VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

UŽIVATELSKÉ ROZHRANÍ PRO DECENTRALIZOVANÉ NÁRODNÍ VOLBY

USER INTERFACE OF DECENTRALIZED NATIONAL ELECTIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **LIBOR MALÍNEK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. IVANA STANČÍKOVÁ**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Malínek Libor**
Program: Informační technologie
Název: **Uživatelské rozhraní pro decentralizované národní volby**
User Interface of Decentralized National Elections
Kategorie: Bezpečnost

Zadání:

1. Seznamte se s decentralizovanými aplikacemi na bázi web3 (DAPPs), Ethereem a blockchainy. Prostudujte programovací jazyk RUST a jeho vývojové rámce.
2. Studujte vlastnosti hlasovacího protokolu 1-out-of-k poskytnutého vedoucí práce.
3. Vytvořte návrh aplikace pro poskytnutý hlasovací protokol a analyzujte možnosti pro konkrétní fáze protokolu.
4. Implementujte aplikaci založenou na jazyce RUST dle vytvořeného návrhu.
5. Analyzujte bezpečnostní funkce jazyka RUST v hlasovací aplikaci.
6. Proveďte studii použitelnosti a zahrňte její zpětnou vazbu do implementované aplikace.

Literatura:

- Venugopalan, Sarad, et al. "BBB-Voting: 1-out-of-k Blockchain-Based Boardroom Voting." *arXiv preprint arXiv:2010.09112* (2020).
- Wood, Gavin. "Ethereum: A secure decentralised generalised transaction ledger." *Ethereum project yellow paper* 151.2014 (2014): 1-32.
- Hao, Feng, Peter YA Ryan, and Piotr Zieliński. "Anonymous voting by two-round public discussion." *IET Information Security* 4.2 (2010): 62-67.

Pro udělení zápočtu za první semestr je požadováno:

- Bez požadavků.

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

Vedoucí práce: **Stančíková Ivana, Ing.**
Konzultant: Homoliak Ivan, Ing., Ph.D., UITS FIT VUT
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 29. července 2022
Datum schválení: 23. června 2022

Abstrakt

Práce popisuje vývoj decentralizované aplikace a uživatelského rozhraní pro hlasovací protokol v jazyce Rust. Cílem práce bylo vytvořit aplikaci, která je jednoduchá pro ovládání uživatelem a zároveň splňuje bezpečnostní prvky podle hlasovací protokolu. Výsledkem práce je decentralizovaná aplikace splňující náležitosti daného hlasovacího protokolu a jejího uživatelské rozhraní, které bylo upraveno dle uživatelské zpětné vazby. Po teoretické stránce se práce zabývá decentralizovanými aplikacemi a principem jejich fungování. Práce též popisuje jazyk Rust a jeho webové frameworky.

Abstract

The thesis describes the development of a decentralized application and user interface for the voting protocol in the Rust language. The main subject of work was to create an application that is easy to control by the user and at the same time meets the security elements according to the voting protocol. The result of the work is a decentralized application that meets the requirements of the voting protocol and its user interface, which was modified according to user feedback. From a theoretical point of view, the work deals with decentralized applications and the principle of their operation. The thesis also describes the Rust language and its web frameworks.

Klíčová slova

uživatelské rozhraní, GUI, Rust, elektronické hlasování, internetové hlasování, e-voting, blockchain, smart kontrakt, decentralizovaná aplikace

Keywords

user interface, GUI, Rust, electronic voting, internet voting, e-voting, blockchain, smart contract, decentralized application

Citace

MALÍNEK, Libor. *Uživatelské rozhraní pro decentralizované národní volby.* Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Ivana Stančíková

Uživatelské rozhraní pro decentralizované národní volby

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením slečny Ing. Ivany Stančíkové. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Libor Malínek
27. července 2022

Poděkování

Rád bych poděkoval Ing. Ivaně Stančíkové za cenné rady, věcné připomínky a vstřícnost při konzultacích a vypracování bakalářské práce.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Volby jsou základním demokratickým prvkem. Ústava dává právo účastnit se voleb jak pasivním tak i aktivním hlasovacím právem. V České republice se však pasivní právo vykonává pouze za osobní účasti ve volebních místnostech, kde hlasujeme na papírových hlasovacích lístcích a kontrolu provádí členové volební okrskové komise.

Pro občany žijící v zahraničí, zejména ve velkých státech je možnost dostavit se do volební místnosti (ambasády, velvyslanectví, honorární konzuláty) často velmi nákladná věc. Samotné sčítání hlasů pak provádí okrsková volební komise, kde možnost kontroly není příliš veliké. Kontrola je možná pouze soudní cestou, avšak i zde musejí být důkazy (občané nejsou účastni sčítání hlasů a jejich přístup k důkazům je omezený) pro soudní přepočet hlasů (porušení zákona, ve výsledcích není započten například přednostní hlas voliče, čestné prohlášení, atd). Člověk dělá chyby a okrsková komise není výjimkou. Ruční sčítání přináší chyby, jak z nepozornosti (únava členů, zejména v případě sčítání v nočních hodinách) tak i pro ovlivnění voleb. Činnost volební komise je jako většina práce placena a odměny tvoří největší položky v rozpočtu voleb, tedy není bezchybná, je nákladná a ruční sčítání není nejrychlejší.

Zde se nabízí otázka, zda i zde není v téhle moderní době vhodné přejít zejména na strojové sčítání, kde by sčítání hlasů probíhalo elektronicky. V dnešním světě již existuje několik států, které přešli na elektronické hlasování v různých podobách (USA, Estonsko). Hlasování přes internet, však přináší problémy zejména v oblasti soukromí (zda opravdu nikdo nebude vědět, jak, kdo hlasoval), důvěryhodnosti (opravdu jsem poslal daný hlas, dané straně, danému kandidátu). Elektronické hlasování proto musí dané problémy správně vyřešit a umožnit občanům ověření správnosti.

Cílem práce je vytvořit klientskou aplikaci s uživatelským rozhraním pro decentralizovanou aplikaci podle dodaného hlasovacího protokolu. Hlavním úkolem tedy je vytvořit funkční klientskou aplikaci s rozhraním zajišťující práci s blockchainem podle dodaného hlasovacího protokolu.

Kapitola 2 se zabývá teoretickou částí. Vysvětluje různé systémy a způsob fungování decentralizované aplikace. Programovacím jazykem Rust se zabývá kapitola 3, která vysvětluje základní informace a principy daného jazyka a následně popisuje frameworky jazyka Rust. V kapitole 4 je vysvětlen princip dodaného hlasovacího protokolu. Kapitola 5 ujasňuje plán vývoje aplikace a požadavky od aplikace. Samotnou implementaci aplikace vysvětluje kapitola 6. Bezpečnostní funkce jazyka Rust a jejich využití v aplikaci popisuje kapitola 7. O testování a zpětné vazbě pojednává kapitola 8.

Kapitola 2

Decentralizované aplikace

Decentralizované aplikace (DAPPs) mají vlastní backendový kód spuštěný v decentralizované síti, nikoliv však na centralizovaném serveru. Pro ukládání dat a smart kontraktů používají blockchain Ethereum. DAPPs tedy využívá blockchain jako jádro zpracování dat a dále i jako svého úložiště. To je implementováno pomocí smart kontraktů.

V současné době se uživatelské rozhraní (znázornění na obrázku 2.6) pro DAPPs obvykle vytváří pomocí tradičního modelu webových stránek. Takže si lze představit kompletní DAPPs jako webovou stránku a k tomu jeden nebo více smart kontraktů. DAPPs má stejné obecné vlastnosti jako tradiční aplikace. Hlavní rozdíl je tedy v tom, že data a výpočty poskytuje blockchain [15].

Používání blockchainů pro DAPPs by se mělo řídit následujícími hodnotami [15]:

1. Uživatel může vidět, co se stane před provedením příkazu nebo odesláním jakýchkoli dat na blockchain.

2. Jakmile uživatel provedl nějakou interakci, nelze ji stáhnout, změnit ani smazat.

3. Správa aplikace může být decentralizovaná tak, aby se na jejím řízení přímo podíleli uživatelé aplikace.

V posledním bodě uvažujeme dva příklady -– jeden využívá prvních dvou vlastností (tyto vlastnosti jsou samy o sobě užitečné a právě tohle ztělesňuje decentralizaci na úrovni protokolu) a druhý, který pomáhá demonstrovat myšlenku decentralizované správy neboli strukturální decentralizace.

První dvě vlastnosti reprezentuje například *CryptoKitties*[1]. Jedna z nejznámějších decentralizovaných aplikací. Hra, která umožňuje hráčům obchodovat, množit, sbírat a prodávat virtuální kočky. Virtuální položky jsou zaznamenány na blockchainu. Tímto způsobem jsou akce transparentní a zaručené, ale na principu správy aplikace není nic zvlášť decentralizovaného.

Dalším příkladem je například *DAO* (decentralizovaná autonomní organizace). Aplikace je řízena obchodovatelnými tokeny, které mají hlasovací právo. V tomto smyslu byl mechanismus aplikace garantován decentralizovanou vrstvou Etherea a samotný koncept byl navržen pro decentralizovanou správu organizace [15].

Termín **smart kontrakt** („chytrá smlouva“) definoval v roce 1996 Nick Szabo jako soubor pravidel/slibů specifikovaných v digitální podobě, včetně protokolů, v rámci kterých tyto pravidla/sliby strany plní. Jedná se tedy o program, který běží na blockchainu

[1] `https://www.cryptokitties.co/`

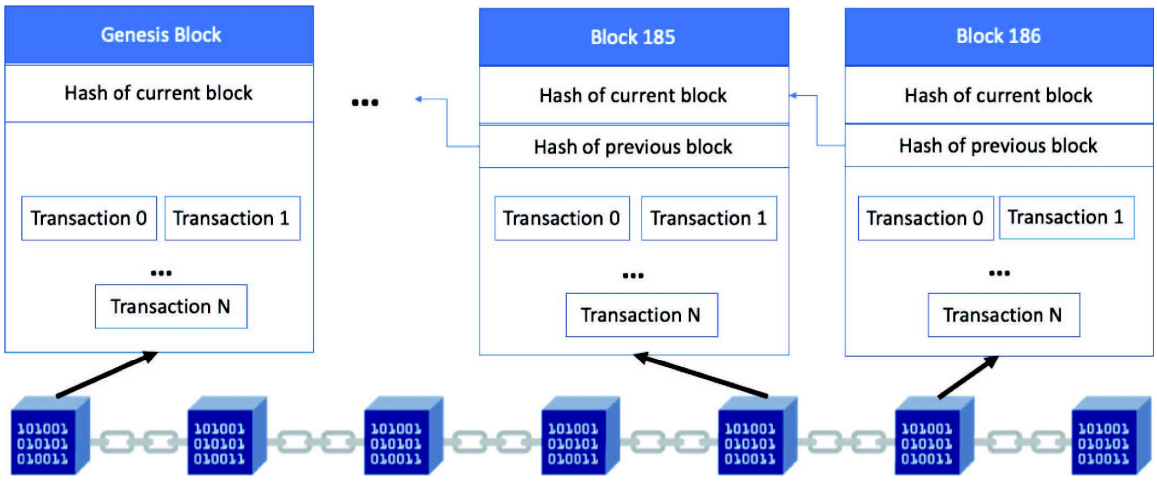
Ethereum. Je to sbírka kódu a dat, která sídlí na konkrétní adrese na blockchainu Ethereum, kde tyhle smart kontrakty využívají právě DAPPs. DAPPs lze decentralizovat, protože jsou řízeny logikou zapsanou v kontraktu, nikoliv jednotlivcem nebo společností [22, 10].

2.1 Blockchain

Blockchain je jedna z technologií distribuované účetní knihy (DLTs). DLTs umožňují stranám bez zvláštní vzájemné důvěry výměnu digitálních dat na bázi peer-to-peer[2] s menším počtem nebo i s žádným zprostředkovatelem třetí strany [1].

Data mohou představovat peníze, kontrakty, různé certifikáty (rodné, oddací listy), nákup či prodej zboží, v podstatě jakýkoli typ transakce nebo aktiva, které lze převézt do digitální podoby.

DTLs je zvláštní druh databáze, ve kterém jsou data zaznamenána, sdílena prostřednictvím distribuované sítě jednotlivých uzlů (účastníku). Blockchain je podmnožinou DTLs, využívající záznam v podobě „řetězce bloků“ (obrázek 2.1), rozdíl se týká způsobu distribuce (privátní/veřejný a bez povolení/s povolením) [1].



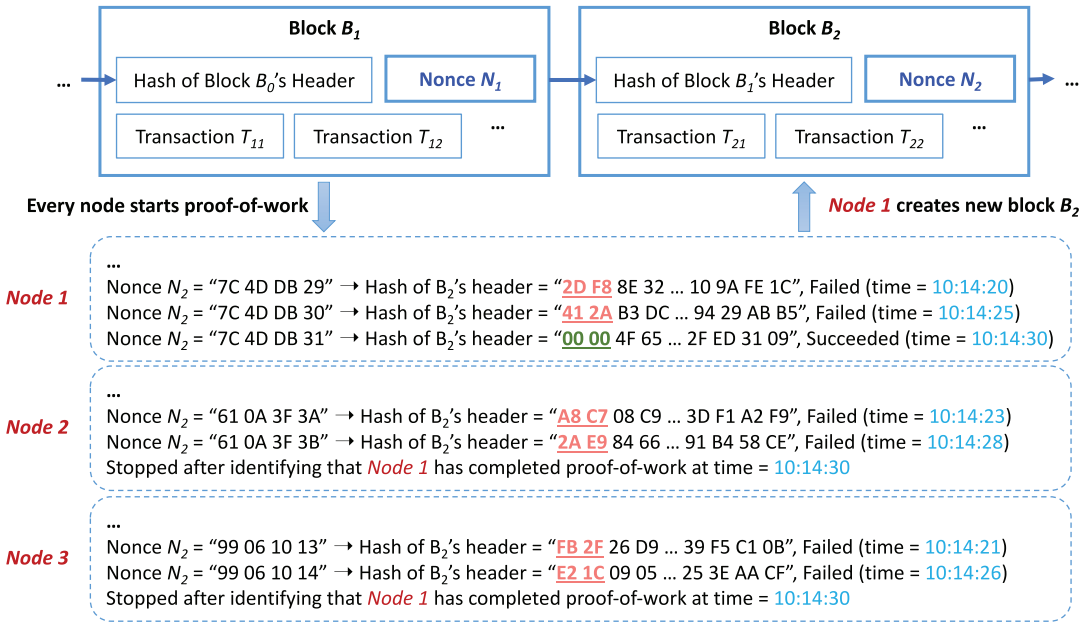
Obrázek 2.1: Datový řetězec v blockchainové síti [28].

Blockchain je „účetní kniha“, která sleduje datové transakce. Transakce je vysílána do distribuované sítě uzlů, které budou transakci ověřovat na základě dohodnutého souboru pravidel. Po ověření bude transakce s případně dalšími spojena do bloku, který bude následně přidán do blockchainu. Celý proces zajišťuje, že nově vytvořený blok je nevyvratitelně spojen s blokem předchozím a „následujícím“, tím vzniká „řetězec bloků“. Záznamy, tvořící blockchain sdílí každý uzel a je neustále synchronizován. Jako „účetní kniha“ uchovává všechny transakce, které byly vytvořeny od spuštění [1].

2.2 Těžba

Těžaři jsou speciální role účastníků v systému, kteří mají za úkol ověřovat transakce. Tento proces se nazývá *těžba*. Pokud někdo odešle někomu transakci, neznamená to přímé zaslání od jednoho uživatele k druhému, ale daná transakce se posílá do celé sítě. Tahle transakce

[2] `https://www.lexico.com/definition/peer-to-peer`



Obrázek 2.2: Příklad hledání správné hodnoty *nonce* při zadané složitosti [14]

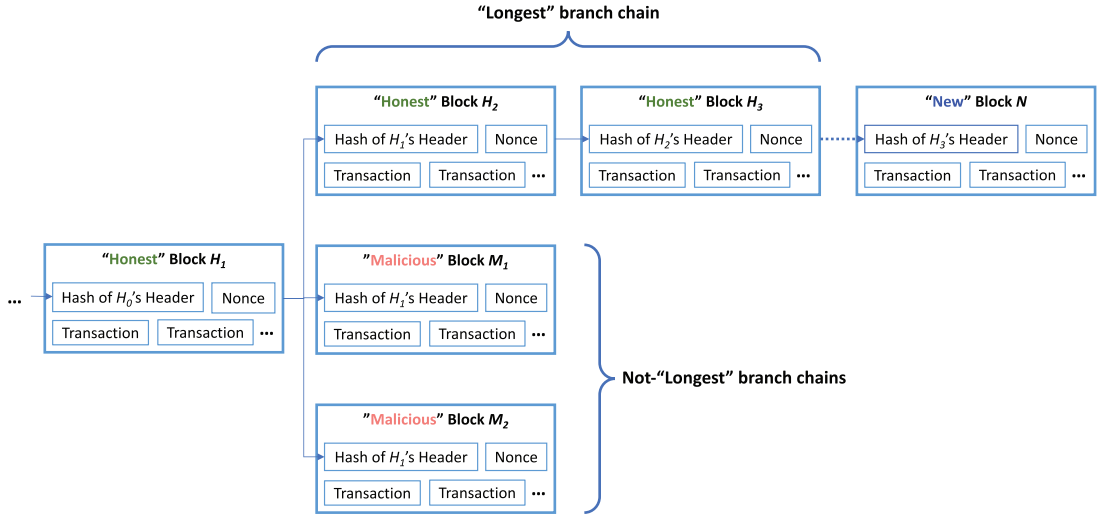
je označována jako *nevyřízená/čekající* (anglicky pending), aby mohla být úspěšně vložena na blockchain musí být ve stavu *ověřena* [1].

Těžaři sbírají nevyřízené transakce a skládají je do bloku (blok též nese nějaká specifická označení jako velikost bloku, čas, hash, ...). Hash bloku tvoří 64 znaků a cílem těžaře je vytvořit hash, který začíná předdefinovaným počtem nul, známý jako „obtížnost" (anglicky difficulty, na obrázku 2.2 vyznačeno zelenými číslicemi), avšak těžař nemůže odhadnout jakým vstupem by byl dán výstup, jediná možnost je zkoušet co možná nevíc vstupů – měnit parametry bloku, zejména hodnotu zvanou *nonce* [1].

Pokud je daný blok těžařem nalezen, dostane patřičnou odměnu v podobě kryptoměny. Když byl blockchain Bitcoin vytvořen (první blok se nazývá *Genesis*), odměna činila 50 Bitcoinů[3], avšak systém je navržen tak, aby se každé čtyři roky snížila odměna na polovinu. Těžař společně s odměnou dostane i transakční poplatky [1]. Nynější odměna činí 6,25 Bitcoinů[4].

V případě, že ve stejnou dobu najde několik těžařů blok, jsou vygenerovány do 2 větví (anglicky *fork*, částečně znázorňuje obrázek 2.3), případně více, pokud je nalezeno současně několik bloků, avšak takhle záležitost je nepravděpodobná na základě složitosti. Těžaři pracují na obou větvích a nově vygenerovaný blok přidávají na jednu z nich, těžaři pracující na jiné větvi se přesouvají na onu delší. Větev ze které odešli těžaři se stává „osiřelou", protože se již nezvětšuje. V systému je zpravidla potřebný určitý počet vytěžených bloků, aby daná větev mohla být považována za autentickou (na Bitcoinovém blockchainu je to šest bloků). To souvisí s tzv. Modelem finality (konečnost), což je jedna z vlastností, které je třeba dosáhnout prostřednictvím konsensuálních protokolů, které zajišťují, že transakce s kryptoměnami nelze po zveřejnění v blockchainu změnit, zvrátit nebo zrušit. Finalitu v blockchainu Bitcoinu je dosaženo pomocí protokolu Proof of Work (více o konsensuálních

[3] `https://www.blockchain.com/btc/block/0`
[4] `https://www.blockchain.com/explorer?view=btc`



Obrázek 2.3: Příklad, **vypořádávání se s rozvětvenými řetězci v případě, kdy útočníci vytvářejí škodlivé bloky** (M1, M2), aby „soutěžili“ s poctivým blokem (H2), ve snaze převzít poctivý řetězec (autentický, H1 a jeho předchozí bloky). Předpokládá se, že výpočetní výkon poctivých uzlů je větší než výpočetní výkon „škodlivých“ uzlů, proto poctivý blok H3 je vytvořen hned po bloku H2, ještě než útočníci vytvoří nové škodlivé bloky po bloku M1 a M2. Na základě mechanismu blockchainu, každý uzel nejprve identifikuje platný blok na základě délky řetězce a vytvoří nový blok (N) pouze na konci nejdelšího řetězce (H1-H2-H3) a řetězce H1-M1 a H1-M2 budou ignorovány, protože jsou krátké. Většinové hlasování (výpočet výkonu) se vyjádří jako „jeden CPU, jeden hlas“, proto nejdelší řetězec představuje většinové rozhodnutí). Vzhledem k tomu, že proces těžby je drahý a poctivé uzly mají vyšší výpočetní výkon (tj. mají více CPU „hlasů“) než škodlivé uzly, pravděpodobnost, že útočník úspěšně modifikuje blok a všechny následující bloky (vytvoří škodlivý řetězec) je velice nízká [14].

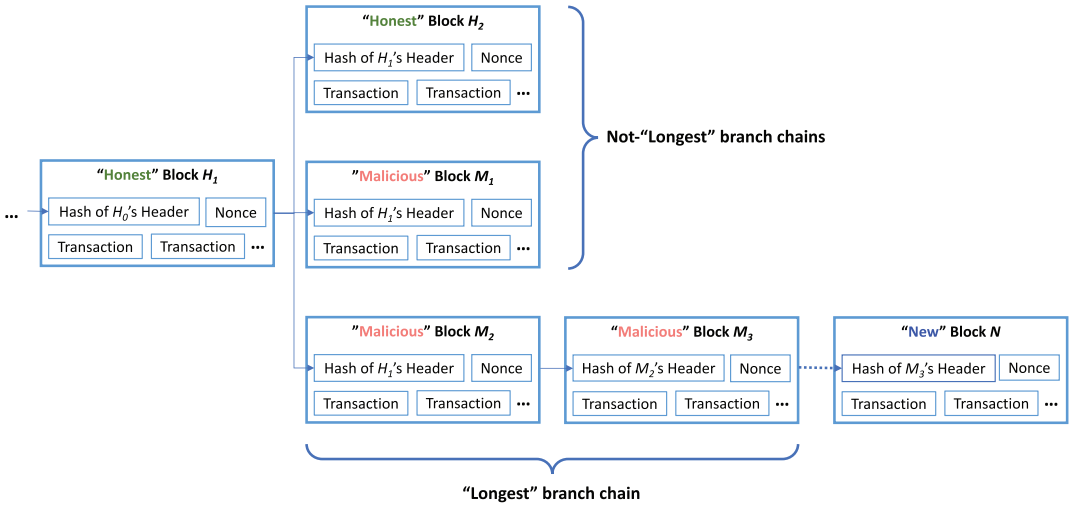
protokolech v sekci 2.3) [26, 28]. Blok na Bitcoinu se generuje přibližně jednou za 10 minut, zatímco blok Ethereu se generuje přibližně jednou za 13 sekund[5] [28]. Větvení používají útočníci pro změnu záznamu na blockchainu, útoky znázorňují obrázky 2.3 a 2.4.

Distribuce

Rozlišují se podle přístupu k blochainu na public (veřejný) a na private (soukromý). U veřejné distribuce může kdokoli číst a přistupovat k blochainu, zatímco k přístupu a čtení u privátní distribuce je potřeba autorizovat daný uzel. Dále se dělí podle provádění a ověřování transakcí na „permissionless“ a „permissioned“. U permissionless může kdokoli provádět a ověřovat transakce, zatímco u permissioned je třeba povolit daný uzel neboli uživatel musí získat povolení [1].

Public permissionless blockchains Je blockchainový systém, kde se každý může účastnit na fungování systému. Každý, kdo je připojen k internetu může provádět transakce a

[5] `https://ycharts.com/indicators/ethereum_average_block_time`



Obrázek 2.4: Příklad, **útoku 51%**. V tomhle případě se vytvářejí škodlivé bloky M1 a M2, aby „soutěžili“ s poctivým blokem (H2), ve snaze převzít poctivý řetězec (autentický, H1 a jeho předchozí bloky). Zde je však výpočetní výkon poctivých uzlů menší než výkon škodlivých uzlů, protože ovládají 51% a více výpočetního výkonu sítě, proto hned po bloku M2 vzniká škodlivý blok M3. Na základě mechanismu blockchainu bude nový blok (N) vytvořen pouze na konci nejdelšího řetězce, v tomhle případě H1-M2-M3 – útočník tedy úspěšně vložil záznam na blockchain (z H2 na M2) a přebírá řetězec získáním většiny hlasů [14].

číst celý protokol transakcí. Mezi tyhle blockchainy řadíme například **Bitcoin**, **Ethereum**, **Litecoin** [1].

Public permissioned blockchains Je blockchainový systém, umožňující každému připojenému na internetu provádět transakce a číst transakční protokol, avšak pouze někteří účastníci se můžou podílet na fungování systému. Například privátní verze Etherea nebo **Ripple** [1].

Private permissioned blockchains Je blockchainový systém, který omezuje provádění transakcí, tak i čtení transakčního protokolu. Majitel blockchainové sítě je schopen určit, kdo se může účastnit na blockchainu a kdo i na fungování celého systému. Řadí se zde například **Rubix**, **Hyperledger** [1].

Private permissionless blockchains Je blockchainový systém, který omezuje provádět transakce a čtení protokolu transakcí na určené uzly, avšak na podílení fungování sítě je otevřený komukoliv, kdo je připojený k internetu. Částečně se zde řadí **Exonum** [1].

2.3 Konsenzus

Dohoda mezi uzly v blockchainové síti, která předává transakční informace (zejména vytváření nových bloků) je jednou z nejdůležitějších součástí blockchainové technologie. Blockchainová síť je aktualizována prostřednictvím nasazeného konsensuálního protokolu, aby bylo

zajištěno, že transakce a bloky jsou seřazeny správně a aby byla zaručena integrita a konzistentnost distribuované účetní knihy [28].

Jako konsensuální algoritmus, který využívá výpočetní sílu CPU se nazývá „Proof of work“ (PoW). PoW má však nevýhodu v obrovské spotřebě energie. Proces nalezení nového bloku se odvíjí od správného nalezení hashe (znázornění na obrázku 2.2), kde je tohle možno definovat jako „závod mezi všemi těžaři“, ten který disponuje větší výpočetní kapacitou, má větší šanci najít správný hash. Jakmile však jeden těžař najde daný blok, práce všech ostatních těžařů je považováno za zbytečnou (znázornění na obrázku 2.2). Blok, kteří ostatní těžaři hledali, pravděpodobně obsahoval transakce zahrnuty v posledním nalezeném bloku a již nelze považovat za platný, což znamená, že musí změnit jeho kompozici. Automaticky musí zahájit proces těžby nového bloku od nuly [1]. Tento konsensuální algoritmus využívá například Bitcoin, Litecoin, Ethereum, Dogecoin [6].

Kvůli výše uvedenému problému ve spotřebě energie, bylo motivací nacházet nové konsensuální algoritmy. Mezi neoblíbenější patří „Proof of stake“ (PoS). V tomto konsensuálním algoritmu je těžební síla rozdělena mezi zúčastněné uzly podle procenta kryptoměny, které vlastní. Pokud někdo vlastní například 10% z celkového počtu dané kryptoměny, je daný uzel schopen vytěžit 10% bloků. Použití PoS je mnohem energetičtěji účinější než PoW, protože zde nehraje roli výpočetní výkon [1]. PoS využívají například Solana, Cardano, Cosmos [6].

Mezi další konsensuální algoritmus, který jsou však využíván méně, je například „proof of authority“ (PoA), kde ověřování transakcí a vytváření nových bloků zajišťují pouze schválené uzly [1]. PoA využívá například VeChain [6].

2.4 Bitcoin

První a nejznámější blockchainová síť z roku 2009 vytvořená osobou nebo skupinou osob pod pseudonymem Satoshi Nakamoto. Jedná se o blockchain, který sice nepodporuje smart kontrakty, avšak stal se průkopníkem v oblasti blockchainů. Mince neboli kryptoměna se nazývá Bitcoin a jedná se o nejobchodovanější měnu na kryptoburzách. Od tohohle blockchainu následně vznikaly další blockchainy pod různými názvy s různými měnami. Tenhle blockchain však nepodporuje smart kontrakty a slouží pouze pro obchodování s kryptoměnami [1].

2.5 Ethereum

Ethereum je další typ blockchainé sítě, který se však chová jako virtuální stroj (Ethereum Virtual Machine). Ethereum bylo spuštěno v roce 2015 a jeho autorem je Vitalik Buterin. Hlavním záměrem Etherea bylo vytvořit alternativní protokol pro budování decentralizovaných aplikací na základě smart kontraktů. U téhle sítě se kryptoměna nazývá Ether a jedná se druhou o nejobchodovanější měnu na kryptoburzách. U tohohle blockchainu se rozlišují dva typů účtů – externě vlastněné účty (EOA, externally owned accounts) a účty kontraktů. EOA je kontrolován pomocí soukromého klíče, nemá žádný přidružený kód, avšak může odesílat transakce. Účet kontraktu má přidružený kód, který se spustí, když obdrží transakci z EOA. Sám o sobě nemůže poslat transakci, transakce tedy musí vždy pocházet z EOA [5].

Smart kontrakt

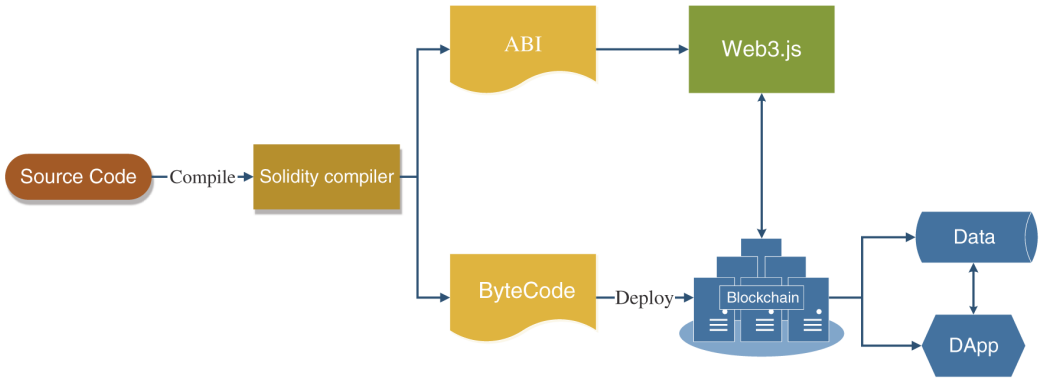
Koncept smart kontraktů, tedy soubor pravidel specifikovaný v digitální podobě byl integrován do blockchainové sítě Ethereum za účelem usnadnění, ověření a vymáhání smluv a tím zlepšením jejich plnění. Tento kontrakt je zaznamenán jako spustitelný počítačový kód. Smart kontrakt je uložen a sdílen v distribuované síti, ke které mají přístup všichni účastníci. Tyto kontrakty se automaticky vykonají, když jsou splněny všechny předem nastavené podmínky v rámci blockchainové sítě. Zúčastněné strany, které se dohodly na smart kontraktu si navzájem více důvěřují a mají snížené riziko chyb a podvodu.

Některé výhody smart kontraktů [28]:

- **Úspora nákladů** – není zde žádný prostředník třetí strany, zkrácení doby procesu
- **Přesnost** – všechny kontrakty jsou zaznamenány pomocí počítačového kódu, který je přesný a efektivní prostředek pro uložení informací
- **Transparentnost** – kontrakty jsou dostupné a viditelné pro všechny uzly

Ethereum Virtual Machine

Virtuální stroj Ethereum (EVM) provádí logiku definovanou ve smart kontraktech a zpracovává změny stavu, ke kterým dochází na blockchainu. EVM nerozumí jazykům na vysoké úrovni, jako je například Solidity, které se používají k psaní smart kontraktů. Místo toho se musí zkompilovat jazyk vysoké úrovně do bajtového kódu (obrázek 2.5 popisuje nasazení smart kontraktu), který pak může EVM spustit [13]. Mezi další blockchainy, které využívají virtuální stroje můžeme zařadit NEO[6], NEAR[7] nebo AVM[8].



Obrázek 2.5: Proces nasazení smart kontraktu. Zdrojový kód smart kontraktu je přeložen pomocí Solidity kompileru na bytecode (bajtový kód) a na ABI (aplikační binární rozhraní), což lze jednoduše chápat jako soubor popisu rozhraní smart kontraktu, který obsahuje názvy polí, názvy metod, názvy parametrů a typy parametrů ve smart kontraktu [20]

EVM je Turingovsky kompletní, tedy model výpočtů, který může provádět jakýkoli algoritmus bez ohledu na to, jak je složitý, jaké datové struktury se používají a kolik úložného prostoru nebo času by bylo potřeba k jeho vyhodnocení, avšak efektivita výpočtů již nemusí

[6] `https://neo.org/`
[7] `https://near.org/`
[8] `https://developer.algorand.org/`

být ideální [18]. Případný cyklus v kódu se však musí zastavit po určitém počtu provedení, to je nutné z hlediska toho, aby se zabránilo provádění daného cyklu nekonečně dlouho, což by omezilo provádění dalších instrukcí v kódu aplikace [16]. EVM je zejména zodpovědný za všechna následující provedení v síti [3]:

- Potvrzuje platnost transakce. Kontroluje správný počet hodnot, podpisy, hodnotu *nonce*. Pokud není správný některý z daných parametrů, vrátí chybu.
- Zkontroluje, zda je dostatek *gas* na provedení transakce, v případě že není, transakce se nezdaří. Transakční poplatek se už ale nevrátí a stane se součástí odměny těžaře.
- Převádí hodnoty Etherea na adresu příjemce.
- Vypočítává celkový spotřebovaný *gas* a transakční poplatek, aby inicializoval platbu pro těžaře.

Gas

Gas je zastavovací parametr Etherea a závisí na dané instrukci, kdy každá instrukce spotřebuje určité množství [16]. Cena jednotky gas není konstantní, ale záleží na vytíženosti dané sítě, tedy „čím víc je síť využívána v daný moment, tím více bude stát jednotka gas“. Celková cena je však i dána podle složitosti a potřebného výpočetní výkonu pro zapsání jednotlivých bloků do sítě. Uživatel musí zaplatit za vynaložený výpočetní výkon bez ohledu na to, zda byla transakce úspěšná či nikoli.

Gas je proto pobídka pro vývojáře, aby vytvářeli smart kontrakty s nízkými náklady, a pro těžaře, aby ověřovali transakce. Tímto způsobem se komunita brání proti plýtvání výpočetního výkonu, spamování a zamezuje náhodným či úmyslným nekonečným smyčkám instrukcí [3].

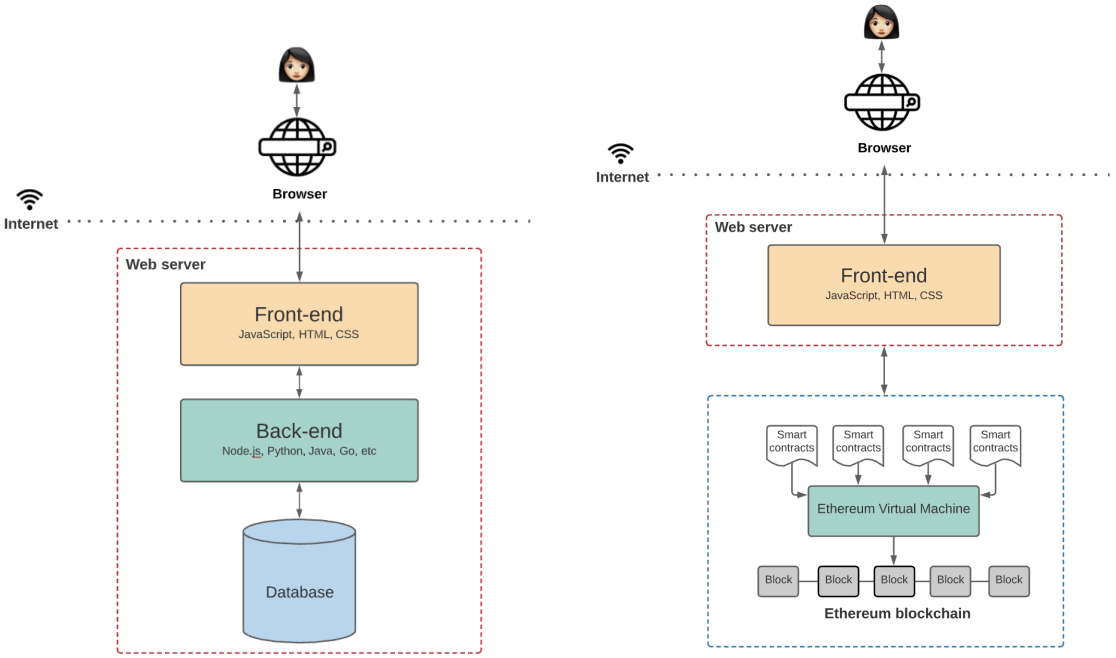
Cena gas[9] se uvádí v jednotkách *gwei*, kdy 1 $gwei = 10^{-9}\ ETH$. Ceny také ovlivňují dobu trvání potvrzení transakce. Čím více se za gas nabídne, tím se zpravidla rychleji potvrdí transakce.

2.6 Web3

První generace internetu je období 60.–90. let 20. století, od vojenského a vědeckého internetu až po příchod komerčního internetu [17]. Web2 je podoba internetu, kterou většina uživatelů zná. Princip je výměna vašich osobních údajů společnostem, které poskytují služby na internetu. Aplikace pro web 2.0 (obrázek 2.6) musí mít místo pro uložení základních dat, to vyžaduje neustálou aktualizaci databáze. Backendový kód musí definovat logiku aplikace a frontendový kód uživatelské rozhraní [13].

Web3 v kontextu Etherea označuje decentralizované aplikace (obrázek 2.6), běžící na blockchainu, které umožňují poskytovat nějakou službu a přitom nepotřebují osobní údaje uživatelů [27]. Na rozdíl od aplikací Web 2.0, Web 3.0 eliminuje prostředníka. Neexistuje žádná centralizovaná databáze, která by ukládala stav aplikace a neexistuje žádný centralizovaný webový server, kde by sídlila backendová logika. Místo toho se využívá blockchain, který je spravován anonymními uzly na internetu. Backend v tomhle případě tvoří smart kontrakty a každý uživatel, kdo chce vytvořit aplikaci běžící na blockchainu nasadí vlastní kód do sítě [13].

[9] `https://etherscan.io/gastracker`



Obrázek 2.6: Architektura web 2.0 a web 3.0 aplikace [13]

Benefity

Následující odrážky vychází z [27].

- Každému v síti je oprávněno využívat služby
- Nikdo nemůže zablokovat, ani odepřít přístup ke službě
- Platby pomocí tokenu ether (ETH)
- Ethereum je Turingovsky kompletní (anglicky Turing-complete)

Limity

Následující odrážky vychází z [27].

- **Rychlost** – transakce na web3 jsou pomalejší z důvodu, že musí být zpracovány těžařem a šířeny po síti
- **UX** – interakce s Web3 aplikacemi může vyžadovat další software a znalosti uživatele
- **Dostupnost** – nedostatečná integrace Web3 aplikací v prohlížečích, tím je méně dostupný pro uživatele
- **Cena** – čím větší kód aplikace, tím dražší poplatky na blockchainu, snaha mít co nejmenší kód

Srovnání s centralizovaným systémem

Centralizovaný systém je rychlejší, protože předávání informací zajišťuje centrální orgán, který disponuje vysokým počtem výpočetních zdrojů, mají většinou větší propustnost (výkon). V rámci konfliktů má rozhodující „slovo“ centrální orgán, což se někdy může zdát jako výhoda, naproti tomu u decentralizovaného systému je zapotřebí, často složitý protokol pro vyřešení problému, pokud účastníci mají protichůdná „tvrzení“ [27].

Koordinace u centralizace je snazší, centrální orgán může donutit účastníky sítě, aby přijali upgrady nebo aktualizace. Centrální orgán může cenzurovat data, čímž potenciálně odřízne části sítě od interakce se zbytkem sítě [27]. U některých věcí se zdá být centralizovaný systém lepší, avšak je zde si potřeba uvědomit, že vše záleží na centrálním orgánu a jeho rozhodnutí platí, i když s ním většina uživatelů nebude souhlasit.

U centrální sítě mohou být útočníci schopni zničit síť tím, že se zaměří na centrální orgán. Zatímco decentralizovaná síť může stále fungovat, i když je velká část účastníků napadena [27].

Kapitola 3

RUST

Programovací jazyk RUST[1] si zakládá na třech pilířích – **výkon**, **spolehlivost** a **produktivita**. RUST je staticky typovaný, univerzální, open-source programovací jazyk. Klade důraz na bezpečnost paměti, aniž by ohrozil běh aplikace. Tento jazyk se poprvé objevil v roce 2010 jako projekt Mozilly[2] a první stabilní verzi měl v roce 2015 [9].

Syntaxe jazyka je velice podobná programovacímu jazyku C++, avšak sémanticky je od tohoto jazyka odlišný. RUST je si zakládá na bezpečnosti, mezi kterou patří i paměťová, tím že nepovoluje ukazatele NULL a znemožňuje neplatné ukazatele. Jazyk obsahuje speciální typ s názvem *trait*, který je podobný typu třída. Jedná se o nástroj, který se v jiných jazycích často nazývá rozhraní (interface).

3.1 Verze

RUST se vydává ve třech verzích – Stable, Beta, Nightly. Stable (Stabilní) verze jsou vydávány každých 6 týdnů. Beta verze jsou verze, které se objeví v příštím stabilním vydání. Nightly (Noční) vydání se vydávají každou noc [23].

Většina vývojářů primárně používá stable verzi, ale pro vývojáře, kteří chtějí vyzkoušet experimentální nové funkce, mohou použít nightly nebo beta [23].

3.2 Vlastnictví

Vlastnictví je unikátní funkce jazyka RUST, umožňující poskytovat záruky bezpečnosti paměti, aniž by potřeboval garbage-collection. Všechny programy musí řešit způsob jak využívat paměť během běhu, některé používají garbage-collection (např. C#, Java), který během běhu programu neustále hledá nepoužívanou paměť. U některých jazyků musí programátor explicitně alokovat a uvolňovat paměť (např. C). RUST používá třetí způsob, kdy paměť je spravována prostřednictvím systému vlastnictví se sadou pravidel, která kompilátor kontroluje v době kompilace. Žádná z funkcí vlastnictví nezpomaluje váš program, když běží. Pravidla [24]:

- Každá hodnota v RUSTu má proměnnou, která se nazývá vlastník
- V jednu chvíli může být pouze jeden vlastník.
- Když vlastník přejde mimo rozsah (scope), hodnota bude uvolněna.

[1] `https://www.rust-lang.org/`
[2] `https://research.mozilla.org/rust/`

3.3 Produktivita

Pro produktivitu vývojáře pomáhá [9]:

- **Rustup** – instalátor a upgrade toolchain
- **Cargo** – správce balíčků pro stahování závislostí, publikování kódu
- **Rustfmt** – nástor pro automatické formátování textu
- **Clippy** – nástroj na zachycení běžných chyb, zlepšení výkonu a čitelnosti

RUST si zakládá i na kvalitní, přívětivé dokumentaci a jednoduchém hledání publikovaných knihoven[3].

3.4 Webové frameworky

Následující sekce popisuje frameworky jazyka Rust pro webové aplikace a následně i frameworky pro práci s blockchainem a operacemi na eliptické křivce.

Framework Rocket

Rocket je webový framework pro programovací jazyk RUST. Tenhle framework má za cíl být flexibilní a „přátelštější“, tím se myslí, aby vývojář psal málo kódu ke splnění určitého úkolu/funkčnosti. S dalšími frameworky, které jsou popsány v dalších sekcích *Actix* a *Yew* paří mezi nejpoužívanější, avšak Rocket disponuje velkou komunitou a z velmi dobrou dokumentací. Pro začátečníka v jazyce RUST je Rocket nejideálnější volbou.

Rocket se snaží být konstruován pomocí třech základních pilířů:

- *Bezpečnost a korektnost*, kdy tyhle funkce jazyka by neměli být rozdílné na základě zkušenosti vývojáře, tedy „*i méně zkušený vývojář bude mít stejně bezpečnou a korektní aplikace jako zkušený vývojář*“. Rocket se snadno používá a zároveň dělá opatření, která zajistí, že bude aplikace bezpečná a správná bez větší režie [2].
- *Informace o zpracování požadavku by měly být typové a samostatné.* Web a HTTP jsou samy o sobě netypové, znamená to, že někdo nebo něco musí převézt řetězce na nativní typy. Rocket to udělá s nulovou programovací režií. Zpracování požadavků Rocketu je samostatné s nulovým globálním stavem, handlery (funkce propojená s nějakou událostí) jsou běžné funkce s argumenty [2].
- *Rozhodnutí by neměla být vynucená.* Šablony, serializace, relace a téměř vše ostatní jsou připojitelné, volitelné součásti. Rocket má oficiální podporu a knihovny jsou zcela volitelné a vyměnitelné [2].

Tera Jedná se o balíček frameworku Rocket, který se využívá pro vykreslení webu podle šablony. Do HTML kódu se přidají další znaky, podle kterých knihovna vykreslí danou stránku podle zadaných parametrů. Pro vývojáře je to velké zjednodušení, kdy může dynamicky poskytnou danou stránku pro uživatele bez větších problémů.

Pro podporu téhle knihovny je však nutné mít u souboru/šablony příponu *.tera* nebo *.hbs*, to rozhoduje, který engine bude při vykreslování použit. Pokud se šablona upraví

[3] `https://crates.io/`

v režimu *debug* (ladění), může se aktualizováním prohlížeče načíst změny, místo znovu sestavováním celé Rust aplikace.

Tera se především inspiruje ze šablonového systému *Jinja2* pro programovací jazyk Python, která umožňuje oddělit kód aplikace od její prezentace uživateli.

Framework Actix

Framework Actix[4]. Je výkonný, pragmatický a rychlý webový framework pro Rust postavený na modelu Actor[5], který umožňuje psát aplikace jako skupinu nezávisle se vykonávajících, ale spolupracujících objektů. Tyhle objekty se nazývají „Actor“ a komunikují výhradně výměnou zpráv.

Tenhle framework byl dlouho dobu neudržovaný a ztratil mnoho příznivců, kteří pak přešli na jiné frameworky. V poslední době však začala komunita Rustu na Actixu znova pracovat a očekává se, že si v brzké době najde svoje příznivce, zejména mezi zkušenějšími vývojáři. Kvůli dlouhodobé stagnaci na vývoji se však nedoporučuje pro nové vývojáře v Rustu.

Framework Yew

Framework Yew[6]. Je moderní framework jazyka Rust pro vytváření vícevláknových front-end webových aplikací pomocí WebAssembly. Obsahuje makro pro deklarování interaktivního HTML kódu. Je založený na komponentách, který usnadňuje vytváření interaktivních uživatelských rozhraní, minimalizuje volání DOM API a pomáhá vývojářům snadno přesouvat zpracování vlákna na pozadí. Podporuje efektivní a vzájemnou spolupráci s JavaScriptem. Často se používá s frameworkem Rocket, případně Actixem.

Tenhle framework je stále udržován a má i své příznivce, kteří ho používají. Komunita je však menší než například u Rocketu a proto se nachází i méně dostupných návodů. Dokumentace zde není moc podrobná, proto se též nedoporučuje pro nové vývojáře v Rustu.

3.5 Framework pro práci s blockhainem

Následující sekce popisuje frameworky pro práci s blockchainem, tedy zasílání transakcí, volání funkcí smart kontraktu, apod. Další alternativy nebyly uvažovány, neboť jiné frameworky pro práci práci s blockhainem nejsou již aktuální a skončili v brzké fázi svého vývoje – svědčí o tom počet stažení a poslední změny daného frameworku a navíc většina frameworků vychází z níže uvedeného Web3.

Framework Web3

Jedná se o framework podle protokolu Ethereum JSON-RPC Klient [8]. Je bezstavový a odlehčený protokol vzdáleného volání procedur (RPC). Tato specifikace definuje několik datových struktur a pravidla pro jejich zpracování. Jako datový formát se používá JSON. Na nasazení smart kontraktu (zobrazení nasazení znázorňuje obrázek 2.5 v sekci Ethereum 2.5) je zapotřebí vygenerovat binární kód a ABI kód (nízkoúrovňové rozhraní) smart kontraktu, pomocí kompilátoru Solidity[7].

[4] `https://actix.rs/`
[5] `https://www.brianstorti.com/the-actor-model/`
[6] `https://yew.rs/`
[7] `https://soliditylang.org/`

Framework ethcontract

Framework ethcontract[8] je založený na frameworku *web3*, pro generování typově bezpečných vazeb smart kontraktů. Interně generované typy používají funkce web3 k interakci se sítí Ethereum, používají vlastní run-time. Pro každý kontrakt se vytvoří samostatný soubor s kódem v Rustu. Tenhle framework se zejména využívá pro jednodušší implementaci a přehlednost kódu.

3.6 Framework pro práci s eliptickou křivkou

Framework Elliptic Curve[9] s knihovnou K-256 se jedná o framework, který podporuje obecnou kryptografii nad eliptickými křivkami (ECC – Eliptic Curve Cryptography), včetně typů a vlastností pro reprezentaci různých forem eliptických křivek, skalárů, bodů [7].

Knihovna K-256 neboli secp256k1 zahrnuje aritmetické funkce poskytující skalární a bodové typy (projektivní bod a afinní bod) s podporou pro skalární násobení. Secp256k1 je Koblitzova křivka běžně používaná v kryptoměnových aplikacích, především v bitcoinu a dalších kryptoměnách, zejména ve spojení s algoritmem ECDSA (Eliptic Curve Digital Signature Algorithm). Díky širokému nasazení v těchto aplikacích je secp256k1 jednou z nejoblíbenějších a běžně používaných eliptických křivek [4, 19].

[8] `https://docs.rs/ethcontract/latest/ethcontract/`
[9] `https://docs.rs/elliptic-curve/0.12.2/elliptic_curve/`

Kapitola 4

Hlasovací protokol

Hlasovací protokol 1-out-of-k, tedy možnost volby z více kandidátů/výběrů, poskytnutý vedoucí práce využívá smart kontrakty na blockchainové síti, takhle kapitola tedy bude vyházet z dané diplomové práce [21]. Tenhle protokol je verifikovatelný, škálovatelný a je též odolný proti neúčasti některých voličů, kteří se zaregistrovali. Tenhle protokol vychází z protokolu BBB-Voting.

Účastníci protokolu jsou:

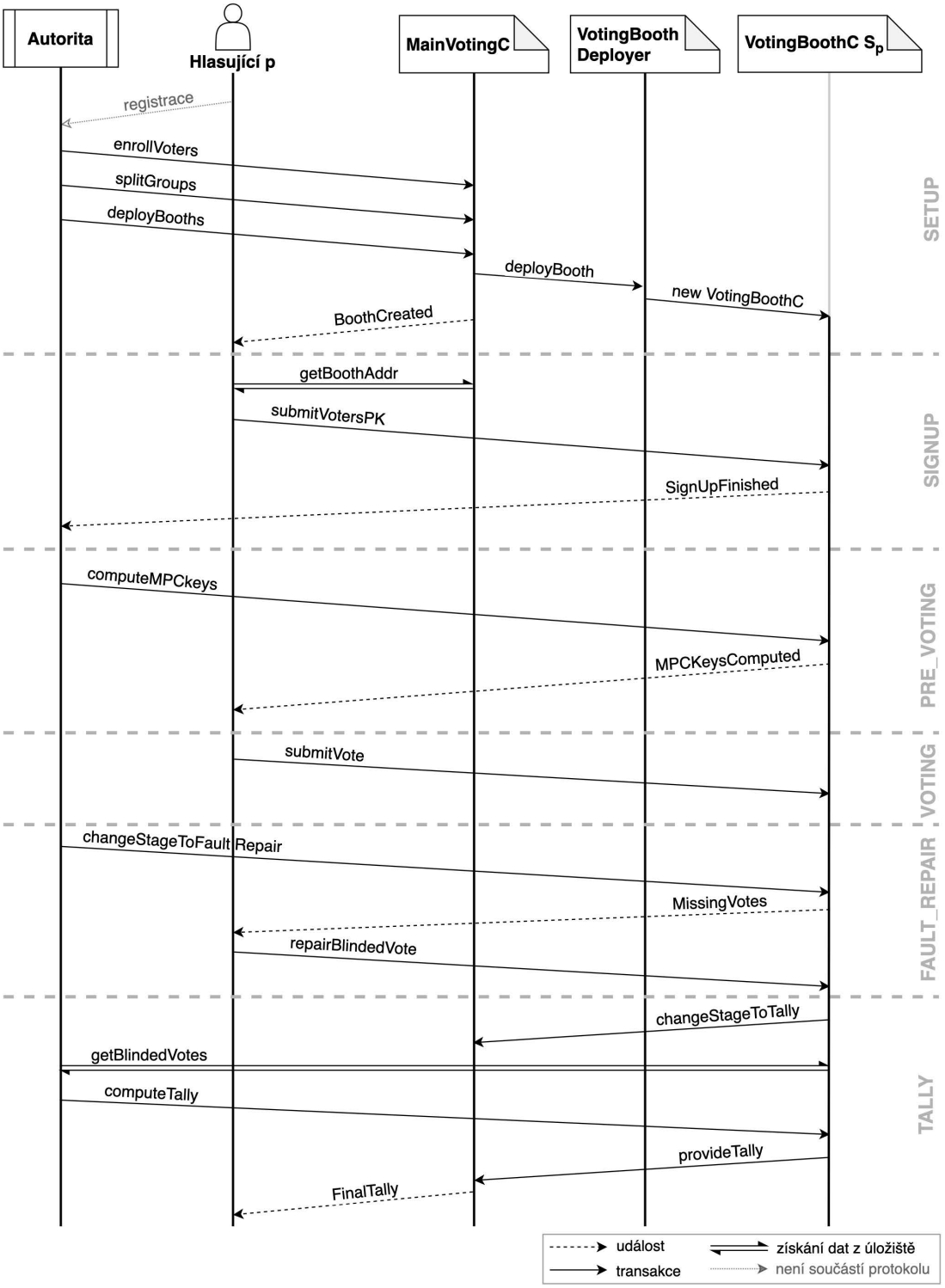
- **Autorita** je „správce“ daného hlasování. Zajišťuje registraci voličů, nasazení kontraktů a udává čas, ve kterém se hlasování provede. Zjistit hlas voliče a případně nějak zasahovat do výsledku voleb nikterak nemůže.
- **Hlasující** jsou zaregistrováni příslušnou autoritou a posléze jsou rozděleny do volebních okrsků. Pro účast musí hlasující poskytnout kryptografický klíč a dále pak při volbě hlasovací lístek.
- **Blockchain** slouží jako výpočetní platforma a dále i jako prostředek pro zveřejnění výsledků voleb. Veškeré informace o hlasování jsou volně dostupné na blockchainové síti.

4.1 Průběh hlasování

Takhle sekce popisuje jednotlivé fáze hlasovacího protokolu podle obrázku 4.1.

Příprava

Autorita odešle adresy jednotlivých hlasujících hlavnímu kontraktu (vykonává úkony zabezpečující chod a agregaci hlasování v jednotlivých volebních okrscích). Následně autorita zavolá funkci na rozdělení hlasujících do jednotlivých skupin. Hlavní kontrakt následně voliče náhodně rozdělí. Každý volební okrsek je tvořen samostatným smart kontraktem. Smart kontrakt volebního okrsku obsahuje počet přiřazených hlasujících, údaje o kandidátech a jiné kryptografické parametry, samotné adresy hlasujících neobsahuje. Tahle část je znázorněna na obrázku 4.1 jako *SETUP*.



Obrázek 4.1: Průběh hlasování a úkony účastníků v jednotlivých fázích protokolu [21]

Registrace

Hlasující získá adresu své skupiny pomocí hlavního kontraktu a následně musí do skupiny zaregistrovat svůj veřejný klíč. Jakmile jsou zaregistrovány všechny klíče, může se přejít do další fáze. Fáze registrace je znázorněna na obrázku 4.1 jako *SIGNUM*.

Parametry hlasování a klíče hlasujících Je-li možnost hlasovat mezi k kandidáty:

- Generátor G, což je bod na křivce nad tělesem $\mathbb{F}_p$, kde p je prvočíslo [21].
- k generátorů $\{F_1, ..., F_k\}$, bodů na křivce generující stejnou subgrupu jako G [21].

Veřejný a soukromý hlasovací klíč jsou vytvořeny podle následujících rovnic:

- soukromý klíč je náhodná hodnota $x_i \in_R \mathbb{Z}_n$, kde n je řád bodu G [21]
- veřejný klíč je hodnota $G \cdot x_i$ [21]

Výpočet veřejné části klíče

Po registraci všech veřejných klíčů hlasujících dané skupiny, může autorita vyvolat výpočet veřejných částí hlasovacích klíčů. Výpočet probíhá samostatně v každé skupině, následně si hlasující může z kontraktu získat veřejné klíče ostatních hlasujících a veřejnou část hlasovacího klíče si vypočítat lokálně, nebo může z kontraktu získat přímo výsledný klíč po ukončení této fáze (obrázek 4.1 *PRE VOTING*).

Veřejná část hlasovacího klíče Je-li v hlasovací skupině N hlasujících, tak pro každého hlasujících P_i se počítá jako:

$$G \cdot y_i = \sum_{j=1}^{i-1} G \cdot x_j - \sum_{j=i+1}^{N} = G \cdot x_j$$

Pro každého P_i je pak samotný hlasovací klíč roven $x_i y_i G$.

Hlasování

Hlasující si vytvoří hlasovací lístek a následně odešle do kontraktu své skupiny. Daný kontrakt zkontroluje korektnost lístku pomocí kryptografického důkazu. Pokud je ověření úspěšné, hlasovací lístek se uloží (obrázek 4.1 fáze *VOTING*). Pokud všichni registrovaní hlasující odevzdali hlasovací lístek, přesouvá se protokol do fáze *TALLY* (obrázek 4.1).

Hlasovací lístek Samotný hlasovací lístek má následně tvar podle níže uvedené rovnice, kde $F_j \in F_1, ..., F_k$ značí zvoleného kandidáta.

$$V_i = x_i y_i G + F_j$$

Spolu s tímto lístkem odešle každý hlasující také **zero-knowledge důkaz** („důkaz s nulovou znalostí“). Jedná se o prokázání, že jedna strana zná nějakou hodnotu bez toho, aby dala druhé straně jakoukoliv jinou informaci mimo tu, že danou hodnotu zná.

Oprava

Jestliže se někteří zaregistrovaní hlasující nezúčastní hlasování, tedy neodevzdají hlasovací lístek, není možné spočítat výsledek, proto je zde zapotřebí další práce ze strany hlasujících. Pokud tedy nastane situace, že v daném volebním okrsku nedošlo k odeslání hlasovacího lístku od všech registrovaných hlasujících, je zde zapotřebí „opravit“ hlasy, aby výsledek hlasování bylo možné správně vypočítat. Tuhle fázi (obrázek 4.1 *FAULT REPAIR*) vyvolává autorita.

Každý hlasující musí vytvořit opravné klíče (jeden za každého neaktivního hlasujícího) pro svůj hlasovací lístek a odeslat do kontraktu skupiny.

Vyloučení neaktivních hlasujících Pro každého aktivního hlasujícího P_i a neaktivního hlasujícího P_j je hodnota opravného klíče $x_i x_j G$, kde x_i je soukromý klíč aktivního hlasujícího a $x_j G$ je veřejný klíč neaktivního hlasujícího. Spolu s tímto opravným klíčem se poskytne i zero-knowledge důkaz korektnosti vypočítaného opravného klíče.

Výsledky

Výsledky hlasování jsou počítány pro kandidáty v rámci volebního okrsku zvlášť. Tenhle výpočet je však náročný pro provádění na blockchainové síti, proto je zde možnost zjistit výsledek pomocí výpočetního stroje autority, případně využít i externí výkonné výpočetní zdroje (například cloudu). V tomhle případě se provádí výpočet pomocí autority, která následně výsledky přepošle kontraktům k ověření správnosti.

Jakmile jsou výsledky za jednotlivé volební okrsky správně ověřeny, odešlou se do hlavního kontraktu, kde se následně agregují do celkového výsledku hlasování. Hlasujícím jsou celkové výsledky zpřístupněny, až když jsou do hlavního kontraktu přeposlány výsledky ze všech volebních okrsků. Fáze zpracování výsledků je znázorněna na obrázku 4.1 jako *TALLY*.

Výpočet výsledků Výsledné počty hlasů c_i, kde $i \in 1, ..., k$ pro jednotlivé kandidáty jsou vypočítány nalezením řešení rovnice.

$$\sum_{i=1}^{n} V_i = \sum_{i=1}^{n} x_i y_i G + F = c_1 F_1 + ... + c_k F_k$$

4.2 Struktura protokolu

Strukturu protokolu tvoří minimálně sedm smart kontraktů z toho 2 slouží jako knihovny kryptografických operací a smart kontrakt volební skupiny se vytváří pro každý volební okrsek, proto se při více okrsků může protokol skládat z více smart kontraktů.

Struktura dodaného hlasovacího protokolu protokolu:

- **Hlavní kontrakt** (MainVotingC) zajišťující zabezpečení chodu a agregaci výsledků v jednotlivých volebních okrscích.
- **Kontrakty skupin hlasujících** (VotingBoothC) jsou smart kontrakty spravující volební okrsky, kdy pro každý volební okrsek se vytvoří právě jeden.
- **Definice funkcí protokolu** (VotingFunc a VotingCalls) Kontrakty, kde jsou implementovaný funkce pro kontrakt VotingBoothC, aby se ušetřili množství kódu v onom

kontraktu, protože je třeba ho nasazovat ve více kopiích a výsledkem je tedy nižší cena. První definuje funkce zasahující do databáze blockchainu (on-chain). Druhý nemodifikuje databázi a funkce můžou být volány lokálně (off-chain). Oba tyhle kontrakty jsou vytvořeny pouze jednou, nehledě na počet kopií VotingBoothC.

- **Vytváření kontraktů skupin** prostřednictvím pomocného kontraktu VotingBoothDeployer, který má pouze za úkol na žádost hlavního kontraktu (MainVotingC) vytvořit kontrakt volebního okrsku (VotingBoothC).
- **Knihovny** EC a FastEcMul jsou určeny pro kryptografické operace.

4.3 Interakce hlasujícího

V rámci celého protokolu je uživatel nucen využít aplikaci nejméně dvakrát – při registraci veřejného klíče a tím zavázání se k účasti v hlasování a následně v samotném hlasování (výběrem kandidáta a zasláním hlasovacího lístku). Mezi těmito fázemi však může být delší časový úsek a hlasující tedy musí „usednou k počítači“ více než jednou a pro některé uživatele by tohle nemuselo být pohodlné. Při testování uživatelské použitelnosti (Kapitola 8) si uživatelé zkoušeli aplikaci lokálně a tedy nedošlo k velkému prodlení mezi jednotlivými fázemi.

Pokud by však alespoň jeden z některých hlasujících, který si zaregistroval veřejný klíč nezúčastnil hlasování, přešel by hlasovací protokol do opravné fáze, což by mělo za následek, že každý uživatel, který si úspěšně zaregistroval veřejný hlasovací klíč, by byl povinen „usednou k počítači“ ještě jednou, aby provedl opravu hlasovacích lístků, jinak by se nedal zjistit celkový výsledek hlasování. V celkovém počtu by tohle znamenalo, že ze strany uživatele je třeba ke třem interakcím s aplikací. V reálném světě by nejspíš volby podle tohoto protokolu trvali několik hodin nebo dokonce i dnů, proto nemusí být pro uživatele příliš pohodlné vykonávat tři interakce během daného časového úseku.

Kapitola 5

Návrh uživatelského rozhraní pro hlasovací protokol

Uživatelské hlasování pro výše uvedený hlasovací protokol zajišťující elektronické hlasování je „doplňkem“ ke stávajícímu hlasování, tedy papírového/osobního a očekávají se alespoň základní technické schopnosti uživatele. Nepředpokládá se, že se bude využívat elektronického hlasování podle zadaného protokolu jako majoritní. Dále je zde zapotřebí i uživatelské rozhraní pro danou autoritu zajišťující volby. V rámci práce bylo však nutností implementovat i klientskou část, nikoli pouze grafické rozhraní.

5.1 Základní požadavky

Základní požadavek pro uživatele je zejména správa svého „profilu“, ze kterého následně mohou volat různé funkcionality, zejména odevzdat svůj hlasovací lístek a tím správně hlasovat. Pro autoritu zejména základní požadavek je správa daných voleb, nasazení kontraktů s patřičnými parametry (například informace o kandidátech) a dále pak výpočty hlasovacích klíčů a nalezení výsledků voleb. Také zde bude patřit kontrola, v jakých fázích se hlasování nachází a jejich případné postoupení.

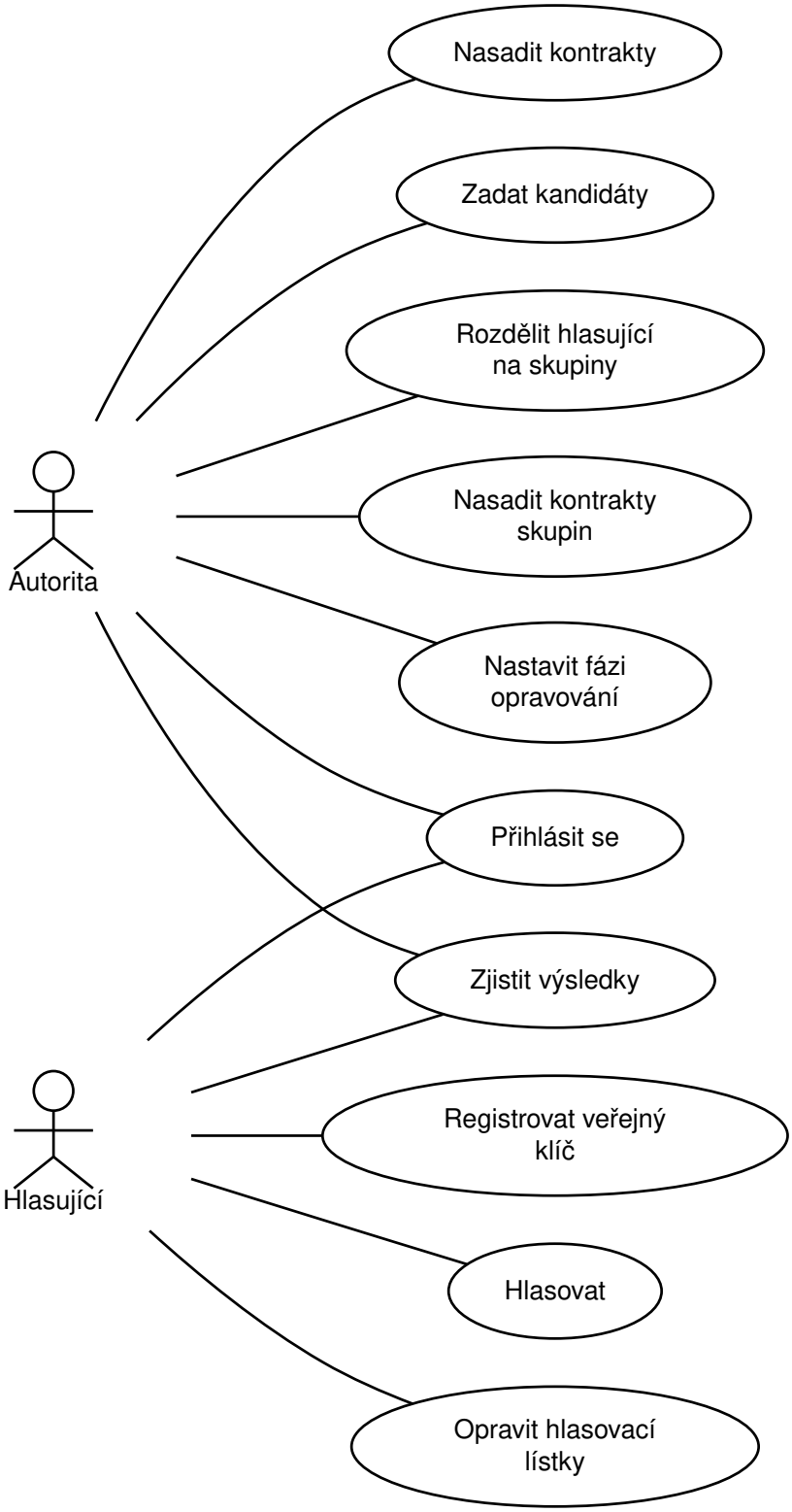
5.1.1 Autorita

Hlavní nabídka v rozhraní pro autoritu by měla zvládat nasadit dané smart kontrakty na blockchain i s danými parametry. Nastavit a následně odeslat adresy peněženek hlasujících, vyvolat rozdělení hlasujících do volebních skupin a následně nasazení kontraktů volebních skupin.

Autorita musí znát aktuální fázi hlasovacího protokolu u daných volebních okrsků, aby mohla následně interagovat s protokolem, například vypočítat veřejný hlasovací klíč k jednotlivým volebním okrskům, přesunout volební okrsek do „opravné fáze“, v závěrečné fázi spočítat výsledky hlasování za jednotlivé volební okrsky a jejich bezpečné odeslání danému kontraktu pro kontrolu správnosti.

5.1.2 Hlasující

Mezi nejdůležitější funkcionalitu je odevzdání hlasovacího lístku s patřičnými kandidáty. Tomu předchází i zjištění, do jakého volebního okrsku patří. Případně i „oprava“, pokud se daném volebním okrsku vyskytuje neaktivní hlasující.



Obrázek 5.1: Diagram návrhu užití uživatelského rozhraní

Zjištění celkového výsledků voleb je samozřejmostí a to i zpětná kontrola svého hlasovací lístku, zda byl korektně odeslán.

5.1.3 Funkční a nefunkční požadavky

Seznam základních funkčních a nefunkčních požadavků decentralizované aplikace.

Funkční požadavky autority

- Přihlášení autority ke svému profilu
- Přidání kandidátů pro hlasování
- Přidání adres peněženek hlasujících
- Rozdělení a nasazení skupin hlasujících
- Výpočet hlasovacích klíčů
- Změna fáze na „opravnou“ při neúčasti některých hlasujících
- Nalézt výsledek hlasování

Funkční požadavky hlasujícího

- Přihlášení hlasujícího ke svému profilu
- Registrovat svůj veřejný hlasovací klíč
- Oprava hlasovacího lístku pokud je potřeba
- Odeslání zašifrovaného hlasovacího lístku
- Zjistit celkové výsledky hlasování

Nefunkční požadavky

- Použití lokálního Ethereum blockchainu
- Aplikace se skládá pouze z částí, které fungují distribuovaným způsobem (žádné využití centralizovaného systému)
- Implementace v Rustu
- Funkční na operačním systému Windows
- Využití designu státní správy

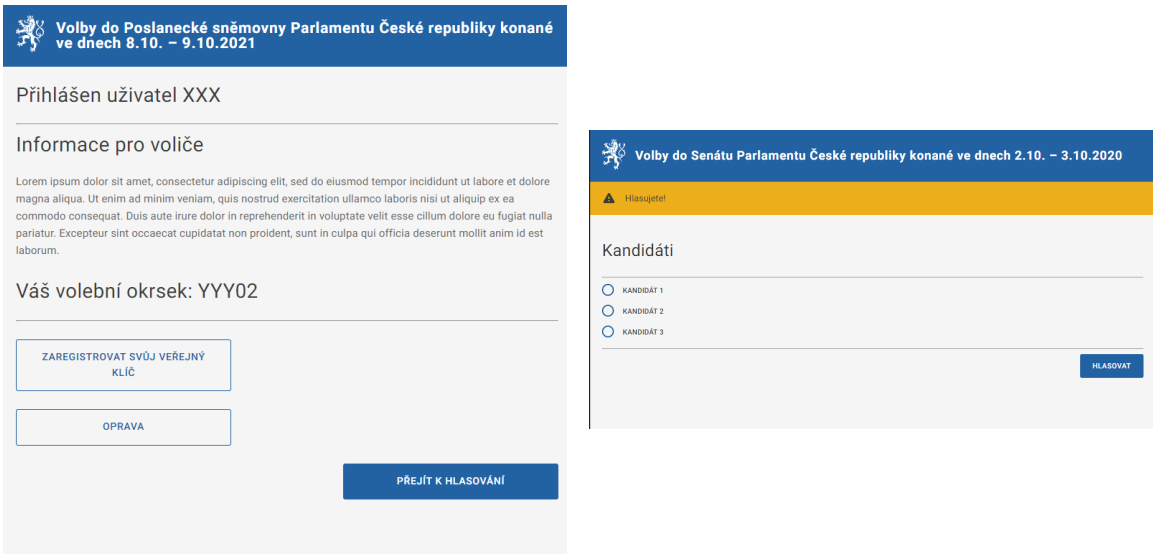
5.2 Vizualizace

V rámci návrhu uživatelského rozhraní bylo potřeba vytvořit grafické vizualizace dané aplikace pro hlasovací protokol. Vizualizace byla navržena pouze v podobě „maket“, tedy grafický design bez handlerů. Vizualizace byla vytvořena pomocí prostého HTML kódu a kaskádových stylů.

Podle komponent (zejména tlačítek u formulářů) se následně vytvořil návrh na případné volání aplikace pomocí HTTP metod POST a GET, které by korespondovali s posloupností a názvy hlasovacího protokolu podle grafu 4.1. Snahou bylo vytvořit, aby jedno kliknutí na tlačítko se rovnalo jedné transakci, případně získání dat z blockchainu podle hlasovacího protokolu. V rámci implementace, o které více pojednává kapitola 6 se vizualizace během vývoje částečně změnila do pohodlnější verze, zejména ve snížení počtu tlačítek a různých interakcí na jedné stránce.

5.2.1 Předběžné návrhy

Návrhy vycházejí podle designu *designsystem.gov.cz*[1]. Tenhle design systém je scénář, podle kterého Ministerstvo vnitra navrhuje a vyvíjí weby a digitální produkty. Design systém má pomoci rychleji vytvářet digitální produkty, které budou konzistentní napříč veřejnou správou [11]. Návrh na profil hlasující (hlavní stránka, ze které bude interagovat s hlasovacím protokolem, obrázek 5.2) a dále návrh na stránku s výběrem kandidáta (obrázek 5.2).



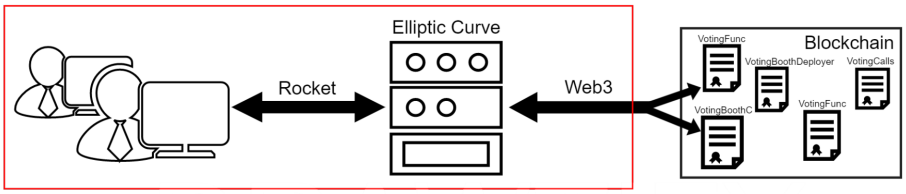
Obrázek 5.2: Návrh „profilu“ hlasujícího a Návrh na hlasování mezi kandidáty

[1] `https://designsystem.gov.cz/`

Kapitola 6

Implementace

Takhle kapitola popisuje implementaci decentralizované webové aplikace. V rámci práce bylo stěžejním zadáním vytvořit uživatelské rozhraní, avšak při implementaci bylo zapotřebí implementovat i funkční klientskou část, kterou budou popisovat následující sekce. V rámci práce decentralizované aplikace bylo zapotřebí implementovat vnitřní uspořádání klientské části, aby aplikace korektně komunikovala s blockchainem. Schéma aplikace s využitím jednotlivých frameworků v níže uvedeném obrázku 6.1.



Obrázek 6.1: Návrh aplikace, kde v červeném ohraničení je klientská aplikace, která musela být implementována s popisem využívání jednotlivých frameworků. Rocket mezi komunikací uživatele s aplikací, Elliptic Curve k vnitřním výpočtům a nakonec Web3 pro komunikaci s blockchainovou sítí.

6.1 Použité frameworky

Při implementaci aplikace se použil webový framework Rocket, zajišťující chod webové aplikace a následně i zajištění zobrazení a vygenerování uživatelského rozhraní za pomocí balíčku Tera. Pro práci s blockchainem, tedy odesílání transakcí, zpracování, volání funkcí smart kontraktů atd. se využil framework ethcontract, který mimo jiné vygeneroval typově bezpečné vazby smart kontraktů. Tenhle framework je však založen na frameworku Web3, který využívá. Pro práci na eliptické křivce byl využit framework Elliptic Curve s knihovnou K-256, který se především uplatnil u fáze hlasovacího protokolu VOTING a FAULT REPAIR při práci s projektivními, afinními body, jejich složení a dekompozicí na souřadnice, případně i doplnil další datové typy, které se v RUSTu nenacházejí.

6.2 Rozdělení aplikace

Aplikace se skládá z několika souborů, přičemž hlavní soubor *main.rs*, který obsahuje jak spouštěcí metodu aplikace, tak funguje i jako „rozcestník“. Pokud uživatel vykoná akci, daný soubor ji zachytí a následně ji přepošle dalším vnitřním metodám. Zachycení daných požadavků se vykonává pomocí HTTP metod, zejména GET a POST.

Dále se používá pro dočasné ukládání cookies. Soubory obsahující html kód mají koncovou příponu *.tera*, které se nachází ve složce *templates* slouží, aby balíček Tera frameworku Rocket rozpoznal, kde se mají nacházet dané „značky“ a ty následně zpracovávat na příslušnou hodnotu, aby se úspěšně vygenerovala daná stránka. Složka *static* obsahuje neměnné soubory – soubory obsahující javascript, kaskádové styly, ikony, atd.

Soubory aplikace jsou přehledně rozděleny podle fází protokolu, pokud by se našla nějaká chyba v určité fázi protokolu, lehce se může dohledat soubor definující danou fázi a zde následně provést úpravu kódu. Mezi další soubory patří soubor na zachytávaní případných chybových stavů (500, 404) vracející stránku s chybou. Pomocný (servisní) soubor obsahující struktury, traity, pomocné metody a soubor obsahující testy (více v sekci o testování 6.6).

Soubor *generate_binding.rs*, při spuštěni vygeneruje podle zadaných kontraktů ve formátu *json*, který vznikne při kompilaci Solidity překladačem, alternativní API pro generování typově bezpečných vazeb smart kontraktů. Tohle generování má výhodu pro podporu automatického doplňování a schopnost kontrolovat vygenerovaný kód. Bez toho by se vývojáři pracovalo obtížněji a kód by se v některých případech stal nepřehledným.

6.3 Lokální Blockchain

Testování aplikace hlasovacího protokolu a spouštění by bylo na blockchainu Ethereum velmi nákladné, proto se k testování využívá spuštěný blockchain na lokální síti pomocí aplikace *Ganache*[1]. Tenhle program dokáže spustit lokální Etherový blockchain s danými parametry a s vygenerovanými adresami peněženek.

6.4 Klientská část

V rámci vývoje aplikace byla zvolena verze RUSTu *nightly*, protože některé knihovny z použitých frameworků byly kompatibilní pouze s touhle verzí, zejména s využitým webovým frameworkem Rocket. Dále se pro práci s decentralizovaným systémem použil framework ethcontract a s operacemi na eliptické křivce Elliptic Curve (práce s afinními a projektivními body, skaláry, konstanty křivky secp256k1, datové typy, ...). V rámci práce bylo pro pohodlnější testování implementovat i nasazení smart kontraktů dle volebního protokolu, tedy jednodušší rozhraní pro volební autoritu a následně i dané operace pro autoritu a hledání výsledků hlasování.

SETUP

Nejprve bylo nutné nasadit smart kontrakty s danými parametry a následně z fáze *SETUP* tedy bylo zapotřebí vytvořit transakci *enrollVoters*, pro odeslání peněženek hlasujících hlavnímu kontraktu, transakci *splitGroups* pro rozdělení hlasujících do volebních skupin a

[1] `https://trufflesuite.com/ganache/`

transakci *deployBooths* pro nasazení smart kontraktů jednotlivých volebních skupin. Adresy smart kontraktů volebních skupin si uložíme, stejně jako všechny ostatní adresy nasazených smart kontraktů hlasovacího protokolu.

SIGNUP

Ve fázi *SIGNUP* bylo zapotřebí získat adresu smart kontraktu volební skupiny daného uživatele, který se přihlašuje, což se získává z blockchainové sítě, kdy zavoláním funkce *votersGroup* hlavního kontraktu s argumentem adresy peněženky přihlašujícího uživatele získáme ID volebního okrsku. Pomocí funkce hlavního kontraktu *groupBoothAddr* s argumentem získaného ID uvedené v předchozí větě, získáme následně adresu volebního okrsku kam uživatel patří. Aby nedošlo k nějakým nežádoucím chybám a správnému vygenerování uživatelského rozhraní je zapotřebí zkontrolovat, jestli se uživatel již nezaregistroval k účasti v hlasování, což se provádí zavoláním funkce *votersWithPk* smart kontraktu volební skupiny. Jako argument dané funkce se využije adresa peněženky přihlášeného uživatele a získává se Booleova hodnota, která značí, jestli se již registroval či nikoli. Pokud se již zaregistroval k hlasování, získá se ID hlasujícího v dané volební skupině pomocí funkce kontraktu volební skupiny *votersPkIdx*. Pomocí takhle získaného ID voláme funkci kontraktu volební skupiny *getBlindedVote*, kterou se případně získá hlasovací lístek (pokud uživatel již hlasoval) a vygeneruje se patřičná stránka.

Na stránce po úspěšném přihlášení a pokud uživatel ještě nehlasoval se vyskytuje tlačítko pro registraci, které vyvolá následující akce. Tahle část vychází z rovnice uvedené v kapitole 4.1 v sekci *Registrace*. Vygeneruje se náhodné 256bitové celé číslo bez znaménka (zkratka U256). Rust však nepodporuje generování U256, proto se vygeneruje pole o velikosti 32 prvků, kdy každý prvek obsahuje vygenerované náhodné 8bitové číslo bez znaménka pomocí knihovny jazyka Rust a následně se poté pole převede na U256. Dále se získá řád grupy eliptické křivky secp256k1. Pokud je náhodné U256 větší než řád grupy, dělíme ho dvěma do doby, než bude menší. Až je menší než řád grupy, získal se výsledek privátního hlasovacího klíče voliče. Následně se pro voliče musí vygenerovat veřejný hlasovací klíč, který se vypočítá skalárním součinem bodu G (generátor cyklické grupy) a privátního hlasovacího klíče. Následně se zasílá transakce do smart kontraktu volební skupiny *submitVotersPk*, kde jako argument se dává x souřadnice a y souřadnice veřejného hlasovacího klíče. Po úspěšné transakci, voláme funkci *votersPkIdx*, kde se získá ID hlasující ve volební skupině a ono ID se zasílá na uživatelské rozhraní, kde se hlasujícímu zobrazí.

PRE VOTING

Ve fázi *PRE VOTING* musí autorita vyvolat výpočet veřejných částí hlasovacích klíčů všech hlasujících ve všech smart kontraktech volebních skupin. Nejdříve aplikace zkontroluje, zda se protokol nachází ve fázi *PRE VOTING*, což se zjišťuje pomocí funkce *Stage* smart kontraktu volební skupiny. Pokud se daný protokol nachází v dané fázi, zašle se transakce *buildRightMarkers4mpc* do smart kontraktu volební skupiny. Event *rightMarkersComputed* informuje o úspěšném předpočítání hodnoty pro další výpočet pro získaní veřejných částí klíče. Pokud je předpočítání úspěšné, získá se voláním funkce *mpcBatch* a *getCntOfVoters* smart kontraktu volební skupiny velikost skupiny a počet voličů. Tyhle hodnoty se následně využívají v předvýpočtu pro získání veřejné části klíče.

Získání veřejných hlasovacích klíčů je rozděleno do dvou částí. První část je **předvýpočet**, kdy se volá funkce *modInvCache4mpcBatched* smart kontraktu hlasovací skupiny, která vrátí hodnoty pro výpočet veřejné části klíči. Druhá část je samotný **výpočet** veřejných

částí klíčů. Odesílá se transakce *computeMpcKeys* s hodnotami získané v předvýpočtu. U předvýpočtu a samotného výpočtu se využívá dávkové zpracování pomocí volání uvedených funkcí.

VOTING

Ve fázi *VOTING* se uživateli zobrazí tlačítko na hlasování, které se přesměruje na stránku s výběrem kandidátů, v rámci téhle akce se zjišťuje, jestli volič již nehlasoval, obdobně jako ve výše uvedené sekci SIGNUP 6.4. Informace o kandidátech se získají tak, že se nejdříve získá počet kandidátů voláním funkce *getcntOfCandidates* hlavního kontraktu a následně iterujeme do počtu kandidátů. V iteracích se volá funkce *candidates*, kde jako argument se zadává index iterace (první iterace je 0, druhá iterace 1, atd.) a získává se tím informace o kandidátovi. Kandidáti se následně přepošlou na uživatelské rozhraní, mezi kterými si hlasující vybere a následně klikne na tlačítko „hlasovat“, které spustí následující akce.

Vytvoření volebního lístku Následující odstavce téhle sekce popisují implementaci zašifrování hlasu, podle rovnice uvedené v sekci *Hlasování* v kapitole 4.1. Při těchto operací je nutné pracovat s projektivními, afinními body, skaláry, patřičné datové typy a konstanty dané eliptické křivky. Tyhle možnosti a operace nad danými typy zajišťuje framework Elliptic Curve. Aplikace musí získat veřejný hlasovací klíč daného voliče. Voláním funkce *mpcPks* a *votersPks* s argumentem ID hlasujícího a následně argumentem 0 pro získání x souřadnici a argumentem 1 pro získání y souřadnici se získá veřejná část hlasovacího klíče voliče a veřejný klíč hlasujícího. Následně se získá generátor kandidáta voláním funkce *candidateGens* s patřičným ID kandidáta, pro kterého hlasující hlasoval.

Následně veřejný hlasovací klíč se vynásobí pomocí soukromého hlasovacího klíče voliče převedeného na skalár, tím se získá projektivní bod. Daný bod se následně sečte s patřičným generátorem kandidáta, získaného v předchozím odstavci. Výsledkem je zašifrovaný hlas, který se označí $bVote$.

Vypočítání prvků důkazu V rámci korektní odeslání transakce a tedy hlasování pro kandidáta je zapotřebí vypočítat prvky důkazu, že na onom hlasovacím lístku je zašifrován jeden z možných kandidátů. Provede se pomocí zero-knowledge důkazu korektnosti hlasovacího lístku při využití eliptických křivek.

FAULT REPAIR

Pokud se autorita rozhodne nastavit fázi FAULT REPAIR u některých volebních skupin, vykoná patřičnou akci stisknutím tlačítka. Do zvolené volební skupiny zašle autorita transakci *changeStageToFaultRepair*, kterým se změní fáze na FAULT REPAIR u dané volební skupiny. Autorita, případně i hlasující může pomocí eventu *missingVotesEvent* získat ID zaregistrovaných hlasujících, kteří však neodevzdali hlasovací lístek.

Pokud se daná volební skupina nachází ve fázi FAULT REPAIR zobrazí se uživateli tlačítko na opravu hlasovacích lístků, které vyvolá následující akce. Následující část se váže na výpočet uvedený v kapitole 4.1 v sekci *Oprava*. Definuje se vektor $faultyVotersPks$ a eventem *missingVotesEvent* se získá pole ID uživatelů, kteří neodevzdali hlasovací lístek. Dané pole se následně iteruje, kdy je nutností získat u každého patřičného uživatele jeho veřejný hlasovací klíč, což se provádí pomocí volání funkce *votersPks*. Veřejný hlasovací klíč se vkládá do $faultyVotersPks$.

Následně se iteruje $faultyVotersPks$, kdy u každé iterace se získá projektivní bod jako součin daného bodu z $faultyVotersPks$ a privátního hlasovacího klíče opravujícího uživatele, převedeného na skalár. Takhle získaný bod slouží pro odečtení od hlasovacího lístku voliče, čímž se lístek opraví, aby šel správně sečíst výsledek. Další nutností je vypočítat Zero-Knowledge důkaz.

Kvůli optimalizaci, tedy převedení do podoby, se kterým smart kontrakt je schopen pracovat se využívá funkce *decomposeScalar* kontraktu FastEcMul a funkce *modInvCache4-RepairVote*. Oprava hlasovacího lístku se provede po odeslání transakce *repairBlindedVote*. O následné opravě se informuje pomocí eventu *repairedBvoteEvent*.

TALLY

Pokud se nachází hlasovací protokol v poslední fázi, může autorita spustit kliknutím na tlačítko dané akce k nalezení výsledků hlasování. Iteruje se seznam adres volebních skupin, kdy u každé iterace probíhá kontrala, zda je daná volební skupina ve fázi TALLY, což se ověřuje voláním funkce *boothsInStageTally* hlavního kontraktu, kde argument je adresa dané volební skupiny, ta nám vrátí Booleovu hodnota. Funkcí *boothTallyCollected* hlavního kontraktu s argumentem volební skupiny, která získáme Booleovu hodnotu značící, že u dané volební skupiny již byl vypočítán výsledek hlasování pro danou volební skupinu.

Pokud se tedy nachází volební skupina ve fázi TALLY a ještě nebyl vypočítán výsledek, tak se dále pokračuje získáním počtu hlasujících v dané skupině voláním funkce *getCntOf-Voters*. Definuje se vektor $blindedVotes$. Následně iterujeme do počtu voličů v dané skupině a v každé iteraci se volá funkce *getBlindedVote* kontraktu volební skupiny s argumentem ID daného uživatele, kterým se získá hlasovací lístek. Pokud je získaný hlasovací lístek prázdný, tedy má souřadnici x a y rovno 0, značí, že se jednalo pouze o zaregistrovaného uživatele, který však nehlasoval. Pokud tedy budou souřadnice rozdílné od 0, značí, že uživatel hlasoval a daný hlasovací lístek se přidává do vektoru $blindedVotes$. Dále se posílá transakce *computeBlindedVotesSum*, kterým se sečtou hlasovací lístky v zašifrované podobě a o úspěchu dané akce se informuje pomocí eventu *blindedVotesSumComputed*.

Následně se získává počet kandidátů pomocí funkce *getCntOfCandidates* a délka vektoru $blindedVotes$ a provede se výpočet všech možných výsledků. Hledání výsledku se provádí mezi všemi možnými výsledky (exhaustive search) což je vlastností protokolu a jiná možnost neexistuje. Implementace se prováděla za pomocí cyklu, kdy každý krok cyklu vezme jeden z možných výsledků, což je pole, kdy na každém indexu v poli je potenciální počet hlasů pro kandidáta. Tyhle hodnoty se sečtou a pokud se rovnají počtu odevzdaných hlasů, pokračuje se. Pokud se nerovnají, tenhle krok končí a pokračuje se v dalším kroku, kdy se vezme další možný výsledek.

Následující odstavce vycházejí z rovnice uvedené v sekci *Výsledky* v kapitole 4.1. Pokud se tedy rovnají, volá se pomocná funkce *sumOfBvotes* aplikace, kde se definuje hodnota sum s výchozí hodnotou bodu G a následně se iteruje do velikosti vektoru $blindedVotes$, kdy se při každé iteraci získá souřadnice x a y daného volebního lístku (hlasovací lístek na pozici indexu iterace vektoru $blindedVotes$) a následně se dané souřadnice převedou na projektivní bod p. Na konci iterace se k hodnotě sum přičte p. Po ukončení všech iterací funkce *sumOfBvotes* vrátí hodnotu sum.

Následně se volá pomocná funkce aplikace *expCandGens*, kde se definuje hodnota sum s výchozí hodnotou bodu G následně se iteruje do počtu generátorů kandidátů. V iteraci se nejdříve zjistí, jestli platí že potenciální možný výsledek na indexu iterace je roven 0, pokud je pokračuje se v další iteraci. Pokud není rovno nule, spočítá se součin generátoru kandidáta

na indexu iterace a potenciálního možného výsledku na indexu iterace, převedeného na skalár a získává se hodnota *prod*, následně se k hodnotě *sum* přičte *prod*. Po konci iterací funkce *expCandGens* vrátí hodnotu *sum*. Pokud je vrácená hodnota z funkce *sumOfBvotes* a z funkce *expCandGens* stejná, našel se výsledek hlasovaní v dané volební skupině. Pokud je hodnota různá, značí, že se nenašel a je nutné pokračovat dalším krokem cyklu, tedy vzít další možný výsledek a provést předchozí akce znovu.

Kvůli optimalizaci, tedy převedení do podoby, se kterým smart kontrakt je schopen pracovat se využívá funkce funkce *decomposeScalar* kontraktu kryptografických operací FastEcMul a funkce *modInvCache4Tally* kontraktu hlasovací skupiny. Získaný výsledek se následně zasílá transakcí *computeTally* kontraktu hlasovací skupiny, která ověří jeho správnost vůči množině odevzdaných hlasovacích lístků.

Pokud jsou všechny volební skupiny zpracovány jak uvádějí předchozí odstavce, zavolá se funkce hlavního kontraktu *getFinalTally* čímž se získá vektor $finTally$ s hodnotami agregovaných výsledku hlasování napříč volebními skupinami jako rozložené skaláry. Finální operací je tedy definování vektoru $finTallyStr$ a následnou iterací do počtu kandidátů. V iteraci se vezme hodnota z vektoru $finTally$ nacházející se na indexu dvojnásobku indexu iterace (bi_a) a hodnota z vektoru $finTally$ nacházející se na indexu dvojnásobku indexu iterace plus jedna (bi_b). Do vektoru $finTallyStr$ se následně přidá hodnota vypočítaná jako $bi_a + bi_b * lambda \mod n$, kde n je řád grupy eliptické křivky secp256k1. Až se provedou všechny iterace, je ve vektoru $finTallyStr$ uložen celkový výsledek hlasování, kdy na každém indexu je počet hlasů pro daného kandidáta.

6.5 Uživatelské rozhraní

Na hlavní stránce webové aplikace jsou uvedeny základní informace o volbách a jednoznačný navigační bar a zápatí. Design a rozložení jednotlivých prvků se řídí podle balíčku *designsystem.gov.cz*[2], jedná se o scénář, který má pomoci při digitalizaci České republiky (návrhy, vývoj webu a další digitální produkty), aby státní organizace měli v budoucnu jednotný a přívětivý design a stejné rozestavění prvků na stránce. Tenhle design systém se již používají na některých webových stránkách jako například Portál veřejné správy, Covid Portál, případně i v mobilních aplikací Tečka, čTečka.

Protože se aplikace zabývá hlasováním na blockchainu a tenhle systém hlasování by případně mohl mít v budoucnu přesah do národních věcí (volba do senátu, volba prezidenta, volby v organizacích – například na předsedu dané organizace, . . .) je použití daného design systému adekvátní. Pokud by se v budoucnu změnil design systém, nemělo by to mít velký dopad na funkčnost aplikace.

Mezi základními pravidly daného balíčku je zobrazení navigačního baru, těla a zápatí. V navigačním baru vlevo nahoře zobrazen státní znak a dále titulek. Na některých stránkách (například `https://gov.cz/` se zobrazuje tlačítko „PŘIHLÁSIT SE DO PORTÁLU OBČANA“, tenhle systém by případně mohl v budoucnu využívat tahle aplikace pro přihlášení do hlasování, aby autorita mohla spolehlivě zaregistrovat daného voliče (v registru obyvatel by případně mohlo vzniknou přidání veřejné peněženky).

V zápatí jednoduché informace, jako kontakty na dané instituce (například na média), zpracování osobních údajů a cookies, odkazy na sociální sítě, apod. Balíček také má pravidla pro případnou modifikaci, jako například barevné škály. Vztahy mezi barvami se nesmí významně změnit — poměr, kontrast, sytost. Nesmí se měnit typografie — font, vzhled. Ne-

[2] `https://designsystem.gov.cz/`

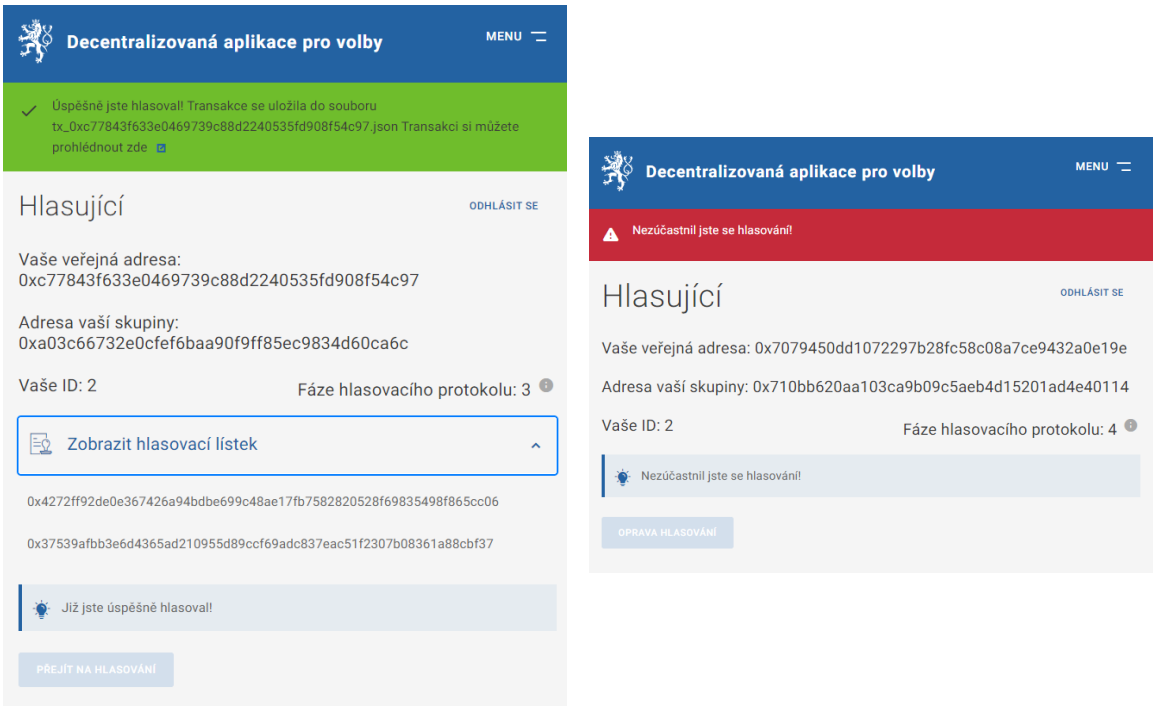
smí se měnit vzhled a funkce existujících komponent design systému, jako příklad zakázané modifikace se uvádí přepínač ve vertikální poloze [12].

Hlasující

V rámci profilu hlasujícího musí být jednoznačné informace o dění hlasování a aby hlasující věděl a pochopil, co se právě děje. Na profilu ze zobrazují základní informace – adresa volební skupiny, ID hlasujícího po odeslání hlasovacího klíče, veřejná adresa hlasujícího a fáze hlasování protokolu.

Tlačítka pro odevzdání hlasovacího klíče, jedná se „zavázání se k účasti v hlasování“, které pošle do smart kontraktu veřejný klíč hlasujícího, jakmile se hlasovací klíč uloží, získáme jeho ID, které přepošleme na profil hlasujícího. Soukromý klíč se uloží do souboru *sec_key_{veřejná adresa hlasujícího }.txt*, který se následně bude používat pro hlasování. Je nutné soukromí klíč neztratit/nesmazat neboť hlasování by daný hlasující nemohl dokončit. Pokud hlasující se již odevzdal hlasovací klíč, tlačítko se stane neaktivním a nedá se již dále používat. Na profilu se v tenhle moment zobrazuje fáze 1.

Po úspěšném zaregistrování hlasovacích klíčů ode všech hlasujících tlačítko zmizí, fáze se posune na úroveň 2 a čeká se na autoritu, až provede určitý krok v rámci výpočtu veřejných klíčů hlasujících. Po výpočtu se zobrazuje fáze 3 a tlačítko „přejít na hlasování“, jehož funkcí je přesměrovat uživatele na stránku s upozorněním, že se jedná již o pevné hlasování a s jednotlivými kandidáty, kde si uživatel vybere jednoho pomocí přepínače a



Obrázek 6.2: Na levé straně je profil hlasující po úspěšném odhlasování. Na obrázku jde vidět upozornění se souborem transakce, odkaz na prohlížeč blockchainu a rozkliknutý panel, který zobrazí hlasovací lístek. Na pravé straně je profil hlasujícího, který zaregistroval svůj hlasovací klíč, ale nehlasoval. Není zde tedy zobrazení hlasovacího lístku, tlačítko je neaktivní, pouze se zobrazuje upozornění ohledně neúčastni v hlasování.

následně klikne na tlačítko odevzdat a vyčká na potvrzení o volbě. V případě, pokud volič si stále není jistý volbou, může kliknout na tlačítko „zpět“, což ho nasměruje zpátky na profil.

Volba probíhá pomocí odeslání formuláře pomocí metody POST. Formulář obsahuje přepínače (anglicky „radio buttons“) se jmény kandidátů. Do aplikace se odesílá ID kandidáta a následně se poté zpracovává hlasování o které pojednávala část VOTING v sekci Klientská část 6.4. Po úspěšném hlasování se přejde na profil hlasujícího s upozorněním o úspěšném hlasování a následně o informaci, že se vygeneroval soubor s danou transakcí ve formátu json, kde si uživatel může zkontrolovat a zjistit dané informace například hash transakce, který může vyhledat v prohlížeči blockchainu (blockchain explorer). V upozornění se zobrazí i odkaz, který po rozkliknutí zobrazí blockchainový prohlížeč s danou transakcí. Na profilu se zobrazí hlasovací lístek, upozornění, že již volič hlasoval a tlačítko pro hlasování se stane neaktivním (Příloha B.1).

V případě, že všichni zaregistrovaní hlasující nehlasovali a autorita přesune hlasovací protokol do 4. fáze, je ze strany uživatele vyžadováno, aby provedl opravu hlasů. Na profilu se zobrazuje zvýrazněná informační hláška a tlačítko na opravu, pokud hlasující úspěšně opraví hlas, je o tom informován hláškou a tlačítko se stane neaktivním. Ze strany uživatele tohle byla poslední přímá interakce s protokolem, neboli se smart kontraktem. V případě, že profil bude hlasujícího, který však neodevzdal hlasovací lístek a kvůli němu byla spuštěna 4. fáze (Příloha B.1).

Pokud jsou všechny lístky opraveny nebo nebylo potřeba opravovat, protože všichni hlasovali, tak ze strany hlasujícího již žádné další interakce nejsou. Vyčkává se pouze na zveřejnění výsledků.

Kontrakty

Při nasazení kontraktů autoritou se vygeneruje soubor *address_of_contract.json*, kde jsou ve formátu *json* uloženy adresy daných kontraktů. V rámci aplikace pak hlasující tyhle adresy používá a komunikuje s nimi. Při spuštění aplikace se soubor do aplikace automaticky načte. Autorita však musí zveřejnit daný soubor, aby hlasující znali adresu hlavního kontraktu od kterého zjistí adresu kontraktu volební skupiny.

Při zadání adres peněženek hlasujících se dané adresy uloží do souboru *address.json*. V rámci aplikace se nepoužívá a slouží jako informační soubor (zejména pro testování vývojářem).

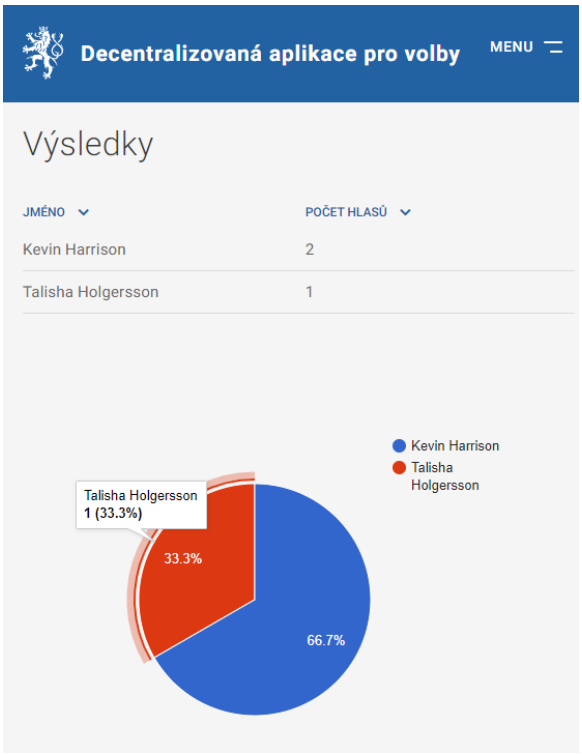
Opravná fáze

Pokud se autorita rozhodne spustit opravnou fázi hlasovacího protokolu, uloží do souboru *not_voted_idxs.json* ID neaktivních hlasující, tedy těch, kteří se zavázali k hlasování, ale nehlasovali. Tento dokument podobně jako ve výše uvedené kapitole 6.5 slouží pouze jako informační soubor. Při opravě hlasovacích lístků hlasujícím aplikace vyčte na blockchainu ID neaktivních hlasujících a následně začne opravu.

Uživatelům se zobrazuje fáze 4 a dále tlačítko „Oprava hlasování“, které po úspěšném opravě stane neaktivním. Opravná fáze je tedy (pokud ji budeme uvažovat) posledním případným krokem ze strany hlasujícího.

Výsledky

Na stránce celkových výsledků se vygeneruje jednoduchá tabulka se jmény kandidátů a jejich počty hlasů. Tabulka se následně může libovolně seřadit pomocí záhlaví tabulky. V případně, že výsledky nejsou stále k dispozici, zobrazí se hlášky s danou informací.



Obrázek 6.3: Výsledky celkového hlasování

V rámci testování pomocí uživatelů, byl následně pod tabulku implementován koláčový graf (obrázek 6.3) zobrazující celkové výsledky hlasování s možností rozkliknutí a následně zobrazení údajů. Více o koláčovém grafu v části 8.1.

Hledání výsledků je výpočetně náročné. V rámci protokolu hledáme mezi všemi možnými výsledky ten správný (exhaustive search), což je vlastností hlasovacího protokolu, protože šifrování v protokolu neumožňuje získání výsledku jiným možným způsobem. V aplikaci tenhle výsledek hledáme pomocí permutace s opakováním, tenhle výpočet není nejideálnější, avšak pro potřeby téhle aplikace postačuje – v rámci testování pracujeme se třemi hlasujícími a dvěma kandidáty. Vzorec permutace s opakováním pro hledání výsledků voleb je $P = (h+1)^k$ kde k značí počet kandidátů a h počet hlasujících zvětšené o jeden (musíme počítat s možností nula hlasů pro některého kandidáta). Pokud pracujeme s výše uvedenými počty dostaneme výsledek 16 možností, přičemž pří procházení rovnou ignorujeme kombinace pokud součet dané kombinace nesouhlasí s počtem odevzdaných hlasů (příklad hledání správného výsledku v tabulce 6.1), avšak pokud bychom například pracovali s 5 kandidáty a 10 hlasujícími již máme 161051 kombinací, které aplikace musí projít, i když většina bude ignorována z hlediska podmínky počtu odevzdaných hlasů.

[**Počet hlasů kandidáta 1 a kandidáta 2**]	**Součet**	**Ignorováno**?
[0, 0]	0	ANO
[0, 1]	1	ANO
[0, 2]	2	ANO
[0, 3]	3	**NE**, potenciální výsledek
[1, 0]	1	ANO
[1, 1]	2	ANO
[1, 2]	3	**NE**, potenciální výsledek
[1, 3]	4	ANO
[2, 0]	2	ANO
[2, 1]	3	**NE**, potenciální výsledek
[2, 2]	4	ANO
[2, 3]	5	ANO
[3, 0]	3	**NE**, potenciální výsledek
[3, 1]	4	ANO
[3, 2]	5	ANO
[3, 3]	6	ANO

Tabulka 6.1: Hledání výsledků v hlasovacím protokolu probíhá mezi všemi možnými výsledky. Tabulka uvádí příklad, kdy se hlasování v aplikaci uvažuje o **2 kandidátech** a **3 hlasujících**, kde všichni hlasující hlasovali. Aplikace nejdříve sečte počty hlasů v možných výsledcích a pokud souhlasí s počtem odevzdaných hlasů, máme potenciální výsledek. Pokud nesouhlasí, aplikace danou možnost přeskočí a tím ušetří místo pro výpočty.

6.6 Testovaní

Jazyk RUST obsahuje přehledné a pro programátora přívětivé testování. Nabízí unit testy, kde je účelem otestovat každou jednotku kódu izolovaně od zbytku kódu, aby bylo možné rychle určit, kde kód funguje a kde nefunguje podle očekávání. Dalšími testy jsou *integrační testy*, tyhle testy jako zcela externí vůči vaší knihovně a používají váš kód stejným způsobem, jakým by to dělal jakýkoli jiný externí kód, pouze s využitím veřejného rozhraní a potenciálně s využitím více modulů na test [25].

K testům se přidává anotace `#[cfg(test)]`, která se používá i v téhle aplikaci. Tahle anotace oznamuje RUSTu, aby zkompiloval a spustil testovací kód pouze při spuštění testu, nikoli při spuštění sestavování aplikace. Tahle metoda šetří čas kompilace, když je například potřeba pouze sestavit knihovnu. Mimo jiné šetří místo i ve výsledném kompilovaném souboru, protože testy u daného souboru nejsou zahrnuty [25].

V rámci testů se vyskytují dva moduly. První testovací model testuje korektní hlasování – od nasazení kontraktů autoritou, přiřazení hlasujících, zaregistrování klíčů, náhodnému hlasování jednotlivých hlasujících a výpočet celkových výsledků. Druhý testovací model testuje hlasování, kdy jeden hlasující nehlasuje po zaregistrování klíče – nasazení kontraktů autoritou, přiřazení hlasujících, zaregistrování klíčů, oprava hlasovacích klíčů, náhodnému hlasování výpočet celkových výsledků.

6.7 Nasazení autoritou

V rámci aplikace bylo vhodné udělat i uživatelské rozhraní pro autoritu. Kdy během nasazování autorita zadává dané údaje podle pořadí v jakém jsou potřeba, které následně potvrdí tlačítkem – od zadávání jmen kandidátů, přidáním peněženek hlasujících až o úplné nasazení všech kontraktů. Pokud vše proběhne v pořádku, vygeneruje se soubor s adresy daných kontraktů a soubor s adresy hlasujících. Tenhle soubor je důležitý pro komunikaci s kontrakty u uživatelů (o souboru pojednává výše uvedená část 6.5).

Autorita zde může nalézt adresy kontraktů hlasovacích skupin, kde u každé lze nalézt fázi ve které se nachází. Autoritě se zde nabízí i tři tlačítka – první pro výpočet hlasovacích klíčů, druhé pro nastavení opravné fáze a třetí pro výpočet celkových výsledků hlasování.

Kapitola 7

Analýza bezpečnostních funkcí jazyka RUST

Tahle kapitola se bude zabývat použitím bezpečnostních vlastností jazyka Rust v hlasovací aplikaci.

7.1 Prázdný ukazatel

RUST nepovoluje `null`, tedy prázdný ukazatel, místo toho používá stavy, které jsou však nepovinné a vývojář je nemusí využít. Pokud vývojář využije tuhle možnost, jsou následně definovány dva stavy – `SOME(T)` – stav, kdy daná metoda/funkce obsahuje hodnotu/objekt `T`, nebo `NONE`, tedy kdy daná metoda neobsahuje hodnotu, jedná se o alternativu k ostatním programovacím jazykům, které využívají ukazatel `null`. Stavy si můžeme i definovat vlastní, což se v aplikaci uplatnilo. Jako příklad stavů se může uvést `OK(T)` a `ERR(e)`, kde v případě chyby se vrací stav `ERR(e)` s chybou uloženou v `e`, ze kterou se následně může pracovat, například vypsání chybové hlášky nebo jako ze stavem samotným bez potřeby znát hodnotu, například využití k ukončení cyklu, atd.

7.2 Práce se stavy

Zjištění a práce ze stavy má různé podoby, funkcí `match` se může přímo definovat, co se stane v případě, kdy nastane jeden z možných stavů. Funkcí `unwrap` vracíme hodnotu `T` pokud je vrácen stav `OK(T)` a ignoruje, že by daná funkce mohla vrátit chybový stav (na tohle si musí dát vývojář pozor, neboť při vrácení chyby, program spadne), případně `try`, který dokáže zachytit chybu a následně vrací s dané metody stav `ERR(e)`. Případně často využívaný v téhle aplikaci funkce `expect(e)`, která očekává korektní výsledek tedy `OK(T)`, avšak v případě chyby vrací chybovou hlášku definovanou v `e`. Tahle funkce se často používá při práci s kontrakty (call, send).

Vlastní stavy se definovaly i v případě vrácení hodnoty z dané metody. Na základě těchto stavů pak aplikace mohla správně zobrazit uživateli relevantní informace na uživatelské rozhraní, jako například upozornění s danou hláškou. Vlastní stav se definoval i pro vrácení vygenerovaných šablon, případně právě i pro přesměrování. Například pokud hlasující již hlasoval, tak url `/voter/voting`, na které si hlasující vybírá kandidáta a poté následně hlasuje (obrázek 5.2) by pro uživatele již neměla být k dispozici, proto se zde vrátí stav na přesměrování, tedy zpátky na profil hlasujícího, místo aby se vrátila stránka pro hlasování.

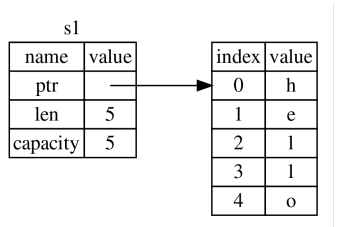
7.3 Testy

Rust umožňuje psaní testů, které se v téhle aplikaci využívají pro snadnější odstranění chyb. Testy urychlili vývoj aplikace, protože při spuštění testů se snáze hledali chyby a místa, kde se vytvořili. Urychlení vývoje spočívalo v automatizovaných krocích jednotlivých hlasujících a autority, tedy se zde nemuselo ručně „klikat“ a zadávat patřičné údaje pro nasazení kontraktů.

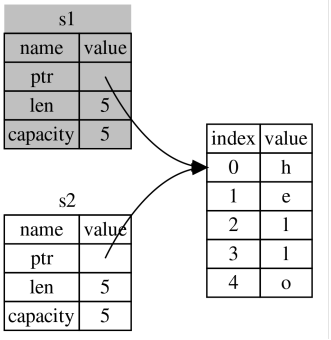
7.4 Vlastnictví

V rámci vývoje bylo zapotřebí vždy vyřešit problém s vlastnictvím. Při vývoji se muselo dbát, aby při kompilaci programu nedošlo k porušení jednoho z pravidel, které popisuje sekce 3.2. S tímhle problémem však pomáhal nástroj *clippy*, který na to upozorňoval při kompilaci aplikace. Předávání hodnot Rust implicitně neklonuje, ale propůjčuje. To znamená, pokud existuje vlastník `s1` (obrázek 7.1) a daný zdroj propůjčí `s2` (obrázek 7.2), již přestává být vlastníkem a novým vlastníkem se stává `s2`, což v některých případech při vývoji aplikace bylo nežádoucí. Propůjčování totiž znamená, že se zkopíruje objekt na zásobníku nikoli na haldě. Hrozilo by zde totiž, že by se původní vlastník dealokoval společně s daty na haldě a došlo by chybám na straně nového vlastníka.

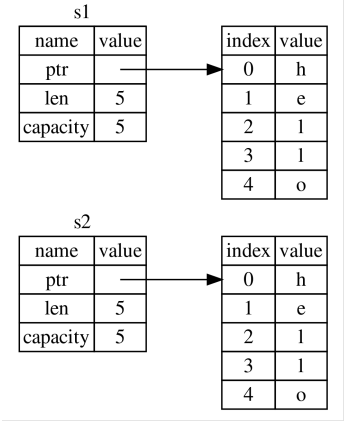
V těchto případech se v aplikace, kdy propůjčení bylo nežádoucí sloužila metoda klonování (obrázek 7.3), tedy kdy se objekt zkopíroval na zásobníku, ale také i na haldě a déle se mohly využívat a zpracovávat nezávisle na sobě. Rust implicitně propůjčuje, protože je to z hlediska běhového výkonu levnější. Například v aplikaci bylo stěžejní odesílání transakcí, protože při vložení dat do metod zajišťující odesílání transakce se data již nevrátila a bylo nutné je získat znovu. Klonování sice není z hlediska běhového výkonu aplikace ideální, ale v rámci téhle aplikace byly nežádoucí účinky pro běh aplikace zanedbatelné [24].



Obrázek 7.1: Alokování hodnoty v Rustu. Vlevo je zásobník (stack), kde se nachází objekt `s1` (v tomhle případě se jedná o datový typ `string`) s ukazatelem a vpravo halda (heap) [24].



Obrázek 7.2: Propůjčení hodnoty vlastníkem `s1` objektu `s2`, ten se stává novým vlastníkem. Původní vlastník `s1` se v tenhle moment nemůže využívat [24].



Obrázek 7.3: Klonování, kdy se vytvoří nezávislé kopie. Objekt na zásobníku se zkopíruje a dále se zkopíruje i nově na haldě [24].

Kapitola 8

Uživatelská použitelnost

Aplikaci si vyzkoušelo 8 uživatelů (6 z nich ve věkové skupině 20-24 let a 2 ve věku 53 let a 64 let, zároveň všichni účastníci testování mají nejvyšší dosažené vzdělání střední s maturitou) a poté na základě osobní zpětné vazby bylo zjišťováno, co by se mohlo vylepšit, případně doplnit, změnit. Testování uživatelů se provádělo za osobní účasti uživatele a vývojáře s okamžitou zpětnou vazbou.

Kvůli složitosti mít zapnutý lokální blockchain, aby aplikace správně fungovala, se zvolil tenhle typ testování za osobní účasti. Nepředpokládalo se, že by uživatelé měli zkušenosti s instalací *ganache*, případně zájem instalovat na svůj osobní počítač aplikaci třetí strany pouze za účelem otestování hlasovací aplikace.

8.1 Zobrazení výsledků hlasování

V rámci stránky celkových výsledků hlasování (obrázek 6.3), by uživatelé k jednoduché tabulce přidali graf výsledků. Graf byl následně implementován pomocí interaktivní webové služby Google Charts[1]. Při načtení stránky se do přidané skriptu dodají potřebné údaje (zejména jména kandidátů a počet jejich hlasů) a následně se vygeneruje koláčový graf. Uživatel si následně jednoduše mohl pohledem na graf představit celkové výsledky.

8.2 Blockchainový prohlížeč

Uživatelé, kteří testovaly danou aplikaci, měli požadavky, aby mohli zpětně dohledat, případně zkontrolovat korektnost hlasování. Na základě téhle zpětné vazby se doplnil blockchainový prohlížeč, kde na základě vyplněného hashe transakce nebo hashe bloku vyhledá patřičný objekt. Prohlížeč zobrazí základní údaje daného objektu a v objektu *transakce* mezi položkou *data* si uživatel může vyhledat hlasovací lístek a tím zkontrolovat korektnost hlasování.

Jako alternativu a také způsob, aby hlasující zjistil za pozdější dobu údaje dané transakce se transakce při úspěšném hlasování uloží do souboru ve formátu *json*. Tenhle soubor obsahuje totožné údaje jako u výše uvedeného prohlížeče, tedy hlavně kolonku *data*.

[1] `https://developers.google.com/chart`

8.3 Rychlost

Uživatelé si stěžovali na pomalou rychlost aplikace při odesílání hlasu, ta je z důvodu náročných výpočtů důkazů a jejich potvrzení. V reálném prostředí by uživatelé čekali i na zapsání transakce na blockchain. Jedno z možných řešení by byla implementace informační hlášky například „čekejte prosím, hlas se zpracovává. Tahle akce může trvat desítky sekund.“, aby si uživatelé nemysleli, že jim „zasekl“ počítač. Rychlost byla dána i výpočetním strojem, na kterém testování probíhalo.

8.4 Informace k fázím

Mezi doporučeními bylo též zavést stručnou informaci, o jakou fázi hlasovacího protokolu se jedná. Fáze se sice ukázána pouze jako číslo, ale pro uživatele nebylo příliš informativní, proto se přidala ikonka „i“ (informace), kdy po najetí myší na danou ikonku se zobrazily číselné ohodnocení fází a jejich stručná informace – název a účel.

Dále byla vylepšeno informační hláška pro uživatele. Zprvu měla hláška pouze informaci, že hlasující stále nehlasoval, což mohlo být pro některé hlasující matoucí, proto se doplnily o další hlášky podle fáze ve které se hlasování nachází, jako například o tom, že hlasující ještě nezaregistroval svůj veřejný klíč, čeká se na pokyn autority, čeká se na opravu hlasovacích lístků, hlasování je u konce a čeká se na jejich vyhodnocení, apod.).

Kapitola 9

Závěr

Cílem téhle práce bylo vytvořit decentralizovanou aplikaci, tedy uživatelské rozhraní pro decentralizované elektronické volby podle zadaného hlasovacího protokolu. V rámci práce byla dána pozornost na jednoduché ovládání a co nejstručnější, ale co možná nejvýstižnější informace pro hlasující. Aplikace nesměla být velice složitá na ovládání, protože uživatelé jsou z celého spektra skupin, jak věkových tak případně i podle zkušeností s prací na počítači. Mezi dalšími vlastnostmi, které aplikace musela splňovat bylo šifrování. Pomocí frameworku, který provádí operace nad eliptickou křivkou, bylo šifrování podle dodaného protokolu možné a proto se zachovala i bezpečnost hlasovacího protokolu, která je u volební aplikace žádoucí a nelze bez ní pracovat. Pro rozvržení komponentů na webové stránce se vybral balíček, který připravila Česká republika v rámci plánu digitalizace. Tenhle balíček je stále ve vývoji, ale pro účely aplikace byl adekvátní. Balíček byl vybral na základě dostupnosti, ale především pro jeho účel, tedy aby veřejná správa měla v budoucnu jednotný design. Tahle aplikace zabývající se volbami by v budoucnu mohla mít přesah pro národní věci a proto by měla mít stejný design jako případně ostatní státní weby/aplikace.

V práci lze jistě pokračovat a to například implementací již dostupných softwarových nebo hardwarových peněženek do uživatelského rozhraní aplikace. Bohužel použitý webový framework není kompatibilní s využitím daných peněženek a proto se nabízí otázka například o rozšíření daného frameworku o možnost pracovat s danými peněženkami. Další možností případného vylepšení aplikace je najít výhodnější pro časovou a výpočetní náročnost algoritmus na vyhledávání možností výsledků hlasování.

Polemizovat se ovšem dá u finanční stránky téhle aplikace. Sice se práce nezabývá přímo finanční stránkou, ale je úzce spjatá, protože za transakce, které se provádějí, jak ze strany autority, tak ze strany voliče, při odesílání volebního lístku se musí zaplatit poplatek, tedy kolik *GASu* je potřeba pro provedení a kolik zrovna stojí jednotka *GAS*. Takhle otázka by měla vznést především nad autoritou, jímž například můžeme uvažovat stát. Jak by vykompenzoval poplatek, který by musel volič zaplatit za odevzdání hlasovacího lístku? Je vůbec tohle možné, aby volič musel platit, za to, že chce volit? Systém blockchainu je však takhle nastavený a bylo by pouze na autoritě, jak by tyhle poplatky kompenzovala. Případně jak by se volby přes blockchain projevili na trhu s kryptoměnami.

Bezpečnostní rizika, která se uváděla v teoretické části prakticky říkají, že je nutností mít co největší počet uzlů, aby se zamezilo případnému útoku a zfalšovaní voleb nebo, aby útočník dostal pod kontrolu dané volby a mohl tím ovlivnit v podstatě několik let dění v oblasti voleb.

Literatura

[1] ANDERBERG, A., ANDONOVA, E., BELLIA, M., CALÈS, L., INAMORATO DOS SANTOS, A. et al. *Blockchain Now And Tomorrow*. Scientific analysis or review, Anticipation and foresight KJ-NA-29813-EN-N (online),KJ-NA-29813-EN-C (print),KJ-NA-29813-EN-E (ePub). Luxembourg (Luxembourg), 2019.

[2] BENITEZ, S. *Meet Rocket.* [online]. 2021 [cit. 2022-09-03]. Dostupné z: https://rocket.rs/.

[3] BHARDWAJ, P., CHANDRA, Y. a SAGAR, D. *Ethereum Data Analytics: Exploring the Ethereum Blockchain*. Září 2021.

[4] BROWN, D. R. L. SEC 2 : Recommended elliptic curve domain parameters. *Standars for Efficient Cryptography*. Certicom Corp. 2010. Dostupné z: https://cir.nii.ac.jp/crid/1572824501046106880.

[5] BUTERIN, V. et al. *Ethereum: A next-generation smart contract and decentralized application platform*. 2014.

[6] *Coins archive* [online]. 2021 [cit. 2021-18-11]. Dostupné z: https://cryptoslate.com/coins/.

[7] *Crate elliptic curve* [online]. 2022 [cit. 2022-12-04]. Dostupné z: https://docs.rs/elliptic-curve/latest/elliptic_curve/.

[8] *Crate Web3* [online]. 2021 [cit. 2021-15-11]. Dostupné z: https://docs.rs/web3.

[9] DAHL, D. B. Writing R Extensions in Rust. *CoRR*. 2021, abs/2108.07179. Dostupné z: https://arxiv.org/abs/2108.07179.

[10] *Decentralized applications (DAPPS)* [online]. 2021 [cit. 2021-15-11]. Dostupné z: https://ethereum.org/en/dapps/#what-are-dapps.

[11] *GOV.CZ* [online]. 2021 [cit. 2022-24-04]. Dostupné z: https://code.gov.cz/gov-cz/gov-design-system.

[12] *Design Systém Gov.cz* [online]. 2021 [cit. 2022-24-04]. Dostupné z: https://designsystem.gov.cz/#/pravidla/pravidla-pro-modifikaci.

[13] KASIREDDY, P. *The architecture of a web 3.0 application* [online]. Sep 2021 [cit. 2021-22-11]. Dostupné z: https://www.preethikasireddy.com/post/the-architecture-of-a-web-3-0-application.

[14] Kuo, T.-T., Kim, H. a Ohno Machado, L. Blockchain distributed ledger technologies for biomedical and health care applications. *Journal of the American Medical Informatics Association*. Listopad 2017, sv. 24, s. 1211–1220. DOI: 10.1093/jamia/ocx068.

[15] Metcalfe, W. Ethereum, Smart Contracts, DApps. In:. Springer, 2020, kap. Chapter 5, s. 77–93. DOI: 10.1007/978-981-15-3376-1_5.

[16] Ojha, G. *Feasibility Study of Pipelining in Ethereum Virtual Machine Architecture*. Září 2020. DOI: 10.13140/RG.2.2.18334.15687/1.

[17] Potts, J. a Rennie, E. Web3 and the Creative Industries: How Blockchains are reshaping business models. *SSRN Electronic Journal*. 2019. DOI: 10.2139/ssrn.3372108.

[18] Reprintsev, A. Turing Completeness. In:. Duben 2018, s. 235–242. DOI: 10.1007/978-1-4842-3372-6_10. ISBN 978-1-4842-3371-9.

[19] RustCrypto. *Elliptic curves K256 at master* [online]. 2022. Dostupné z: https://github.com/RustCrypto/elliptic-curves/tree/master/k256/.

[20] Shen, X., Jiang, S. a Zhang, L. Mining Bytecode Features of Smart Contracts to Detect Ponzi Scheme on Blockchain. *Computer Modeling in Engineering & Sciences*. Leden 2021, sv. 127, s. 1069–1085. DOI: 10.32604/cmes.2021.015736.

[21] Stančíková, I. *Škálovatelné hlasování s ochranou soukromí hlasů založené na blockchainu*. Brno, 2021. Diplomová práce. Vedoucí práce Homoliak, I.

[22] Szabo, N. Smart contracts: building blocks for digital markets. *EXTROPY: The Journal of Transhumanist Thought,(16)*. 1996, sv. 18, č. 2, s. 28.

[23] *The rust programming language* [online]. 2021 [cit. 2021-15-11]. Dostupné z: https://doc.rust-lang.org/book/appendix-07-nightly-rust.html.

[24] *The rust programming language* [online]. 2021 [cit. 2021-18-11]. Dostupné z: https://doc.rust-lang.org/book/ch04-01-what-is-ownership.html.

[25] *The rust programming language* [online]. 2022 [cit. 2022-20-04]. Dostupné z: https://doc.rust-lang.org/book/ch11-03-test-organization.html#the-tests-module-and-cfgtest.

[26] Tsoulias, K., Palaiokrassas, G., Fragkos, G., Litke, A. a Varvarigou, T. A. A Graph Model Based Blockchain Implementation for Increasing Performance and Security in Decentralized Ledger Systems. *IEEE Access*. 2020, sv. 8, s. 130952–130965. DOI: 10.1109/ACCESS.2020.3006383.

[27] *Web2 vs web3* [online]. 2021 [cit. 2021-15-11]. Dostupné z: https://ethereum.org/en/developers/docs/web2-vs-web3/.

[28] Zhang, J. Deploying Blockchain Technology in the Supply Chain. In: Thomas, C., Fraga Lamas, P. a Fernández Caramés, T. M., ed. *Computer Security Threats*. Rijeka: IntechOpen, 2019, kap. 5. DOI: 10.5772/intechopen.86530. Dostupné z: https://doi.org/10.5772/intechopen.86530.

Příloha A

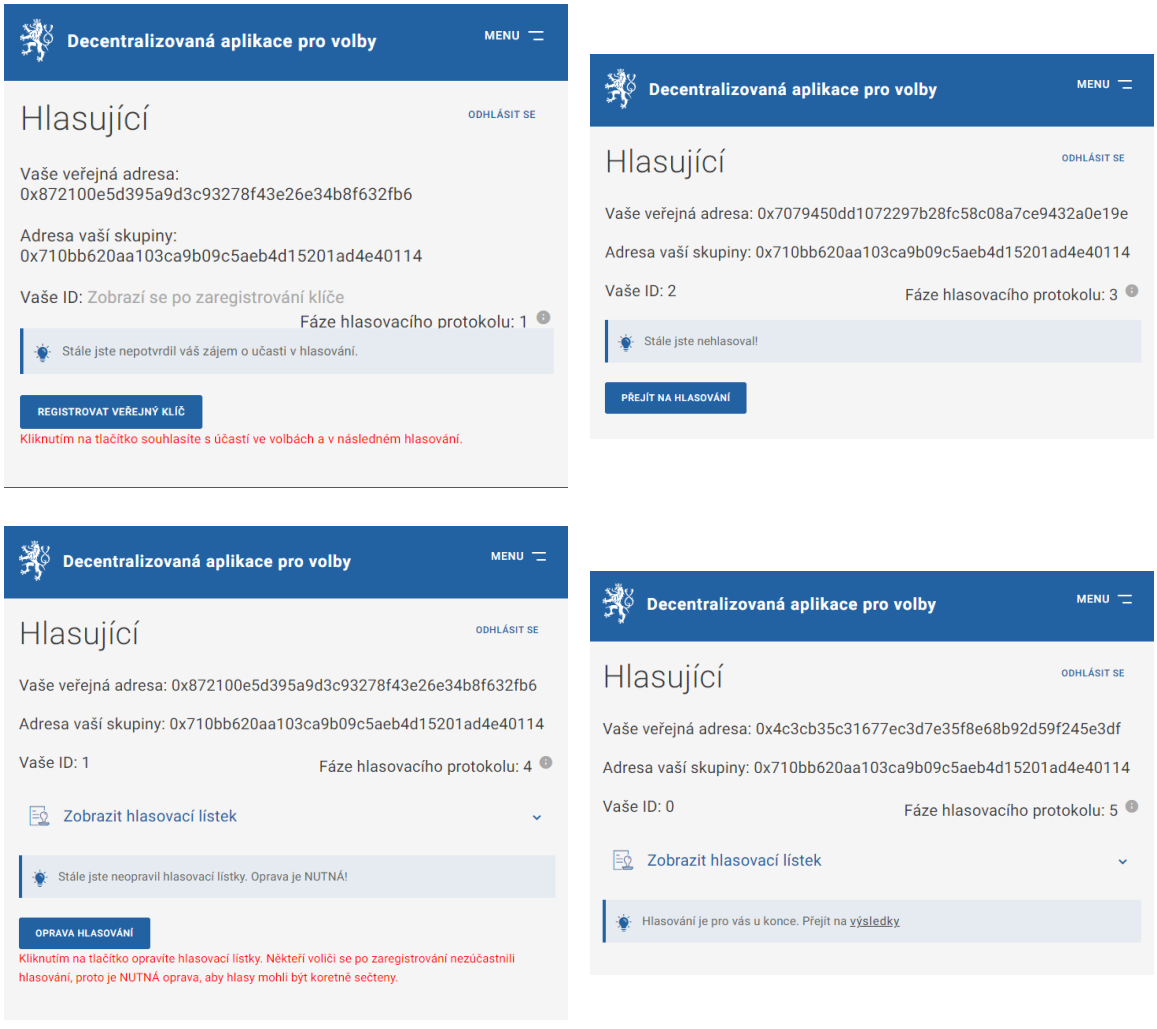
Obsah přiloženého DVD

Na přiloženém DVD jsou nahrané příslušné položky

- xmalin30.pdf – text bakalářské práce
- latex/ – zdrojové soubory textu bakalářské práce
- src/ – zdrojové kódy aplikace
  - src/ – kódy v Rustu
  - static/ – neměnné soubory (kaskádové styly, javascripty)
  - templates/ – šablony
  - Cargo.lock – konfigurační soubor
  - Cargo.toml – konfigurační soubor
  - README.md – návod pro kompilaci a spuštění
  - Rocket.toml – konfigurační soubor
- bin/ – aplikace ve spustitelné formě.

Příloha B

Uživatelské rozhraní



Obrázek B.1: Některé vizualizace při používání aplikace. **Horní levý**: profil po přihlášení, čeká se na zaregistrování svého veřejného hlasovacího klíče. **Horní pravý**: Uživatel již může hlasovat, přejít na výběr mezi kandidáty. **Dolní levý**: Uživatel musí opravit svůj hlasovací lístek. **Dolní pravý**: Ukončené hlasování, uživatel může přejít na výsledky.