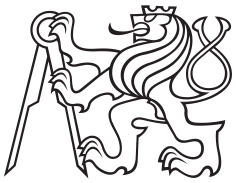
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra kybernetiky

Decentralizovaný meziuniverzitní informační systém

Jakub Strnad

Vedoucí: Ing. Karel Frajták, Ph.D.
Leden 2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Strnad** Jméno: **Jakub** Osobní číslo: **498832**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra kybernetiky**

Studijní program: **Otevřená informatika**

Specializace: **Základy umělé inteligence a počítačových věd**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Decentralizovaný meziuniverzitní informační systém

Název bakalářské práce anglicky:

Decentralized Inter-University Information System

Pokyny pro vypracování:

Proveďte analýzu systémů pro výměnu dat a komunikaci mezi univerzitami, např. ISIC. Prozkoumejte možnosti decentralizace univerzitního systému za pomoci technologie Hyperledger Fabric. Navrhněte jednotný datový model použitelný napříč univerzitami a smart kontrakty, které budou zajišťovat logiku s daty na zmíněném blockchainu.
Dále navrhněte konfiguraci pravidel pro jednotlivé univerzity uvnitř tohoto systému, včetně toho, kdo má přístup k jakým datům a kolikrát se data replikují na jednotlivých uzlech.
Nakonec vytvořte kritéria pro funkčnost takového decentralizovaného systému a otestujte řešení na implementaci tohoto systému, ověřte splnění stanovených kritérií.

Seznam doporučené literatury:

[1] Stranger, Alvaro Pina, German Varas, and Gaëlle Mobuchon. "Managing Inter-University Digital Collaboration from a BottoStranger, Alvaro Pina, German Varas, and Gaëlle Mobuchon. "Managing Inter-University Digital Collaboration from a Bottom-Up Approach: Lessons from Organizational, Pedagogical, and Technological Dimensions." Sustainability 15, no. 18 (2023): 1-20.m-Up Approach: Lessons from Organizational, Pedagogical, and Technological Dimensions." Sustainability 15, no. 18 (2023): 1-20.
[2] Androulaki, Elli, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart Androulaki, Elli, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart et al. "Hyperledger fabric: a distributed operating system for permissioned blockchains." In Proceedings of the thirteenth EuroSys conference, pp. 1-15. 2018. et al. "Hyperledger fabric: a distributed operating system for permissioned blockchains." In Proceedings of the thirteenth EuroSys conference, pp. 1-15. 2018.
[3] Suliyanti, W Suliyanti, Widya Nita, and Riri Fitri Sari. "Blockchain-based implementation of building information modeling information using hyperledger composer." Sustainability 13, no. 1 (2020): 321.idya Nita, and Riri Fitri Sari. "Blockchain-based implementation of building information modeling information using hyperledger composer." Sustainability 13, no. 1 (2020): 321.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Karel Frajták, Ph.D. laboratoř inteligentního testování systémů FEL

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

Datum zadání bakalářské práce: **22.09.2023** Termín odevzdání bakalářské práce: **09.01.2024**

Platnost zadání bakalářské práce: **16.02.2025**

Ing. Karel Frajták, Ph.D. podpis vedoucí(ho) práce	prof. Ing. Tomáš Svoboda, Ph.D. podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

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

. _____ _____

Datum převzetí zadání Podpis studenta

Poděkování

Děkuji panu doktoru Karlu Frajtákovi za vedení této bakalářské práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 4. ledna 2024

.....
podpis autora práce

Abstrakt

Nejcennější věc, kterou v dnešní době máme, jsou data. Všechny systémy moderního světa s nimi nějak pracují, analyzují je a na jejich základě se rozhodují. Podmnožinou takových dat jsou i data akademická, avšak nakládání s tímto typem dat je v dnešním kontextu poněkud zastaralé. Interně je manipulace s daty administrativně náročná a zdlouhavá, data se uchovávají zpravidla v jedné centralizované databázi, což představuje riziko zneužití či úniku dat. Externí systémy nad nimi nemůžou stavět komplexní logiku, což limituje rozvoj komerčního potenciálu.

Cílem této práce je zmapovat aktuální informační systémy a jejich zacházení s akademickými daty ať už interně nebo externě. Dále se práce zabývá návrhem decentralizovaného informačního systému, který umožní inovativní zacházení s akademickými daty, a implementací demo verze systému. Nakonec práce posoudí využitelnost takového systému, jeho bezpečnost a škálovatelnost.

Klíčová slova: blockchain, informační systém, hyperledger fabric

Vedoucí: Ing. Karel Frajták, Ph.D.

Abstract

Data is the most important thing we own today. Every current system processes data, analyzes it, and makes decisions based on it. Academic data represents a distinct subset; however, its usage has become outdated: internal data manipulation is time-consuming and administratively challenging, and data is stored in one centralized database, implying the risk of data leak or misuse. Furthermore, external systems cannot build complex logic above this data, considerably limiting its commercial potential.

This thesis aims to survey the existing information systems and their use of academic data internally and externally. Against this background, it proposes a decentralized inter-university information system that enables an innovative use of academic data while implementing a proof-of-concept version of this system. Finally, it evaluates this system's degree of usability, security, and scalability.

Keywords: blockchain, information system, hyperledger fabric

Title translation: Decentralized Inter-Univerzity Information System

Obsah

Obrázky

Kapitola 1
Úvod

Cílem této bakalářské práce je zkoumat aktuální univerzitní informační systémy optikou decentralizovaných distribuovaných systémů. Práce provede rešerši aktuálních univerzitních informačních systémů používaných na území České republiky. Zhodnotí, zdali jsou tyto systémy dostačující z hlediska požadavků dnešní doby. Definuje a navrhne meziuniverzitní informační systém pomocí decentralizovaného distribuovaného systému s využitím technologie blockchain jako databáze a implementuje "proof of concept". Součástí je analýza a zhodnocení využitelnosti takového systému.

Bakalářská práce má následující strukturu:

První kapitola je úvodem.

Druhá kapitola se zabývá aktuálními informačními systémy. Provádí jejich rešerši a kategorizuje je na interní a externí. Závěrem shrne aktuální informační systémy a vytyčí jejich nedostatky optikou dnešní doby.

Třetí kapitola se zabývá definicí meziuniverzitního informačního systému a formuluje kritéria, která by měl splňovat. Navrhne realizovat meziuniverzitní informační systém pomocí decentralizované technologie na základě technologie blockchain.

Čtvrtá kapitola osvětluje technologie blockchain. Demonstruje jejich vlastnosti a sumarizuje kritéria, ve kterých se blockchainové technologie liší. Nakonec doporučí technologii Hyperledger Fabric pro realizaci decentralizovaného meziuniverzitního informačního systému.

Pátá kapitola detailně představuje technologii Hyperledger Fabric a její jednotlivé komponenty.

V šesté kapitole se práce věnuje implementaci decentralizovaného meziuniverzitního informačního systému pomocí technologie Hyperledger Fabric.

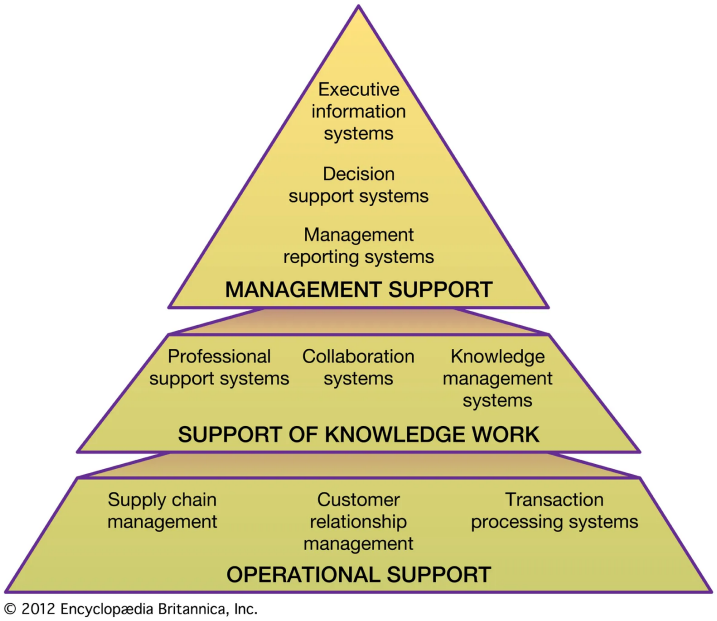
Sedmá, poslední kapitola shrne dosažené výstupy a zhodnotí naplnění zadání.

Kapitola 2
Aktuální systémy

Tato kapitola mapuje aktuální univerzitní informační systémy. Definuje pojmy *externí* a *interní* univerzitní informační systém. Známé systémy rozřadí do těchto kategorií a shrne jejich klady a zápory. Dále zavede pojem *meziuniverzitní informační systém* a vymezí, jaká kritéria by takový moderní systém měl splňovat jak interně, tak externě.

2.1 Informační systém

Informační systém je souhrnem hardwaru, softwaru a lidí. Jedná se o systém, jehož součástí jsou aplikace, uživatelé a procesy. Systém má za úkol sbírat, uchovávat, zpracovávat a distribuovat informace pro plánování, rozhodování a kontrolu [1]. V kontextu akademického informačního systému se jedná zpravidla o hardware univerzity a software buď univerzitní nebo licencovaný v závislosti na možnostech a potřebách jednotlivých univerzit. V takovémto systému jsou aktéry studenti, zaměstnanci či externisté. Jedná se tedy o univerzitní ekosystém, ve kterém dochází k výměně dat mezi jednotlivými aktéry. Jelikož je pojem univerzitní informační systém rozsáhlý, v další kapitole jej rozdělím na univerzitní informační systém interní a externí.



Obrázek 2.1: Ilustrace úrovní informačního systému [2].

2.2 Definice interní vs. externí univerzitní informační systém

Interní univerzitní informační systém

Je takový systém, který nějakým způsobem pracuje s interními daty v rámci jedné univerzity. Mezi funkcionality takového systému patří management profilů studentů a lektorů, management předmětů, závěrečných prací, citací, přestupků (např. plagiáty), management rozvrhů, administrační rozhraní pro uživatele aplikace, export dokumentů či přehled studia. Jedná se tedy o systém, který slouží uživatelům, kteří mají vztah k dané univerzitě. Příkladem takového systému jsou KOS nebo UIS.

Externí univerzitní informační systém

Je takový systém, který nabízí rozhraní pro subjekty mimo akademické prostředí dané univerzity, tzn. univerzity jiné či externí subjekty. Jedná se o zaměstnavatele, vládní sektor, instituce či byznysy. Příklad takového systému je ISIC.

2.2.1 Interní univerzitní informační systémy

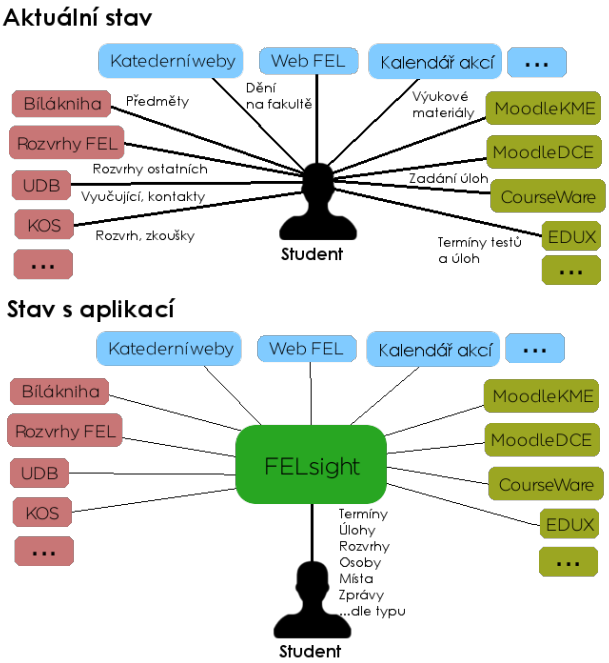
Komponenta Studium - KOS

Jedná se o interní aplikaci ČVUT pro studenty, která eviduje všechny potřebné informace o studiu. Studenti si zde mohou tvořit rozvrh, přihlásit se ke zkoušce,

nalézt informace o svém studiu atd. Slouží jako rozhraní pro komunikaci mezi studentem a univerzitou. KOS nabízí REST API, ale jedná se pouze o službu určenou KOS aplikaci, jelikož k jeho využívání je zapotřebí vlastnit účet. Tento způsob implementace API neumožňuje externím subjektům získat data z KOS a zamezuje tak integraci logiky v externích systémech. Pokud by bylo možné založit API klíč externím subjektům, logika by využívat šla. Z těchto důvodů řadím KOS software do interního univerzitního informačního systému.

FELSight

FELSight je interní aplikace vyvíjená studenty elektrotechnické fakulty ČVUT pod vedením Centra znalostního managementu FEL ČVUT. Aplikace nabízí studentům jednotnou platformu pro podporu studia tím, že extrahuje všechny informace z důležitých fakultních portálů a prezentuje je v přehledné podobě. Student v aplikaci najde svůj kalendář, termíny úloh a testů, události, přehled semestru a dokonce i jídelníček v univerzitní menze či jiných restauracích. Aplikace pracuje s informacemi uvnitř jedné fakulty, proto ji řadím mezi interní univerzitní informační systémy.



Obrázek 2.2: Ilustrace zjednodušení informačních portálů pro studenta pomocí FELSight aplikace [3].

Univerzitní Informační Systém - UIS

Jedná se o univerzitní informační systém, který není vázaný pouze na jednu univerzitu, ale slouží jako “as a service” řešení pro jakoukoliv univerzitu.

Systém zjednodušuje administrativu, uchovává veškeré informace všech relevantních aktérů dané univerzity, podporuje jak studijní, tak vědecko-výzkumné procesy. Aplikace se řadí mezi interní, jelikož nepodporuje komunikaci vně univerzity (ať už s jinými univerzitami nebo externími klienty).

2.2.2 Externí univerzitní informační systémy

International Student Identity Card - ISIC

Jedná se o jediný celosvětově uznávaný průkaz studentů a lektorů. Registruje všechny osoby, které mají vztah k určité univerzitě. Tento systém nabízí API rozhraní, přes které se mohou jednotliví externí partneři dotazovat na informace o držiteli karty. Data, na která se lze dotazovat, jsou velmi omezená. Zjednodušeně řečeno lze pouze ověřit, jestli je ISIC průkaz platný. Dále lze přes toto rozhraní zapisovat aktivní slevy, které jsou relevantní pro průkaz držitele. Rozhraní není veřejné, lze se dotazovat pouze s API klíčem registrovaného partnera v ISIC systému. V následující části uvádím hlavní REST API endpoints, které ISIC vystavuje.

Verifikace

ISIC podporuje čtyři typy verifikace: podle čísla průkazu, telefonního čísla, jména držitele a čipu. Množina dat, se kterou všechny metody pracují, je stejná. Budu proto v práci dále uvažovat pouze metodu s číslem karty, jelikož ji lze využít v externích webových aplikacích. Z ISIC API dokumentace vidíme, že jsou podporovány pouze čtyři atributy (viz. obrázek 2.3). Tři z nich nejsou povinné. ISIC API vrací pouze ty atributy, které byly poslány v dotazu. To znamená, že nesdílí informace, které partner nezná předem. V kontextu expirace karty lze jen odhadovat, do kdy je karta platná. Je nutno poskytnout datum, do kterého je žádáno vědět, jestli je daná karta platná (viz. obrázek 2.4). Podíváme-li se na tohle rozhraní optikou dnešní doby, zjistíme, že není dostačující pro většinu užitečných případů. ISIC rozhraní vůbec neuvažuje detailní informace studentů, učitelů a zaměstnanců univerzit. Nelze stavět komplexnější logiku nad univerzitními daty, čímž se limitují inovace v sektoru použitelnosti zmíněných dat.

Transakce

ISIC také podporuje management slev. Tento modul však není pro naši problematiku relevantní, nebudu ho proto dále rozvádět.

POST /verifications

Number mode

A card verification based on a card number.

Request Body Parameters

Parameter Name	Data Type	Mandatory	Description
cardNumber	String	Yes	Card number including initial and end letters.
discountId	Integer	No	Unique discount identifier provided by the client's ISIC account manager. Card verification service will check discount restrictions.
cardholderDateOfBirth	Date	No	Cardholder's date of birth.
cardValidLongerThan	Date	No	To check if the verified card is valid longer than specified date.

Obrázek 2.3: ISIC verifikace — atributy [4].

Sample Response Body – Success


```
{
    "id": 3044020,
    "createdOn": "2021-01-12T18:21:45+00:00",
    "cardNumber": "S420539013044P",
    "mode": "NUMBER",
    "discountId": 12700,
    "result": "SUCCESSFUL",
    "cardType": "ISIC",
    "cardholderDateOfBirth": "1997-03-12",
    "cardValidLongerThan": "2021-12-31"
}
```


Obrázek 2.4: ISIC verifikace — úspěšná odpověď [4].

2.3 Shrnutí aktuálních systémů

V kontextu dnešní doby se aktuálně používané systémy jeví jako zastaralé. Jedná skupina univerzit má vlastní informační systém, který je vyvíjen a udržován interním IT oddělením. Inovace jsou v těchto systémech náročné a ekonomicky nevýhodné. Druhá skupina univerzit využívá systém poskytnutý externí společností, nemá tak absolutní kontrolu nad daty a možnost úprav na míru je velmi omezená. U obou případů jsou data uložena v jedné centralizované databázi (i když jsou periodicky zálohována), což vytváří prostor pro únik dat, jejich zneužití či neoprávněnou manipulaci. Zároveň existuje čím dál větší množství externích subjektů mimo univerzitní prostředí se schopností data komerčně využívat či nad nimi stavět složitější logiku a tím zlepšit kvalitu určitých služeb. Takovéto zhodnocení dat však nepodporuje žádný interní informační systém. Jediná možnost, jak alespoň trochu využít akademická data, je API rozhraní od asociace ISIC, které je však velmi omezené a nedostačující v moderních případech užití.

V další části navrhuji spojit interní a externí informační systém v jeden obecný informační systém, který by dostál současným nárokům, a takovýto systém zobecnit nad rámec jedné konkrétní univerzity.

Kapitola 3

Meziuniverzitní informační systém

V této kapitole představuji obecný meziuniverzitní informační systém, definuji kritéria jeho funkčnosti a obhajuji použití *permissioned* blockchain technologie pro jeho realizaci.

3.1 Kritéria systému

V této sekci definuji kritéria meziuniverzitního informačního systému.

3.1.1 Interní využití systému uvnitř univerzity

Systém by měl podporovat přesně to, co dnešní univerzitní systémy dokáží. Jedná se zejména o administrativu spojenou se studiem, managementem osob, administrací předmětů, závěrečných prací, rozvrhů, přestupků atd., zkrátka vším, co je spjaté s fungováním univerzity.

3.1.2 Externí využití systému mimo univerzitní prostředí

Systém by měl externím subjektům povolit čtení dat dané univerzity, pokud to povolí vlastník dat. Díky tomuto přístupu pak mohou různé společnosti stavět sofistikovanou logiku nad těmito daty. Jako příklad můžeme uvést zaměstnavatele, kterého zajímá, zdali uchazeč jako absolvent dané univerzity úspěšně splnil relevantní předměty s lepším prospěchem než známka C.

3.1.3 Jednotný datový model

Systém by standardizoval datový model akademických dat a vynutil ho pro všechny univerzity. Zrychlilo by to vývoj nových univerzitních informačních systémů a přispělo ke standardizaci vývoje externích aplikací v akademickém sektoru.

3.1.4 Škálovatelnost

Systém by měl být škálovatelný. Měl by být schopen vyhovět zvýšené poptávce o data určité univerzity.

3.1.5 Bezpečnost

Systém by měl být bezpečný. Měl by vynucovat ukládání dat na půdě univerzit a kontrolovat přístupy k nim. Všechny data by se měli duplikovat a zálohovat, kdyby došlo k výpadku.

3.1.6 Transparentnost

Systém by měl být transparentní. Všechny úpravy dat by měly být dohledatelné a ověřitelné. K veřejným datům univerzit by měly mít přístup všechny účty dané univerzity.

3.1.7 Shrnutí

Systém, který splňuje předešlé kritéria, uchovává všechny data o univerzitách a nabízí rozhraní pro externí subjekty pro jejich regulované čtení. Dokáže obsluhovat všechny administrativní potřeby jednotlivých univerzit a deklaruje standardizovaný datový model, který následují všechny univerzity používající tento systém. Univerzity vlastní svá data na svých datových uložištích a jednotlivý vlastníci je mohou zpeněžit. Systém je bezpečný, transparentní a velmi škálovatelný.

3.2 Návrh použít blockchain

Meziuniverzitního informačního systému není možné dosáhnout klasickými technologiemi používaných v informačních systémech dodnes. Pokud bychom používali konvenční databázové řešení, každá univerzita by měla svůj vlastní datový model. Standardní datový model by byl nevynutitelný. Dále by si každá univerzita musela hlídat zálohování dat, pravidla přístupu k datům a uživatelské role. V této sekci navrhuju pojmout meziuniverzitní systém decentralizovaně pomocí technologie blockchain.

3.2.1 Proč použít blockchain?

Meziuniverzitní informační systém je podle mého názoru ideální kandidát pro použití decentralizované technologie blockchain. Mluvíme o spoustu jednotlivých organizacích (univerzitách), které operují ve stejném sektoru a produkují stejná data, tudíž by měly mít jednotný datový model. Univerzity chtějí mít absolutní kontrolu nad svými daty a preferují ukládat data ve svých lokalitách ať už z právních či preferenčních důvodů. Systém by měl být transparentní a jakékoliv nečestné chování odhalitelné. Bezpečnost a vysoká škálovatelnost je samozřejmostí. Data musí být redundantní a replikovatelná. V další kapitole vysvětlím, co to blockchain je a vyberu nejideálnější blockchainovou technologii pro můj specifický případ.

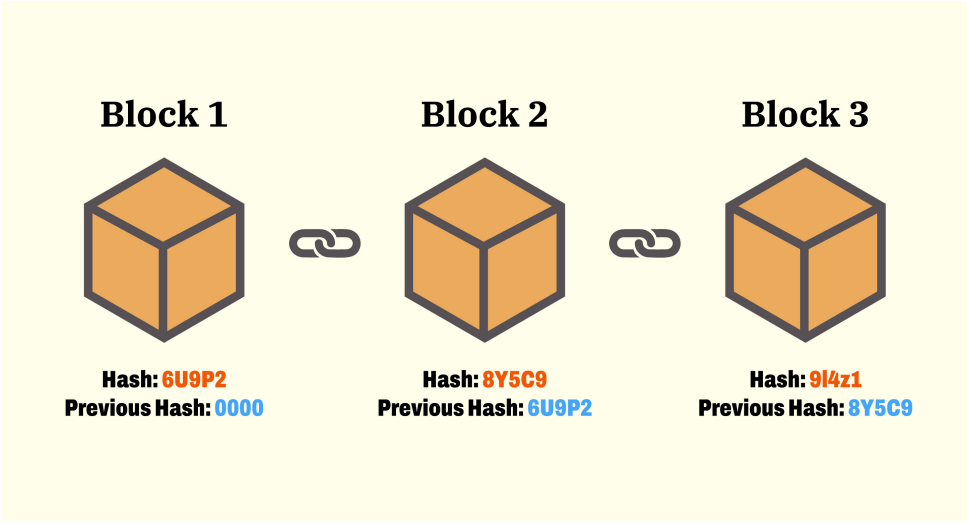
Kapitola 4
Blockchain

4.1 Charakteristika

Pojem blockchain je mnohovýznamný. Jako blockchain můžeme nazývat jak typ databáze, tak distribuovaný systém, který tuto databázi využívá.

Blockchain jako databáze

Popisuje blockchain jako typ databáze. Zápis do takové databáze se dělá pomocí tzv. blocků, které v sobě nesou data. Jednotlivé blocky na sebe navazují pomocí hashe a tvoří tak historii blockchainu; poslední block reprezentuje poslední změnu databáze. Každý nový block mění stav databáze. První block je tzv. genesis block a určuje konfiguraci blockchainu (např. množinu validátorů, maximální gas price atd.). Taková databáze může existovat mezi několik servery, ale i pouze na jednom serveru. Tento pohled na blockchain neimplikuje distribuovaný systém, jeho použití v takovém systému je však velmi výhodné.



Obrázek 4.1: Ilustrace propojení blocků v blockchainu [5]. *Autor obrázku neznámý.*

Blockchain jako distribuovaný systém

Popisuje blockchain jako systém uzlů, které udržují globální stav databáze, tj. blockchainu. Každý uzel uchovává kopii databáze ve svém uložišti a shoduje se na novém stavu databáze pomocí konsenzus algoritmu s ostatními uzly. K tomu, aby se globální stav blockchainu změnil zápisem nového blocku, musí se určité procento serverů (zpravidla většina) shodnout na novém stavu. Pokud taková shoda nastane, většina uzlů zapíše nový block do své kopie blockchainu a globální stav databáze se změní.

Takovéto vymezení dále zahrnuje rovněž aplikační logiku na blockchainu. Většina blockchainů podporuje tzv. Smart Contracts, což je kód, jehož implementace je zapsána na blockchainu a nabízí rozhraní pro čtení a zápis komplexnějších dat. Koncept Smart Contracts byl poprvé implementován blockchainem Ethereum, kde je implementace smart contractu veřejně dohledatelná. Externí klienti jsou tedy obeznámeni s implementací smart contractů a mohou si být jisti s výsledky volání.

Dále v bakalářské práci budu uvažovat blockchain jakožto distribuovaný systém.

4.2 Vlastnosti

4.2.1 Z pohledu technologie

V bodech níže jsou shrnuty nejdůležitější vlastnosti blockchainu jako takového.

- **Asynchronicita** — Blockchain je distribuovaný systém. To znamená, že jakákoliv změna jeho stavu trvá déle než u konvenčních centralizovaných systémů. Je to způsobeno tím, že se nové blocky tvoří periodicky a na jejich zařazení do blockchainu se musí většina uzlů shodnout. Existuje metrika block time, která popisuje průměrný čas tvoření nového blocku a jeho potvrzení ostatními uzly. Konkrétní block time je závislý na typu konsenzu blockchainu, počtu a výkonu uzlů a kvalitě internetového připojení. Zpravidla je tento čas v řádu sekund až minut.
- **Poplatek** — U většiny blockchainů existuje tzv. *native cryptocurrency* s monetární hodnotou. Pro vykonání zápisu na blockchain je nutné zaplatit určitou částku (dle velikosti transakce a rychlosti jejího zpracování) prostřednictvím tzv. *gas fee*. Tento mechanismus slouží k odměnění vlastníků uzlů za poskytnutou práci a rovněž jako ochrana proti DoS[1] útoku na jednotlivé uzly.
- **Transparentnost** — Nezávisle na tom, jestli je blockchain privátní, nebo veřejný, existuje vždy důkaz všech akcí, které se na blockchainu od počátku udály.

[1] Denial of Service

- **Bezpečnost** — Všechny uzly v síti uchovávají svou vlastní kopii blockchainu. Pokud jeden uzel přepíše svůj blockchain a navrhne nový block, ostatní uzly ho nepřijmou, protože se nový block neshoduje se stavem jejich databáze, anebo kvůli chybějícímu podpisu relevantního účtu. To znamená, že pokud uživatel vlastní nějakou hodnotu na blockchainu, bez jeho oprávnění mu tuto hodnotu nemůže nikdo vzít. U konvenčních systémů může kdokoliv autorizovaný změnit řádek v SQL databázi u uživatelova účtu a odebrat mu hodnotu. Tohle je zákonně ošetřeno, ovšem zákon nepokrývá vše.
- **Výkon** — Distribuované systémy typu blockchain mají v porovnání s konvenčními centralizovanými systémy několikanásobně menší výkon. Většina blockchainů dokáže zpracovat pouze 15 až 4000 transakcí za sekundu. Není to však aktuální problém, protože blockchainové technologie nejsou rozšířené natolik, aby se zmíněného limitu dosáhlo. V současnosti se pracuje na nových způsobech konsenzu a implementace blockchainů, které mají limit posunout a být tak připravené na budoucí adopci.
- **Fork** — Negativní vlastnost blockchainu. Jedná se rozdělení blockchainu na dvě části (sdílí stejnou historii, ale v novějších blocích se liší). Existují dva typy:
  - **Soft fork** — Zpravidla se jedná o plánovaný fork. Nastává v případě, když se v protokolu blockchainu změní nějaká hodnota, např. block size z 10 MB na 2 MB. Uzly, které přejdou na novou verzi protokolu, mohou pořád emitovat nové blocky a ostatní uzly je přijmou. Naopak ty uzly, které nepřejdou na nový protokol, přijímají blocky mezi sebou, avšak uzly s novou verzí už takové blocky nepřijímají. Postupem času by měly všechny uzly přejít na novou verzi protokolu.
  - **Hard fork** — Nenapravitelné rozdělení historie blockchainu. Toto nastalo například u Ethereum blockchainu, kde byl využitím chyby v protokolu ukraden velký počet nativní kryptoměny ETH. Jedna část komunity přijala opravu protokolu a vrátila stav blockchainu před útokem, druhá část pokračovala dále pod názvem ETH Classic. Výsledkem jsou blockchainy dva.

4.2.2 Z pohledu uživatele

V následujících bodech jsou shrnuty nejdůležitější vlastnosti z pohledu interakce s blockchainem.

- **Úspěšnost transakce** — Většina blockchainů implementuje poplatek ve formě *gas fee* (viz. kapitolu 4.2.1). Aby se vykonaná transakce zapsala do blockchainu, musí transakce vyhrát pomyslný závod s ostatními transakcemi v mempoolu. Transakce se řadí podle výše *gas price*[2] od

[2] Cena jednotky gas

nejvyšší po nejnižší. Transakce s nejvyšším *gas price* mají největší šanci být zvalidovány či vytěženy uzly kvůli jejich finanční odměně. Pokud tedy uživatel podcení zadání dostačující výše *gas price*, nemusí se jeho transakce do blockchainu zapsat v nejbližších hodinách, dnech či vůbec. Úspěšnost transakce není zaručena.

- **Asynchronicita** - Blockchain je asynchronní systém; transakci trvá určitý čas než se zahrne do blocku a zapíše do blockchainu. Z aplikačního hlediska nedává smysl čekat na vykonání transakce. Aplikace by měla upozornit uživatele, že transakce probíhá a že se čeká na její potvrzení. Toto je důležité zvážit z pohledu UX[3] v aplikacích, které využívají blockchain ke svému fungování.

4.3 Typologie

Typů blockchainů existuje mnoho. Mohou se lišit způsobem, jakým se uzly dohodnou nad novým stavem databáze, podporovaným jazykem pro psaní Smart Contracts, implementací Virtual Machine pro vyhodnocení volání Smart Contracts, ACLs, přístupností blockchainu, škálovací řešení atd. V dalších sekcích proto rozebírám nejdůležitější rozdíly jednotlivých typů.

Blockchainy se dělí podle těchto kritérií:

4.3.1 Podle přístupnosti

Podle participace uzlů

Podle toho, jestli se uzel může připojit do blockchain sítě sám, nebo ne, rozlišujeme dva typy blockchainů.

- **Veřejné** — Uzel se může do sítě kdykoliv připojit a podílet se na chodu blockchainu.
- **Privátní** — Uzel potřebuje oprávnění, aby se do blockchainu připojil. Toto oprávnění zpravidla uděluje admin blockchainu. Oprávnění může mít podobu zápisu do blockchainu nebo například X509 certifikátu, jak je tomu u sítí vytvořených pomocí technologie Hyperledger Fabric.

Podle participace uživatelských účtů

- **Veřejné** — Uživatel může participovat v blockchainu, tj. posílat transakce, bez jakéhokoliv omezení.
- **Permissioned** — Uživatel musí mít oprávnění, aby mohl provádět transakce na blockchainu. Toto oprávnění může mít podobu zápisu do blockchainu nebo X509 certifikátu, jak tomu je u blockchainů vytvořených

[3] User Experience

pomocí technologie Hyperledger Fabric. Stejně jako u uzlů, oprávnění musí být uděleno administrátorem blockchainu.

4.3.2 Podle algoritmu konsenzu

Aby blockchain fungoval, musí se většina uzlů shodnout na nových blocích. Shodování se řídí podle tzv. konsenzus algoritmu, který může být implementován několika způsoby. V bodech níže popisuji tři nejznámější a nejpoužívanější rodiny algoritmů.

- **Proof Of Work** — Aby uzly mohly navrhnout nový block, musí před tím vyřešit výpočetně náročnou úlohu. První uzel vyřeší úlohu, vytvoří block a rozešle jej ostatním uzlům, které ověří správnost výpočtu. Samotný výpočet je náročný, avšak jeho ověření je velmi jednoduché [6]. Algoritmus funguje takto: Transakce se shlukují do blocku. Jakmile je block připraven k vytěžení, Proof Of Work algortimus vygeneruje hash pro daný block. Jednotlivé uzly se potom snaží vygenerovat tzv. target hash, který je menší nebo roven hash blocku. Pokud se to uzlu podaří, dostane odměnu ve formě kryptoměny. Takovýto algoritmus implementuje první a nejznámější blockchain s názvem Bitcoin [7].
- **Proof Of Stake** — Aby uzly mohly navrhnout nový block, musí být zvoleni algoritmem jako leader. Proof Of Stake algoritmus volí uzly podle jistiny na tzv. Staking Smart Contractu. Algoritmy z rodiny Proof Of Stake volí leadera podle velikosti dané jistiny. Čím větší jistina, tím větší šance, že uzel bude zvolen jako leader a získá privilegium vytvořit block a poslat ho ostatním uzlům. Participace v tomto blockchainu je zpravidla možná, pokud je vklad větší než stanovený limit. Například u blockchainu Ethereum [8], který relativně nedávno přešel z Proof Of Work algoritmu na Proof Of Stake, je tento limit 32 ETH.
- **Proof Of Authority** — Tato rodina algoritmů je ideální pro privátní blockchainy. Proof Of Authority algoritmy jsou založené na vzájemné důvěře mezi uzly. Každý uzel blockchainu se považuje za důvěryhodný, nepočítá se tedy s tím, že by uzel navrhoval nesprávné blocky. Na rozdíl od Proof Of Stake algoritmů je v sázce reputace uzlu (namísto jistiny) a tím i autority, která jej provozuje. Algoritmy této rodiny se řadí mezi nejrychlejší z hlediska tvoření konsenzu. Algoritmy z rodiny Proof Of Authority zpravidla uvnitř implementují CFT algoritmy (viz. následující část), je však možné použít i BFT.

V bodech výše jsem popsal rodiny algoritmů, které se používají v blockchainových decentralizovaných distribuovaných systémech. V další části vymezuji dva nejzákladnější typy algoritmů.

Crash Fault Tolerant

Crash Fault Tolerant (CFT) algoritmus zaručuje fungování konsenzu i v tom případě, že dojde k selhání určitého počtu uzlů nebo dojde k rozdělení uzlů v

blockchainu kvůli problémům s připojením. Zaručuje tedy fungování sítě při selhání některých komponent, avšak nezaručuje správné fungování sítě, pokud se určitý uzel nechová čestně nebo navrhuje blocky se špatnými daty. Pro takové případy existuje rodina algoritmů Byzantine Fault Tolerant, zkráceně BFT (viz další část). Mezi nejznámější CFT algoritmy se řadí následující:

- **RAFT** — Nejznámější algoritmus rodiny CFT. Využívá se v distribuovaných systémech; v kontextu blockchainu se využívá v privátních permissioned blockchainech postavených nad technologií Hyperledger Fabric a několika dalších. Funguje takto: Každý uzel musí být v právě jednom stavu z následujících stavů — Leader, Follower a Candidate. RAFT algoritmus definuje časově omezené etapy. Každá etapa začíná volbou, ve které se jednotlivé se uzly ve stavu Candidate uchází o pozici Leadera. Pokud Candidate vyhraje volbu, stává se z něj Leader a ostatní uzly se podřídí se statusem Follower. Jakmile je Leader zvolen, zůstává jím až do doby, dokud jiný uzel nevyvolá novou volbu (např. kvůli ztrátě spojení s Leader uzlem) [9]. V blockchainu je Leader uzel zodpovědný za řazení transakcí do blocků a jejich rozesílání ostatním uzlům.
- **Clique** — Známý algoritmus z rodiny CFT, který je mimo jiné implementován v blockchainech postavených nad technologií Hyperledger Besu[4]. Využívá se v Proof Of Authority privátních sítí a funguje následovně: V každém kroku je zvolen vůdčí uzel a N/2 – 2 uzlů asistentů. Tato množina uzlů se stará o sestavení blocku. Aby byl úspěšně vytvořen block, zvaliduje každý uzel ze zmíněné množiny n transakcí, které se nachází v mempoolu uzlů. Po validaci rozešlou jednotlivé uzly svůj block ostatním uzlům mimo svou množinu. Je zde však jeden problém. Jednotlivé uzly rozesílají své blocky nezávisle na sobě. To znamená, že uzel mimo zvolenou množinu uzlů dostane block dříve od asistenta než od vůdce, což vytvoří tzv. Hard Fork. Nicméně existují metodiky, jak tomuto zabránit, např. GHOST protokol [10]. Pro produkční blockchainy není tento algoritmus doporučen právě kvůli častému Hard Forku.

Byzantine Fault Tolerant

Byzantine Fault Tolerant (BFT) algoritmus zaručuje fungování konsenzu i u případů, kde se jednotlivé uzly chovají nečestně. Aby blockchain fungoval správně, musí se na novém stavu databáze, tj. novém blocku, shodnout alespoň většina uzlů (u většiny algoritmů podporující BFT je nutno mít pro správné fungování konsenzu celkový počet uzlů větší než 3N, kde N je počet nečestných uzlů).

- **IBFT** — Istanbul BFT Consensus Algorithm je deterministický algoritmus pro replikaci databázového stavu mezi jednotlivými uzly, který zaručuje v čestných epochách časovou složitost komunikace O(n2). Algoritmus zavedl 4 kroky v každé epoše, konkrétně pre-prepare, prepare, commit a round-change.

[4] https://www.hyperledger.org/projects/besu

- **QBFT** — Vylepšení IBFT algortimu s cílem vyřešit jeho bezpečnost a špatnou latenci. IBFT algoritmus se může zaseknout na dvou čestných uzlech, které se zamknou na dvou odlišných blocích. QBFT algoritmus též implementuje tři první kroky v epoše, tj. pre-prepare, prepare a commit. Uzel na začátku každé epochy vytvoří návrh na block a rozešle jej ostatním uzlům s hlavičkou pre-prepare. Ostatní uzly tuto zprávu přijmou a rozešlou zprávu prepare ostatním uzlům. Jakmile uzel přijme prepare zprávu, rozešle zprávu commit. Nakonec uzel vloží block do své kopie blockchainu, pokud dostane alespoň 2N/3 commit zpráv od ostatních uzlů. Pokud nedojde ke konsenzu během určité doby, dojde k round-change a resetují se interní hodiny všech uzlů [10] [11]. QBFT algoritmus je plně implementován v blockchainech postavených nad technologií Hyperledger Besu. Tento algoritmus se doporučuje pro enterprise-grade blockchainy, jelikož je průměrně rychlejší než IBFT.
- **PolyBFT** — Tento algoritmus rozšiřuje IBFT 2.0 algoritmus a přidává k němu Proof Of Stake řešení, aby byl implementovatelný pomocí Smart Contracts. Tohle řešení využívají blockchainy postavené nad technologií Polygon Edge [12].

4.4 Výběr blockchainu pro meziuniverzitní systém

V systému se vyskytují citlivá data. To znamená, že veřejné blockchainy nejsou pro tento případ vhodné. Jelikož tuto síť budou provozovat univerzity, jedná se o privátní systém, v rámci kterého si jednotliví aktéři věří. Síť musí být taktéž dostatečně rychlá, protože počet potenciálních uživatelů se pohybuje v řádu milionů. Blockchain musí být permissioned, jelikož jednotlivé osoby, které budou chtít tento systém využívat, budou muset mít ověřenou identitu. Každá osoba tak bude muset vlastnit určitý certifikát opravňující k participaci v systému. Je zcela klíčové, aby k datům na blockchainu měly přístup jen oprávněné entity. Pro účely meziuniverzitního informačního systému považuji za nejvhodnější blockchain postavený nad technologií Hyperledger Fabric s využitím konsenzus algoritmu RAFT.

Kapitola 5
Hyperledger Fabric

Tato kapitola se zabývá frameworkem na tvorbu enterprise-grade permissioned blockchainů. Vymezuje jeho vlastnosti a komponenty.

5.1 Charakteristika

Hyperledger fabric je open-source platforma pro tvorbu privátních enterprise-grade permissioned blockchainů. Nabízí modulární a flexibilní komponenty pro tvorbu aplikací. Platforma byla vytvořena pod záštitou Linux Foundation a je aktuálně vyvíjena 200 vývojáři z 35 různých organizací. Blockchainy stavěné nad touto technologií nepotřebují nativní kryptoměnu, jelikož její jediný záměr je ochrana před DDOS útoky na jednotlivé uzly a tento typ útoku nehrozí v systému, kde všechny uzly znají identitu všech aktérů v síti [13].

5.2 Komponenty

V této sekci objasňuji nejdůležitější komponenty frameworku Hyperledger Fabric.

5.2.1 Infrastruktura

Orderer

Jedna ze dvou nejdůležitějších komponent Hyperledger Fabric systému. Orderer uzel je zodpovědný za správné řazení transakcí do blocků a rozesílání těchto blocků ostatním uzlům sítě. Dále kontroluje, jestli jsou jednotlivé transakce v souladu s nastavením pravidel sítě. Zpravidla se v síti vyskytuje více než jeden uzel typu Orderer. Dohromady tato množina uzlů typu Orderer tvoří tzv. Ordering Service.

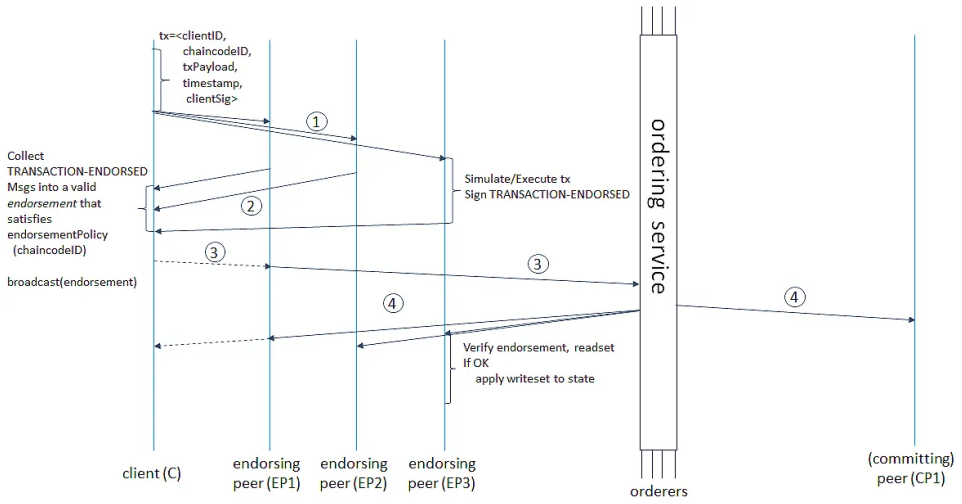
Ordering Service

Slouží jako *mozek* blockchainu implementující jeden z možných konsenzus algoritmů. I když se doporučuje Crash Fault Tolerant RAFT algoritmus pro

enterprise-grade blockchainy, je možné využívat i Byzantine Fault Tolerant PBFT algoritmus, Kafka nebo Solo ordering.

Peer

Druhá nejdůležitější komponenta systému zodpovědná za aplikační logiku a uchovávání stavu blockchainu. Jednotlivé uzly typu Peer uchovávají kopii databáze ve svém uložišti a starají se o management Smart Contracts, kterým se v kontextu Hyperledger Fabric technologie říká Chaincode.



Obrázek 5.1: Transaction flow uvnitř Hyperledger Fabric blockchainu [14].

5.2.2 Databáze

Blockchainové řešení implementované pomocí technologie Hyperledger Fabric může využívat dva typy databází:

- **LevelDB** — Key-value databáze. Všechny zápisy reprezentuje jako klíč a hodnotu ve tvaru textových řetězců. Řadí se do rodiny *NoSQL* databází.
- **CouchDB** — Charakterizuje se též jako *NoSQL* databáze. Narozdíl od LevelDB implementuje indexy a umožňuje tak rychlé vyhledávání, filtrování a řazení.

LevelDB je v blockchainových řešeních implementovaných pomocí technologie HyperLedger Fabric používaná vždy. Slouží k uchovávání historie blockchainu zápisem bloků. CouchDB je možné do řešení přidat — narozdíl od LevelDB reprezentuje aktuální globální stav a nabízí aplikační rozhraní pro optimalizované čtení.

5.2.3 Certificate Authority

Jako Certificate Authority (CA) se v kryptografii nazývá entita, která podepisuje a zprostředkovává certifikáty. Může se jednat o tzv. Root Certificate

Authority, která si podepíše svůj certifikát sama, nebo o tzv. Intermediate Certificate Authority, která svůj certifikát obdrží od jiné Intermediate Certificate Authority nebo od Root Certificate Authority. Hyperledger Fabric blockchain je privátní a permissioned, což znamená, že pro interakci s blockchainem je potřeba vlastnit určitý certifikát. Hyperledger Fabric nabízí svou vlastní implementaci Certificate Authority v podobě docker image pod názvem Fabric CA, pomocí které se lze takové certifikáty generovat.

5.2.4 Identita

V této sekci vysvětluji, jak funguje identita v Hyperledger Fabric frameworku.

Organizace

Hlavními entitami v Hyperledger Fabric blockchainu jsou organizace. Organizace je dána root certifikátem vygenerovaným pomocí Certificate Authority, která je ve vlastnictví dané organizace. Díky tomuto přístupu lze všechny certifikáty vygenerované CA organizací ověřit vůči root certifikátu dané organizace a tím zjistit, zda-li certifikát patří ke zmíněné organizaci. Jednotlivé certifikáty nesou mimo jiné informaci o tom, jakou má certifikát v organizaci roli. Všechny certifikáty generované CA ve vlastnictví organizací mohou mít jednu ze čtyř rolí v dané organizaci. Těmito rolemi jsou:

- **Orderer** — Používá se pro certifikáty, které bude využívat uzel typu Orderer.
- **Peer** — Používá se pro certifikáty, které bude využívat uzel typu Peer.
- **Admin** — Používá se pro uživatele s právem dělat administrativní úkony vzhledem k organizaci, která jim certifikát vydala (pokud konfigurace sítě neříká něco jiného).
- **Client** — Používá se pro všechny ostatní klienty zahrnující aplikace a uživatele.

Člen organizace

Člen organizace patří k právě jedné organizaci. Je určen X.509 certifikátem, který mu poskytla daná organizace. V základním nastavení blockchainu může entita s certifikátem typu **Client** číst data a psát je do blockchainu. **Admin** může upravovat definici Chaincode, **Peer** může schvalovat transakce a **Orderer** může provádět řazení transakcí. Celý ekosystém Hyperledger Fabric pracuje pouze s čtyřmi typy rolí; vlastní role nelze přidávat. Pokud je nutná úprava práv rolí, lze tak učinit pouze v konfiguraci blockchainu.

Každá entita, která není organizací, musí patřit k určité organizaci. Díky tomuto přístupu můžeme sledovat aktivitu uživatelů na blockchainu. Toto se hodí v případě, kdy některý uživatel provádí ilegální transakce. Pomocí adres a historie blockchainu jsme schopni jej identifikovat a penalizovat.

X.509 certifikáty

Jedná se o standard certifikátu, který reprezentuje dvojici veřejného a privátního klíče. X.509 funguje na bázi vizitky vlastníka. Veřejný klíč, který lze získat z klíče privátního, je zapsán v certifikátu a potvrzuje tím, že se skutečně jedná o certifikát k této dvojici klíčů. Tento typ certifikátů se využívá převážně v internetových protokolech jako TLS/SSL, ale i na digitální podpis jako např. v technologii Hyperledger Fabric. Jako signature algoritmus se používá ECDSA pomocí SHA256, stejně jako v blockchainech podporující EVM.

ECDSA

Jedná se o protokol digitálního podpisu, který pro podepisování využívá eliptické křivky. V Hyperledger Fabric se používá křivka secp256r1.

SHA256

Secure Hashing Algorithm 256 je rozšířený hashovací algoritmus, který zobrazí jakýkoliv datový vstup na řetězec s velikostí 256 bitů.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            28:ac:1c:73:48:80:1a:04:85:63:0a:22:d4:ae:dd:d4:3b:e1:02:8b
    Signature Algorithm: ecdsa-with-SHA256
        Issuer: CN=ca-orderer-server
        Validity
            Not Before: Jul 26 11:05:00 2023 GMT
            Not After : Jul 22 11:05:00 2038 GMT
        Subject: CN=ca-orderer-server
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:18:8d:e7:4b:7d:9d:2e:be:32:12:20:8a:da:49:
                    91:4b:38:35:66:bd:0c:98:bd:6c:a9:b0:58:a4:ef:
                    7a:8f:f7:aa:4a:63:10:61:71:f3:71:b7:73:d1:ef:
                    e0:0b:7a:77:ab:39:c5:e5:c4:89:a7:d4:31:3d:eb:
                    68:cf:cc:c4:01
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Key Usage: critical
                Certificate Sign, CRL Sign
            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:0
            X509v3 Subject Key Identifier:
                22:4A:1D:2F:99:21:FC:F4:35:2B:20:1C:C0:72:5C:DD:75:E4:59:13
    Signature Algorithm: ecdsa-with-SHA256
         30:45:02:21:00:a7:dd:08:70:5c:09:01:8e:6b:72:54:26:ac:
         86:4f:9b:c3:d3:8c:b0:5c:93:8a:f1:e3:a3:35:44:44:db:ef:
         8e:02:20:2c:c8:6e:8d:9d:23:f1:0d:c2:04:e0:de:1b:01:3e:
         84:b1:be:a1:d5:d3:83:ba:44:d9:5c:20:7b:66:17:bf:f2
```

Obrázek 5.2: Příklad root certifikátu orderer organizace

5.2.5 Channel

Channel a blockchain jsou synonyma. Channel vzniká pomocí počáteční konfigurace a uchovává vlastní historii blocků. Hyperledger Fabric framework

umožňuje existenci více Channels na jednotlivých uzlech. Jeden uzel tak může participovat na více blockchainech zároveň. Tento přístup umožňuje pro jednotlivé organizace založit speciální blockchain, se kterým budou komunikovat pouze dané organizace. Uzly typu Peer a Orderer tvoří dohromady infrastrukturu, v které může běžet jeden a více Channels neboli blockchainů. Channel jako takový je pak společným datovým prostorem jedné či více organizací uchovávající v sobě pravidla o tom, co smí jednotlivé organizace. Channel taky registruje Chaincodes, které při volání zapisují a čtou data daného Channelu. Chaincodes, které jsou registrovány v Channel A, mohou interagovat s Chaincode z Channel B a naopak.

5.2.6 Chaincode

Chaincode je kód, jehož implementace je zapsána na blockchainu (Channel). Chaincode je tedy určitá logika, která pracuje nad daty daného Channelu a nabízí rozhraní pro interakci s tímto Chaincode. Aby mohl být Chaincode využíván klienty, musí být nainstalován alespoň na jednom uzlu typu Peer. To samo o sobě nestačí, administrátor organizace musí potvrdit transakcí, že souhlasí jménem své organizace s implementací daného Chaincode. Po tomto kroku mohou všechny entity organizace komunikovat s Chaincode. Podle pravidel sítě nemůže administrátor nainstalovat a potvrdit Chaincode na uzlech jiných organizací. Hyperledger Fabric dále umožňuje měnit implementaci Chaincode. Stačí nainstalovat novou verzi na uzly své organizace, potvrdit novou implementaci a aktualizovat verzi Chaincode. Tato pravidla se však mohou lišit v závislosti na konfiguraci daného Channel. Může existovat pravidlo, že privilegium instalace a potvrzení implementace Chaincode mají pouze administrátoři z organizace A. Můžeme mít i pravidlo, že definici Chaincode musí potvrdit alespoň dva administrátoři z každé organizace v Channel. Pravidla mohou být opravdu velmi flexibilní.

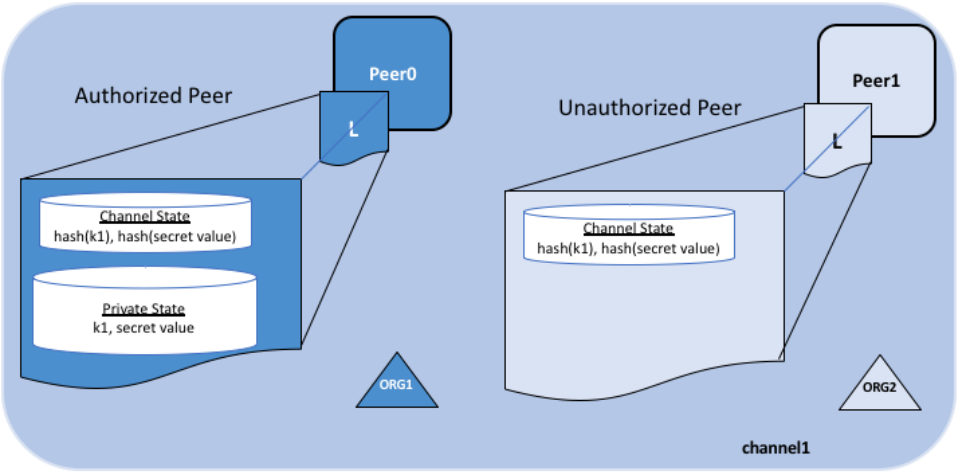
5.2.7 Private Data Collection

Dle mého názoru největší přednost Hyperledger Fabric frameworku. Jedná se o způsob, jak sdílet citlivá data mezi jednotlivými uzly typu Peer v různých organizacích. Private Data Collection (PDC) se skládá ze dvou částí:

- **Data** — Ta jsou posílána peer-to-peer metodou mezi jednotlivými uzly typu Peer pomocí Gossip protokolu. Data dostanou a uloží pouze autorizované uzly, ostatním uzlům se posílá pouze hash dat [15].
- **Hash dat** — Hash privátních dat, který si uloží všechny uzly. Slouží jako důkaz, že daná transakce proběhla [15].

Pro každý Chaincode může existovat N Private Data Collections. Jedná se tedy o konfiguraci organizací, které mají přístup k určitým citlivým informacím. Tato konfigurace se může během času měnit v závislosti na požadavcích entit v blockchainu. Existuje řada metodik, jak sdílet privátní data. Níže uvádím hlavní z nich:

- **Chaincode** — Konfigurace sítě může umožňovat ostatním entitám mimo určitou organizaci interagovat s uzly typu Peer té organizace. Pokud by se citlivá data nacházela pouze na uzlu té organizace, může se před tato data umístit jakási přístupová brána pomocí Chaincode. Na úrovni Chaincode by následovala kontrola, jestli entita doptávající se na data má povolení. Pokud ano, uzel vrátí data, pokud ne, přístup bude odepřen [15].
- **Změna Private Data Collection konfigurace** — Můžeme změnit konfiguraci PDC a povolit přístup k citlivým datům ostatním organizacím. Ve chvíli, kdy se transakce o změně konfigurace PDC potvrdí, začnou uzly ostatních organizací ukládat citlivá data [15].
- **Offchain** — Data lze sdílet mimo blockchain konvenčními metodami. Organizace, která data dostala, si může data upravit pomocí hashing algoritmu a porovnat správnost hashe s hashem na blockchainu. Tímto způsobem si ověří, že data jsou opravdu autentická [15].



Obrázek 5.3: Ilustrace Private Data Collection [15].

5.2.8 Fabric Gateway

Je to externí knihovna umožňující aplikacím inteligentně komunikovat s uzly typu peer a orderer jménem uživatele, který knihovnu používá — pro používání knihovny je nutné poskytnout uživatelský certifikát). Knihovna je schopná z adresy jednoho uzlu typu peer extrahovat informace o potřebném endorsementu transakce pro její správné vykonání. S touto informací pošle požadavek na endorsement potřebným uzlům (i mimo organizaci dotazujícího se uživatele). Po získání všech endorsementů pošle Fabric Gateway požadavek commit uzlu typu orderer a transakce se zapíše do Channel všech uzlů. Pokud Fabric Gateway interaguje s privátními daty, zohledňuje konfiguraci Private

Data Collection. Ilustrace tohoto procesu je k vidění na obrázku 5.1. Tuto knihovnu použiji dále v implementační části.

Kapitola 6
Implementační část

6.1 Návrh řešení

V této části představuji návrh řešení decentralizovaného meziuniverzitního informačního systému s využitím technologie Hyperledger fabric. Vymezuji všechny úrovně systému a navrhuji jednotlivé entity, které budou uchovávat data. Dále navrhuji pár jednoduchých případů užití a sestavuji jejich sekvenční diagramy.

6.1.1 Organizace

Decentralizovaný meziuniverzitní informační systém pracuje s akademickými daty interně a externě. Interní entity v systému budou převážně autoři akademických dat, tzn. jednotlivé univerzity. Externí entity budou ryze příjemci dat, tzn. zaměstnavatelé, externí společnosti, vládní sektor či nadnárodní řetězce. Z toho vyplývá, že v takovémto systému budou figurovat dva typy Hyperledger Fabric organizací:

Univerzita

Jedná se o typ organizace, která zaštituje všechny své osoby a data. Každá organizace tohoto typu vlastní alespoň jednu orderer komponentu a alespoň jednu peer komponentu. Orderer komponentu vlastní proto, aby se univerzita podílela na konsenzu v blockchainu. Na peer komponentě uchovává univerzita svá privátní data o studentech, známkách a dalších důležitých entitách. Taky zde eviduje veřejnou část dat ostatních univerzit — přispívá k redundanci dat, pokud nějaký peer přestane fungovat, systém ho nahradí dalším uzlem. Kvůli existenci privátních dat na blockchainu musí mít tato entita registrovanou tzv. Private Data Collection (viz. kapitolu 5.2.7). Do této kolekce má přístup pouze univerzita sama. Sdílení těchto dat bude probíhat přes aplikační logiku smart contractu, jak ukazuji v následujících kapitolách. Účty registrované pod tímto typem organizace mohou komunikovat pouze s peer komponentami vlastní organizace (psaní a čtení dat).

Veřejná organizace

Jedná se o typ organizace, která nevlastní žádné komponenty ani typu peer, ani typu orderer. Organizace pouze zaštiťuje uživatelské účty externích entit a vymezuje jim určitá pravidla v blockchainu. Organizace je velmi jednoduchá. Účty registrovány pod touto organizací mohou provolávat peer komponenty všech univerzit. Je tu však jedno pravidlo – mohou data pouze číst. Jestli mohou nebo nemohou číst určité zápisy v blockchainu závisí na povoleních, která jsou daná aplikační logikou na smart contractech.

Vymezil jsem dva typy organizací, které se budou vyskytovat v modelovém systému. Pro ukázku systému stačí mít pouze tři příklady těchto typů.

Univerzita A

Jedná se o organizaci typu Univerzita. S přiřazeným identifikátorem UviversityAMSP reprezentuje první univerzitu v systému.

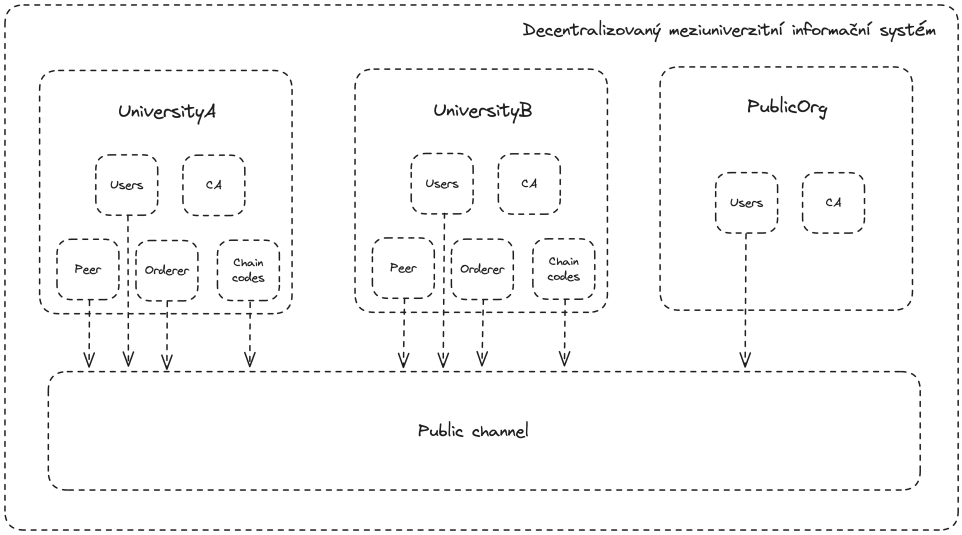
Univerzita B

Jedná se o organizaci typu Univerzita. S přiřazeným identifikátorem UniversityBMSP reprezentuje druhou univerzitu v systému.

Veřejná organizace

Jedná se o organizaci typu Veřejná organizace s identifikátorem PublicOrgMSP. Reprezentuje všechny externí entity, které mají zájem číst data z decentralizovaného meziuniverzitního informačního systému.

Tímto jsem vymezil všechny organizace, které se budou v systému vyskytovat. Na obrázku níže je znázorněna topologie sítě s jednotlivými organizacemi.



Obrázek 6.1: Ilustrace topologie organizací v systému

Všechny organizace budou komunikovat pouze s skrze jeden channel s názvem public (viz. kapitolu 5.2.5). Šipky představují komunikaci komponent skrze public channel.

6.1.2 Chaincode

Vymezil jsem jednotlivé organizace, které se budou v systému nacházet, a také jejich pravidla na úrovni channel. Abstraktní infrastruktura je připravena, nyní je řada na aplikační logiku. Systém se bude skládat ze dvou smart contractů:

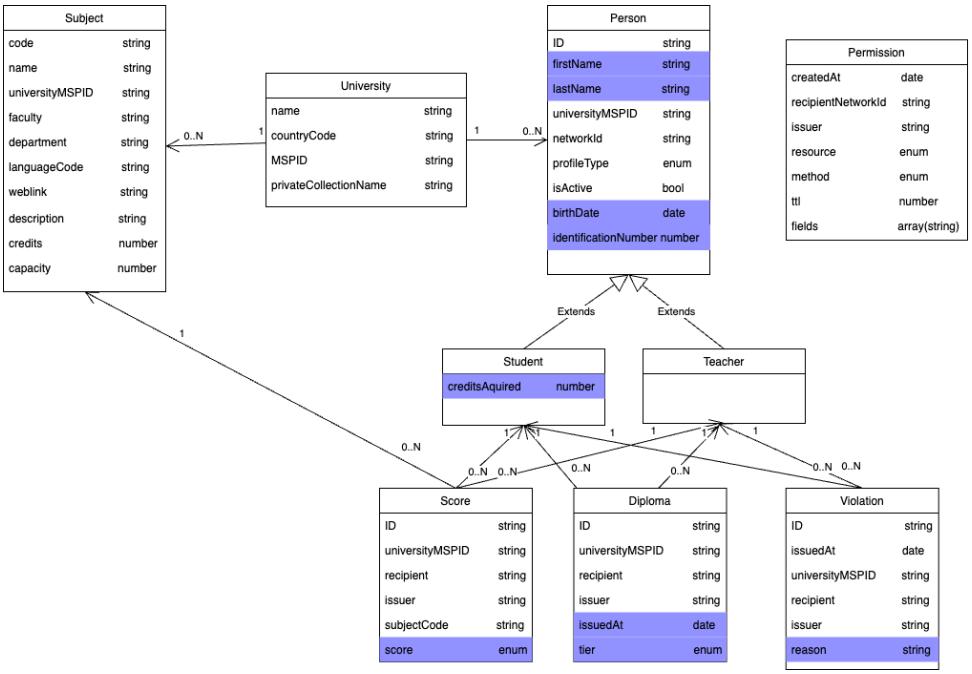
- **UniversityRegistry** — Smart contract je zodpovědný za ukládání dat do blockchainu a čtení z nich. Bude zprostředkovávat rozhraní pro tvorbu a čtení entit, které budou moci využívat pouze účty registrované pod danou univerzitou. Tato pravidla budou vynucená na úrovni tohoto smart contractu a nelze je obejít. Tento smart contract bude dále nabízet dvě funkce pro čtení dat, které budou moci využívat externí účty registrované pod veřejnou organizací (nebo jakoukoliv jinou). Tyto dvě funkce budou chráněny pomocí AccessManager smart contractu.

- **AccessManager** — Jedná se o smart contract zabývající se manipulací s povoleními pro čtení dat prostřednictvím externích účtů. Nabízí rozhraní pro zápis a čtení povolení. Čtení povolení bude využívat UniversityRegistry smart contract. Zápis bude možné provést pouze přes studentský účet.

Detaily a implementaci smart contractů podrobně popisuji v implementační části (viz. kapitolu 6.2.4).

6.1.3 Datové entity

V předchozí části jsem nastínil, jaké smart contracty jsou v systému potřeba. AccessManager smart contract bude pracovat s povoleními (permissions) a UniversityRegistry smart contract bude pracovat s akademickými daty. Aby byl systém využitelný v reálném světě, bylo by nutné modelovat několik desítek datových entit. V tomto případě postačí pro ukázku entit sedm. Nejprve představuji Class diagram jednotlivých entit a následně popisuji detailněji, co dané entity představují.



Obrázek 6.2: Class diagram

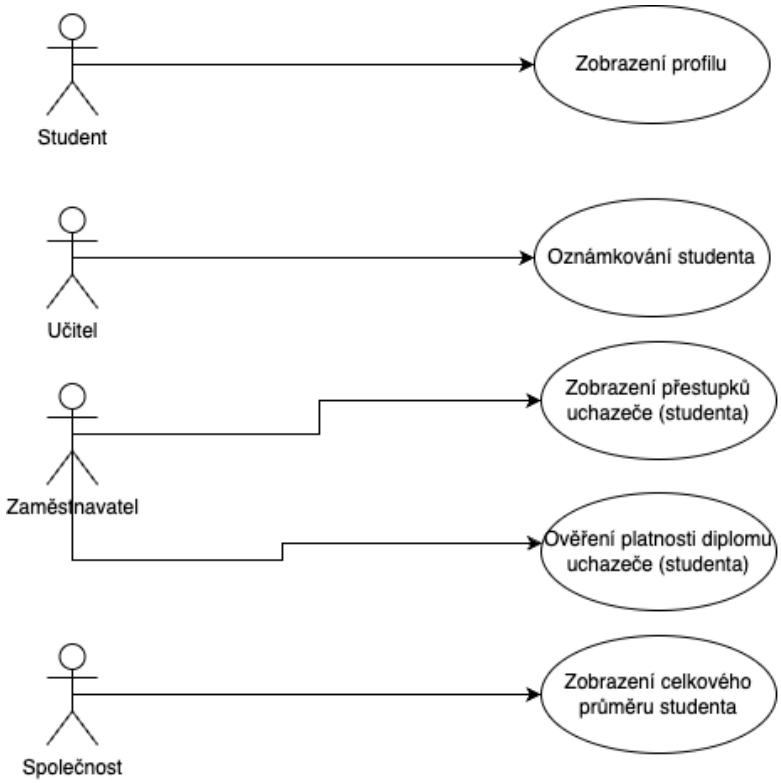
Jelikož nepracuji s relační databází, asociace jsou realizovány pomocí identifikátorů, nikoliv cizích klíčů. Barevně jsou vyznačena pole, která nejsou veřejná a nesdílejí se napříč univerzitami. Vysvětlení jednotlivých entit:

- **University** — Reprezentuje informace o univerzitě a ukládá velmi důležitá pole privateCollectionName. Všechna privátní data této univerzity se zapisují do dané kolekce. Private Data Collection stejného jména musí být registrována na úrovni smart contractů.
- **Subject** — Reprezentuje předmět na univerzitě. Všechna data jsou u tohoto modelu veřejná.
- **Person** — Reprezentuje uživatele patřící k univerzitě. Uživatelský účet je dán certifikáty, které generuje Certificate Authority. V aplikační logice se musí tento účet referencovat, proto jsem navrhl tuto datovou entitu pro ukládání informací.
- **Score** — Reprezentuje hodnocení studenta učitelem v určitém předmětu.

- **Diploma** — Reprezentuje diplom, zatím zahrnuje pouze Bc., Ing. a Ph.D.
- **Violation** — Reprezentuje porušení školního řádu, např. plagiát
- **Permission** — Reprezentuje povolení číst data externím účtem.

6.1.4 Případy užití

Pro ukázku funkčnosti systému bude můj systém implementovat následující případy užití.

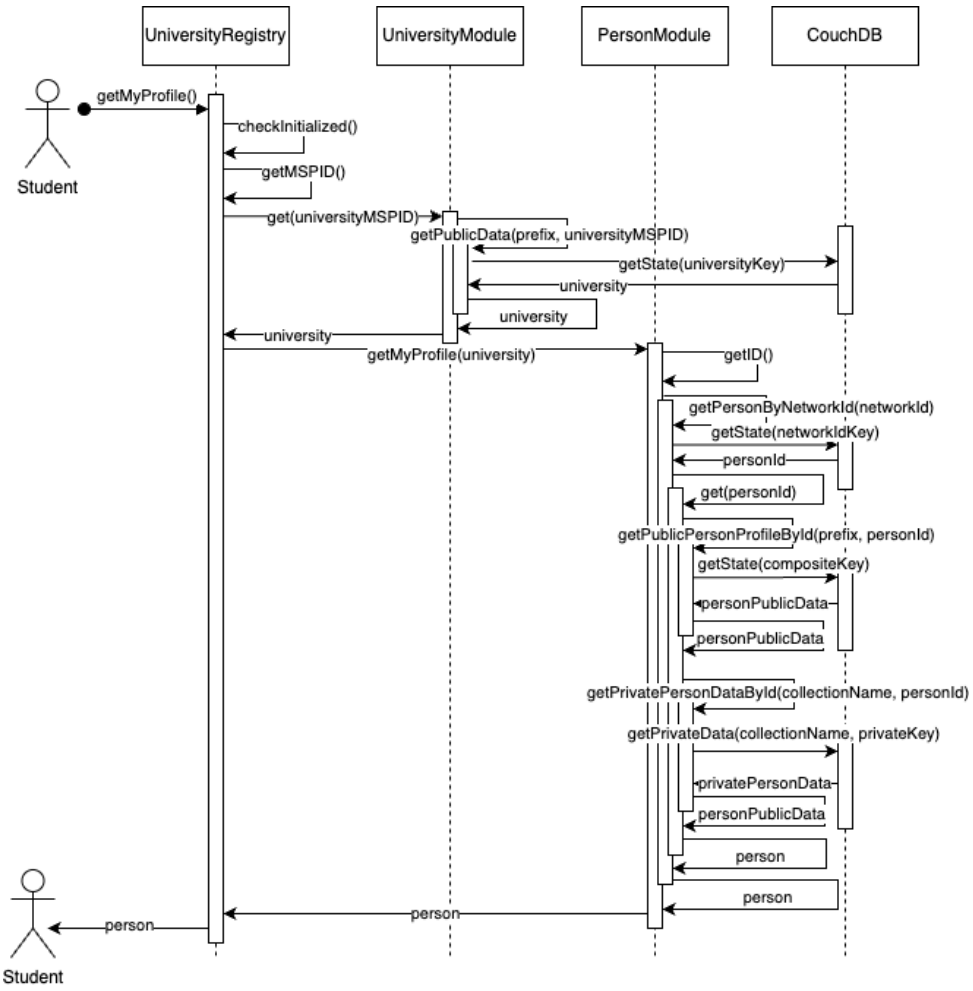


Obrázek 6.3: Usecase diagram

Sekvenční diagramy uvádím pouze pro dva případy užití, a to zobrazení profilu studenta a zjištění všech jeho přestupků.

Zobrazení osobních údajů studentem

Základní vlastností jakéhokoliv systému je možnost zobrazit si svá uložená data. V tomto případě se jedná čistě o interní komunikaci v rámci jedné univerzity, není třeba druhého smart contractu AccessManager. Na obrázku níže je zobrazen sekvenční diagram tohoto případu užití.



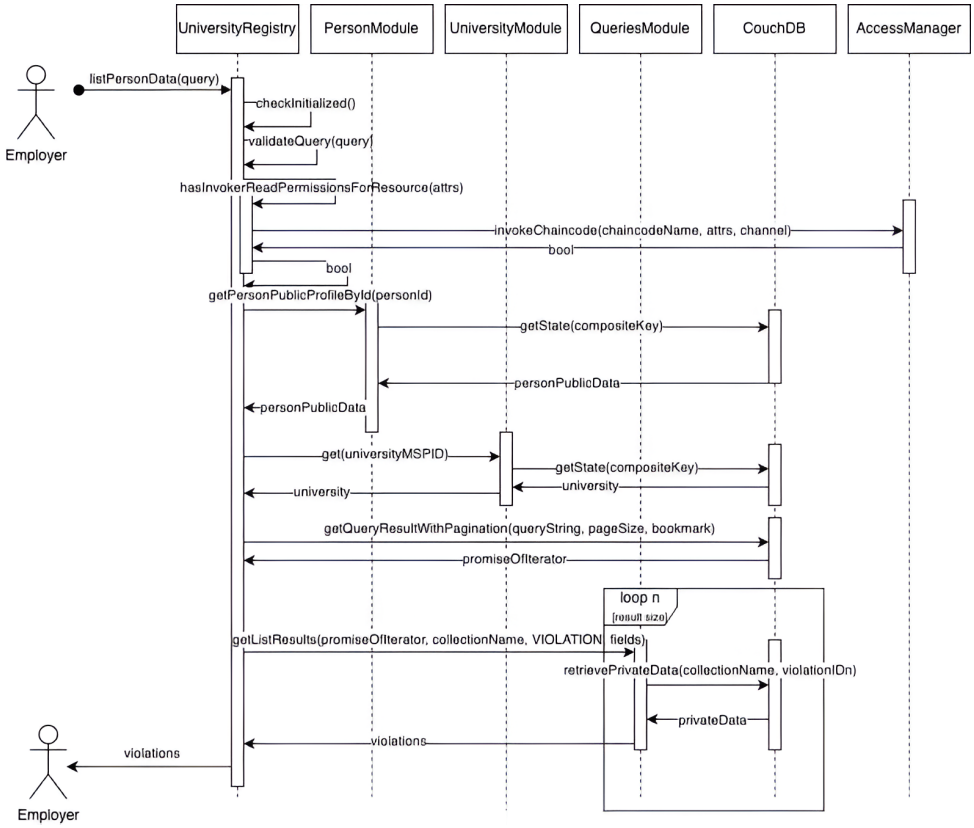
Obrázek 6.4: Sekvenční diagram zobrazení svého profilu

Student zavolá metodu getMyProfile() na UniversityRegistry smart contractu, který je nainstalován a schválen studentovou univerzitou na uzlu typu peer. Jako kontext této transakce se používá studentova identita, která je automaticky získána z podpisu transakce pomocí klíče studentova certifikátu. V první řadě se kontroluje, zda je smart contract připraven přijímat dotazy, tj. checkInitialized() funkce proběhne bez chyby. Pokud je smart contract připraven k vyřizování požadavků, získá MSPID své univerzity z kontextu a dotáže se na informace o této univerzitě. Pokud univerzita neexistuje, vrátí smart contract chybovou hlášku. Dále se získá profil studenta pomocí interní metody getMyProfile(university) nacházející se v person části smart contractu. Nejprve se musí získat tzn. Network Id — pomocí funkce getID(), která identifikuje certifikát uživatele. Podle tohoto networkId se smart contract dotáže databáze na personId, který už identifikuje profil uživatele v tomto systému. Pomocí tohoto personId se extrahuje veřejná a privátní data, složí se dohromady do datové struktury a vrátí se uživateli. Person module a University module je čistě logické členění UniversityRegistry smart contractu (tyto moduly uvádím jako samostatné javascriptové soubory dále v

implementační části).

Zjištění všech přestupků studenta zaměstnavatelem

Jedná se o dotaz externího subjektu na citlivá akademická data studenta určité univerzity. Zde je již potřeba kontrolovat přístup k těmto datům pomocí druhého smart contractu s názvem AccessManager. Na obrázku níže je vypracován sekvenční diagram takového případu užití.



Obrázek 6.5: Sekvenční diagram zobrazení všech přestupků studenta

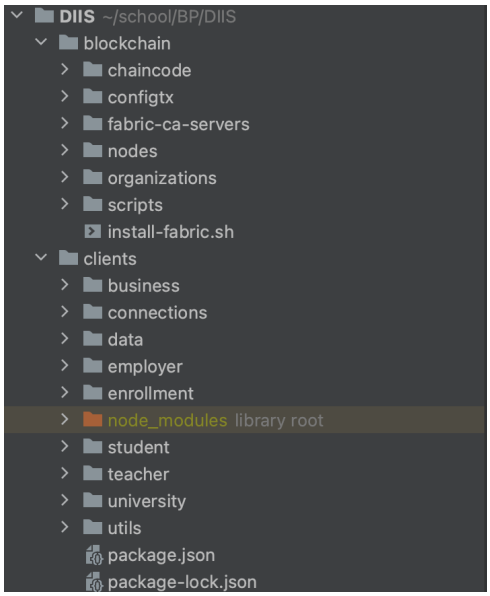
Zaměstnavatel zavolá funkci listPersonData(query) v určitém formátu. Funkce bude velmi obecná, aby podporovala operaci LIST nad všemi typy objektů. V první řadě se zkontroluje, jestli je UniversityRegistry smart contract připraven zpracovávat požadavky. Pokud ano, zkontroluje se, zda je query parametr ve správném formátu. UniversityRegistry smart contract se zeptá AccessManager smart contractu, zda má zaměstnavatel práva číst daná data. Pokud ano, zjistí se, do které univerzity patří student, jehož data chce zaměstnavatel číst. Toto je důležité, protože potřebujeme znát název Private Data Collection univerzity, abychom mohli získat všechna privátní data k jednotlivým objektům. Jakmile UniverzityRegistry smart contract zná univerzitu, do které student patří, extrahuje si z databáze všechny veřejné zápisy hledaných dat. Jeden po druhém je pak prochází a ptá se na privátní

data pomocí názvu Private Data Collection. Následně výsledky složí do jednoho pole a vrátí zaměstnavateli.

6.2 Implementace

V této kapitole detailně popisuji architekturu systému a implementaci smart contractů.

6.2.1 Adresářové rozdělení



Obrázek 6.6: Adresářová struktura projektu

Projekt jsem rozdělil na dvě hlavní části — *blockchain* a *clients*. V *blockchain* adresáři se nachází všechny soubory, které se podílejí na tvorbě blockchainu a jeho aplikační logiky. V *clients* adresáři jsou ukázky využití mého systému. V další části vysvětluji, jaká je úloha každého adresáře v projektu.

chaincode

Jedná se o adresář, který v sobě má veškerou implementaci smart contractů. Těmi jsou UniversityRegistry a AccessManager.

configtx

Adresář má v sobě defaultní konfigurace uzlů typu peer a orderer, které jsou přepisovány při vytvoření jednotlivých docker kontejnerů. Je zde i nejhlavnější soubor celého blockchainu – configtx.yaml. Ten v sobě nese konfiguraci organizací a jejich pravidel. Podle něj jsem vytvořil hlavní channel s názvem public.

fabric-ca-servers

Jedná se o adresář se všemi konfiguracemi Certificate Authority serverů. Lze zde najít i hlavní identity pro každou organizaci, kterou jsem generoval při implementaci tohoto projektu. Každý CA server má dvojí uplatnění – podporuje generování jak TLSCA, tak CA certifikátů pro všechny entity v blockchainu. Jsou zde přiloženy rovněž jednotlivé docker compose konfigurace pro jednoduché vytvoření kontejnerů pro registraci nových uživatelů a entit.

nodes

V tomto adresáři jsou docker compose konfigurační soubory ke všem uzlům typu peer a orderer všech organizací.

organizations

Zde jsou uloženy všechny potřebné identity ke správnému chodu blockchainu. Můžeme zde najít certifikáty všech uzlů typu peer a orderer, dále certifikáty administrátorských účtů všech organizací, které se podílejí na blockchainu.

scripts

Adresář, ve kterém jsou naprogramovány jednoduché skripty pro tvorbu, zrušení a restart blockchainu. Dále jsou zde skripty pro instalování smart contractů, udělování souhlasu s používáním a commit logika.

podadresáře clients adresáře

Jednotlivé subjekty mají vlastní adresář, který obsahuje kód simulující používání systému. Podadresář *data* v sobě nese předvyplněná data pro rychlejší využívání systému. Adresář *enrollment* obsahuje proces pro registraci nových uživatelů do blockchainu ve smyslu vydávání nových certifikátů, nikoliv registrace profilu studenta či učitele. Podadresář *connections* v sobě obsahuje JSON konfigurační soubor pro připojení k uzlům pomocí externí javascriptové knihovny. Adresář *utils* obsahuje pomocné funkce.

6.2.2 Certifikáty

Tato část popisuje generování certifikátů pro všechny typy organizací, které se podílí na chodu blockchainového ekosystému. Stačí zde popsat pouze dva typy organizací: Univerzita a Veřejná organizace, jelikož první typ vlastní uzly typu peer a orderer a druhá obsahuje čistě uživatelské účty.

Univerzita

Pro generování certifikátů pro tento typ organizací je třeba založit CA server. K tomu stačí pouze docker image hyperledger/fabric-ca (v tomto projektu používám verzi 1.5.6). Docker compose konfigurační soubor vypadá takto.

```yaml
version: '3.7'


services:
  DIIS-public:
    image: hyperledger/fabric-ca:1.5.6
    container_name: DIIS-public
    ports:
      - "7054:7054"
    environment:
      - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
      - FABRIC_CA_SERVER_TLS_CLIENTAUTH_TYPE=NoClientCert
    volumes:
      - ./config/tlsca:/etc/hyperledger/fabric-ca-server
      - ./config:/config
    command: sh -c 'fabric-ca-server start --cafiles /config/ca/fabric-ca-server-config.yaml'
```

Obrázek 6.7: Fabric CA server konfigurace univerzity A

Po vytvoření Fabric CA kontejneru se vygenerují hlavní klíče pro CA a TLSCA servery. Tyto klíče umožňují podepisovat nové certifikáty pro ostatní entity. V konfiguračním souboru pro daný server je předurčený hlavní administrátorský účet a další důležité proměnné jako hostname, csr, typ certifikátů, který chceme podporovat atd. Pokud chceme registrovat účet, tj. získat certifikát, je potřeba provést dva kroky: registraci a enrollment.

Registrace

Registrovat nové účty pod danou Certificate Authority může pouze účet s administrativními pravomocemi. Každý server nahraje takového administrátora při spuštění do své interní databáze (administrátorský účet je dán konfiguračním souborem). V mém případě jsem registroval všechny účty přes terminál. Příklad registrace účtu peer typu TLS vypadá následovně.

```
./fabric-ca-client register -u https://localhost:7054
--id.name peer --id.secret peerpw --id.type peer
--tls.certfiles tls-root-cert/tls-ca-cert.pem --mspdir
tls-ca/tlsadmin/msp
```

Kde fabric-ca-client je binární soubor, který implementuje logiku manipulace s Fabric CA serverem. Fabric CA server je tzv. double headed. Pokud nezadáme –caname flag, komunikujeme s TLSCA serverem; pro komunikaci s CA serverem je nutné použít správný –caname flag. –id.name je název účtu, –id.secret je heslo účtu, –id.type je typ účtu, –tls.certfiles jsou root TLSCA certifikáty specifické pro daný Fabric CA server a –mspdir je soubor s certifikáty administrátora. Po provedení tohoto příkazu se účet registruje do interní databáze serveru a je připraven k enrollmentu.

Enrollment

Účet existuje v databázi, pokud se registruje pomocí administrátorského účtu. K enrollmentu je potřeba znát jméno a heslo účtu. Po správném zadání se

vygenerují a stáhnout certifikáty daného účtu. Pro enrollment je nutné zadat tento příkaz.

```
./fabric-ca-client enroll -u
https://peer:peerpw@localhost:7054
--enrollment.profile tls --csr.hosts
"localhost,peer1.university-a.diis.io"
--tls.certfiles tls-root-cert/tls-ca-cert.pem --mspdir
tls-ca/examplepeer/msp
```

Kde –enrollment.profile je typ účtu, v tomto ukázkovém případě se jedná o typ TLS, který bude využitý na komunikaci s ostatními uzly kvůli bezpečnosti. –csr.hosts je množina hostnames, ze kterých bude uzel komunikovat. –mspdir má nyní trochu jiný význam — lokace uložení vygenerovaných certifikátů. Je nutné zadávat –csr.hosts pro typ TLS; pokud se jedná o typ CA (používaných pro podpis transakcí a endorsmentu) či uživatele, který není server, není nutné zadávat hostnames.

Ekvivalentně se vygenerují certifikáty pro ostatní účty. Je zjevné, že každý účet má dva podúčty (jeden TLSCA a jeden CA). Pokud je organizace typu Univerzita, je nutné vygenerovat certifikáty pro uzly typu peer a orderer. Také je nutné mít alespoň jednoho administrátora pro schvalování definicí smart contractů a jejich commitment.

Veřejná organizace

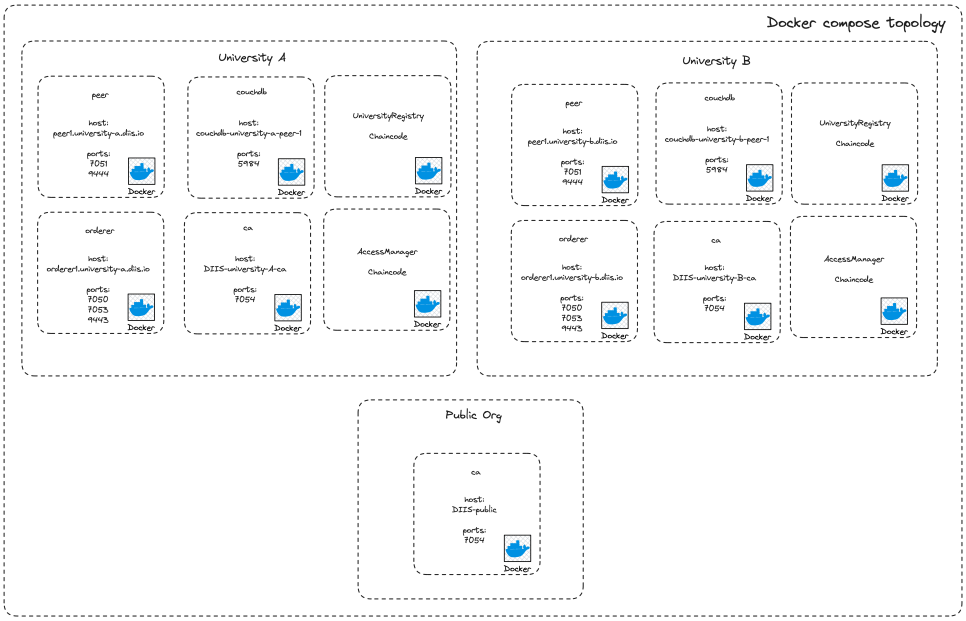
Postup je zcela ekvivalentní generovaní certifikátů u organizace typu Univerzita. Jediný rozdíl je, že není potřeba generovat certifikáty pro uzly typu peer a orderer, jelikož tento typ organizace žádné nemá.

6.2.3 Topologie sítě

Řešení se dá kategorizovat jako tzv. cluster. Jedná se o množinu spolupracujících počítačů (v mém případě uzlů a dalších pomocných komponent), které spolu komunikují. Jednotlivé komponenty budou kontejnerizovány pomocí technologie docker. V systému mám čtyři docker images:

- **Orderer** viz. kap. 5.2.1
- **Peer** viz. kap. 5.2.1
- **Fabric CA** viz. kap. 5.2.3
- **Chaincode** — Jedná se o smart contract, který běží v kontejnerizovaném prostředí. Tento chaincode kontejner je spouštěn peer kontejnerem právě tehdy, když je nutné psát do blockchainu nebo z něj číst prostřednictvím tohoto smart contractu. Po vykonání požadavku smart contractem zůstává chaincode kontejner spuštěn. Díky tomu se sníží latence při sekundárních spouštění smart contractu, není tedy nutné znovu vytvořit chaincode kontejner.

V mém případě budu tvořit cluster pomocí technologie docker compose. Tento nástroj slouží ke spouštění několika docker kontejnerů současně a řeší jejich síťové propojení. Celý cluster bude běžet na mém osobním počítači na dedikované privátní síti, kde spolu mohou jednotlivé komponenty komunikovat. Porty těchto komponent budou svázány s porty mého osobního počítače, abych mohl komunikovat s blockchainem. Topologie docker compose sítě má následující podobu.



Obrázek 6.8: Ilustrace topologie kontejnerů

Po vytvoření blockchainu je nutné zkontrolovat body níže:

- Všechny uzly se spustili;
- Vygeneroval se genesis block public channel;
- Ordering service se shodl na vůdci;
- Ordering service začal řazení blocků nad public channel;
- Všechny uzly typu peer na sebe vidí a komunikují spolu;
- Všechny uzly typu peer se připojili do public channel;

Spustil jsem skript s názvem setup.sh v adresáři blockchain/scripts/infrastructure/. Obrázky níže jsou jeho výstupem.

```
-e Creating channel config block...
2023-12-29 13:15:19.120 CET 0001 INFO [common.tools.configtxgen] main -> Loading configuration
2023-12-29 13:15:19.131 CET 0002 INFO [common.tools.configtxgen.localconfig] completeInitialization -> orderer type: etcdraft
2023-12-29 13:15:19.132 CET 0003 INFO [common.tools.configtxgen.localconfig] completeInitialization -> Orderer.EtcdRaft.Options unset, setting to tick_interval:"500ms" elec
tion_tick:10 heartbeat_tick:1 max_inflight_blocks:5 snapshot_interval_size:16777216
2023-12-29 13:15:19.132 CET 0004 INFO [common.tools.configtxgen.localconfig] Load -> Loaded configuration: /Users/jakubstrnad/school/BP/DIIS/blockchain/configtx/configtx.ya
ml
2023-12-29 13:15:19.137 CET 0005 INFO [common.tools.configtxgen] doOutputBlock -> Generating genesis block
2023-12-29 13:15:19.137 CET 0006 INFO [common.tools.configtxgen] doOutputBlock -> Creating application channel genesis block
2023-12-29 13:15:19.137 CET 0007 INFO [common.tools.configtxgen] doOutputBlock -> Writing genesis block
-e Block created
```

Obrázek 6.9: Genesis block public channel je úspěšně vytvořen

Genesis block našeho blockchainu je úspěšně vytvořen. Podíváme se, jestli ho ordering service zaregistroval.

```
-e Joining orderers...
Status: 201
{
        "name": "public",
        "url": "/participation/v1/channels/public",
        "consensusRelation": "consenter",
        "status": "active",
        "height": 1
}

Status: 201
{
        "name": "public",
        "url": "/participation/v1/channels/public",
        "consensusRelation": "consenter",
        "status": "active",
        "height": 1
}
```

Obrázek 6.10: Oba uzly typu orderer se přihlásili k řazení blocků na public channel

Celý ordering service nyní bude řadit přicházející transakce na public channel. V našem případě jsou pouze dva uzly typu order v celém ordering service.

```
-e Joining peers into public channel...
2023-12-29 12:15:52.063 UTC 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
2023-12-29 12:15:52.250 UTC 0002 INFO [channelCmd] executeJoin -> Successfully submitted proposal to join channel
2023-12-29 12:15:52.749 UTC 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
2023-12-29 12:15:52.942 UTC 0002 INFO [channelCmd] executeJoin -> Successfully submitted proposal to join channel
-e Peers joined into public channel
```

Obrázek 6.11: Oba uzly typu peer se přihlásili k public channel

Uzly typu peer se připojily na public channel. Výstup ze skriptu vypadá dobře, podívám se přímo do docker kontejnerů a ověřím správnost nasazení.

```
2023-12-29 12:15:37.409 UTC 001d INFO [orderer.consensus.etcdraft] Step -> 2 is starting a new election at term 1 channel=public node=2
2023-12-29 12:15:37.411 UTC 001e INFO [orderer.consensus.etcdraft] becomePreCandidate -> 2 became pre-candidate at term 1 channel=public node=2
2023-12-29 12:15:37.413 UTC 001f INFO [orderer.consensus.etcdraft] poll -> 2 received MsgPreVoteResp from 2 at term 1 channel=public node=2
2023-12-29 12:15:37.414 UTC 0020 INFO [orderer.consensus.etcdraft] campaign -> 2 [logterm: 1, index: 2] sent MsgPreVote request to 1 at term 1 channel=public node=2
2023-12-29 12:15:37.484 UTC 0021 INFO [orderer.consensus.etcdraft] poll -> 2 received MsgPreVoteResp from 1 at term 1 channel=public node=2
2023-12-29 12:15:37.485 UTC 0022 INFO [orderer.consensus.etcdraft] stepCandidate -> 2 [quorum:2] has received 2 MsgPreVoteResp votes and 0 vote rejections channel=public no
de=2
2023-12-29 12:15:37.485 UTC 0023 INFO [orderer.consensus.etcdraft] becomeCandidate -> 2 became candidate at term 2 channel=public node=2
2023-12-29 12:15:37.485 UTC 0024 INFO [orderer.consensus.etcdraft] poll -> 2 received MsgVoteResp from 2 at term 2 channel=public node=2
2023-12-29 12:15:37.485 UTC 0025 INFO [orderer.consensus.etcdraft] campaign -> 2 [logterm: 1, index: 2] sent MsgVote request to 1 at term 2 channel=public node=2
2023-12-29 12:15:37.488 UTC 0026 INFO [orderer.consensus.etcdraft] poll -> 2 received MsgVoteResp from 1 at term 2 channel=public node=2
2023-12-29 12:15:37.488 UTC 0027 INFO [orderer.consensus.etcdraft] stepCandidate -> 2 [quorum:2] has received 2 MsgVoteResp votes and 0 vote rejections channel=public node=
2
2023-12-29 12:15:37.489 UTC 0028 INFO [orderer.consensus.etcdraft] becomeLeader -> 2 became leader at term 2 channel=public node=2
2023-12-29 12:15:37.489 UTC 0029 INFO [orderer.consensus.etcdraft] run -> raft.node: 2 elected leader 2 at term 2 channel=public node=2
2023-12-29 12:15:37.490 UTC 002a INFO [orderer.consensus.etcdraft] run -> Raft leader changed: 0 -> 2 channel=public node=2
2023-12-29 12:15:37.492 UTC 002b INFO [orderer.consensus.etcdraft] run -> Start accepting requests as Raft leader at block [0] channel=public node=2
```

Obrázek 6.12: Leader ordering service zvolen

Vůdce ordering service byl úspěšně zvolen.

Obrázek 6.13: Uzly typu peer na sebe vidí.

Tohle je výstup z uzlu typu peer univerzity A. Uzly typu peer jsou schopny komunikace mezi sebou.

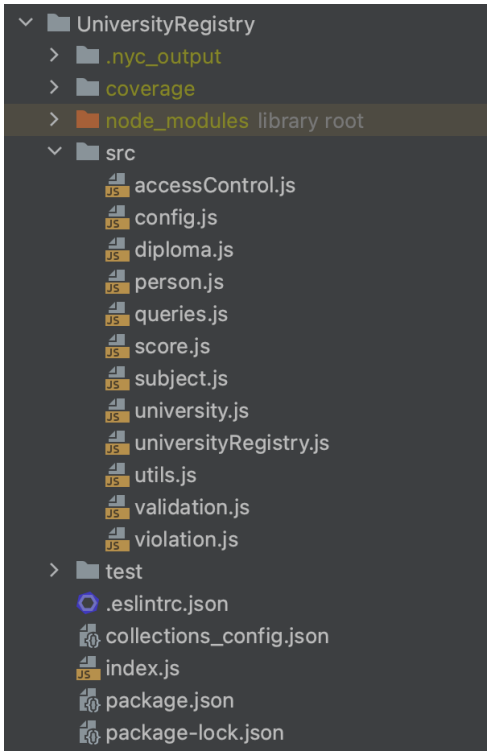
Úspěšně jsem vytvořil blockchain se čtyřmi uzly. Infrastruktura je připravena pro aplikační logiku zprostředkovanou smart contracty. Jejich implementaci popisuji v další části.

6.2.4 Chaincode

V této části detailně rozvádím implementaci smart contractů.

UniversityRegistry

Tento smart contract slouží jako hlavní rozhraní mého decentralizovaného univerzitního informačního systému. Je zodpovědný za manipulaci s akademickými daty v systému, umožňuje jejich ukládání a čtení. Smart contract vynucuje nejzákladnější pravidla pro práci s daty. Příkladem takových pravidel je, že pouze administrátor univerzity může zaregistrovat svou univerzitu do systému, pouze učitelský účet může hodnotit studentské účty v rámci jedné univerzity anebo že studentský účet může data pouze číst. Na obrázku níže je zobrazena strukturu smart contractu.



Obrázek 6.14: Adresářová struktura UniversityRegistry smart contractu

Vysvětlení souborů v *src* adresáři:

- **src** - adresář obsahuje veškerý kód potřebný ke správnému fungování smart contractu
  - **accessControl.js** — rozhraní pro komunikaci s AccessManager smart contractem;
  - **config.js** — konstanty;
  - **diploma.js** — modul, který se stará o manipulaci s diplomy;
  - **person.js** — modul, který se stará o manipulaci s osobami, tzn. registrace, čtení profilu atd.;
  - **queries.js** — modul, který se stará o parsování, paginaci a vyhledávání privátních dat; je používán pouze metodou listPersonData();
  - **score.js** — modul, který se stará o manipulaci se známkováním studentů učiteli;
  - **subject.js** — modul, který se stará o manipulaci s předměty;
  - **university.js** — modul, který se stará o registraci a čtení univerzit;
  - **universityRegistry.js** — hlavní rozhraní smart contractu;
  - **utils.js** — pomocné funkce; jedná se hlavně o generování UUID, abstrakce zapisování a čtení veřejných nebo privátních dat atd.;
  - **validation.js** — jedná se o množinu schémat, podle kterých se validují data v dotazech;
  - **violation.js** — modul, který se stará o manipulaci s přestupky;
- **test** — adresář s unit-testy pro tento smart contract;
- **collections_config.json** — soubor s konfigurací privátních kolekcí tohoto smart contractu. Obsahuje pouze dvě Private Data Collections. Každá kolekce je určena pro jednu univerzitu. V tomto systému má každá univerzita právě jeden uzel typu peer. Proto je počet uzlů, na které se mají distribuovat privátní data, roven nule (nepočítá se peer uzel, který zpracovává transakci, tzn. privátní data budou vždy uložena právě na jednom autorizovaném uzlu typu peer);
- **index.js** — hlavní soubor, který importuje UniversityRegistry smart contract. Tento soubor využívá uzel typu peer, když inicializuje kontejner;

Ostatní soubory nejsou pro tento případ důležité.

Tímto jsem popsal adresářovou strukturu UniversityRegistry smart contractu. Nyní popíši jeho rozhraní, které je možné volat aplikací (klientem). UniversityRegistry smart contract implementuje metody těchto tří typů:

- Metody, které mohou používat pouze administrátorské účty univerzit. Příklad takové metody je metoda registerUniversity().

- Metody, které mohou používat pouze účty univerzit. Příklad takové metody je metoda getMyProfile(). Dále je možné dělit metody podle toho, jestli je může používat jen univerzitní účet s typem učitel. To už je pouze detail.
- Metody, které může volat kdokoliv, kdo má účet u jakékoliv organizace. Tyto metody jsou chráněny smart contractem AccessManager. Příklad takové metody je listPersonData().

Rozhraní, které dává smysl využívat pouze v rámci samotné univerzity. Parametry metod jsem nechal prázdné, pro větší detaily navštivte dokumentaci přímo v kódu v příloze této bakalářské práce.

- **initialize()** — Funkce pro inicializaci smart contractu, je nutné uložit jméno AccessManager smart contractu pro správné fungování. Pokud tato metoda nebyla zavolána, není možné používat smart contract. Je možné ji volat pouze jednou.
- **checkInitialized()** — Funkce, která kontroluje inicializaci smart contractu. Je nutné ji uvést jako první v jakékoliv funkci pro maximální bezpečnost.
- **registerUniversity()** — Funkce, kterou může volat pouze administrátor univerzity. Slouží k registraci univerzity jako datové entity to blockchainu.
- **getCollectionName()** — Funkce pro zjištění názvu Private Data Collection vlastní univerzity.
- **getMyUniversity()** — Funkce pro zjištění profilu univerzity, do které účet patří.
- **registerPerson()** — Funkce pro registraci účtů jako datových entit do blockchainu. Jsou podporovány pouze dva typy těchto účtů tj. TEACHER a STUDENT. Funkci může volat pouze administrátor univerzity a účty, které zaregistruje, budou automaticky spadat do jeho univerzity.
- **getMyProfile()** — Funkce, která slouží ke čtení vlastního profilu (osobní data).
- **addSubject()** — Funkce, která slouží k přidání nového předmětu do systému. Tuto funkci může volat pouze ten účet, který je registrovaný v blockchainu a má typ TEACHER. Předmět, který tento účet vytvoří, automaticky spadá pod jeho univerzitu.
- **getSubject()** — Funkce, která slouží pro čtení určitého předmětu.
- **setScore()** — Funkce, která slouží ke známkování studenta z určitého předmětu. Funkci může volat pouze účet s registrovaným účtem na blockchainu typu TEACHER.
- **getScore()** — Funkce, která slouží pro čtení určitého známkování.

- **addDiploma()** — Slouží k vystavení diplomu studentovi. Je možné volat pouze s učitelským účtem.
- **getDiploma()** — Slouží ke čtení určitého diplomu.
- **addViolation()** — Slouží k přidání nového porušení školního řádu. Je možné volat pouze s učitelským účtem.
- **getViolation()** — Slouží ke čtení určitého přestupku.

Rozhraní, která mohou využívat externí účty mimo univerzitu.

- **getPersonData()** — Funkce, která slouží pro čtení určitého zdroje dat. Tato metoda je chráněna AccessManager smart contractem. Pokud má volající povolení číst data, vrátí se mu smysluplná odpověď. Pokud povolení nemá, vrátí se chybová hláška.
- **listPersonData()** — Funkce, která slouží pro čtení většího počtu dat jednoho zdroje. Není nutné znát žádný identifikátor, pouze personId studenta, na kterého se volající doptává. Tato metoda je též chráněna AccessManager smart contractem.

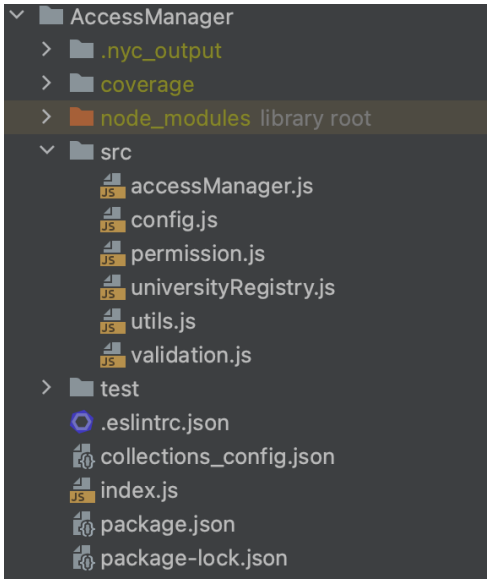
Ukázka implementace metody listPersonData() na úrovni rozhraní UniversityRegistry smart contractu.

```
async listPersonData(ctx, query) : Promise<{bookmark: string, results: ...}> {
    await this.checkInitialized(ctx);
    const jsonQuery = JSON.parse(query);
    validation.validateListQuery(jsonQuery);

    const permAttrs : (...)[] = [config.LIST, jsonQuery.queryParams.type, JSON.stringify(jsonQuery.fields), jsonQuery.query
    const hasPermissions : boolean | undefined = await accessControl.hasInvokerReadPermissionsForResource(ctx, permAttrs);
    if (!hasPermissions) throw new Error("You do not have a permission to read this data.");

    const publicPerson = await person.getPublicPersonProfileById(ctx, jsonQuery.queryParams.recipient);
    const uni = await university.get(ctx, publicPerson.universityMSPID);

    const [queryString : string , pageSize : any | number , bookmark : string | any ] = queries.parseRequestQueryString(jsonQuery);

    const promiseOfIterator : Promise<...> & AsyncIterable<iterators.KV> = ctx.stub.getQueryResultWithPagination(queryString, pa

    let results : any[] = await queries.getListResult(
        ctx,
        promiseOfIterator,
        uni.privateCollectionName,
        config.getPrefixFromObjectType(jsonQuery.queryParams.type),
        jsonQuery.fields
    );

    const metadata : QueryResponseMetadata = (await promiseOfIterator).metadata;

    return {
        results: results,
        bookmark: metadata.bookmark,
    }
}
```

Obrázek 6.15: Implementace listPersonData() metody

AccessManager

Tento smart contract je zodpovědný za management povolení číst určité zdroje dat. Do datového prostoru tohoto smart contractu zapisují pouze účty typu STUDENT a dávají tím povolení externím účtům mimo svou univerzitu číst vlastní data. Tento smart contract je hlavně volán UniversityRegistry smart contractem, který zjišťuje zda-li má volající přístup k chtěným datům. Nejprve popíšu adresářovou strukturu.



Obrázek 6.16: Adresářová struktura AccessManager smart contractu

Vysvětlení:

- **src** — stejné jak u UniversityRegistry smart contractu;
  - **accessManager.js** — hlavní rozhraní smart contractu;
  - **config.js** — konstanty;
  - **permission.js** — modul, který se stará o management povolení;
  - **universityRegistry.js** — rozhraní pro komunikaci s UniversityRegistry smart contractem;
  - **utils.js** — pomocné funkce;
  - **validation.js** — validace dat dotazů;
- **test** — adresář s unit-testy pro tento smart contract;
- **collections_config.json** — prázdný, tento smart contract privátní data nemá;
- **index.js** — stejně jako u UniversityRegistry smart contractu;

Tento smart contract implementuje pouze tyto metody.

- **initialize()** — To stejné jako u UniversityRegistry smart contractu.
- **checkInitialized()** — To stejné jako u UniversityRegistry smart contractu.
- **grantReadPermissionToResource()** — Funkce, kterou může volat pouze účet, který má registrovaný účet v blockchainu prostřednictvím UniversityRegistry smart contractu. Volající může touto funkcí povolit externímu účtu (nebo i účtu uvnitř své univerzity) povolení pro čtení určitého zdroje vlastních dat.
- **hasInvokerReadPermissionsForResource()** — Funkce, která slouží k ověření dotazu na zdroj dat.

Ukázka implementace metody hasInvokerReadPermissionsForResource() na úrovni rozhraní AccessManager smart contractu.

```
async hasInvokerReadPermissionsForResource(ctx, method, resource, fields, personId) : Promise<boolean>  {
    await this.checkInitialized(ctx);
    const parsedFields : any  = fields === "*" ? fields : JSON.parse(fields);
    validation.validateHasPermissionRequest(method, resource, parsedFields, personId);

    let perm;
    try {
        perm = await permission.get(ctx,  attrs: [personId, ctx.clientIdentity.getID(), resource, method]);
    } catch (error) {
        if (
            error instanceof permission.PermissionExpired ||
            error instanceof permission.PermissionDoesNotExist
        ) {
            return false;
        }
    }

    if (Array.isArray(parsedFields)) {
        for (let key of parsedFields) {
            if (!perm.fields.includes(key)) {
                return false;
            }
        }
    } else if (parsedFields !== "*") {
        return false;
    } else if (parsedFields !== perm.fields) {
        return false;
    }

    return true;
}
```

Obrázek 6.17: Implementace hasInvokerReadPermissionsForResource() metody

Jelikož je povolení velmi důležitá datová entita v tomto systému, převedu její strukturu uloženou v blockchainu obrázkem níže.

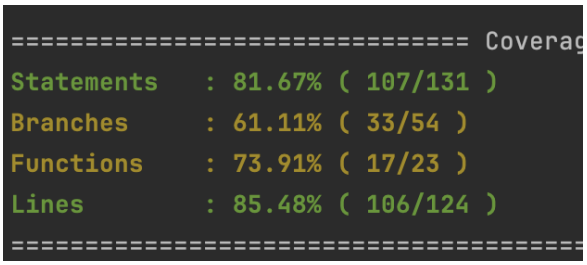
```
{
  "recipientNetworkId": "x509::/OU=client/CN=employer::/CN=diis-public-ca-server",
  "issuer": "43040196-0196-440196699a0-a40196699a0a1f-0a1fd71",
  "resource": "VIOLATION",
  "fields": ["reason", "issuer"],
  "ttl": 600,
  "method": "list",
  "createdAt": "2023-12-26T10:17:08.000Z"
}
```

Obrázek 6.18: Struktura datové entity povolení v blockchainové databázi

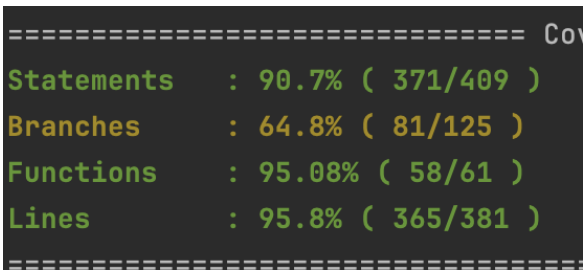
Kde recipientNetworkId je identifikátor certifikátu uživatele, kterému se povolení dává. Atribut issuer je identifikátor účtu uživatele uloženého v blockchainu, který povolení vydal. Atribut resource je typ dat, který se může číst. Pole fields je množina polí, které se můžou z dat číst. Atribut ttl je časový interval v sekundách, po jeho uplynutí už není možné číst data. Atribut method je metoda čtení dat. Atribut createdAt je čas vzniku povolení.

Testování

Oba smart contracty obsahují adresář *test* s unit-testy k danému smart contractu. Unit-testy jsou implementovány pomocí frameworku chai[1], sinon[2] a fabric-shim[3]. Testy simulují transakce na blockchainu bez spouštění lokální sítě. Na obrázcích níže je uvedeno pokrytí smart contractů pomocí unit-testů.



Obrázek 6.19: Výstup z modulu coverage pro UniversityRegistry smart contract s celkovým počtem 24 unit-testů.



Obrázek 6.20: Výstup z modulu coverage pro AccessManager smart contract s celkovým počtem 10 unit-testů.

[1] https://www.chaijs.com/
[2] https://sinonjs.org/
[3] https://www.npmjs.com/package/fabric-shim

```
it( title: "Should get subject", fn: async() : Promise<void> => {
    sinon.stub(universityRegistry, 'checkInitialized').resolves(true);
    mockClientIdentity.getMSPID.returns("UniversityAMSP");
    mockStub.createCompositeKey.withArgs("university", ["UniversityAMSP"]).returns("universityUniversityAMSP");
    mockStub.createCompositeKey.withArgs("subject", ["UniversityAMSP", "B4B36SIN"]).returns("subjectUniversityAMSPB4B36SIN");
    mockStub.getState.withArgs("subjectUniversityAMSPB4B36SIN").returns(Buffer.from(JSON.stringify(subjectProfile)));
    mockStub.getState.withArgs("universityUniversityAMSP").returns(Buffer.from(JSON.stringify(universityProfile)));

    const response = await universityRegistry.getSubject(ctx, universityMSPID: "UniversityAMSP", subjectCode: "B4B36SIN");
    expect(response).to.be.deep.equal(subjectProfile);
});
```

Obrázek 6.21: Příklad jednoho z unit-testů kontrolující vrácení objektu subject.

Nasazení smart contractů

Všechny smart contracty jsem popsal a vysvětlil. Nyní je potřeba všechny do systému nasadit a začít využívat. K nasazení smart contractů využiji skripty, které se nachází v adresáři blockchain/scripts/deployment/. Nejprve musím začít instalací smart contractů na uzly typu peer všech univerzitních organizací. Využiji skript install-chaincode.sh. Pro nasazení smart contractu UniversityRegistry stačí přepsat proměnnou CHAINCODE_LABEL na UniversityRegistry a spustit script. Výstup by měl vypadat následovně.

```
jakubstrnad@jakubs-MacBook-Pro deployment % sh install-chaincode.sh
2023-12-30 10:06:10.169 CET 0001 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Installed remotely: response:<status:200 payload:"\nSUniversityRegistry:8d9bc9e3d73
d4d2d54288474eb2227b17f255936 8e231ec94a67fd0110fadd9a\022\022UniversityRegistry" >
2023-12-30 10:06:10.170 CET 0002 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Chaincode code package identifier: UniversityRegistry:8d9bc9e3d73d4d2d54288474eb222
7b17f255936 8e231ec94a67fd0110fadd9a
2023-12-30 10:06:32.175 CET 0001 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Installed remotely: response:<status:200 payload:"\nSUniversityRegistry:8d9bc9e3d73
d4d2d54288474eb2227b17f255936 8e231ec94a67fd0110fadd9a\022\022UniversityRegistry" >
2023-12-30 10:06:32.176 CET 0002 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Chaincode code package identifier: UniversityRegistry:8d9bc9e3d73d4d2d54288474eb222
7b17f255936 8e231ec94a67fd0110fadd9a
```

Obrázek 6.22: Výstup z install-chaincode.sh skriptu pro UniversityRegistry smart contract

UniversityRegistry smart contract je nasazen, nyní je třeba, aby každý administrátor všech organizací souhlasil s jeho definicí. Pokud tato situace nastane, stačí, aby kterýkoliv administrátor provedl commit operaci na public channel. Pro tyto operace použiji skript commit-chaincode.sh, který udělí souhlas za všechny administrátory a provede commit. Stačí nastavit CHAINCODE_NAME na UniversityRegistry a CHAINCODE_PACKAGE_ID na ID nainstalovaného smart contractu, které se získá z výstupu skriptu install-chaincode.sh. Po spuštění skriptu je očekáván následující výstup.

```
jakubstrnad@jakubs-MacBook-Pro deployment % sh commit-chaincode.sh
2023-12-30 10:12:49.936 CET 0001 INFO [chaincodeCmd] ClientWait -> txid [6f17d29e6bc676de53fb3b0fc464c711fadab0801d42ac221e07d257eac4e9da] committed with status (VALID) at
localhost:7051
2023-12-30 10:12:52.158 CET 0001 INFO [chaincodeCmd] ClientWait -> txid [92ca58d71e9a912581f9548494c445ddc3db344e98089ead34eead1ae177686] committed with status (VALID) at
localhost:7151
2023-12-30 10:12:54.475 CET 0002 INFO [chaincodeCmd] ClientWait -> txid [5546c3a6942379a1887b7b7469ce77074e3e95e9e4e3402142e2684ca8eb36c8] committed with status (VALID) at
localhost:7151
2023-12-30 10:12:54.475 CET 0001 INFO [chaincodeCmd] ClientWait -> txid [5546c3a6942379a1887b7b7469ce77074e3e95e9e4e3402142e2684ca8eb36c8] committed with status (VALID) at
localhost:7851
```

Obrázek 6.23: Výstup z commit-chaincode.sh skriptu pro UniversityRegistry smart contract

Pro AccessManager smart contract provedu stejné operace. Po dokončení instalací a commit operací pro všechny smart contracty je potřeba provést

finální inicializaci. Pro tuto operaci využiji skript initialize-chaincodes.sh. Po úspěšném spuštění se očekává takovýto výstup.

```
jakubstrnad@jakubs-MacBook-Pro deployment % sh initialize-chaincodes.sh
2023-12-30 10:17:20.510 CET 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200
2023-12-30 10:17:20.806 CET 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200
```

Obrázek 6.24: Inicializace smart contractů

Smart contracty jsou připraveny k používání.

6.3 Výsledky

V této části demonstruji několik možných využití mého systému. Dále provedu rozbor jeho nedostatků a nastíním jakým směrem by se měla aplikace vyvíjet v budoucnu.

6.3.1 Průchod systémem

Celý systém je připraven k používání, infrastruktura běží a smart contracty jsou nasazeny. V systému však nejsou zatím žádná data. Nejprve zaregistruju všechny univerzity pomocí skriptu addUniversity.js v clients adresáři. Dále je potřeba zapsat alespoň jeden předmět. Toho dosáhnu pomocí scriptu addSubject.js. Dále zaregistrujeme učitele a studenta pomocí skriptu register.js v enrollment adresáři. Výstup z registrace uživatelských účtů by měl vypadat takto.

```
jakubstrnad@jakubs-MacBook-Pro clients % node enrollment/register.js
Built an in memory wallet
ef797c81-7c81-497c8118f02-a97c8118f02dfb-2dfb649
Built an in memory wallet
96b552a9-52a9-4552a9142b1-a552a9142b1fb1-1fb12a1
done
```

Obrázek 6.25: Výstup z register.js skriptu

První UUID je ID studenta, druhé je ID učitele. Řekněme, že se student provinil z určitého přestupku vůči školnímu řádu univerzity. Učitel mu tento přestupek zapíše do blockchainu pomocí skriptu addViolation.js v teacher adresáři. Výstup by měl být následující.

```
jakubstrnad@jakubs-MacBook-Pro clients % node teacher/addViolation.js
Built an in memory wallet
160e4400-4400-4e44007c438-ae44007c438198-8198324
```

Obrázek 6.26: Výstup z addViolation.js

Přestupku bylo přiřazeno UUID. Řekněme, že po nějaké době student uchází o pracovní pozici v určité firmě. Zaměstnavatel této firmy chce zjistit,

zda-li se student dopustil nějakých přestupků vůči školnímu řádu během jeho studia. Dotážu se jako zaměstnavatel na blockchain se zájmem zjistit zmíněné přestupky. Použiji na to skript getViolations.js v employer adresáři.

```
Error: You do not have a permission to read this data.
    at SingleQueryHandler.evaluate (/Users/jakubstrnad/school/BP/DIIS/clients/node_modules/fabric-network/lib/impl/query/singlequeryhandler.js:64:57)
    at processTicksAndRejections (node:internal/process/task_queues:96:5)
    at async Transaction.evaluate (/Users/jakubstrnad/school/BP/DIIS/clients/node_modules/fabric-network/lib/transaction.js:322:25)
    at async listScores (/Users/jakubstrnad/school/BP/DIIS/clients/employer/getViolations.js:31:17) {
  isEndorsed: false,
  payload: Buffer(0) [Uint8Array] [],
  status: 500
```

Obrázek 6.27: Chybová hláška z getViolations.js skriptu

Bohužel není možné číst dané data, nemám jako zaměstnavatel žádné povolení. Dám tedy studentovi na výběr, buď mi dá povolení číst data ze systému nebo bude vyřazen z výběrového řízení kvůli utajování důležitých informací. Student dá povolení zaměstnavateli pomocí skriptu grantPermission.js v student adresáři. Dá mu však pouze povolení na pole reason. Jakmile je povolení zapsáno do blockchainu, může zaměstnavatel znovu zkusit získat data.

```
jakubstrnad@jakubs-MacBook-Pro clients % node employer/getViolations.js
Built an in memory wallet
{
  results: [ { reason: 'Plagiat' } ],
  bookmark: 'g1AAAACoeJzLYWBgYMpgSmHgKy5JLCrJTq2MT8lPzkzJBYrbMJRl5ucklmTm5zEYmhmkmpgYGOhCCDDbPNnE2EI3EcE2tLTQBRHGRiYMIDM5YGZSbFoWAF6DKkM'
}
```

Obrázek 6.28: Výstup z getViolations.js skriptu

Čtení je úspěšné. Zaměstnavatel chce však zkusit vytěžit dat více, než mu bylo povoleno studentem. Zkusí se tedy dotázat i na pole issuer.

```
Error: You do not have a permission to read this data.
    at SingleQueryHandler.evaluate (/Users/jakubstrnad/school/BP/DIIS/clients/node_modules/fabric-network/lib/impl/query/singlequeryhandler.js:64:57)
    at processTicksAndRejections (node:internal/process/task_queues:96:5)
    at async Transaction.evaluate (/Users/jakubstrnad/school/BP/DIIS/clients/node_modules/fabric-network/lib/transaction.js:322:25)
    at async listScores (/Users/jakubstrnad/school/BP/DIIS/clients/employer/getViolations.js:31:17) {
  isEndorsed: false,
  payload: Buffer(0) [Uint8Array] [],
  status: 500
```

Obrázek 6.29: Chybová hláška z getViolations.js skriptu

Tohle bohužel nejde, jelikož dotaz nesedí se studentovým povolením.

6.3.2 Splnění kritérií meziuniverzitního informačního systému

V bodech připomenu kritéria definovaná v kapitole 3.1. a podívám se, jestli jsem daná kritéria splnil.

- **Interní využití systému uvnitř univerzity** — Ano, je možné provádět administrativní operace nad akademickými daty uvnitř jednotlivých univerzit.
- **Externí využití systému mimo univerzitní prostředí** — Ano, pokud je povoleno externímu subjektu data číst.
- **Jednotný datový model** — Ano, public channel vynucuje následování datového modelu.

- **Škálovatelnost** — Ano, každá univerzita musí mít alespoň jeden uzel typu peer, prakticky bych jich měla mít více kvůli zálohování dat. Dá se říct, že každá univerzita si řeší škálování interně nezávisle na jiných univerzitách. To znamená, pokud by byl problém s velkým počtem požadavků může si univerzita inicializovat další uzel typu peer, který pomůže zpracovávat požadavky. Jediný problém může nastat u Private Data Collections, jelikož se musí zvažovat při commitmentu smart contractů. Čím více je univerzit v systému, tím více je privátních kolekcí a management je náročnější. I u velkého počtu univerzit to není kritické, ale zvážil bych jiný způsob ukládání privátních dat. Bohužel nemám možnost to vyzkoušet.

- **Bezpečnost** — Ano, univerzitní účty mohou komunikovat jen s uzly typu peer své univerzity. Uvnitř univerzity jsou nadefinovány role, které dále zajišťují bezpečnost dat. Externí subjekty nemohou číst data bez povolení.

- **Transparentnost** — Ano, všechny účty určité univerzity mohou vidět všechny veřejná data té univerzity z historie blockchainu. Privátní data nevidí v historii nikdo, veřejný je pouhý hash privátních dat. Reálná privátní data existují pouze na autorizovaném uzlu typu peer.

6.3.3 Nedostatky

Můj systém slouží jako "proof of concept"decentralizovaného meziuniverzitního informačního systému, to znamená, že pár důležitých věcí jsem v implementaci přehlížel a celkově jsem systém nekoncipoval jako systém do produkčního prostředí. V následujících bodech jsem sepsal nedostatky, o kterých vím, že se v systému vyskytují.

- **Nepoužívá se salt při tvorbě UUID** — Tohle je pouze problém u vytváření UUID pro osoby (studenty a učitele). UUID se tvoří pomocí funkce SHA256 nad identifikačním číslem osoby (číslo občanského průkazu), jelikož se délka čísla občanského čísla pohybuje v 9 cifrách, není těžké odhadnout podle výsledného UUID, jaké číslo bylo použito. Toto představuje riziko úniku informací ze systému, jelikož je výsledný UUID veřejný a je sdílen napříč uzly typu peer všemi univerzitami.

- **Datové entity** — Modelování akademických dat v systému je velmi stručné, neobstálo by v produkčním prostředí. Mé modelování však stačí pro ukázku funkčnosti systému.

- **Registrace univerzit** — V systému jsou na pevno dány dvě univerzitní organizace. Hyperledger Fabric technologie podporuje přidávání organizací do existujícího channel, bohužel jsem k tomu nevytvořil žádný skript — mimo rozsah této práce.

6.3.4 Další možný vývoj systému

Pár bodů, které mě napadají v souvislosti s vylepšením celkového systému.

- **Uživatelské rozhraní** — Systém by si zasloužil uživatelské rozhraní ve formě webové či mobilní aplikace. V aplikaci bych si představil jednoduchý způsob připojení uživatelského certifikátu a jednoduchou logiku přidělování povolení pro čtení dat, přidávání předmětů atd. Stojí za zmínku, že v tomto typu systému není potřeba heslo, stačí pouze privátní klíč, který by byl uložený lokálně.

Kapitola 7

Závěr

Tato bakalářská práce si kladla za cíl provést rešerši jednotlivých informačních systémů používaných v komunikaci mezi univerzitami, jako např. ISIC. Měla prozkoumat možnosti decentralizace takových systémů pomocí technologie Hyperledger Fabric a zhodnotit benefity s tím spojené. Primárně však měla vytvořit jednotný datový model akademických dat použitelný napříč všemi univerzitami a navrhnout smart contracty, které budou zajišťovat logiku se zmíněnými akademickými daty. Měla navrhnout konfiguraci blockchainu a definovat pravidla pro všechny aktéry uvnitř tohoto ekosystému. Nakonec měla definovat kritéria takového systému a ověřit, jestli výsledný systém těmto kritériím vyhoví.

Zadání práce jsem rozšířil o interní univerzitní informační systémy, jelikož jsou nedílnou součástí akademických dat a v podstatě i jejich jediní autoři. Decentralizaci systému pro sdílení akademických dat, jako již zmíněný ISIC, nemá podle mého názoru smysl řešit bez interního systému pro administraci dat jednotlivých univerzit. Externí a interní systémy nahlíží na akademická data ze dvou různých úhlů. Tato práce se tedy pokusila o decentralizaci meziuniverzitního informačního systému, který je hodnotný jak směrem dovnitř, tak směrem ven. Dále má tato práce přesah do využitelnosti akademických dat. Ukazuje, že lze standardizovaně vyhovět požadavkům externích subjektů, kteří se snaží využít tato data inovativně. Má demo verze systému ukazuje, že je možné stavět komplexní logiku nad akademickými daty mimo univerzitní prostředí a tím zlepšit služby studentům, lektorům a ostatnímu akademickému personálu.

Bakalářská práce tak zadání nejen splnila, nýbrž i mírně rozšířila. Provedla rešerši aktuálních informačních systémů a nahlížela na ně optikou dnešní doby. Ukázala, že se systémy dají sloučit do jednoho systému, který řeší jak interní komunikaci uvnitř univerzity, tak externí komunikaci z univerzity. Provedla rešerši blockchainových řešení a z analýzy vybrala technologii Hyperledger Fabric, čímž potvrdila doporučení ze zadání. Navrhla jednotný datový model napříč univerzitami a definovala pravidla zapisování, čtení a distribuce dat mezi univerzitami a externími subjekty. Popsala datové entity a několik možných případů užití, řadu z nich detailně. V neposlední řadě byla vytvořena

demo verze decentralizovaného meziuniverzitního informačního systému a provedena ukázka průchodu takovýmto systémem na případu užití získání všech přestupků zaměstnavatelem. Nakonec jsem zanalyzoval, zda můj systém vyhověl kritériím, které jsem si stanovil.

Výstup této bakalářské práce může sloužit jako vhled do budoucnosti práce s akademickými daty. Praktické využití blockchainových technologií pozorujeme teprve několik let a existuje pouze pár systémů, které využívají blockchain smysluplně. Decentralizace meziuniverzitního informačního systému v každém případě smysluplná je a přináší nemalý počet benefitů.

Literatura

[1] Blackstone JR. and John H., *APICS Dictionary, 17th Edition.*, (2022), 93.

[2] Encyclopædia Britannica, Inc. *information system*, (2023). [Online]. Available: https://www.britannica.com/topic/information-system/Computer-software

[3] Michal Roch, *Jednotná platforma pro studium I. - FELSight*, (2016). [Online]. Available: http://blog.czm-cvut.cz/2016/02/04/jednotna-platforma-i-felsight/

[4] GTS ALIVE, s. r. o., *ISIC Check API Integration Manual*, (2022). [Online]. Available: https://data.aliveplatform.com/isic-check/isic-check-api-integration-manual.pdf

[5] Gabriel O. Rodriguez Cruz, *What Is Blockchain?*, (2022). [Online]. Available: https://money.com/what-is-blockchain/

[6] Amitai Porat, Avneesh Pratap, Parth Shah, and Vinit Adkar, *Blockchain Consensus: An analysis of Proof-of-Work and its applications.*, (2017), 93.

[7] Satoshi Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, www.bitcoin.org, October 2008.

[8] Vitalik Buterin, *Ethereum Whitepaper*, (2014). [Online]. Available: https://ethereum.org/en/whitepaper/

[9] Dongyan Huang , Xiaoli Ma and Shengli Zhang *Performance Analysis of the Raft Consensus Algorithm for Private Blockchains*, IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS: SYSTEMS, VOL. 50, NO. 1, January 2020, 173

[10] Md. Mainul Islam, Mpyana Mwamba Merlec, Hoh Peter In *A Comparative Analysis of Proof-of-Authority Consensus Algorithms: Aura vs Clique*, 2022 IEEE International Conference on Services Computing (SCC), 329

[11] Caixiang Fan, Changyuan Lin, Hamzeh Khazaei, Petr Musilek *Performance Analysis of Hyperledger Besu in Private Blockchain*, 2022 4th IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)

[12] Rohit Gupta, *Polygon Byzantine Fault Tolerance (PolyBFT)*, October 2023. [Online]. Available: https://wiki.polygon.technology/docs/edge/design/consensus/polybft/polybft-overview/

[13] Hyperledger Foundation, *Hyperledger Fabric*. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.5/whatis.html

[14] Hyperledger Foundation, *Transaction flow*. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/latest/txflow.html

[15] Hyperledger Foundation, *Private data*. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/latest/private-data/private-data.html

Příloha A

Slovník pojmů

Databáze — datová struktura, která umožňuje ukládat a číst data

EVM — Ethereum Virtual Machine

ČVUT — České vysoké učení technické

FEL — Fakulta elektrotechnická

UUID — Univerzální unikátní identifikátor

NoSQL — Databázový koncept používající jiné prostředky než tabulkové relační schémata

Mempool — Memory pool, jedná se o řadu transakcí čekající na validaci či vytěžení