebná data ke zprocesování transakce. Kromě toho poskytují jednoduchý způsob práce s podepisováním transakcí ECDSA algoritmem. Než je transakce podepsána, knihovna umožnuje jednoduše vytvořit Keccak-256 hash, který je v procesu vyžadován. Podepsaná transakce je nadále poslána RPC uzlu, jenž transakci zvaliduje a zařazuje do transaction pool. Knihovny a jejich funkce jsou popsány v sekci 2.2.3.

Komunikační vrstva

Zajišťuje veškerou komunikaci mezi libovolnou komponentou a procesní vrstvou, tedy blockchainem. V síti ethereum je tato komunikace zajištěna JSON-RPC službou, která funguje na protokolu HTTP.

Decentralizované úložiště

Decentralizované úložiště, jako je například služba IPFS, podporuje ukládání statických dat či souborů. Narozdíl od blockchainu však není možné data

a soubory dynamicky měnit. Je vhodné využívat decentralizované úložiště pro neměnná data vetšího rozsahu, jejichž uložení přímo v procesní vrstvě na blockchainu by bylo nákladné kvůli vysokému transakčnímu poplatku. [14]

2.2.2 Chytré kontrakty

Chytrý kontrakt je automatizovatelná a vymahatelná úmluva. Ačkoliv je automatizovatelná pomocí počítače, některé součásti úmluvy vyžadují lidský vstup a ovládání ze strany člověka. Chytrý kontrakt je vymahatelná úmluva na základě zákonů o vymáhání práv a povinností nebo na základě vykonání počítačového kódu, jehož nelze manipulovat. [15]

V prostředí EVM je chytrý kontrakt reprezentován bajtkódem. Chytrý kontrakt lze implementovat v jazycích, které lze zkompilovat do tohoto bajtkódu. Nejčastěji využívané jazyky pro implementaci chytrého kontraktu jsou programovací jazyky Solidity a Vyper.

2.2.3 Knihovny pro integraci s webovým rozhraním

Jsou to open-source knihovny, které vznikly, aby uživateli zjednodušily implementaci v aplikační a komunikační vrstvě. Mezi příklady takových knihoven se řadí již zmíněné javascriptové knihovny Wagmi či Ethers.js. V jazyce Python mají tyto knihovnu svou obdobu, která se jmenuje web3.py. Na straně aplikační vrstvy uživateli zpravidla pomáhají s

- načítáním funkcí chytrého kontraktu,
- prováděním transakcí čtení,
- prováděním transakcí zápisu,
- formulací transakce zápisu a jejím podpisem,
- správou a využíváním dostupných účtů,
- komunikací s RPC uzlem,
- a dalšími operacemi.

2.3 Blockchainový uzel

Blockchainový uzel je základním stavebním kamenem decentralizovaných ekosystémů. Jeho úkolem je uchovávat historii informací, zajišťovat dodržování pravidel nastavených v síti a vytvářet nové bloky. Tato kapitola představí základní kategorizaci a konkrétní implementace uzlů.

2.3.1 Kategorizace uzlů

Blockchainové uzly lze kategorizovat dle množství dat, která jsou v delším časovém horizontu lokálně uchovávána dostupnými. Dalším posuzovanou vlastností uzlu je, zda se může či nemůže účastnit validačního procesu.

- **Full node** — Je uzel, který zajišťuje primární chod blockchainu validací. Nehledě na to, ve kterém bloku se plný uzel zapojil do validace, jeho povinností není uchovávat veškeré historicky zvalidované bloky. Pro účely šetření paměti si plný uzel ponechává pouze relativně krátkou historii bloků, typicky alespoň posledních 128 bloků.
- **Archive node** — Funguje na stejném principu jako plný uzel, ale jeho účelem je uchovávat kompletní historii dat blockchainu od prvního bloku, který se nazývá genesis blok. Tento typ uzlu je typicky využíván prohledávacími a analytickými službami.
- **Light node** — Neuchovává celé objemy dat bloků, ale pouze jejich hlavičky. Hlavičky bloků uchovávají pouze přehled informací v něm uložených. Pokud lehký uzel potřebuje informaci, kterou nemá dostupnou, doptává se plných uzlů. Spuštění takového uzlu bývá nejjednodušším způsobem pro uživatele, jak získat lokální přístup do blockchainu. Tato kategorie uzlů neumožňuje participaci na validaci bloků. [16]

2.3.2 Konkrétní implementace blockchainových klientů

Blockchainový klient implementuje protokol, který popisuje, jak jsou bloky ověřovány, odsouhlaseny a propagovány skrze peer-to-peer síť. [17] Právě proto, že veřejný blockchain je peer-to-peer síť, do které se za předpokladu doržování pravidel může připojit jakýkoliv klient, konkrétní implementace uzlu není vynucena. Z tohoto důvodu existuje na trhu řada implementací uzlů pro EVM blockchainy. Mezi nejvyužívanější řešení patří například tyto implementace.

- Geth
- Nethermind
- Hyperledger Besu
- Erigon

Kapitola 3

Současné RPC služby

Kapitola vytyčuje současná dostupná řešení RPC služeb a popisuje problémy, které jsou adresovány v návrhu decentralizované služby poskytovatele RPC uzlu s implicitní platbou.

3.1 Úvod do dostupných RPC služeb

Aktuální dostupná řešení RPC služeb můžeme dělit na dva typy, které se odlišují svou dostupností věřejnosti.

3.1.1 Veřejné RPC uzly

Veřejné uzly jsou provozovány úzkým okruhem lidí, typicky úvodních tvůrců a vývojářů, kolem daného blockchainu. Tato skupina bývá označována jako foundation. Jejich motivace k provozu takového uzlu je zpřístupnění blockchainu co největšímu počtu uživatelů a zajištění jednoduché konektivity. Kategorie veřejných RPC uzlů je význačná vícero charakteristickými znaky.

Autentizační proces

Veřejné RPC uzly nemají autentifikační proces. Uzel má vystavené veřejné rozhraní, které není ničím podmíněno. Při volání služby není potřeba získat autentifikační token a dotaz na uzel je tak velice snadný.

Cena

Vzhledem k tomu, že při přístupu k uzlu není vyžadována autentifikace, nelze při dotazování na uzel sledovat, jaký uživatel se k uzlů připojuje. Neexistuje možnost, jak tento typ dotazu zpoplatnit.

Limity požadavků

Zásadní nevýhodou takového uzlu je neschopnost odbavit větší počet požadavků. Provozovatelé uzlu vytváří limity dle svého uvážení. Zpravidla limitují počet transakcí z jedné IP adresy. Tento limit bývá ve využívaných systémech nedostačující. Limitování dostupnosti má jednoduché odůvodnění. Cena za

provozování uzlu není zanedbatelná vzhledem k požadavkům na hardware a infrastrukturu.

Garance dostupnosti

Vzhledem k tomu, že služba není placená, nebývá garantována dostupnost. V praxi je tak běžné, že se veřejný uzel odpojí a není dostupný.

Využití

Primární využití veřejných RPC uzlů je pro implementaci decentralizovaných aplikací a běžné využívání blockchainu. Průměrný uživatel zvládne vzhledem k časové náročnosti zpracování transakce pouze velmi omezený počet dotazů. Konkrétní využití je například v blockchainových peněženkách, kde musí uživatel zadat webovou adresu rozhraní RPC uzlu, aby mohl vykonávat transakce. Stejně tak se veřejné RPC uzly používají v implementaci na klientovi decentralizované aplikace. Kód klienta je veřejný a nelze zajistit bezpečnou autentifikaci z jeho strany.

3.1.2 Privátní RPC uzly

Privátní RPC uzly, na rozdíl od veřejných, jsou provozovány společnostmi za účelem zisku. Provozovatelů takové služby je na trhu nespočetné množství, proto je v následující části uveden pouze reprezentativní vzorek.

Autentifikační proces

Privátní RPC uzly vždy vyžadují autentifikaci uživatele. Každý požadavek na uzel je zaznamenáván poskytovatelem služby.

Cena

Ceny za službu se liší dle byznys modelů konkrétních společností. Lze ale odhadnout, že ceny vždy reflektují náklady na infrastrukturu, výpočetní výkon, cenu využívaného hardwaru a udržování softwaru. Služby nabízejí vícero plánů pro uživatele dle intenzity a počtu zasílaných požadavků.

Limity požadavků

V porovnání s veřejnými RPC uzly jsou limity požadavků výrazně vyšší, protože je za ně poskytovateli placeno.

Garance dostupnosti

I v případě služeb privátních RPC uzlů garance nebývá smluvně zajištěna. V mnohých případech nelze předpokládat stoprocentní dostupnost uzlů. Vzhledem k tomu, že jsou služby placené a poskytovatel je motivován si platící klienty udržet, bývá většinou dostupnost zajištěna v maximální možné míře.

Využití

Privátního přístupu k RPC uzlu je využíváno subjekty, které nejsou technicky vybaveny zajistit si svůj vlastní RPC uzel či to pro ně ekonomicky není výhodné. Typickým příkladem jsou služby, které uživatelům záměrně spravují jejich peněženky. Toto je motivováno ochranou uživatele před ztrátou přístupu k peněžence a zjednodušením procesu vykovávání transakcí. Takové služby jsou například kryptoměnové směnárny či blochchainové hry.

3.2 Problém současných služeb

Tato část práce se zaměřuje na služby poskytující privátní přístup k RPC uzlu. Vzhledem k veliké početnosti takových služeb na trhu kapitola adresuje pouze ty nejvyužívanější. Dále jsou zde popsané problémy aktuálně existujících služeb.

3.2.1 Proces zajištění přístupu ke službě a následné obsluhy

Proces zajištění přístupu ke službě a následné obsluhy obsahuje tři základní problematické případy užití a to sice registraci, platby a omezené plány. Problémy v těchto případech užití jsou dále rozepsány.

Registrace

Prvním identifikovaným problémem je registrace. Blockchainová komunita je význačná svoji potřebou zůstat anonymní. Registrací do RPC služby jsou data o transakcích a využívání blockchainu sdílena s poskytovatelem a přímo spojena se subjektem, který ji využívá. S registrací je také spojeno riziko ztráty přihlašovacích údajů a tudíž případné ohrožení provozu systému.

Platba

Druhým identifikovaným problémem je proces platby. Současná řešení implementují klasické měsíční a roční plány a uživatel je nucen využít k zaplacení kreditní kartu. V produkčním provozu je vývojář povinen sledovat, zda je služba zaplacena, a implementovat kontrolní mechanismus integrací API rozhraní služby třetí strany.

Omezené plány

Třetím identifikovaným problémem jsou cenové plány. Omezení počtu transakcí v jednotlivých plánech znamená ohrožení provozu aplikace, protože uživateli služba nedovolí dále vykonávat transakce. Zároveň pokud uživatel nevyužije přidělený počet dotazů, zvyšuje se mu průměrná cena za jeden dotaz. Tyto tři problémy jsou nevyhnutelné při zajišťování možnosti vykonávat veliký počet dotazů na službu poskytovatele veřejného přístupu k RPC uzlu.

Služby	Registrace	Platba	Plány
QuickNode	Ano	Ano	Ano
Infura	Ano	Ano	Ano
Tatum	Ano	Ano	Ano
Moralis	Ano	Ano	Ano

Tabulka 3.1: Vybraná současná řešení a přítomnost problému

3.2.2 Zhodnocení aktuálních řešení

Pro účely zhodnocení a potvrzení existujících problémů byly vybrány čtyři služby a to sice QuickNode, Infura, Tatum a Moralis. V tabulce 3.1 je vyhodnoceno, zda daná služba disponuje zmíněnými problémy.

Kapitola 4

Návrh řešení RPC služby

Navrhované řešení má za cíl adresovat a eliminovat problémy vytyčené v třetí kapitole. Tato kapitola zajišťuje stručný popis principů, kterými tohoto cíle navrhované řešení dosahuje. Finální a podrobný návrh aplikace je v práci rozveden v další kapitole. Současně je zde proveden výběr vhodné technologie pro implementační část práce.

4.1 Stručný popis navrhovaného řešení

Klíč k optimalizaci procesu získání přístupu a následné obsluhy je v přesunu části funkcionalit do blockchainového chytrého kontraktu. Navrhované řešení RPC uzlu s implicitní platbou řeší všechny tři vytyčené problémy v předchozí kapitole následujícím způsobem.

Registrace

V blockchainových ekosystémech je uživatel reprezentován svým účtem, tedy peněženkou. Vytvoření peněženky je pro vývojáře blockchainových ekosystémů výrazně jednodušší, než výběr a registrace služby třetí strany. Peněženka zároveň není nosičem osobní informace, jako je například e-mail, telefon či adresa uživatele. V navrhovaném řešení proběhne proces obdobný registraci výrazně rychleji a to pouhým připojením peněženky k aplikaci.

Platba

Vývojáři na blockchainu mnohem častěji přijdou do kontaktu s platbou v kryptoměně než s platbou v tzv. fiatové oficiálně uznávané měně. Taková platba je vykonána bez potřeby zadávání údajů jako je tomu u platby platební kartou. Zároveň je pro vývojáře komfortnější sledovat platby za službu přímo na blockchainu a to z důvodu, že nemusí implementovat rozhraní třetí strany. Pracuje pouze se standardizovaným API existujících knihoven.

Omezené plány

V navrhovaném řešení je omezenost transakcí vyřešena částečně. Uživatel má možnost si službu předplatit a tím vzniká riziko, že mu nedostatečný zůstatek

na účtu znepřístupní službu. Řešením tohoto problému je implicitní platba. Při použití implicitní platby uživatel s každou transakcí převádí poplatek na účet poskytovatele, čímž si zajišťuje nekonečnou dostupnost služby.

4.2 Analýza požadavků

V této sekci je dostupný rozbor z hlediska funkčních a nefunkčních požadavků na systém. Dělí skupinu uživatelů na tři typy. Dále sekce poskytuje soupis funkčních požadavků dle typu uživatele.

4.2.1 Nefunkční požadavky

Nefunkční požadavky na vytvořený software jsou popsány v následujících bodech.

Podpora na zařízeních

Pro účel vytvoření prototypu softwaru je vyžadováno, aby klientská aplikace podporovala webový prohlížeč Chrome verze 124 a vyšší pro operační systém macOS.

Uživatelská zkušenost

Klientská aplikace zajišťuje standardní použitelnost. Aplikace nedostane uživatele do nedefinovaného chování, jež znemožní její další používání.

Bezpečnost

Aplikace nevystaví zkušeného uživatele blockchainových aplikací do rizika přímé ztráty jeho prostředků.

4.2.2 Funkční požadavky

V definici funkčních požadavků jsou požadavky rozděleny na tři skupiny dle typu uživatele.

Nepřihlášený uživatel

Nepřihlášený uživatel má možnost:

- přihlásit se do služby.

Přihlášený uživatel

Přihlášený uživatel má možnost:

- zobrazit informaci o aktuálním zůstatku na službě,
- zobrazit informaci o celkovém počtu utracených prostředků,

- zobrazit informaci o celkovém počtu vytvořených transakcí,
- zobrazit přehled využívání aplikace za posledních 60 dnů,
- předplatit službu,
- vybrat nespotřebovaný zůstatek na službě,
- zobrazit a prohlížet historii svých transakcí předplacení či výběru,
- vykonávat jednotlivé transakce.

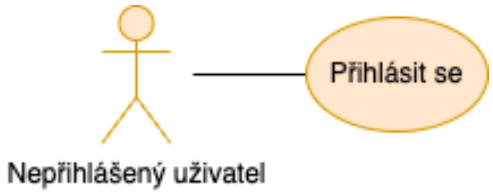
Majitel chytrého kontraktu

Majitel má stejné možnosti jako přihlášený uživatel, k tomu však navíc:

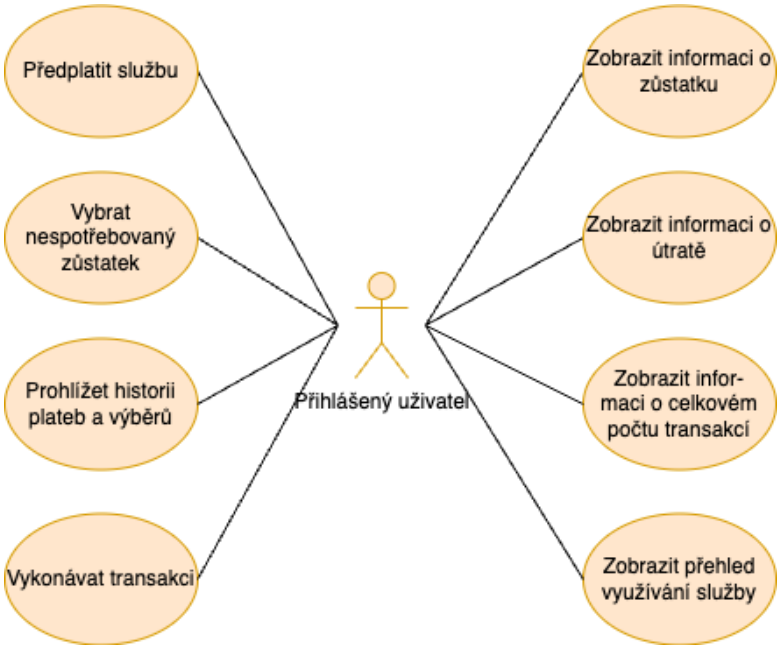
- vybrat z chytrého kontraktu platby za službu.

4.2.3 Případy užití

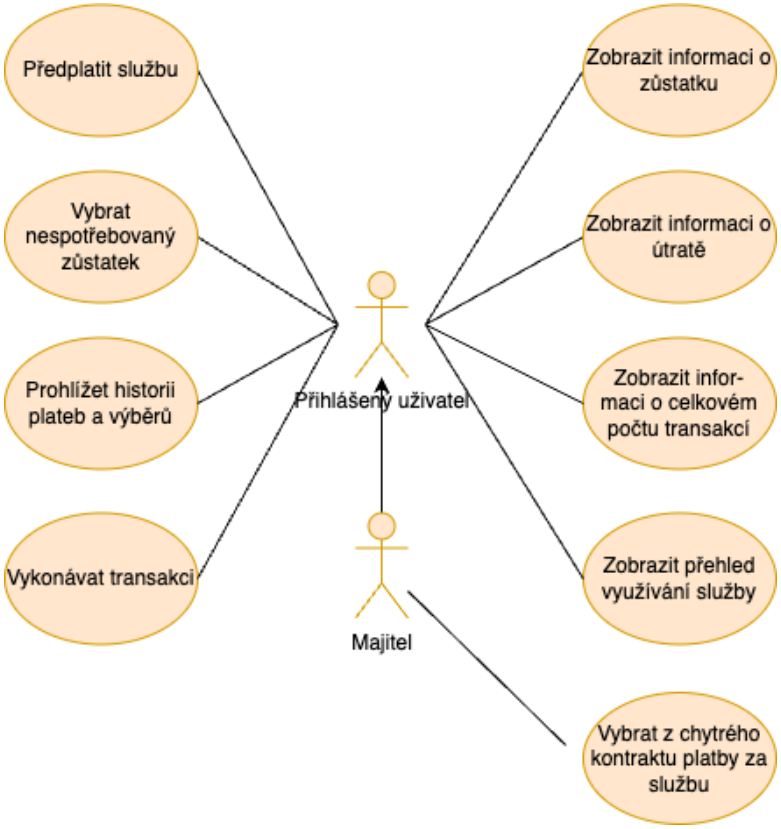
Podle definovaných funkčních požadavků jsou vytvořeny diagramy popisující případy užití pro jednotlivé uživatele. V diagramu 4.1 lze vidět jednoduchý diagram nepřihlášeného uživatele s jediným případem užití a tím je přihlášení. V diagramu 4.2 lze najít případy užití dle funkčních požadavků pro přihlášeného uživatele. Diagram 4.3 zobrazuje dědičnost majitele chytrého kontraktu od přihlášeného uživatele. Navíc má majitel možnost výběru plateb za službu.



Obrázek 4.1: Případy užití nepřihlášeného uživatele.



Obrázek 4.2: Případy užití přihlášeného uživatele.



Obrázek 4.3: Případy užití majitele chytrého kontraktu.

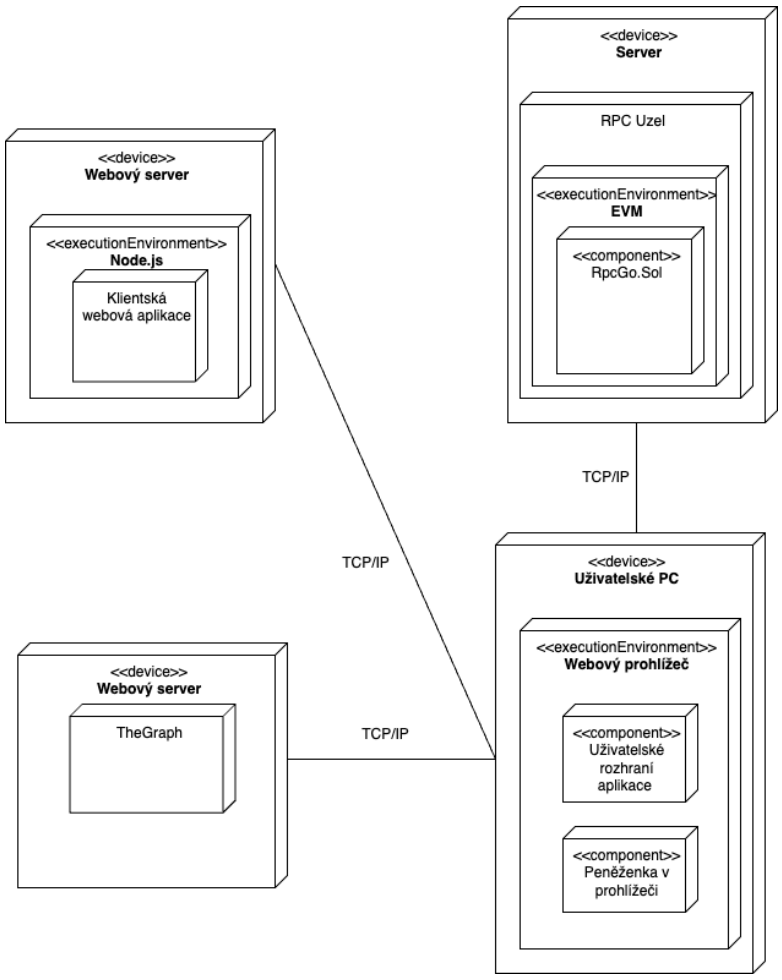
4.3 Architektura aplikace

Decentralizované aplikace zpravidla komunikují s externími službami. Pro zajištění minimální funkčnosti decentralizovaných aplikací je třeba zajistit komunikaci s jedním či více blockchainy, RPC uzly a indexovacími službami blockchainových dat. Dále pak komunikují s vrstvami, které zajišťují připojení kryptoměnových peněženek a podepisování transakcí uživateli. Protože taková architektura není triviální, je třeba vytvořit její dokumentaci, která umožní pochopit uživateli vazby mezi komponentami. Pro popsání těchto důležitých informací byl zvolen diagram nasazení. Tato sekce nabízí diagram nasazení, ve kterém jsou uvedeny i služby, které jsou externí a tedy již nasazené.

4.3.1 Diagram nasazení

Diagram nasazení 4.4 ukazuje na komponenty, které jsou stěžejní pro fungování celého systému. Mezi ně se řadí i blockchain a služba TheGraph. Služba TheGraph indexuje data z blockchainu, která lze pak jejím prostřednictvím jednoduše získat, viz kapitola 5.4. U decentralizované aplikace, narozdíl od klasických aplikací, lze vidět, že komunikace probíhá primárně mezi klientem a dalšími komponentami.

V diagramu lze vidět čtyři hlavní zařízení a prostředí. Máme zde dvě zařízení webového serveru. První z nich je webový server, na kterém je nasazena klientská webová aplikace. Jejím účelem je zpřístupnit klientovi uživatelské rozhraní. Uživatelské rozhraní lze vidět na zařízení uživatelského PC. Na uživatelském PC se využívá prostředí prohlížeče. V prohlížeči má uživatel možnost kromě uživatelského rozhraní také interagovat s peněženkou, která je nutná při obsluze aplikace. Druhý webový server je externí a je v něm nasazena aplikace TheGraph. Toto zařízení je v režii poskytovatele služby TheGraph. Pro fungování aplikace je však nutné. Posledním zařízením je server, na kterém je nasazen uzel blockchainu. Veškeré EVM kompatibilní blockchainy jsou provozovány z uzlu, který umí spustit prostředí EVM. V prostředí EVM jsou následně spouštěny funkce chytrých kontraktů.



Obrázek 4.4: Diagram nasazení.

4.4 Výběr technologie

Výběr technologie je důležitou přípravnou součástí v návrhu softwaru. Tato kapitola se věnuje zvážení a výběru vhodné technologie pro implementaci optimalizovaného řešení služby poskytovatele RPC uzlu.

4.4.1 Obecné předpoklady vyžadované technologie

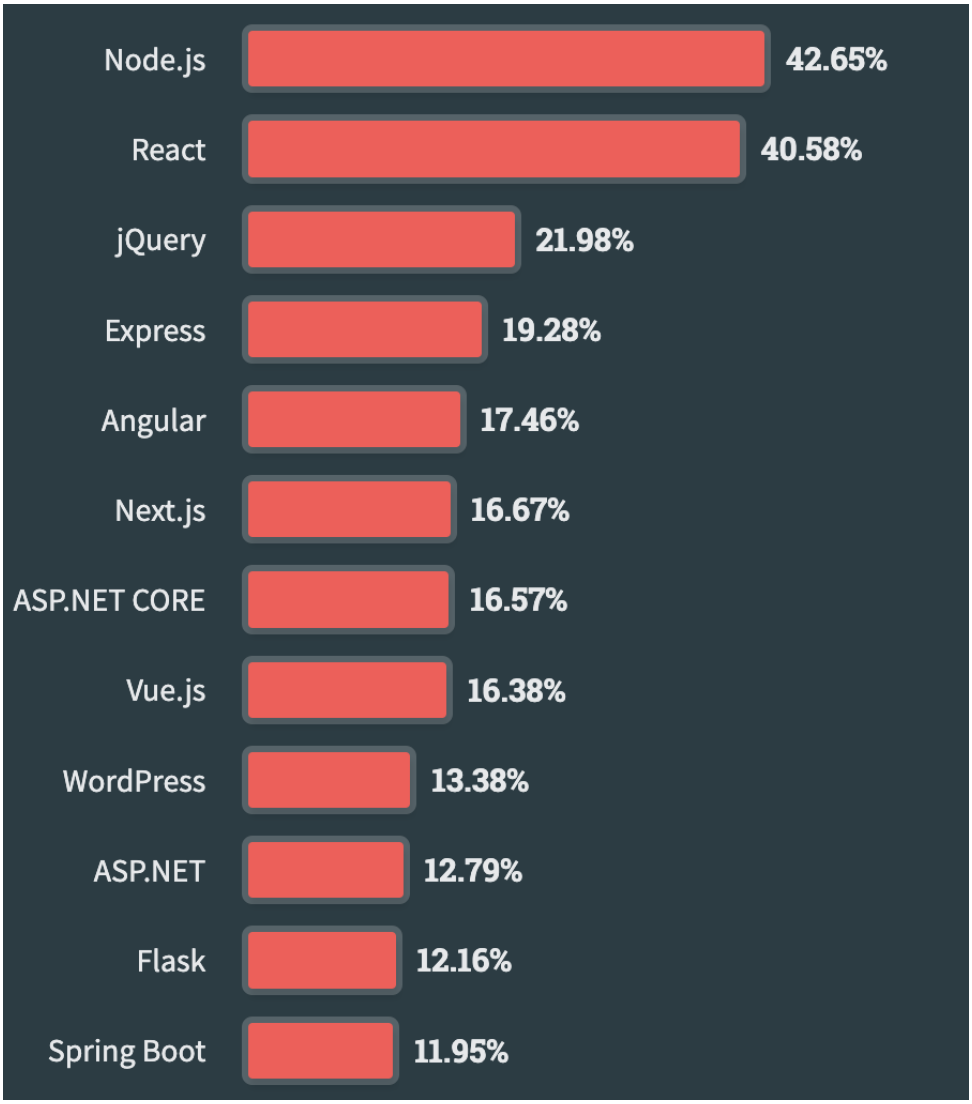
Vzhledem k povaze decentralizovaných aplikací je zde potřeba navrhnout specifickou architekturu. Nabízí se zde využít dobře známý koncept klient-server aplikace. V kontextu decentralizovaných aplikací server reprezentuje blockchain a klienta webová aplikace komunikující s blockchainem. Vetšinu decentralizovaných aplikací lze kategorizovat jako MVC architekturu, přičemž model a view jsou reprezentovány chytrými kontrakty. Často se však lze v praxi lze setkat s hybridními aplikacemi, kde je vedle této sestavy vytvořen i backend aplikace obsluhující logiku, jež by bylo nevýhodné stavět na blockchainu. Pro účely této práce dodatečnou backendovou část aplikace pomineme. Software

prezentující optimalizaci lze postavit s využitím pouze blockchainu a klientské aplikace.

4.4.2 Technologie pro implementaci klientské aplikace

Pro výběr vhodného frameworku pro implementaci klientské aplikace je důležité zvážit několik faktorů. Vhledem ke specifikcké povaze decentralizovaných aplikací je kritickým bodem uvažovat pouze frameworky, které mají dostupné knihovny pro integraci blockchainu. Taková knihovna jednoduše umožňuje připojení blockchainové peněženky, tvorbu a procesování transakcí. V současné době nejrozšířenějšími takovými knihovnami jsou javascriptové knihovny Ether.js a Web3.js. Z tohoto důvodu bude třeba zahrnout do úvahy javascriptové frameworky pro tvorbu webových rozhraní. Dalším důležitým bodem pro výběr vhodného frameworku je bezpečnost. Velikost komunity udržující framework je v tomto bodu kritická. Čím větší je podpora komunita udržující framework, tím méně chyb a lepší kvalitu můžeme očekávat. Pro účely škálovatelnosti je taktéž důležité volit framework, který je široce využíván, a tudíž lze na trhu práce jednoduše najít kompetentní vývojáře.

Z obrázku 4.5 je patrné, že nejvíce využívanými frameworky jsou Node.js a React.js. Protože pro tvorbu decentralizované aplikace není stěžejní implementace backendu, zvoleným frameworkem pro tvorbu webového klienta je framework React.js.



Obrázek 4.5: Data o využívanosti webových frameworků. [18]

4.4.3 Použité technologie pro implementaci chytrých kontraktů

Programovací jazyky, které lze využít pro implementaci chytrých kontraktů, mají v EVM prostředí jednu podmínku, a tou je možnost kompilace do bajtkódu spustitelného v EVM. Dva nejpoužívanější jazyky, které toto splňují, jsou Vyper a Solidity. Vzhledem k dostupným implementacím standardů chytrých kontraktů a širší podporou, ze strany komunity vývojářů, je zde logickou volbou programovací jazyk Solidity.

Solidity je programovací jazyk vyšší úrovně a je orientovaný na implementaci chytrých kontraktů. Jeho tvorba byla ovlivněna jazyky C++, Python a JavaScript. Jazyk je navržen tak, aby v něm bylo možné vyvíjet kód s cílem spuštění v EVM. Je to jazyk staticky typovaný, podporuje dědičnost a tvorbu

knihoven a komplexních typů definovaných uživatelem. [19]

Dále je pro tvorbu chytrých kontraktů využíván framework Hardhat. Tento framework zajišťuje uživateli pohodlnou práci při tvorbě, testování a nasazení chytrých kontraktů do EVM. Hardhat byl zvolen z důvodu široké podpory v komunitě vývojářů.

4.4.4 Technologie pro testování

Důležitou součástí tvorby software je zajištění testů, které hrají důležitou roli ve vývojovém cyklu. Pro účely aplikace, jež je popisována v rámci této práce, je třeba zajistit jednotkové testy chytrých kontraktů. Testování frontendové aplikace bude zajištěno automatizovanými testy, které jsou schopné ovládat uživatelské rozhraní aplikace obdobným způsobem jako sám uživatel.

Jednotkové testy chytrých kontraktů

Pro účel testování chytrých kontraktů byl využit framework Hardhat, který byl zároveň zvolen pro implementaci. Hardhat umožňuje tvorbu jednotkových testů, které mají za účel kontrolovat správné vykonání jednotlivých funkcí na úrovni chytrého kontraktu.

E2E testy frontendové aplikace

Pro tvorbu E2E testů aplikace byl zvolen framework Cypress v rozšíření zvaném Synpress. Cypress je automatizační nástroj, který obstarává testování webových aplikací. Hlavní výhodou Cypress oproti jiným testovacím nástrojům, jako je například Selenium, je automatické vyčkání na dostupnost webových komponent v prohlížeči. [20] Framework Cypress v rozšíření Synpress byl zvolen, protože jako jediná dostupná technologie podporuje pohodlné testování s využitím kryptoměnové peněženky MetaMask.

Kapitola 5

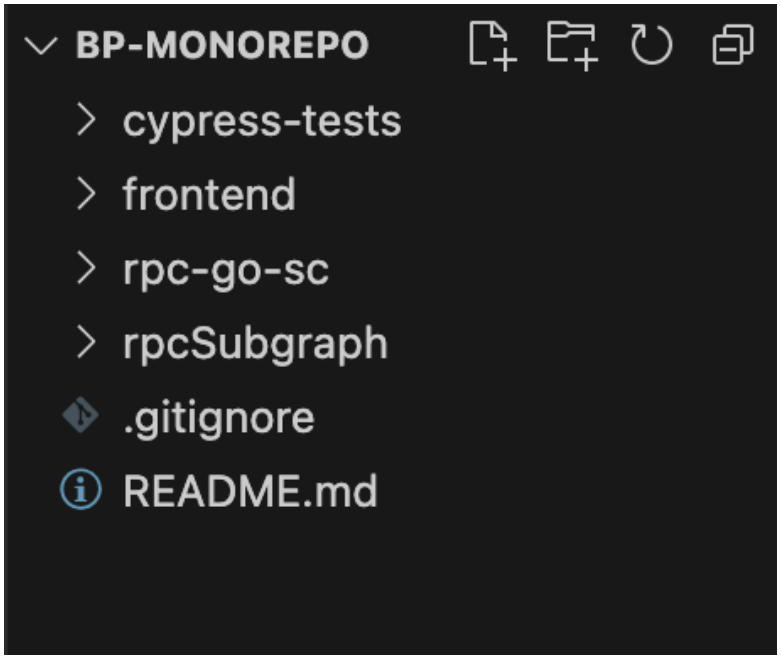
Implementace řešení RPC služby

Kapitola obsahuje konkrétní rozbor kódu a implementovaného řešení. Závěrem kapitoly je provedeno zhodnocení celého řešení. Současně jsou zde navržena rozšíření. Je zde uvedena další přidružená problematika, jež nebyla součástí rozsahu této práce.

5.1 Vývojové prostředí

Softwarové řešení je implementováno na lokálním počítači v rozhraní Visual Studio Code. Verzování systému je zajištěno pomoci GITu v cloudové službě GitHub.

Pro uchování kódu bylo zvolen monorepozitář s názvem implicit-payment-rpc-node. Jeho strukturu lze vidět na obrázku 5.1. Tento repozitář obsahuje veškerý kód, který je potřeba ke spuštění aplikace v lokálním prostředí.



Obrázek 5.1: Struktura monorepozitáře.

Implementovaný kód ve všech složkách využívá javascriptový framework Node.js.

5.1.1 Inicializace a správa modulu frontend

Složka frontend obsahuje klientskou typescriptovou aplikaci implementovanou pomocí frameworku React.js. Pro instalaci potřebných knihoven je nutné spustit příkaz *pnpm install*. Spustění lokálního testovacího serveru je možné za použití příkazu *pnpm run dev*.

5.1.2 Inicializace a správa modulu rpc-go-sc

Složka rpc-go-sc obsahuje implementaci chytrého kontraktu RpcGo.Sol a ERC20.Sol. Pro instalaci potřebných knihoven je nutné spustit příkaz *pnpm install*. Jako prostředí, ve kterém jsou chytré kontrakty implementované, byl zvolen framework Hardhat. Hardhat dále podporuje řadu potřebných příkazů.

- *npx hardhat compile*

  Tento příkaz kompiluje vytvořené soubory typu .Sol ve složce contracts do bajtkódu spustitelném v EVM.

- *npx hardhat test*

  Příkaz spouští testy ze složky tests implementované v typescriptu.

- *npx hardhat run scripts/deploy.ts –network holesky*

  Příkaz spouští skript deploy.ts ze složky scripts. Tímto příkazem se spustí nasazení chytrých kontraktů do blockchainové sítě. V tomto příkladu používame testovací blockchainovou síť holesky.

- *npx hardhat verify 0xC3Bf5ba7874FA863794B427DEef0ec866a492fBe –network holesky*

  Příkaz ověřuje chytrý kontrakt v blockchainovém prohlížeči. Díky ověření kontraktu si může kdokoliv veřejně ověřit kód chytrého kontraktu a spouštět jeho funkce přímo v prohlížeči. V příkladu je využité reálné nasazení chytrého kontraktu RpcGo.Sol na síti holesky.

5.1.3 Inicializace a správa modulu rpcSubgraph

Složka rpc-go-sc obsahuje konfigurační rozhraní služby TheGraph. Instalaci potřebných knihoven lze spustit příkazem *npm install*. Před použitím dalších potřebných příkazů je nutné nainstalovat službu graph-cli pomocí příkazu *npm install -g @graphprotocol/graph-cli*. Graph-cli dále podporuje řadu potřebných příkazů.

- *graph auth –studio <token>*

  Příkaz zajistí autentifikaci uživatele do služby TheGraph, která je vyžadována pro nasazení potřebných komponent.

- *graph codegen && graph build*

  Pomocí tohoto příkazu lze spustit generování a build typescriptového kódu, který je vyžadován pro nasazení.

- *graph deploy –studio holeskyrpcgo*

  Tento příkaz spouští nasazení do služby TheGraph a vystavuje veřejné rozhraní pro dotazování.

5.1.4 Inicializace a správa modulu cypress-tests

Složka cypress-tests obsahuje implementaci Cypress testů v rozšíření Synpress. Před použitím příkazu pro spuštění testů je třeba nainstalovat potřebné knihovny. Toto lze dosáhnout spuštěním příkazu *npm install* v kořenové složce cypress-tests. Spuštění testů poté lze vykonat příkazem *npm synpress:run*.

5.2 Frontend

Pro tvorbu frontendu implementovaného řešení byl zvolen javascriptový framework React.js. Pro zajištění kontroly nad kódem a vynucení typů je kód aplikace implementovaná v typescriptu.

5.2.1 Zvolený typ aplikace

Frontendová aplikace je implementována jako SPA (z angl. Single Page Application). SPA je aplikace, která se jako celek spouští na klientovi, typicky v prohlížeči. Na rozdíl od SSR (z angl. Server Side Rendering), kde se jednotlivé strany aplikace generují na serveru, se zde veškeré komponenty tvoří přímo v prohlížeči.

5.2.2 Zdrojové kódy

Adresářová struktura je v tomto nastavení důležitá. Framework je původně nastavený k vyhledávání komponent ve složce pages. Každá taková komponenta je poté dostupná na adrese dle rozřazení a názvu v této složce. React.js je framework postavený na komponentách, které se vnořují a tvoří tak stromovou strukturu. Rodičovské komponenty svým potomkům předávají hodnoty, se kterými potomkové následně pracují.

Struktura adresáře je rozdělena tak, aby poskytovala čtenáři přehlednost a současně čitelnost kódu.

Adresář pages

V adresáři pages se nachází pět hlavních komponent, které aplikace využívá jako stránky.

- **login** — Zde se nachází implementace přihlašovací logiky do rozhraní služby.
- **dashboard** — Tato komponenta obsahuje implementaci rozhraní přehledu dat náležících k účtu.
- **payments** — V této komponentě je dostupná implementace výběrů a plateb za službu. Zároveň komponenta poskytuje přehled všech historických transakcí.
- **transaction** — Komponenta obsahuje implementaci volání chytrého kontraktu třetí strany skrze službu.

Adresář components

Adresář components obsahuje komponenty využívané v hlavních komponentách, jež reprezentují jednotlivé stránky webové aplikace. Tyto komponenty

jsou importovány a následně využívány v jednotlivých komponentách v adresáři pages.

Adresář abi

Obsahuje ABI chytrých kontraktů RpcGo.Sol a ERC20.Sol. ABI jsou následně importována do jednotlivých komponent a využívána knihovnou Wagmi.

Adresář utils

V adresáři utils se nachází implementace funkcí, které jsou v aplikaci využívány ve vícero komponentách. Tyto funkce byly extrahovány, aby byla zajištěna větší přehlednost a lepší kvalita kódu implementovaného v jednotlivých komponentách.

5.2.3 Využité knihovny

V této části jsou popsány důležité knihovny, které byly využity pro tvorbu webové aplikace.

- **Next.js**

  Knihovna Next.js umožňuje širokou škálu nastavení frameworku aplikace. Mimo jiné umožňuje také rozhraní pro tvorbu aplikace implementované pomocí React.js.

- **React.js**

  Pro implementaci webové aplikace byl zvolen framework React.js. V aplikaci jsou využívány tři knihovny zajišťující základní funkčnost. Knihovna *react* umožňuje využívat značkovací strukturu JSX. Značkovací struktura podporuje tvorbu kódu podobného HTML, jež je vracen jako návratová hodnota komponentami. Značkovací struktura povoluje vkládání proměnných přímo do JSX struktury. Knihovna nadále umožňuje použití tzv. háčků (z angl. Hooks). Základním takovým háčkem je například funkce *useState*. Funkce zaručuje reakci komponent na změnu hodnoty proměnné. Nová hodnota se následně propíše i do grafického rozhraní komponenty. Dále je v aplikaci využita knihovna *react-router-dom*, která zajištuje podporu stránkování v prohlížeči. Aplikace při změně stránky nedotazuje server, aby obdržela nový zdrojový kód. Po změně stránky sama v browseru vytvoří instance nových komponent. Příklady využití knihoven a jejich funkcí lze vidět na obrázcích 5.2 a 5.3.

```
<Box sx={{ textAlign: 'left' }}>
  <Typography variant="body1" sx={{ fontSize: '35px', fontWeight: 'bold' }}>
    {number}
  </Typography>

  <Ty (parameter) description: string ntSize: '20px' }}>
    {description}
  </Typography>
</Box>
```

Obrázek 5.2: Vložení hodnoty proměnné do značkové struktury JSX.

```
const TransactionForm: React.FC = () => {
  const [formData, setFormData] = useState<{ [key: string]: string }>({});
  const [value, setValue] = useState<string>('');
  const [contractAddress, setContractAddress] = useState<string>('');
  const [functionDeclaration, setFunctionDeclaration] = useState<string>('');
  const [args, setArgs] = useState<string[]>([]);
  const [modalOpen, setModalOpen] = useState<boolean>(false);
  const [verificationPassed, setVerificationPassed] = useState<boolean>(false);
  const { writeContractAsync } = useWriteContract();
```

Obrázek 5.3: Použití funkce useState.

- **MUI a Emotion**

  Material UI je populární knihovna komponent pro React.js. Komponenty jsou stylizované do materiálního designu. Materiální design je standard pro tvorbu grafických prvků a komponent od společnosti Google. Knihovna poskytuje celou řadu API pro manipulaci s komponentami a s grafickými úpravami komponent. Emotion je knihovna umožňující psaní stylů přímo v komponentách aplikace.

- **GraphQL**

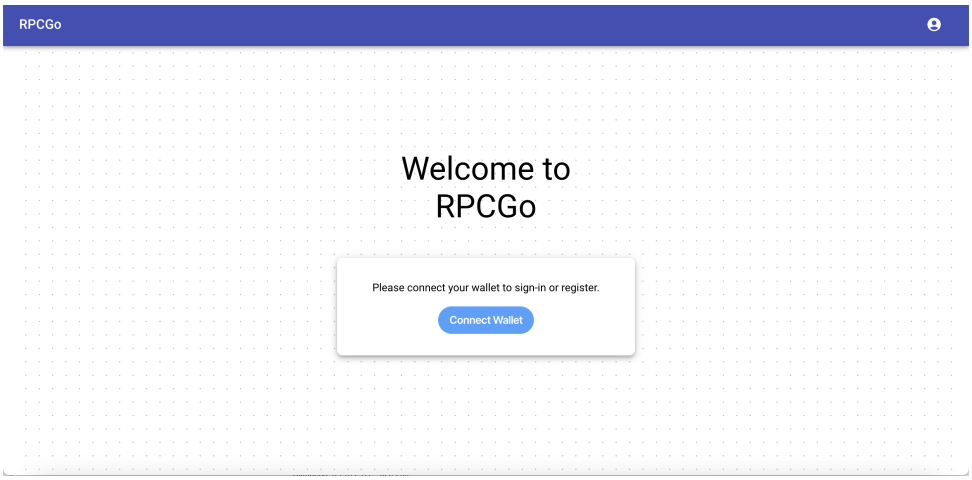
  GraphQL je dotazovací jazyk, s jehož pomocí je realizována komunikace se službou TheGraph. Je to způsob jakým lze získavat data ze serveru. Uživatel si může přesně nadefinovat data, která vyžaduje. Využitím GraphQL tak lze získat minimální potřebné informace, což snižuje objem přenášených dat. [21]

5.2.4 Uživatelské rozhraní klientské aplikace

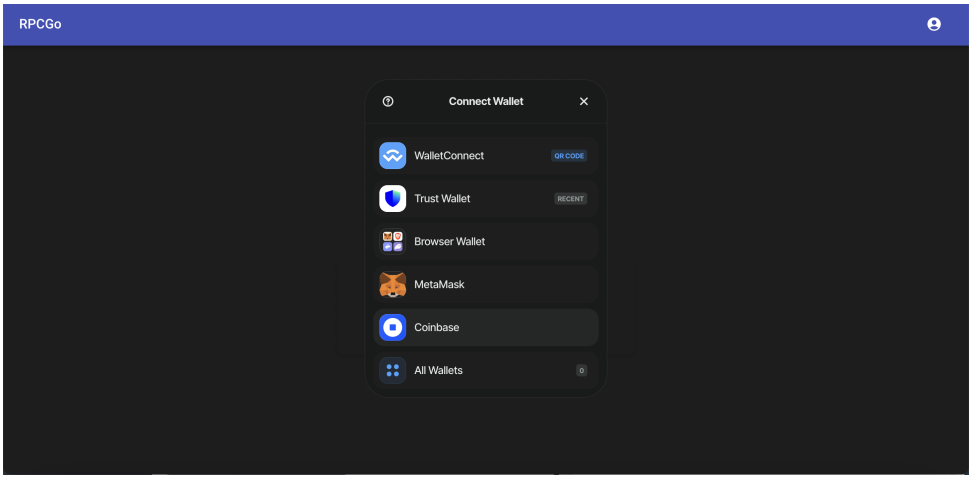
Klientská aplikace má jednoduché uživatelské rozhraní, které zaručuje používatelnost služby pro člověka, jenž má alespoň základní zkušenost s používáním kryptoměnových peněženek. Rozhraní je členěno do několika částí, které splňují požadavky dle návrhu aplikace ve čtvrté kapitole.

Přihlášení

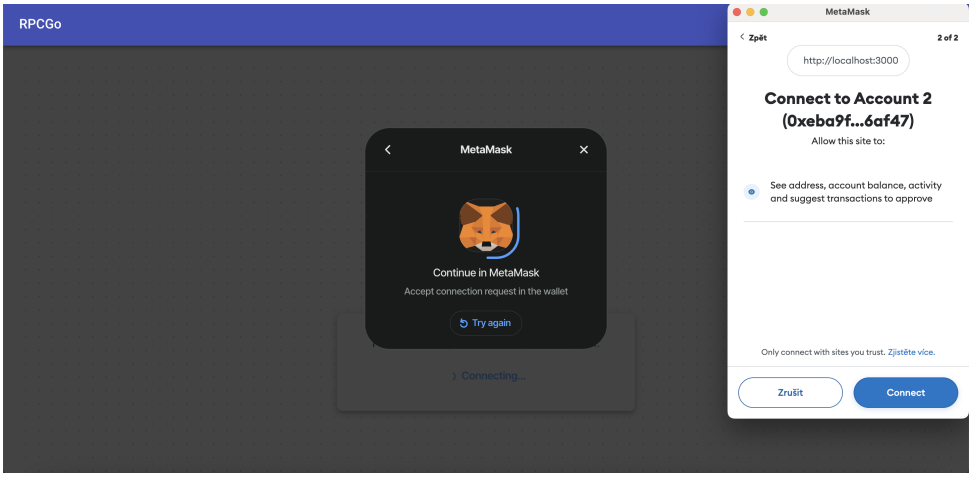
Na dostupné cestě aplikace /login se zobrazí okno pro přihlášení do aplikace. Okno je možné vidět na obrázku číslo 5.4. Uživatel se může přihlásit do aplikace pomocí kryptoměnové peněženky. Základní takovou podporovanou peněženkou je MetaMask. MetaMask lze používat jako rozšíření prohlížeče Chrome či jako mobilní aplikace. Uživatel si připojí peněženku stisknutím tlačítka Connect Wallet a vybráním MetaMasku či jiné příslušné peněženky. Výběr peněženek je vidět na obrázku číslo 5.5. MetaMask ve verzi prohlížečového rozšíření vyvolá pro každou akci vyskakovací okénko, ve kterém uživatel může tuto akci potvrdit (viz obrázek číslo 5.6). Po potvrzení akce je uživatel přihlášen do rozhraní služby.



Obrázek 5.4: Přihlašovací rozhraní do služby.



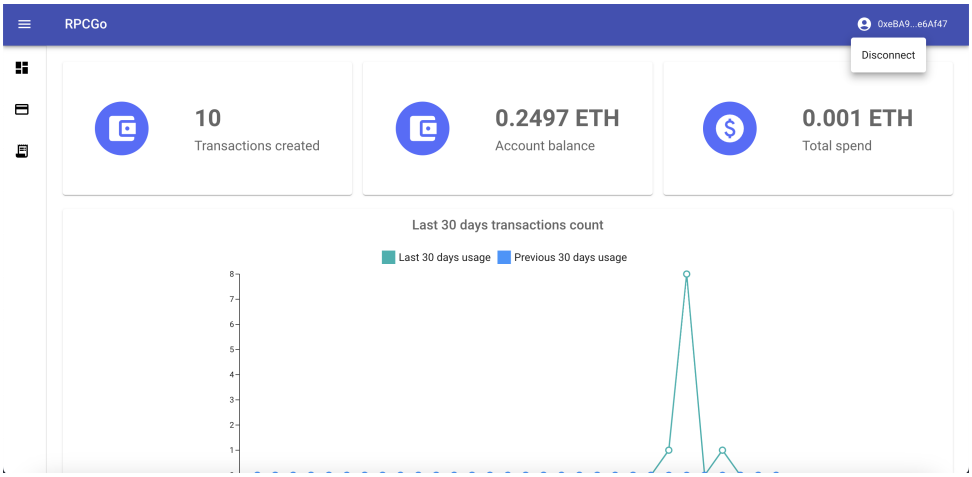
Obrázek 5.5: Volba peněženky, se kterou se uživatel připojí do služby.



Obrázek 5.6: Potvrzení spojení na straně MetaMasku.

Odhlášení

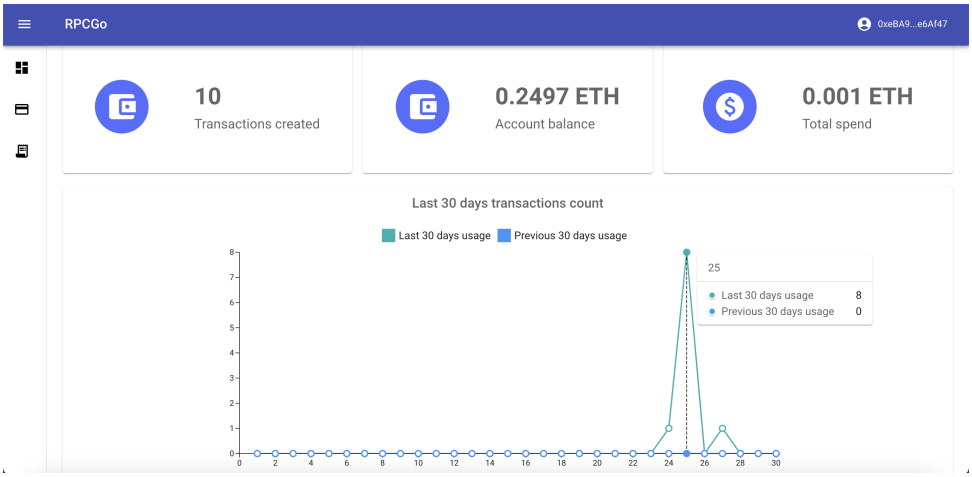
Pro odhlášení ze služby stačí kliknout v pravém horním rohu na ikonu profilu a následně kliknout na tlačítko disconnect. Tlačítko lze vidět na obrázku číslo 5.7. Odhlášením se aplikace vrátí do původního stavu, tedy stavu před přihlášením. Uživatel se může dále přihlásit i s jinou peněženkou.



Obrázek 5.7: Odhlášení ze služby.

Dashboard

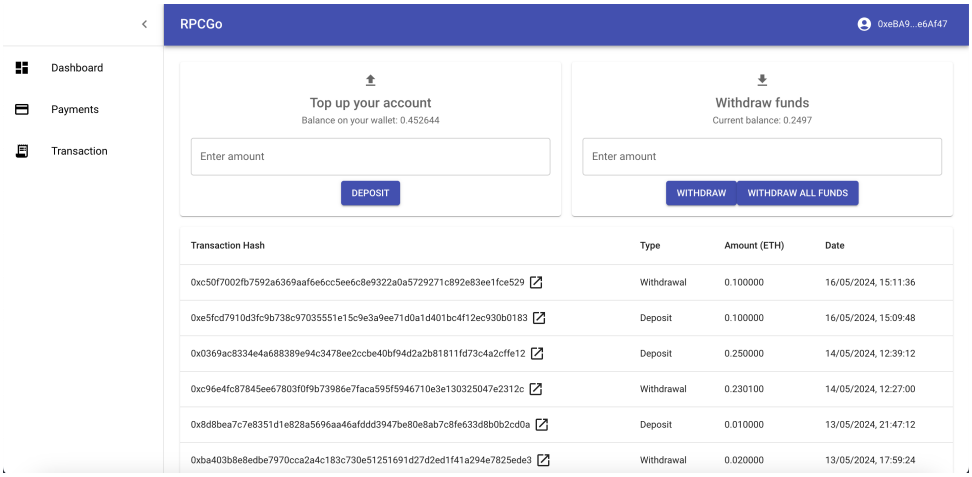
Po přihlášení je uživatel nasměrován na hlavní stránku dostupnou na cestě /dashboard. Tato stránka poskytuje uživateli přehled o počtu vykonaných transakcí v rámci využívání služby, aktuálním zůstatku na jeho účtu ve službě, celkové zaplacené částce za službu a využívání služby v posledních třiceti dnech a v předchozích třiceti dnech. Stránku je možné vidět na obrázku 5.8.



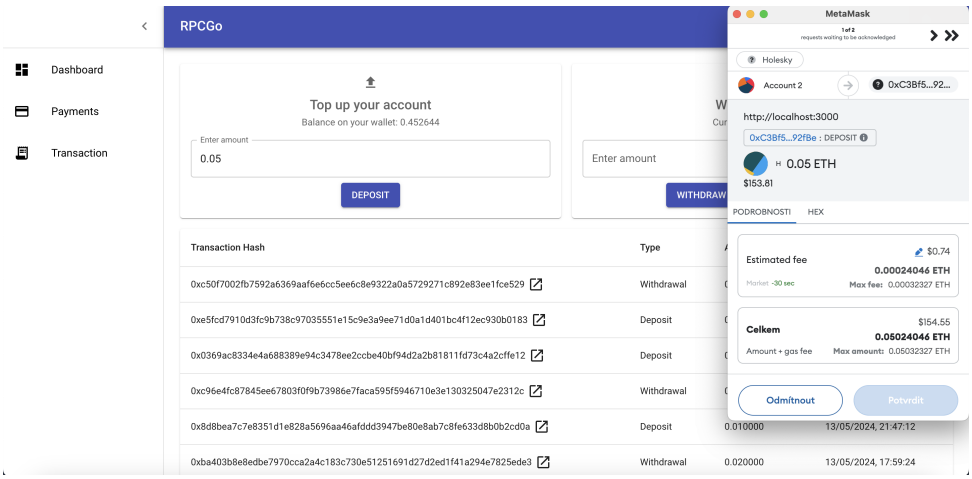
Obrázek 5.8: Stránka s přehledem informací u účtu uživatele.

Platby

Po stisknutí tlačítka *Payments* je uživatel přesměrován na stránku dostupnou na cestě /payments. Tento modul umožňuje uživateli zobrazit si přehled svých plateb a výběrů ze služby. V tomto přehledu jsou dostupné veškeré transakce tohoto typu, které peněženka uživatele vykonala. Dále lze v modulu ovládat rozhraní pro samotné zaslání či výběr prostředků (viz obrázek číslo 5.9). Výběr a zaslání prostředků jsou funkce, které jsou v rozhraní rozděleny do dvou částí, ale fungují z pohledu uživatele obdobně. Při zasílání prostředků do služby uživatel nejprve vyplní částku, kterou chce do služby vložit. Následně MetaMask vygeneruje transakci, kterou uživatel potvrdí. Transakce vyvolaná peněženkou MetaMask je vidět na obrázku číslo 5.10. Po potvrzení transakce uživatelem probíhá její vykonání. Transakce se objeví ve stavu *pending* v přehledu transakcí.



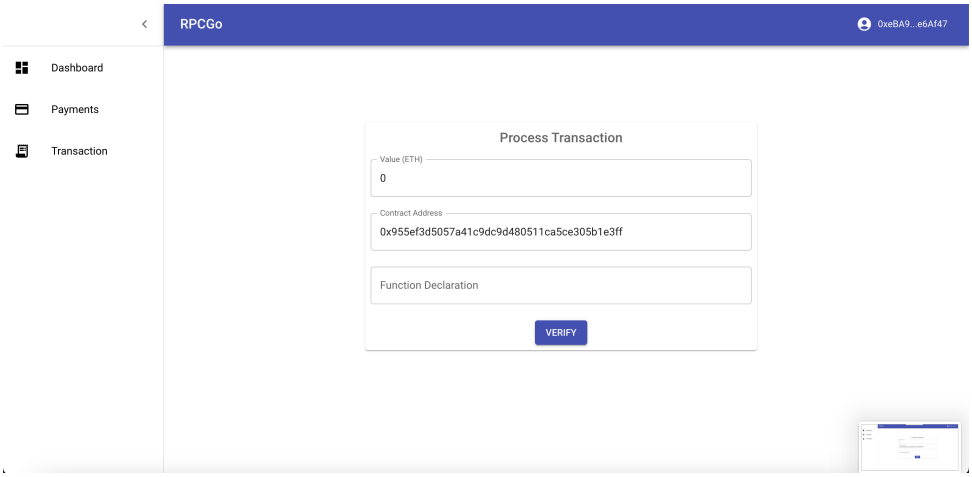
Obrázek 5.9: Platební modul aplikace.



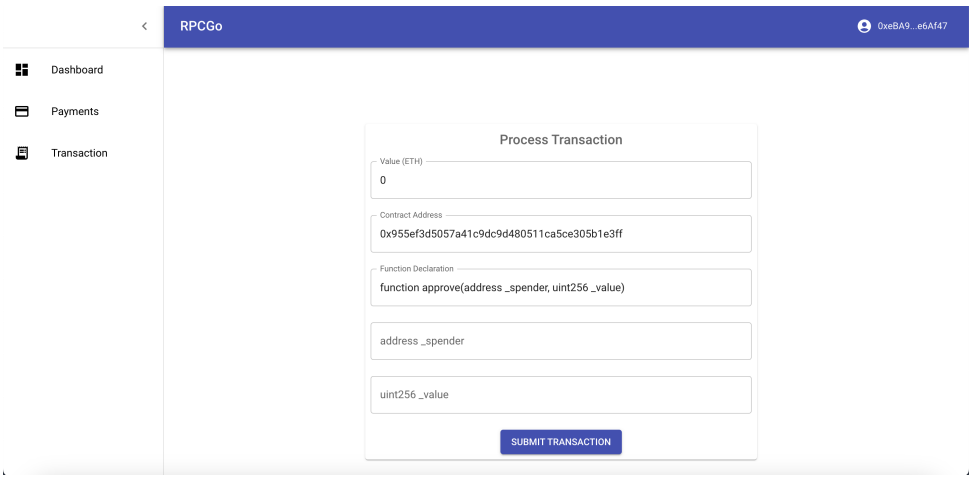
Obrázek 5.10: Transakce vyvolaná peněženkou MetaMask.

Transakce

Modul *transakce* umožnuje uživateli provést transakci přímo skrze službu, tedy skrze chytrý kontrakt. Chytrý kontrakt následně transakci přepošle na chytrý kontrakt, který je předmětem volání. Na obrázku 5.11 lze vidět rozhraní s vícero vstupy. Do prvního pole se zadává množství prostředků, které mají být zaslány do služby. Tyto prostředky lze považovat za platbu za službu obdobně jako při předplacení služby. Nevyužité prostředky se přičítají k zůstatku účtu uživatele. Druhé pole obsahuje adresu chytrého kontraktu, který chceme skrze službu volat. Další pole slouží k identifikaci funkce, která má být volána na chytrém kontraktu. Na základě vložené deklarace funkce, která má být volána, se po stisknutí tlačítka *VERIFY* vygenerují pole pro vložení argumentů funkce (viz obrázek číslo 5.12). Po stisknutí tlačítka *SUBMIT TRANSACTION* se následně objeví vyvolané okno peněženkou MetaMask a uživatel potvrdí transakci obdobně jako v modulu *payments*.



Obrázek 5.11: Modul transakce.



Obrázek 5.12: Rozhraní po verifikaci volané funkce.

5.3 Chytré kontrakty

Chytrý kontrakt tvoří v systému hlavní logiku. Vzhledem k tomu, že za službu se platí kryptoměnami a veškeré potřebné funkce jsou transakční, lze chytrý kontrakt považovat za backend aplikace. V chytrém kontraktu je udržován mimo jiné také stav zůstatků na všech účtech..

5.3.1 Zdrojové kódy

V adresáři jsou dvě základní složky. Jednou z nich je složka contracts, ze které jsou následně kontrakty kompilovány. Druhá důležitá složka je složka test, která obsahuje jednotkové testy kódu chytrého kontraktu.

Ve složce contracts se nachází dva chytré kontrakty.

- **RpcGo.Sol** — Tento chytrý kontrakt obsahuje veškerou logiku aplikace. S použitím kontraktu lze vykonávat transakce na dalších chytrých kontraktech třetích stran. Kontrakt zajišťuje základní funkce pro práci s platbami, umožňuje platby provádět a vybírat zůstatek na účtu. Dále zajišťuje mapování účtů na jejich zůstatky. Při provádění transakcí a volání kontraktů třetích stran také strhává poplatek ze zůstatku na účtu.
- **ERC20.Sol** — Chytrý kontrakt ERC20 reprezentuje jednoduchou implementaci L2 tokenu. Kontrakt je zahrnut v repozitáři aplikace pro účely uživatelského testování aplikace. Jeho funkce jsou standardní pro většinu L2 tokenů a zajišťují základní manipulaci s tokeny.

5.3.2 Hlavní funkce a logika kontraktu RpcGo.Sol

V této sekci jsou popsány hlavní důležité funkce chytrého kontraktu RpcGo.Sol. Je zde poskytnut rozbor jednotlivých funkcí a knihoven.

Mapování účtu a zůstatku

Mapování v Solidity není obdobně jednoduché jako v jiných, rozšířenějších programovacích jazycích. Pro tvorbu datové struktury typu klíč - hodnota se využívá mapping. Při deklaraci datové struktury využijeme syntax *mapping(_KeyType => _ValueType)*. Toto řešení však není dostačující, protože tato třída objektů neumožňuje pohodlnou práci s polem. Aby bylo možné pole hodnot typu klíč - hodnota iterovat a zajistit možnost základní manipulace s tímto objektem, je zde vytvořena vlastní knihovna s datovou strukturou. Vytvořená knihovna IterableMapping umožňuje jednoduchou manipulaci s objektem. Ukázku kódu lze vidět na obrázku 5.13.

Úvodem je v knihovně vytvořena struktura *Map*, která je potřebná pro definici uchovávaných informací. Tato struktura obsahuje klíče, v tomto případě adresy účtů, hodnotu, index a Booleovskou hodnotu, zda je informace vložena či není.

Dále knihovna obsahuje funkce pro základní práci s datovým typem.

- **get(Map storage map, address key)** — Funkce vrací hodnotu na základě klíče.
- **getKeyAtIndex(Map storage map, uint256 index)** — Funkce vrací klíč na základě indexu v poli.
- **size(Map storage map)** — Funkce vrací velikost pole dvojic klíč - hodnota.
- **set(Map storage map, address key, uint256 val)** — Funkce tvoří novou dvojici hodnot klíč - hodnota, pokud neexistuje v poli. Pokud je prvek s tímto klíčem již zařazen v poli, nastaví mu novou hodnotu.
- **remove(Map storage map, address key)** — Funkce odstraňuje prvek z pole dvojic.

V další části kódu je tato knihovna využívána pro správu účtů a jejich zůstatků.

```
library IterableMapping {
    struct Map {
        address[] keys;
        mapping(address => uint256) values;
        mapping(address => uint256) indexOf;
        mapping(address => bool) inserted;
    }

    function get(Map storage map, address key) public view returns (uint256) {
        return map.values[key];
    }

    function getKeyAtIndex(Map storage map, uint256 index)
        public
        view
        returns (address)
    {
        return map.keys[index];
    }
```

Obrázek 5.13: Ukázka implementace knihovny IterableMapping.

Funkce kontraktu RpcGo.Sol

Chytrý kontrakt RpcGo.Sol implementuje rozhraní hlavní logiky aplikace. V úvodu implementace jsou zavedeny deklarace a definice proměnných. Současně jsou zde deklarovány události, k jejichž vzniku dochází v průběhu funkcí. Tyto události jsou důležité zejména pro služby, které zajišťují indexování. Příkladem takové služby je TheGraph.

Události zavedené v kontraktu jsou následující.

- **ExecuteTransaction(address indexed from, address indexed to, uint256 value, bytes data)** — Událost je vyvolána při každém zprocesování jednotlivé transakce během volání služby. V události jsou zaznamenány informae o tvůrci transakce, volaném chytrém kontraktu, množství kryptoměny převáděné s transakcí a data, která mají být předána volanému kontraktu.
- **RevertReason(string reason)** — Událost je vyvolána při jakékoliv kontrolované chybě či nesplnění podmínek pro průchod funkcí. Zaznamenává také důvod přerušení průchodu fukncí.
- **LogData(bytes data)** — Data, která jsou předávána kontraktu při volání služby, jsou zaznamenána touto událostí.
- **Withdraw(address indexed account, uint256 amount)** — Po úspěšném vybrání použitelného zůstatku uživatelem se emituje tato událost. Zaznamenává se informace o účtu, který provedl výběr, a hodnotě výběru.

- **TransferAccount(address indexed from, address indexed to, uint256 amount)** — Událost se tvoří v momentě, kdy uživatel převede část svého zůstatku či celý svůj zůstatek na jiný účet. U této události jsou emitována data o účtu, který hodnotu převáděl, a účtu, který hodnotu přijal. Zároveň se předává informace o částce, která byla převedena.

- **OwnerWithdraw(uint256 amount)** — Událost je vytvořena v momentě, kdy majitel chytrého kontraktu vybere dostupný zůstatek. Zaznamenává se částka, kterou majitel z kontraktu převedl.

- **Deposit(address indexed account, uint256 amount)** — Uživatelem zaslaná platba je zaznamenána touto událostí spolu s informací o původci transakce a částce, jež byla zaslána na účet kontraktu.

- **Payment(address indexed account, uint256 amount)** — Stržením poplatku ze zůstatku na účtu uživatele se vytvoří událost Payment. Pro zpětné získání celkového obnosu zaplaceného za službu uživatelem je zde důležité, že se předává hodnota zaplaceného poplatku. Současně se zaznamenává účet, ze kterého byla platba stržena.

Emisi těchto událostí zajišťují samotné funkce chytrého kontraktu. Tvorbu události lze vidět na obrázku číslo 5.14. Následuje popis důležitých funkcí na kontraktu.

- **constructor()** — Konstruktor je spouštěn při nasazení chytrého kontraktu do prostředí EVM. V případě kontraktu RpcGo.Sol zajišťuje přiřazení majitele kontraktu k proměnné owner. Majitel kontraktu je automaticky účet, který podepsal transakci a zaslal bajtkód, čímž chytrý kontrakt nasadil do prostředí EVM.

- **submitTransaction(address to, uint256 value, bytes memory data)** — Tato funkce je v kontraktu nastavena jako *payable*. Může tedy přijímat depozit od volajícího. Jejím účelem je přeposílat transakce do dalších chytrých kontraktů.

- **function deposit()** — Funkce přijímá vklad od volajícího. Tento vklad je považován za předplacenou částku pro užívání služby.

- **function deposit()** — Funkce je nastavena jako *payable* a přijímá vklad od volajícího. Tento vklad je považován za předplacenou částku pro užívání služby.

- **function withdrawBalance(uint256 amount)** — Funkce umožňuje volajícímu vybrat ze svého zůstatku nevyužité prostředky.

- **function transferAccount(address to, uint256 amount)** — Funkce umožňuje volajícímu převést část svého zůstatku či celý zůstatek na jiný účet.

- **function ownerWithdrawAmount(uint256 amount)** — Funkce umožňuje majiteli chytrého kontraktu převést dostupné prostředky z chytrého kontraktu na jeho účet. Suma hodnot, která reprezentuje dostupné prostředky, se počítá dle prostředků, které uživatelům byly strženy z jejich zůstatků.
- **function getAccountBalance(address account)** — Funkce vypíše stav účtu volajícího.

```
contract RpcGo {
    using IterableMapping for IterableMapping.Map;

    event ExecuteTransaction(address indexed from, address indexed to, uint256 value, bytes data);
    event RevertReason(string reason);
    event LogData(bytes data);
    event Withdraw(address indexed account, uint256 amount);
    event TransferAccount(address indexed from, address indexed to, uint256 amount);
    event OwnerWithdraw(uint256 amount);
    event Deposit(address indexed account, uint256 amount);
    event Payment(address indexed account, uint256 amount);

    IterableMapping.Map private accounts;
    address private owner;
    uint256 private feeAmount = 0.0001 ether;

    constructor() {
        owner = msg.sender;
    }
```

Obrázek 5.14: Kód chytrého kontraktu RpcGo a implementace událostí.

5.4 TheGraph

TheGraph je služba poskytující rozhraní, jež umožňuje jednoduchý přístup k datům na blockchainu. TheGraph indexuje veškeré události na blockchainu. Události jsou poté dostupné přes GraphQL rozhraní. V projektu je graphQL využíváno k získání dat o účtu v klientské aplikaci.

5.5 Nasazení na AWS

Aplikace je v současné době nasazena v prostředí Amazon Web Services, dále jen AWS. AWS je poskytovatel cloudových služeb, jako například virtuálních privátních serverů a další potřebné infrastruktury k nasazení aplikací. K úspěšnému nasazení aplikace jsou třeba dva kroky. Nejprve je třeba vytvořit build aplikace a následně samotné nasazení. Dále je v této sekci popsáno, které další služby AWS jsou využity pro zajištění přístupnosti aplikace. Dostupnost aplikace je závislá na dostupnosti služeb poskytovatele. Dostupnost aplikace je závislá na poskytovateli cloudových služeb. Vytvoření vlastní infrastruktury a zajištění 99,9% dostupnosti služby je mimo rozsah této práce. Odpovědnost zajištění dostupnosti je tedy v rámci nasazení implementované služby postoupena na AWS. AWS garantuje vrácení útraty za službu ve formě kreditů či

peněz, pokud není schopno zajistit 100% dostupnost. Vrácení peněz se odvíjí od procentuální doby nedostupnosti. Pokud je dostupnost menší než 99,9% a zároveň vetší než 99%, uživatel má nárok na vrácení 10% z útraty za službu. Pokud je dostupnost menší než spodní hranice předchozího intervalu a vetší než 95%, AWS garantuje vrácení 25% útraty. Jakákoliv menší dostupnost znamená vrácení plné utracené částky. [22]

5.5.1 Nasazení aplikace

Pro úspěšné nasazení aplikace je nejprve potřeba vytvořit Docker image. Docker image je soubor obsahující všechny potřebné informace potřebné k provozu aplikace, jako například kód aplikace, knihovny a závislosti. Tento proces je odbaven příkazem *docker build . -t dockerdomain/imagename:latest –platform=linux/amd64*. Dále je potřeba docker image nahrát do služby Elastic Container Registry, dále jen ECR. Služba ECR umožňuje ukládání a verzování Docker image. Nahrání kódu lze zajistit příkazem *docker push <aws_account_id>.dkr.ecr.<region>.amazonaws.com/my-app:latest*. Do příkazu je třeba přidat identifikátor účtu, region, ve kterém se nachází ECR uživatele, a název registru.

Kód se úspěšně nahrál do služby ECR. Nyní je třeba docker image nasadit do kontejneru služby ECS. Služba ECS zajišťuje orchestraci Docker kontejnerů. Atualizace ECS úlohy zajistí, že se z ECR nahraje nejnovější verze Docker image.

5.5.2 Další využité služby AWS

Dále je pro úspěšný chod aplikace zapotřebí nastavit službu load balancer. Load balancer se stará o směřování dotazů do takzvaných cílových skupin. Load balancer následně dle typu domény dotazu směřuje dotaz do příslušné cílové skupiny. V cílové skupině je nastaveno, kam se má dotaz dále propagovat a přes jaký protokol. V tomto nasazení cílová skupina obsahuje službu ECS, jež má otevřený port 3000. Na tomto portu již naslouchá implementovaná aplikace.

5.6 Možná rozšíření aplikace

Současná verze služby má mnoho příležitostí pro navazující rozvoj. Aktuální verze chytrých kontraktů je nasazena do prostředí testnetu, který se jmenuje *Holesky*. RPC uzel, který je v této verzi využíván, je externí. Aby byla aplikace použitelná v komerčním prostoru, nezbývá však mnoho úkonů, které je třeba zajistit. Primárně se jedná o nasazení vlastního uzlu. Vlastní uzel v rámci této práce nasazen nebyl, a to z důvodu zanechání preference výběru uzlu na provozovatele služby. Služba není vázána na žádný konkrétní typ RPC uzlu ani na konkrétní EVM kompatibilní síť.

5.6.1 Nasazení do konkrétní EVM kompatibilní sítě

Pro nasazení do konkrétní EVM kompatibilní sítě je třeba splnit tři požadavky.

- **Nasazení chytrého kontraktu** — Chytrý kontrakt RpcGo.Sol lze nasadit do libovolné EVM kompatibilní sítě. Za použití informací z kapitol 5.1.2 a 5.3 lze toto nasazení vykonat. Službu TheGraph je třeba téže nasadit na zvolenou síť.
- **Nasazení RPC uzlu** — Prvním krokem tohoto bodu je výběr konkrétní implementace RPC uzlu. Po vybrání vhodné technologie je třeba uzel nasadit na libovolný server a zajistit jeho konektivitu s blockchainem a dostupnost veřejného RPC API rozhraní.
- **Nasazení webové aplikace** — Pro zajištění plné funkčnosti webového rozhraní je potřeba upravit proměnné hodnoty prostředí. Toho lze dosáhnout před nasazením webové aplikace úpravou hodnot v souboru .env v adresáři frontend. Hodnotu *NEXT_PUBLIC_PROVIDER* je třeba nastavit na odkaz na veřejné rozhraní nasazeného RPC uzlu. Hondotu *NEXT_PUBLIC_RPC_GO_CONTRACT_ADDRESS* je třeba nastavit na adresu nasazeného chytrého kontraktu RpcGo.Sol. Hodnota *NEXT_PUBLIC_GRAPH* musí být upravena na odkaz na veřejné rozhraní služby TheGraph.

5.6.2 Zajištění bezpečného volání služby

Po nasazení vlastního RPC uzlu by bylo vhodné kontrolovat přístup k uzlu na úrovni dotazu. Aby bylo zajištěno, že každé volání služby je zpoplatněno, měla by existovat dodatečná logika přímo v uzlu či na aplikační úrovni před ním. Tato logika by měla kontrolovat, že v těle dotazu se nachází volání chytrého kontraktu RpcGo.Sol a ne kontraktu jiného.

5.6.3 Podpora dalších blockchainových sítí

Současnou verzi aplikace lze nasadit pouze na jednu zvolenou EVM kompatibilní síť. Pro udržení kvality služeb současných poskytovatelů RPC uzlu by bylo vhodné implementovat podporu vícero EVM kompatibilních sítí zároveň. Službu je možné rozšířit i do sítí, jež nejsou s EVM kompatibilní. Takový zásah by znamenal výrazně složitější implementaci, protože chytré kontrakty v takových sítích nelze psát v jazyce Solidity. Zároveň služby, jako je TheGraph, nad těmito sítěmi nefungují či fungují odlišným způsobem, než je popsáno v této práci.

Kapitola 6
Testování

Aby byla zajištěna kvalita softwaru, implementace zahrnuje soubor testů. Jedná se o testy klíčové infrastruktury aplikace. První soubor testů obsahuje jednotkové testy chytrých kontraktů. Druhý soubor testů zahrnuje end-to-end testy frontendu aplikace.

6.1 Jednotkové testy chytrého kontraktu

V implementaci jsou zahrnuty testy, které pokrývají jednotlivé funkce chytrého kontraktu. Zde je přehled zásadních testů, které se týkají klíčových funkcí.

- **Funkce deposit** — Testuje se, zda je částka, jenž byla zaslána jako platba, dostupná v zůstatku volajícího účtu.
- **Funkce withdraw** — Testuje se, zda je částka, jenž byla vybrána ze zůstatku účtu volajícího, odečtena ze zůstatku a vrácena na účet uživatele.
- **Funkce submitTransaction** — Testuje se, zda lze vykonat úspěšně transakci skrze službu. Zároveň se kontroluje, jestli funkce odečítá poplatek ze zůstatku volajícího.
- **Funkce ownerWithdraw - 1** — Testuje se, zda majitel kontraktu nemůže zcizit prostředky na účtu uživatele.
- **Funkce ownerWithdraw - 2** — Testuje se, zda volající, který není majitelem, nemůže zcizit uživateli zaplacenou částku z účtu služby.

6.2 E2E testy frontendu aplikace

V implementaci jsou zahrnuty testy, které pokrývají běžné průchody aplikací uživatelem. Zde je přehled zásadních E2E testů, které se týkají klíčových částí rozhraní.

- **Přihlášení** — Testovací framework Synpress nejprve provede založení účtu v MetaMask. Následně se pokusí přihlásit do rozhraní povolením

propojení s decentralizovanou aplikací v MetaMask. Dále testuje, zda byl uživatel přihlášen a jeho adresa účtu je totožná s adresou uvedenou v uživatelském rozhraní.

- **Předplacení služby** — V tomto scénáři je nejprve provedeno přihlášení. Následně se provede navigace do rozhraní pro platby. V rozhraní pro platby se provede transakce pro předplacení služby. Testuje se, zda byla transakce předána a nachází se ve stavu provádění.
- **Provedení transakce** — V tomto scénáři je nejprve provedeno přihlášení. Následně se provede navigace do rozhraní pro vykonání transakce. V rozhraní se provede verifikace zadané volané fuknce a vyplní se chybějící informace. Testuje se, zda bylo uživateli umožněno tuto transakci prostřednictvím MetaMask potvrdit.
- **Vybrání zůstatku** — V tomto scénáři je nejprve provedeno přihlášení. Následně se provede navigace do rozhraní pro platby. V rozhraní pro platby se provede transakce pro výběr zůstatku účtu volajícího. Testuje se, zda byla transakce předána a nachází se ve stavu provádění.

Kapitola 7

Závěr

Cílem bakalářské práce bylo vytvořit decentralizovanou službu poskytovatele RPC uzlu a webovou aplikaci pro správu této služby. Služba měla poskytnout možnost implicitní platby při provádění transakcí na blockchainu. Zároveň při tvorbě účtu ve službě nemělo býti nutné, aby uživatel musel procházet registračním procesem a pro platby za službu využívat platební kartu. Služba má mít nadále dostupnost 99,9% a má být nasazena do prostředí AWS.

Webové rozhraní měla práce řádně navrhnout a implementovat. Součástí návrhu a implemetace bylo splnění důležitých požadavků pro obsluhu svého účtu. Uživateli mělo být umožněno spravovat svůj účet, předplatit si službu a sledovat svoji útratu. K zadání bylo přidáno několik funkcionalit, jejichž účelem je uživateli zpříjemnit využívání služby. V rozšíření má uživatel možnost sledovat historii svých plateb, aktuální zůstatek na účtu a výběr nevyužitého zůstatku. Dále byla přidána možnost sledování využívání služby v posledních třiceti dnech a ve třiceti dnech, které tomuto období předcházely. Poslední přidanou funkcí je možnost vykonání samotné transakce skrze službu.

V zadání byly také prezentovány technologie, se kterými má být systém implementován. Bylo zadáno, aby byla využita technologie React.js pro implementaci webové aplikace. Pro tvorbu chytrých kontraktů byl zadán programovací jazyk Solidity. Testování aplikace mělo proběhnout s využitím technologií Selenium a Hardhat.

Práce zadání bakálářské práce splnila s výjimkou využití Selenia, frameworku pro tvorbu E2E testů. Pro účely tohoto typu testování byl po konzultaci s vedoucím práce zvolen framework Synpress, který je rozšířením široce využívaného frameworku Cypress. Synpress, na rozdíl od Selenia, poskytuje podporu pro testování decentralizovaných aplikací. Nad rámec zadání práce poskytla rozšíření použitelnosti webového rozhraní, přehled důležitých pojmů z oblasti blockchainových technologií a popis nedostatků současných řešení RPC uzlů. Dále byla v práci využita technologie TheGraph, která zajišťuje optimalizované získávání dat z blockchainu.

Na výstup bakalářké práce lze navázat několika způsoby. Protože prová-

dění operací na produkčních blockchainech je nákladné, služba je aktuálně nastavena na práci s testovacím blockchainem. Nasazení služby do produkčního prostředí však není náročné, protože vzniklá implementace služby je kompatibilní s celou řadou blockchainů. Služba může nyní spolupracovat pouze s jedním blockchainem najednou. Pro zajištění konkurenceschopnosti služby na trhu je možné navázat implementací podpory několika blockchainů zároveň. Už v současné implementaci však může služba uspokojit potřeby určité skupiny vývojářů a vyplnit mezeru na trhu.

Bibliografie

[1] Arun Sekar Rajasekaran, Maria Azees a Fadi Al-Turjman. "A comprehensive survey on blockchain technology". In: *Sustainable Energy Technologies and Assessments* 52 (2022).

[2] Maarten Van Steen a A Tanenbaum. "Distributed systems principles and paradigms". In: *Network* 2.28 (2002).

[3] Rebecca Yang et al. "Public and private blockchain in construction business process and information integration". In: *Automation in construction* (2020).

[4] Shahriar Fahim, S Katibur Rahman a Sharfuddin Mahmood. "Blockchain: A comparative study of consensus algorithms PoW, PoS, PoA, PoV". In: *Int. J. Math. Sci. Comput* (2023).

[5] Dominic Grandjean, Lioba Heimbach a Roger Wattenhofer. "Ethereum proof-of-stake consensus layer: Participation and decentralization". In: *arXiv preprint arXiv:2306.10777* (2023).

[6] Shashank Joshi. "Feasibility of proof of authority as a consensus protocol model". In: *arXiv preprint arXiv:2109.02480* (2021).

[7] Binance. *Proof of authority explained.* 2022. URL: `https://academy.binance.com/en/articles/proof-of-authority-explained`.

[8] Gavin Wood et al. "Ethereum: A secure decentralised generalised transaction ledger". In: *Ethereum project yellow paper* 151.2014 (2014).

[9] Jiajing Wu et al. "Analysis of cryptocurrency transactions from a network perspective: An overview". In: *Journal of Network and Computer Applications* (2021).

[10] Vitalik Buterin et al. "Ethereum white paper". In: *GitHub repository* (2013).

[11] YiHan Huangt, Wangzhenfan Qiu a ChunXiao Wut. "Comparison and Analysis of Data and Transaction Structure of Bitcoin and Ethereum". In: *2021 2nd International Conference on Computer Communication and Network Security (CCNS)*. IEEE. 2021.

[12] Muhammad Milhan Afzal Khan, Hafiz Muhammad Azeem Sarwar a Muhammad Awais. "Gas consumption analysis of Ethereum blockchain transactions". In: *Concurrency and Computation: Practice and Experience* 34.4 (2022).

[13] Saurabh Suratkar, Mahesh Shirole a Sunil Bhirud. "Cryptocurrency wallet: A review". In: *2020 4th international conference on computer, communication and signal processing (ICCCSP)*. IEEE. 2020.

[14] Léo Besançon, Catarina Ferreira Da Silva a Parisa Ghodous. "Towards blockchain interoperability: Improving video games data exchange". In: *2019 IEEE international conference on blockchain and cryptocurrency (ICBC)*. IEEE. 2019, s. 81–85.

[15] Christopher D Clack, Vikram A Bakshi a Lee Braine. "Smart contract templates: foundations, design landscape and research directions". In: *arXiv preprint arXiv:1608.00771* (2016).

[16] Alchemy. *Types of Ethereum Nodes: Full vs. Archive vs. Light*. 2022. URL: https://www.alchemy.com/overviews/full-vs-light-vs-archive-nodes.

[17] Javier Ron et al. "Highly Available Blockchain Nodes With N-Version Design". In: *IEEE Transactions on Dependable and Secure Computing* (2023).

[18] Stack Overflow. *Stack Overflow Developer Survey 2023*. URL: https://survey.stackoverflow.co/2023/#section-most-popular-technologies-web-frameworks-and-technologies.

[19] Péter Hegedűs. "Towards analyzing the complexity landscape of solidity based ethereum smart contracts". In: *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*. 2018.

[20] Fatini Mobaraya, Shahid Ali et al. "Technical Analysis of Selenium and Cypress as functional automation framework for modern web application testing". In: *9th International Conference on Computer Science*. 2019.

[21] Olaf Hartig a Jorge Pérez. "Semantics and complexity of GraphQL". In: *Proceedings of the 2018 World Wide Web Conference*. 2018.

[22] AWS. *AWS Fargate and Amazon Elastic Container Service SLA*. 2022. URL: https://aws.amazon.com/ecs/sla/.

Příloha A

Slovník pojmů

Volající — osoba, jenž vykonává transakci

Uzel — člen blockchainové sítě, z angl. node

Chytrý kontrakt — kód, který se vykonává v uzlu blockchainu

Frontend — uživatelské rozhraní implementované jako klientská aplikace

Framework — nástroj pro vývojáře usnadňující tvorbu a správu kódu

Účet — adresa chytrého kontraktu či adresa peněženky uživatele

Transakce — změna stavu blockchainu vyvolaná akcí uživatele

EVM — Ethereum Virtual Machine

Příloha B

Implementace kódu

Přílohou práce je implementace kódu v souboru implicit-payment-rpc-node-main.zip. Tato příloha byla spolu s bakalářskou prací přiložena v systému KOS.