Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Koncept Data Lakehouse pro zpracování medicínských dat

Plzeň 2023 Bc. Lukáš Moučka

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Lukáš MOUČKA**
Osobní číslo:	**A20N0053P**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Informační systémy**
Téma práce:	**Koncept Data Lakehouse pro zpracování medicínských dat**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se systémy, které jsou koncipovány jako datový sklad (Data Lakehouse) nad datovým úložištěm heterogenních dat v různých formátech (Data Lake).
2. Seznamte se se stávajícím úložištěm medicínských dat MRE a analyzujte možnosti koncepce Data Lakehouse nad tímto úložištěm.
3. Navrhněte prototyp Data Lakehouse nad úložištěm medicínských dat ve vybraném systému.
4. Ověřte základní funkcionalitu prototypu.
5. Proveďte vyhodnocení konceptu Data Lakehouse pro oblast medicínských dat.

Rozsah diplomové práce: **doporuč. 50 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

dodá vedoucí diplomové práce

Vedoucí diplomové práce: **Doc. Dr. Ing. Jana Klečková**
Katedra informatiky a výpočetní techniky

Datum zadání diplomové práce: **9. září 2022**
Termín odevzdání diplomové práce: **18. května 2023**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 11. října 2022

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 18. května 2023

Bc. Lukáš Moučka

Abstract

The master thesis deals with a new type of *Data Lakehouse*, which is designed for heterogeneous data storage. In the theoretical part, it deals with the evolution of different types of storage and describes them in detail. The main goal is to store medical records in this type of storage and then work with them. This is currently solved by the *MRE platform*, which is developed and operated at the FAV ZČU. To verify the suitability of the storage for the medical data area, an application was developed that implements the *Data Lakehouse* using the open-source *Delta Lake* project. The first part of the application consists of the *administration*, which provides complete user management and contains sections to ensure the management of the repository. The second part consists of the *storage* and provides all the functionality for its management through a REST API. The final testing of the application shows the suitability of using the storage for medical data. Based on the testing, suggestions for possible improvements and extensions to existing work are also described.

Abstrakt

Diplomová práce se zabývá problematikou nového typu úložiště *Data Lakehouse*, které je koncipováno pro ukládání heterogenních dat. V teoretické části je zmíněna evoluce jednotlivých typů úložišť a zevrubně je popisuje. Hlavním cílem je ukládat medicínské záznamy do tohoto typu úložiště a následně s nimi pracovat. Toto aktuálně řeší *MRE platforma*, která je vyvíjena a provozována na FAV ZČU. Pro ověření vhodnosti úložiště pro oblast medicínských dat byla vyvinuta aplikace, která implementuje úložiště *Data Lakehouse* za pomocí open-source projektu *Delta Lake*. První část aplikace tvoří *administrace*, která poskytuje kompletní správu uživatelů a obsahuje sekce zajišťující správu úložiště. Druhou část tvoří úložiště a skrze REST API poskytuje veškeré funkce pro jeho správu. Výsledné testování aplikace ukazuje vhodnost použití úložiště pro medicínská data. Na základě testování jsou popsány i náměty pro případná zlepšení a rozšíření stávající práce.

Poděkování

Tímto bych rád poděkoval vedoucí mé diplomové práce, paní *Doc. Dr. Ing. Janě Klečkové* za odborné a důkladné vedení, poskytnuté cenné rady, strávený čas a pomoc při řešení mé diplomové práce. Dále bych rád poděkoval panu *Ing. Petru Včelákovi, Ph.D.* za poskytnuté informace ohledně *MRE platformy* a rad v oblasti medicínských dat.

Obsah

1 Úvod

Databáze jsou tu s námi již od doby, kdy počítače první generace (např. ENIAC a MANIAC) používaly pro I/O operace děrné štítky . Obecně je pojem databáze definován jako systém souborů s pevnou strukturou záznamů. S databází je vždy spojen aplikační software a většinou je součástí samotné databáze. Jedná se o SŘBD (*Systém řízení báze dat*), který umožňuje prohledávání databáze, přístup a modifikaci jednotlivých záznamů. Vývoj databází šel paralelně s vývojem počítačů, ale nutnost existence databází tu byla ještě před vznikem prvního počítače. Za takového předchůdce databází je považovaná papírová kartotéka, která ukládá a uspořádává papírové dokumenty s danou strukturou do fyzického média (např. skříň, krabice, trezor atd.). Každý z dokumentů obsahuje některé povinné údaje, podle kterých jsou záznamy řazeny, aby následně mohly být efektivně vyhledávány.

Velkým iniciátorem ve vývoji databází byly od počátku Státní úřady Spojených států. V roce 1890 se ve Spojených státech uskutečnilo jedenácté sčítání obyvatel a americko-německý statistik a vynálezce *Herman Hollerith* byl pověřen vývojem automatu pro zpracování velkého množství dat o obyvatelstvu. Navrhl tedy automat pro zpracování děrných štítků, který následně dokázal sečíst všech 62 622 250 děrných štítků (přepsané sčítací formuláře do děrných štítků) již za šest týdnů. Revolučním byl *Hollerithův patent*, který informaci o každém obyvateli ukládal jako jedno číslo [14]. V roce 1911 se tento vědec spojil s další firmou a tato fúze je dnes známá jako IBM (*International Business Machines*) [31].

Následně vznikaly další systémy pracující s magnetickými páskami a následně s magnetickými disky. Z dnešního hlediska je však významný až rok 1970, kdy britsko-americký matematik a informatik *Tedd Codd* publikoval článek *A Relational Model of Data for Large Shared Data Banks*, který položil základy *relačních databází*. V následujících deseti letech probíhal vývoj databází postavený na těchto základech a v roce 1980 přinesla na trh firma *Oracle* první SQL (*Structured Query Language*) databázi inspirovanou výzkumem *Berkeleyská univerzity* a IBM. V 90. letech 20. století se začínaly objevovat první *objektově-orientované databáze*, které měly nahradit ty *relační*. Ty však nenaplnily původní očekávání a jako kompromis vznikly *objektově-relační databáze* [43].

Se vzrůstajícím množstvím strukturovaných dat a nutností tato data analyzovat, začal vznikat nový typ databáze postavený nad *relačním modelem databází*, který je známý jako *datový sklad*. Množství zpracovávaných

dat stále roste, v posledních letech dochází k potřebě ukládat a analyzovat heterogenní data, a proto v roce 2019 zveřejnila společnost *Databricks* úložiště *Data Lakehouse*, které umí pracovat se strukturovanými, semistrukturovanými a nestrukturovanými daty.

Tato diplomová práce se zabývá zpracováním medicínských dat, která jsou typickým zástupcem heterogenních dat (např. informace o pacientovi, snímky z EKG (*Elektrokardiografie*), CT (*Computed Tomography*), ultrazvuku atd.). Čtenář bude seznámen s architekturou *Data Lakehouse* a s možnostmi jejího využití v praxi. Cílem této práce je implementovat zmíněné úložšitě pomocí vhodné technologie a následně ověřit jeho funkčnost na výše zmíněných medicínských datech. Dále je cílem srovnat nově implementované úložiště se stávajícím úložištěm medicínských dat, které uchovává data v grafové databázi a je vyvíjeno na *Fakultě aplikovaných věd* (FAV) *Západočeské univerzity* (ZČU). Výhody a popř. nevýhody *Data Lakehouse* budou reflektovány na současném řešení, kde musely být řešeny problémy, které by nový koncept úložiště měl implicitně řešit.

2 Vznik Data Lakehouse

Pojem *Data Lakehouse* vznikl jako odpověď na aktuální potřeby společností shromažďující velké množství dat za účelem jejich analyzování. Data jsou získávána z různých typů informačních systémů: ERP (*Customer Relationship Management*, CRM (*Customer Relationship Management*) apod. Dále se jedná o data z hardwarových zařízení jako jsou IoT[1] nebo z marketingových nástrojů spadajících pod E-commerce[2]. Tato data jsou dále centrálně zpřístupňována všem zainteresovaným týmům do práce s daty (analytické, obchodní a vědecké týmy) [39].

2.1 Data Warehouse

Historicky vzniklo několik přístupů k datům, resp. k tomu jak je efektivně ukládat za účelem jejich analyzování. Dlouhou historii mají datové sklady neboli DW (*Data Warehouse*), které jsou využívány jako základní součást informačních systémů pro podporu podnikání BI (*Business Intelligence*). DW agregují data z různých zdrojů do jediného centrálního konzistentního úložiště dat pro podporu analýzy dat, dolování dat, umělé inteligence neboli AI (*Artificial Intelligence*) a strojového učení neboli ML (*Machine Learning*).

První datové sklady vznikaly na konci 80. let a kvůli narůstajícímu objemu dat postupoval vývoj skokově dopředu. Brzy byla implementována architektura MPP (*Massively Parallel Processing*) [25]. Struktura databáze MPP je navržena tak, aby zpracovávala více operací najednou pomocí několika procesorových jednotek. V tomto typu architektury DW pracuje každá procesorová jednotka se svým vlastním operačním systémem a vyhrazenou pamětí. To umožňuje MPP databázím zpracovávat obrovské množství dat a poskytovat mnohem rychlejší analýzy založené na velkých souborech [13].

DW jsou efektivní pro ukládání a práci se strukturovanými daty a jsou považovány za první generaci platforem pro analýzu dat. Společnosti však musí pracovat i s velkým objemem semi-strukturovaných nebo nestrukturovaných dat. Data tohoto typu jsou heterogenní a použití DW pro tyto případy není vhodné ani nákladově efektivní.

[1] Síť fyzických zařízení, které jsou vybaveny různými senzory a připojeny do internetu za účelem výměny dat.

[2] Souhrnné označení pro obchodní činnosti prováděné na internetu za pomocích dalších elektronických prostředků.

2.1.1 Logická struktura

V datovém skladu jsou data členěna do schémat, které odpovídají dané funkční oblasti – tento přístup k datům je označován jako *schema-on-write* a je optimální pro podporu BI. Každé schéma je tvořeno dvěma skupinami tabulek. *Faktové tabulky* tvoří první skupinu – jsou v nich uložena veškerá analyzovaná data a zabírají většinu paměťového prostoru skladu. Může se jednat např. o číselné údaje, které jsou dále agregovány a používány k dalším výpočtům. Druhou skupinu tvoří *dimenzionální tabulky*, které obsahují číselníky ke kategorizaci dat uložených ve *faktových tabulkách*. *Faktové tabulky* jsou s těmi *dimenzionálními* spojeny pomocí cizích klíčů.

2.1.2 Výhody datového skladu

DW umožňují organizacím zpracovávat velké množství dat, extrahovat z nich signifikantní hodnoty a uchovávat historické záznamy. Níže uvedené čtyři vlastnosti, popsané vědcem jménem *William Inmon*, jsou považovány za hlavní výhody DW [20]:

Orientace na subjekt je snahou u relačních databází eliminovat redundanci uložených dat, které jsou normalizovány do třetí normální formy a následně provázány za pomoci primárních a cizích klíčů. U DW tento proces probíhá přesně naopak. Zde je snaha o co největší vnitřní separaci zdrojů za účelem čitelnosti dat pro koncového uživatele nebo oddělení. Zvýšením redundance dat dochází ke zvýšení nároků na paměťový prostor, ale koncový uživatel nebo oddělení může analyzovat data určité funkční oblasti nebo objektu, protože má v danou dobu všechna potřebná data.

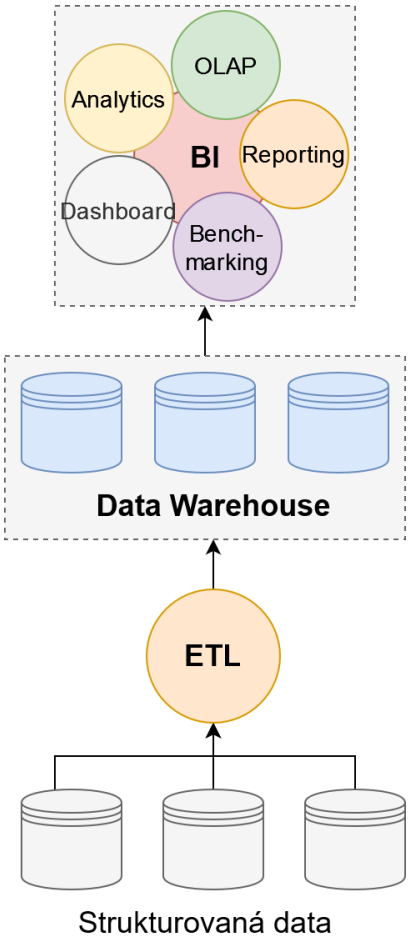
Integrace znamená vytváření konzistence mezi daty různých datových typů pocházejících z heterogenních datových zdrojů (např. jmenná konvence, atributy apod.). Na rozdíl od relačních databází jsou data shromažďována podle logického významu (orientace na subjekt) a ne podle původu (datového typu). Je to klíčová vlastnost, protože DW vrací relevantní data k dané funkční oblasti podle logického významu nezávisle na původu dat.

Nízká proměnlivost zaručuje, že jednou nahraná data jsou stabilní a časem neměnná.

Verzování v relačních databázích znamená, že jsou data ukládána jen v aktuální podobě. Analýzy prováděné nad DW mají i retrospektivní charakter, a proto je nutné ukládat historické záznamy.

2.1.3 Architektura

Na diagramu 2.1 je znázorněna běžná architektura DW. Zdrojem jsou strukturovaná data, která jsou extrahována pomocí procesu ETL (*Extract Transform Load*)[3]. Následně jsou nahrána do DW, kde jsou data členěna podle funkční oblasti do jednotlivých schémat. Nad DW funguje informační systém BI a pomocí dotazovacího jazyka jsou vytvářeny analytiky, reporty apod., které jsou následně prezentovány v rámci nástroje dashboard. Je potřeba velkého spektra datových analytický nástrojů, aby bylo možné pokrýt všechny uživatelské požadavky. Nástroje musí být zároveň dostatečně flexibilní, srozumitelné a poskytovat výsledky v rámci přívětivého uživatelského prostředí. Základem je vytvoření kompatibilního ekosystému nástrojů, které budou splňovat všechny uživatelské požadavky či obchodní cíle.



Obrázek 2.1: Struktura komponent v architektuře Data Warehouse.

[3]Proces extrakce, transformace a nahrání dat z jednoho či více zdrojů. Používá se při přenosu dat jednoho systému do druhého.

2.2 Data Lake

Datové jezero neboli *Data Lake* je druh úložiště, které mělo řešit problémy s ukládáním a analýzou nestrukturovaných a semi-strukturovaných dat popsaných v podkapitole 2.1. Jeho vývoj začal kolem roku 2010 jako reakce na limity datových skladů, které jsou velmi výkonné, poskytují škálovatelnou analýzu, ale jsou poměrně drahé a nedokáží si poradit s případy, které v dnešní době společnosti požadují. Datová jezera jsou považována za druhou generaci platforem pro datovou analýzu a používají se ke konsolidaci všech dat napříč společností v rámci jednoho centrálního místa. Hlavní výhodou datového jezera je ukládání velkého množství dat v jeho nativním formátu nebo s mírnou transformací. Tato data mohou být ukládána hned vedle strukturovaných databázových dat, a to je hlavní výhoda datového jezera, protože na rozdíl od většiny databází a datových skladů může přijímat a ukládat nestrukturovaná a semi-strukturovaná data. Jedná se především o obrázky, videa, zvukové stopy a dokumenty, které jsou důležité pro pokročilé analýzy a strojové učení. Datové jezero využívá *flat* strukturu a objektové úložiště pro ukládání dat. Data obsahují příslušná metadata a unikátní identifikátory, které usnadňují vyhledávání a načtení dat napříč funkčními oblastmi. Využitím levného objektového úložiště a otevřeného formátu dává jezeru potenciální možnost využívat data většímu spektru aplikací [2].

Do jisté míry byl tento záměr naplněn, ale datové jezero postrádá kritické funkce, které známe z relačních databází. Hlavním nedostatkem je chybějící podpora pro transakce, a proto i související problémy s konzistencí a izolací dat. Díky těmto nedostatkům není možné kombinovat čtení dat s dávkovými a streamovacími úlohami [25]. Dále ve srovnání s DW jsou ukládána méně spolehlivá data. Jsou stahována z různých zdrojů a ponechána v nezpracovaném stavu – není zkontrolována jejich přesnost a konzistence. Použitím datového jezera dojde ke ztrátě výhod datových skladů, a proto je v praxi využívána *dvouvrstvá architektura*, která integruje DW i datové jezero. Tato architektura nyní dominuje v průmyslu – používáno prakticky ve všech společnostech patřících na list *Fortune 500*[4]. Tento typ architektury je zdánlivě levný díky oddělenému úložišti od výpočetních operací. V první generaci (řešení jen s DW) jsou veškerá data z provozních systému ukládána přímo do DW. V této druhé generaci jsou data nejdříve ukládána do datového jezera a poté znovu do DW. Tím se zvyšuje složitost systému, latence a data v DW nemusí být vždy aktuální [28]. Navíc tento typ architektury není vhodný pro případy pokročilé analýzy a strojového učení a má níže popsané problémy:

[4]Fortune 500 je žebříček vydávaný časopisem Fortune, který řadí 500 nejúspěšnějších amerických společností podle jejich hrubého obratu.

Spolehlivost je zásadním parametrem při výběru datového úložiště a udržovat dva konzistentní systémy je obtížné a drahé. Je vyžadována existence ETL mezi systémy a vysoký výkon pro podporu rozhodování a BI. Zároveň může dojít k selhání a zavádění chyb do systému, které snižují kvalitu dat (např. malé diference v datech mezi datovým jezerem a DW) [28].

Aktuálnost dat není v DW zaručena v porovnání s datovým jezerem a dochází k zastarávání, protože načítání nových dat trvá řádově dny. Ve srovnání s první generací systému pro analýzu dat je to krok zpět, protože zde byla nová provozní data okamžitě dostupná [28]. Podle průzkumu společnosti *Dimensional Research and Fivetran* 86 % analytiků používá zastaralá data a 62 % z nich musí čekat na aktuální data, a to i několikrát za měsíc [26].

Omezená podpora pro pokročilé analýzy může být také problémem, protože společnosti potřebují od systému získávat prediktivní reporty, a to např. nabízené slevy zákazníkům na základě ročního období. Existují kvalitní ML nástroje (např. TensorFlow, PyTorch a XGBoost), které dobře fungují nad DW. Tyto nástroje však pracují s velkými datovými sadami pomocí NoSQL (*Not only Structured Query Language*) dotazů, pro které nejsou DW optimalizovány. Čtení dat přes ODBC (*Open Database Connectivity*), popř. proprietárně přes JDBC (*Java Database Connectivity*) je neefektivní a neexistuje přímý přístup k nativním datům pomocí DW. Pro tyto účely je možné exportovat nativní data do souborů a následně je ukládat do DW. Poté je však nutné implementovat třetí ETL, což má za následek větší množství chyb v systému a snížení kvality dat. ML nástroje mohou mít i přímý přístup k datovému jezeru, ale ztratí transakční zpracování ACID (*Atomocity, Consistency, Isolation, Durability*) a všechny další výhody relačních databází [28].

Celková cena vlastnictví je důležitým aspektem při výběru datového úložiště. V tomto případě je nutné počítat s náklady za provoz ETL, ale i za dvojnásobný diskový prostor, resp. za data zkopírovaná z datového jezera do DW [28].

2.2.1 Logická struktura

Data nejsou při procesu ukládání členěna do schémat podle příslušnosti k dané funkční oblasti jako je tomu u datového skladu. Pro ukládání dat

slouží objektová databáze neboli *objektové úložiště*, kde jsou data ukládána v otevřeném formátu *Parquet* nebo ORC (*Optimized Row Columnar*) do flexibilních datových kontejnerů (objektů).

Každý objekt má jednoznačný identifikátor a skládá se ze samotných dat a metadat popisující daný objekt. Tento přístup je velmi odlišný ve srovnání s ukládáním strukturovaných dat v běžných databázích. Zde je využíváno blokově orientovaného rozhraní, které zapisuje a čte data v blocích pevné délky. *Objektové úložiště* obsahuje příkazy pro vytvoření/odstranění objektu, k zápisu přečtených bajtů do jednotlivých objektů, získání atributů objektu a nastavení. Je tedy zodpovědné za ukládání objektů včetně jejich metadat a implementuje přístupová oprávnění. Objekty jsou ukládány ve *flat* struktuře, takže nejsou organizovány do žádných adresářů ani hierarchií. K reprezentaci objektu slouží jen jeho unikátní identifikátor – hierarchii a vyšší logiku zajišťují externí aplikace, které pomocí příslušného API (*Application Programming Interface*) komunikují s datovým jezerem. Návrh a implementace kvalitního API je klíčová, protože je v těsné integraci s aplikací, která přímo řídí a organizuje datové úložiště. V mnoha případech probíhá komunikace aplikace s API rovnou přes zabezpečený protokol HTTPS (*Hypertext Transfer Protocol Secure*) [8].

2.2.2 Architektura

Tří hlavní principy, kterými se datové jezero liší od konvenčních datových úložišť [40]:

1. Žádná data nejsou odmítnuta a jsou ukládána nezávisle na jejich kvalitě. Shromážděná data je možné znovu načíst a pracovat s nimi v původní podobě.

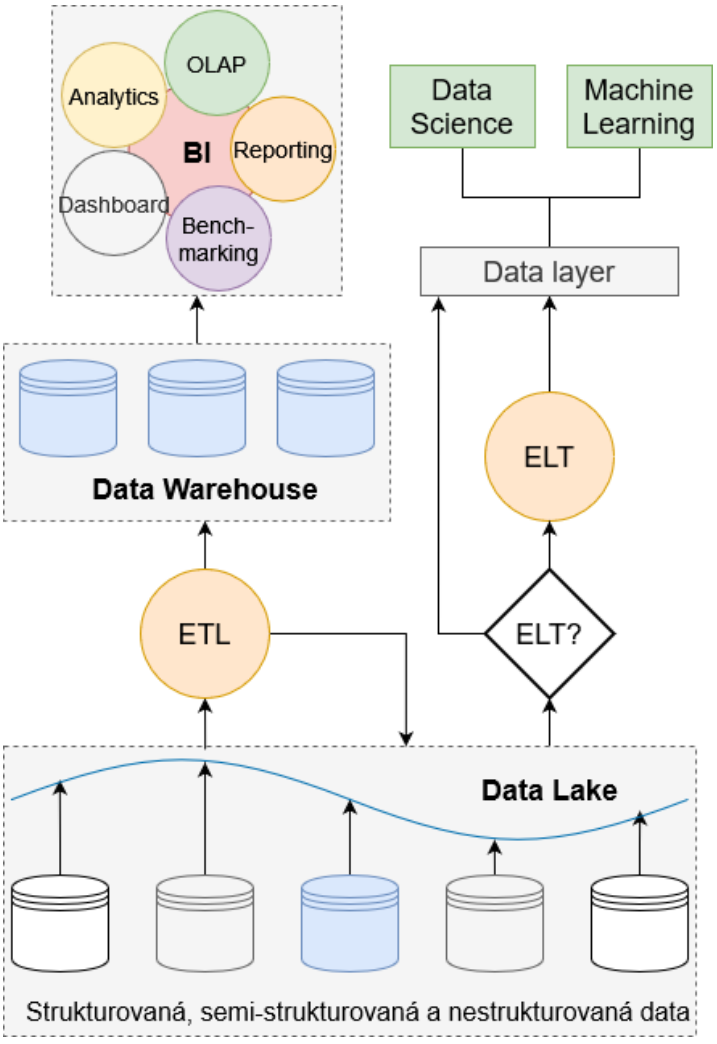
2. Data mohou projít mírnou transformací, ale stále musí být možné je uložit v jejich surovém a netransformovaném stavu.

3. Transformace dat probíhá až podle potřeb na základě specifických požadavků dané analýzy. Tento přístup je známý jako *schema-on-read*.

Z výše uvedených principů vyplývá, že v praxi je úložiště *Data Lake* téměř nepoužitelné. Běžným přístupem je použití více specializovaných úložišť, aby bylo možné ukládat a zpracovávat různé typy dat odděleně. Primárně jsou extrahována data z DW, která jsou využívána informačním systémem BI pro interní potřeby organizace.

Na diagramu 2.2 je znázorněna architektura datového jezera využívaná v praxi. Vstupem jsou strukturovaná, semi-strukturovaná a nestrukturovaná

data, která jsou extrahována pomocí procesu ETL. Levá strana diagramu je téměř srovnatelná s diagramem 2.1. Datový sklad zde umožňuje ukládání strukturovaných dat hned vedle datového jezera. Hlavní změnou je zpětná vazba na data, která slouží pro přístup známý jako *schema-on-read* za pomocí volitelného použití ELT (*Extract, Load, Transform*). Data jsou transformována až podle požadavků nástroje přistupujícího k datovému jezeru přes jeho API. Obecně mají přímou vazbu na datové jezero všechny nástroje, které potřebují pracovat s nativním formátem dat. Jedná se především o strojové učení nebo vědecké týmy zabývající se analýzou dat. S datovým jezerem také komunikuje informační systém BI a pomocí dotazovacího jazyka jsou vytvářeny analytiky, reporty apod., které jsou následně prezentovány v rámci nástroje dashboard.



Obrázek 2.2: Struktura komponent v architektuře Data Lake.

2.3 Data Lakehouse

Datová jezera přinesla řadu výhod, ale mají řadu omezení, kvůli kterým jsou v praxi využívána pouze v kooperaci s DW. Postupem času tento model přestával společnostem vyhovovat, a proto datové sklady začaly přidávat podporu externích tabulek pro formát *Parquet* a ORC. Díky tomu je možné provádět SQL dotazy směřující na datové jezero pomocí DW, ale neodstraňuje správu tabulek v datovém jezeru. Stále zůstává problém s existencí ETL mezi datovým jezerem a skladem a z toho plynoucí nevýhody – zastaralá data v DW a díky tomu vzniklé analytické problémy. V praxi tyto konektory stejně nefungují optimálně, protože *SQL engine* je optimalizován pro jeho interní formát. Například *Spark SQL*, *Presto*, *Hive* nebo *AWS Athena* jsou určeny pro přímé dotazy na datové jezero, neřeší však hlavní problémy datových jezer, a proto nemohou nahradit DW [28].

Začaly se proto objevovat nové koncepty, které měly nedostatky datových jezer řešit. Vznikl nový koncept úložiště *Data Lakehouse*, který je považován za třetí generaci platforem pro datovou analýzu a kombinuje nejlepší vlastnosti datových jezer a datových skladů. Nové úložiště umožňuje všem funkčním oblastem pracovat nad daty v rámci jedné platformy. Je možné používat BI v celém rozsahu, SQL dotazy pro analytiky, ML a data science[5] [39]. Architektura má níže popsané klíčové vlastností:

Podpora transakcí je hlavní výhodou, protože v každé společnosti k datům přistupuje (čtení/zápis) několik uživatelů současně. Každá operace je tedy transakční, takže proběhne celá úspěšně nebo je přerušena. Po přerušení se provádí proces *rollback*, který je schopen databázi uvést do stavu před zahájením neúspěšné operace, a to díky záznamu nazývanému *žurnál*. Před zahájením transakce jsou do *žurnálu* zapsány veškeré budoucí operace, poté je provedena samotná operace a následně je zapsáno, že operace byla úspěšně dokončena a záznam je zrušen. Pokud dojde v rámci vykonávání transakce k přerušení, tak je právě díky *žurnálu* možné vrátit databázi zpět do konzistentního stavu. Každá transakce má ACID (*Atomocity, Consistency, Isolation, Durability*) vlastnosti, které zaručují platnost dat navzdory chybám, výpadkům napájení a dalším externím hrozbám:

- *Atomicita* – série databázových operací proběhne najednou v atomické transakci nebo nedojde k žádné změně.

[5]Data science neboli datová věda je interdisciplinární obor, který využívá vědecké metody, procesy, algoritmy a systémy pro získávání znalostí z dat.

- *Konzistence* – zajišťuje, že transakce převede databázi z jednoho konzistentního stavu do druhého. Nesmí být porušena žádná databázová pravidla (např. doménová omezení) a referenční integrita (vztah mezi primárním a cizím klíčem).
- *Izolace* – dvě navzájem běžící transakce se nesmí ovlivňovat v případě, že přistupují ke stejným datům. Jedna z nich musí počkat na dokončení té druhé, aby byla zachována izolace.
- *Stálost* – změny provedené transakcí jsou stálé a nejsou vymazány výpadkem systému.

Zpracování velkých metadat je realizovatelné díky využití výhod distribuovaného výpočetního výkonu *Apache Spark* ke zpracování všech metadat. Díky tomu je možné zpracovávat tabulky o velikosti petabajtů s miliardami oddílů a souborů [39].

Indexování slouží ke zrychlení vyhledávacích a dotazovacích procesů v databázi. Architektura k tomu využívá dělení tabulek a různé statistické techniky jako jsou *Bloomovy filtry*[6] a přeskakování dat, aby se zabránilo čtení velkých částí datových celků, a to přináší rapidní zrychlení [39].

Vynucení a správa schémat zajišťuje kvalitu vkládaných dat. Možnost vynucování, vývoje a podpora architektury schémat DW (*hvězda* a *sněhová vločka*). Systém je schopen uvažovat o integritě dat a má robustní mechanismy pro řízení a audit [25]. Schéma není nutné definovat jako je tomu u tradičního DW, ale může být definováno samotnými daty a postupně upravováno – tento proces je známý jako *schema evolution*.

Podpora Business Intelligence umožňuje používat nástroje BI přímo na zdrojových datech. BI může pracovat s aktuálními daty, čímž se zvyšuje kredibilita získávaných informací a znalostí. Tímto přístupem jsou snižovány náklady nutné na provoz, protože odpadá nutnost udržování dvou kopií dat pro datové jezero a DW [25].

Izolace úložiště od výpočtů je zajištěno použitím oddělených clusterů stejně jako je tomu u moderních datových skladů. Tímto způsobem lze lépe škálovat více souběžných přístupů k datům a jejich velikost. V praxi je běžné, že datová aktiva rostou exponenciálně, ale výpočetní potřeby nemusí růst stejnou rychlostí [25]. Oddělení dat od vý-

[6]Prostorově efektivní pravděpodobnostní datová struktura, která je využívána na ověřování příslušnosti prvků do množiny.

početních operací umožňuje spravovat náklady na úložiště samostatně a implementovat různé optimalizační funkce pro snížení nákladů [19].

Otevřenost znamená, že formáty pro ukládání jsou otevřené, standardizované a pomocí API mohou k datům efektivně přistupovat různé nástroje, ML, ale rovnou i knihovny (např. knihovny pro Python nebo jazyk R) [25]. Za standardní open-source formát je dnes považován *Parquet*, který téměř vytlačil dříve používaný CSV (*Comma Separated Values*), a to i z důvodu, že využívání *Parquet* formátu je na cloudových řešení (např. Google Cloud, Amazon S3, Oracle Cloud) o několik řádů levnější.

Podpora streamování poskytuje reporty v reálném čase, které jsou v mnoha podnicích standardem a do jisté míry na nich závisí úspěch společnosti. Podpora streamování eliminuje nutnost udržovat systém jen pro tento případ užití – jsou tím i sníženy náklady na provoz systému [19].

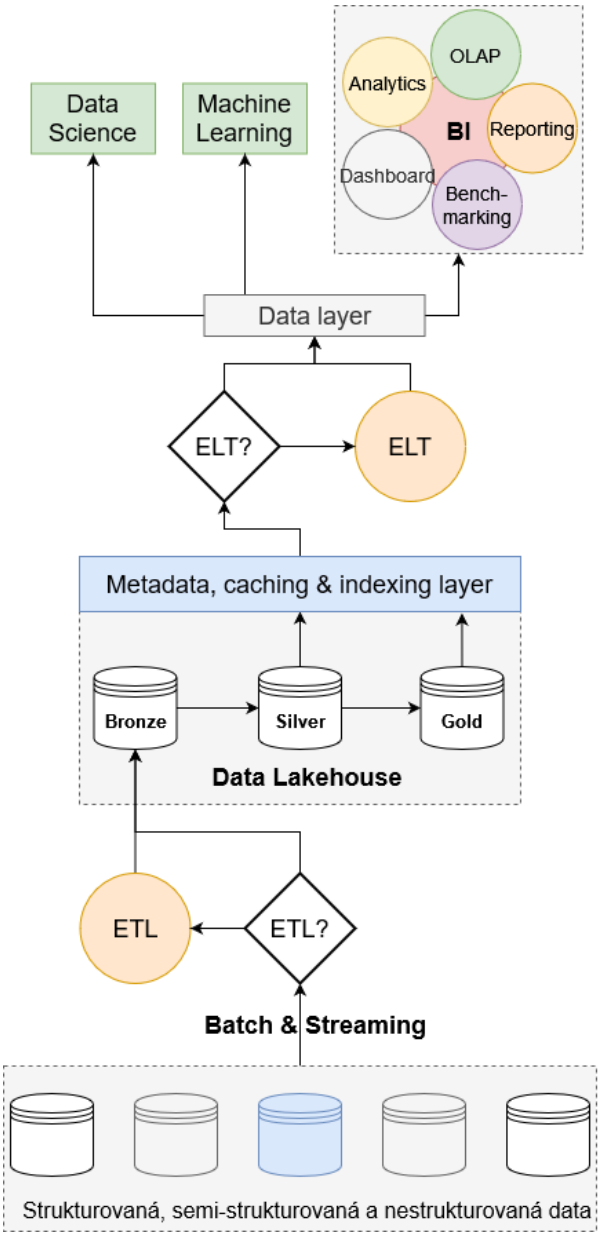
2.3.1 Logická struktura

Data jsou ukládána podobně jako je tomu u datových jezer (viz podkapitola 2.2.1). Pro data je opět použit standardní formát *Parquet* nebo ORC, ale metadata již nejsou součástí objektů. Nad úložištěm objektů je přidána transakční vrstva obsahující metadata, která zajišťuje transakční zpracování ACID při zachování celého objemu dat v datovém jezeře. Díky tomu mohou aplikace číst objekty přímo z tohoto úložiště při zachování standardního formátu.

2.3.2 Architektura

Na diagramu 2.3 je znázorněna architektura *Data Lakehouse*, která je využívána v praxi. Vstupem jsou strukturovaná, semi-strukturovaná a nestrukturovaná data. Většinou se jedná o velké množství dat, takže přichází v dávkách nebo ve formě kontinuálního přenosu dat (stream). Následně mohou být data extrahována pomocí ETL procesu. V této architektuře nemusí být ETL proces vůbec použit a častěji se setkáváme s tím, že jsou netransformovaná surová data rovnou nahrána do první vrstvy – u *Databricks* je to *bronzová vrstva* a u *AWS* se jedná o *přijímací vrstvu*. Při nahrávání jsou k datům přidána relevantní metadata, aby bylo možné provádět transakční zpracování. Nástroje přistupují k datům pomocí poskytnutého API, které

komunikuje přímo s transakční vrstvou metadat. Opět zde může být využit přístup známý jako *schema-on-read* za pomocí volitelného použití ELT. Data jsou transformována až podle požadavků nástroje přistupujícího k datům, a proto může pracovat s nativním formátem dat. Jedná se především o strojové učení nebo vědecké týmy zabývající se analýzou dat. S *Data Lakehouse* také komunikuje informační systém BI a pomocí dotazovacího jazyka jsou vytvářeny analytiky, reporty apod., které jsou následně prezentovány v rámci nástroje dashboard.



Obrázek 2.3: Struktura komponent v architektuře Data Lakehouse.

2.3.3 Definice datových vrstev podle Databricks

Architektura *Data Lakehouse* od společnosti *Databricks* se řídí návrhovým vzorem, která řadí data do třech vrstev podle kvality dat. Každá z těchto vrstev obsahuje příslušné tabulky, které slouží danému účelu v řetězci zpracování dat a zajišťují vysokou dostupnost dat pro všechny případy následného použití.

Bronzová vrstva Získává/uchovává nezpracovaná provozní data ze všech externích zdrojů za pomocí informačních systémů ze skupiny OLTP (*On-Line Transaction Processing*). Tato vrstva přijímá stream dat pomocí pipeline (např. Apache Kafka nebo Amazon Kinesis) nebo přímo z existujícího úložiště objektů (např. Amazon S3). Data při ukládání prochází mírnou transformací, ale struktura tabulek (doplněna o metadata, datum/čas načtení a ID procesu) odpovídá strukturám tabulek ze zdrojového systému. Získaná data různých datových typů (např. JSON, CSV a XML) převádí do standardního formátu *Delta* (formát tabulky u *Delta Lake*). Cílem této vrstvy je rychlé zachycení změn dat a schopnost poskytnout historický archiv zdroje (známé jako *cold storage*), datovou linii, auditovatelnost a číst původní data bez nutnosti opětovného načítání z externího zdroje [7, 37].

Stříbrná vrstva Získává/uchovává čistá atomická data za pomocí informačních systémů ze skupiny OLAP (*On-Line Analytical Processing).* Data prochází větší transformací než u předchozí vrstvy s důrazem na vynucení schémat. Jsou spárovány datové sady z *bronzové vrstvy*, probíhá transformace dat (čistění a filtrování) a rozdělování dat do funkčních oblastí (např. podle obchodní logiky). Dále jsou prováděny různé techniky pro komprimaci dat za účelem zlepšení výkonu dotazů. Tato vrstva přináší podnikový pohled na klíčové obchodní entity, koncepty a transakce (např. obchody, hlavní zákazníci, neduplikované transakce a tabulky křížových dotazů). Takto upravená data jsou využívána pro pokročilé analýzy a ML. Figuruje zde ELT proces, kdy jsou nejdříve data uložena s minimálními úpravami a čištěním a transformována až při vzniku potřeby. Díky tomu je zajištěna rychlost a agilita při získávání a doručování dat v datovém jezeře. Z pohledu datového modelování se v této vrstvě vyskytuje více dat se strukturou odpovídající *třetí normální formě* [7].

Zlatá vrstva Při načítání dat ze *stříbrné* do *zlaté vrstvy* je aplikováno mnoho komplexních transformací a obchodních pravidel pro daný projekt (opět za pomocí OLTP), a proto jsou i data organizována ve specifických

databázích. Jedná se o poslední vrstvu určenou především pro prezentaci dat, takže data jsou více denormalizovaná a existuje menší počet spojení mezi datovými modely. Mnoho z těchto modelů je založeno na *hvězdných schématech*.

2.3.4 Definice datových vrstev podle AWS

Společnost *AWS* rozděluje data v rámci architektury *Data Lakehouse* do pěti vrstev podle jejich kvality za pomocí svých proprietárních služeb (komponent).

Přijímací vrstva První vrstva provádí migraci dat pomocí dávkových a streamovacích metod. Využívá k tomu služby *Data Migration Service* pro RDBMS (*Relational Database Management System*) a *Amazon Kinesis* pro ukládání do vyrovnávací paměti. Následně jsou data uložena na úložiště *Amazon Redshift* nebo *Amazon S3*, které poskytuje levné úložiště pro ukládání objektů a jejich schémat v otevřených formátech (např. Apache Parquet, CSV, JSON). Data jsou kontinuálně upravována, když se pohybují mezi vrstvami jezera. V praxi poté lze vidět, že mnoho DW (popř. datových trhů) přechází z klasického RDBMS na architekturu *Data Lakehouse*. Společnosti poté mohou provádět pokročilé analýzy mezi různými formáty dat, což předtím nebylo možné nebo příliš nákladné. Např. díky spojení dat o IoT/výrobě s daty o prodejích a marketingů je možné plánovat zlepšování kvality na základě defektů, které jsou klasifikovány do různých kategorií (známe jako *Defect Analysis*). Dalším příkladem je potenciální přínos pro zdravotnictví v oblasti genomiky. Datové trhy s klinickými daty *EMR/HL7* jsou propojeny s finančními náklady na daný zdravotní problém [7]. Díky tomu může proběhnout včasná analýza, odhalení možných zdravotních potíží a v neposlední řadě ušetření celkových nákladů za léčbu pacienta.

Vrstva pro ukládání dat Druhá vrstva je zodpovědná za poskytování služeb pro ukládání velkého množství dat. Podporuje škálování a ukládá nestrukturovaná i vysoce strukturovaná data v různých stavech připravenosti pro potřeby další vrstvy nebo pro aplikace využívající data z této vrstvy [33].

Vrstva datového skladu Architektura *Data Lakehouse* může ukládat i zpracovávat strukturovaná data, ale v praxi je vhodnější tento typ dat ukládat do DW, kde jsou strukturovaná do *hvězdicového* nebo *vločkového schéma*. Tato data přicházejí např. z transakčních systémů nebo relačních

databází. Moderní DW mohou ukládat až petabajty dat a díky MPP a rychlému úložišti mohou vykonávat SQL dotazy s nízkou latencí. Data jsou samozřejmě před uložením předzpracována procesem ETL. Následné změny v datech a schématu jsou striktně validovány, aby se zachovala důvěryhodnost dat v rámci všech podnikových sfér [33].

Vrstva datového jezera Centralizované datové úložiště pro ukládání strukturovaných, semi-strukturovaných a nestrukturovaných dat. Další informace spjaté s toutou vrstvou jsou zmíněny v podkapitole 2.3.

2.3.5 Srovnání jednotlivých vrstev

Společnosti *Databricks* a *AWS* přistupují k definici jednotlivých vrstev odlišnými způsoby. *Databricks* definuje jen vrstvy týkající se přímo architektury Data Lakehouse, přičemž každá z nich reflektuje míru zpracování, resp. kvalitu dat na dané vrstvě. *AWS* definuje vrstvy na úrovni komponent, protože v definici jednotlivých vrstev jsou popsány použité technologie pro zpracování dat. Lze konstatovat, že z pohledu *AWS* se nejedná jen o Data Lakehouse, ale o komplexní systém pro zpracování a analýzu dat. Důkazem toho je, že zde existuje samostatná vrstva pro DW i Data Lakehouse. U strukturovaných dat počítají se zpracováním na vrstvě pro DW a u semi-strukturovaných a nestrukturovaných na úrovni Data Lakehouse.

I přestože jsou oba pohledy na vrstvy zcela rozdílné, tak mají jednu společnou vlastnost. Vrstvy jsou definovány pro určitou míru zpracování dat, a to vzestupně. První vrstvu tvoří nezpracovaná surová data, která jsou ve většině případů získaná ze streamu dat. Při pohledu do vyšších vrstev se míra zpracování dat zvyšuje a aplikacím používající dané vrstvy jsou poskytována kvalitnější data.

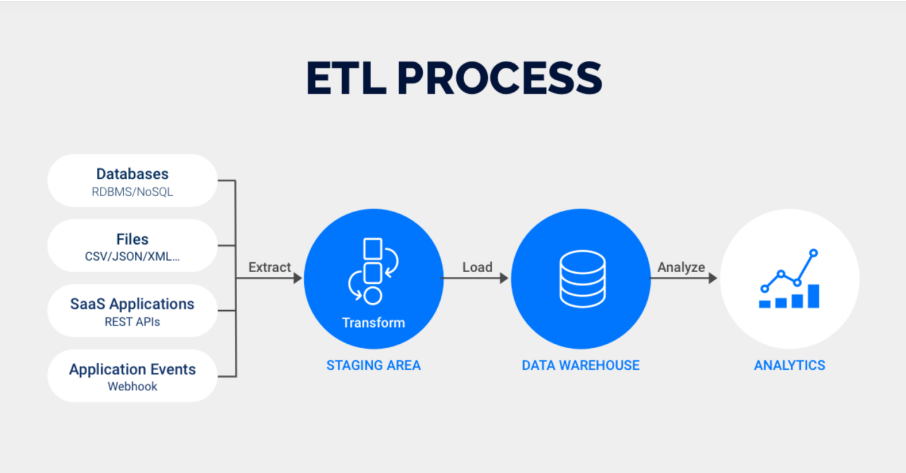
2.4 ETL vs. ELT

ETL (*Extract, Transform, Load*) a ELT (*Extract, Load, Transform*) jsou procesy pro integraci dat, jejichž hlavním cílem je transportovat data ze zdrojového místa do cíle. Každý z těchto procesů je vhodný pro určité případy užití, ale hlavní rozdílem je, že ETL transformuje data před uložením do cílového místa. Naopak ELT nejdříve data uloží a transformuje je až při vzniku potřeby [29]. Lze říci, že ELT ukládá netransformovaná surová data, což je typičtější přístup pro datová jezera.

2.4.1 ETL

„Extrakce, transformace a načítání“ je proces, který extrahuje surová data ze zdrojů, následně je pomocí sekundárního serveru transformuje a načte do cílové databáze. Používá se hlavně v případech, kdy data musí odpovídat podmínkám a schématu cílové databáze. Takovými databázemi jsou DW pro online analytické zpracování (OLAP), které akceptují pouze relační datové struktury založené na SQL [29]. Dalším případem užití mohou být bezpečnostní důvody, kdy je požadováno, aby data byla před samotným uložením předzpracována – potenciálně nebezpečná data jsou upravena nebo úplně vyjmuta. Ve všech případech jsou extrahovaná data přesunuta do cílové databáze až po úspěšné transformaci.

Celý proces lze vidět na obr. 2.4, kde za zdroj jsou považovány databáze, soubory nebo data z cloudových služeb získaná pomocí REST (*Representational State Transfer*) API. Po extrakci jsou data transformována na sekundárním serveru a po úspěšné transformaci nahrána do DW. Data uložená v cílové databázi jsou poté využívána pro analytické účely. Tato metoda se objevila již v 70. letech 20. století a stále převládá v lokálních databázích, které mají omezenou paměť a výpočetní výkon [29].



Obrázek 2.4: Komponenty ETL procesu [29].

Výhody ETL proces je výhodný pro zpracování strukturovaných dat a je ideální pro výpočetně náročné transformace (např. u systémů se starší architekturou) nebo pro manipulaci dat před vstupem do systému – typickým příkladem je de-identifikace, anonymizace nebo vymazání osobních údajů

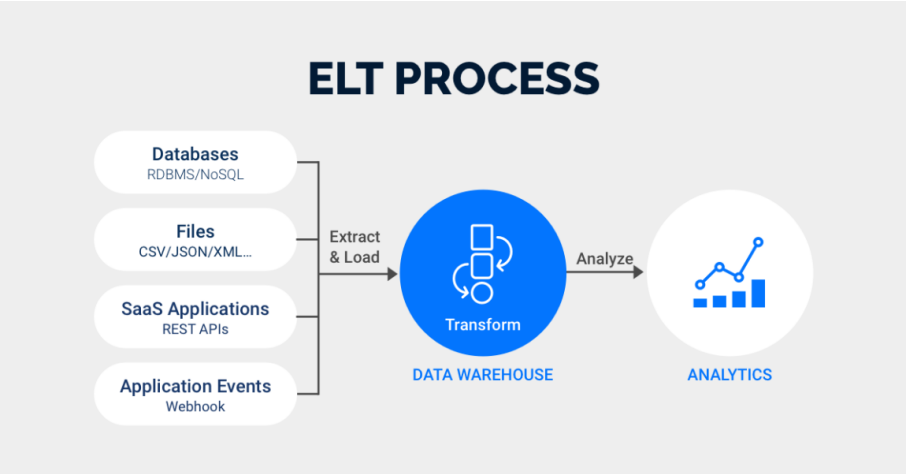
kvůli GPDR (*General Data Protection Regulation*)[7]) [29].

Nevýhody Nevytváří historický archiv se surovými daty, takže jej není možné znovu prohledávat. Resp. je možné jej znovu prohledávat, ale výsledek bude pravděpodobně totožný [29].

2.4.2 ELT

„Extrakce, načítání a transformace“ je proces, který extrahuje data ze zdrojů, ale před samotným uložením do cílové databáze nevyžaduje transformaci dat. ELT načítá nezpracovaná data přímo do cílové databáze bez toho, aby je posílal na sekundární server k transformaci. Nezpracovaná surová data jsou uložena do DW nebo Data Lakhouse po neomezenou dobu, takže je možné vykonávat opakované transformace. Díky tomu, že se jedná o paměťově a výkonově náročný proces, tak je zpravidla vykonáván v rámci cloudových řešení jako jsou např. Snowflake, Amazon Redshift, Google BigQuery a Microsoft Azure. Tyto služby usnadňují ukládání surových dat a poskytují transformace v rámci jejich aplikací. ELT je poměrně nový proces a stává se populárnějším i díky tomu, že společnosti zavádějí cloudovou infrastrukturu [29].

Celý proces lze vidět na obr. 2.5, kde jsou opět za zdroje považovány databáze, soubory nebo data z cloudových služeb získaná pomocí REST API. Po extrakci jsou data nahrána do DW a transformace probíhá až na základě potřebných analytik.



Obrázek 2.5: Komponenty ELT procesu [29].

[7]Obecné nařízení představující právní rámce ochrany osobních údajů v evropském prostoru.

Výhody ELT je rychlejší, protože rovnou načítá data do cílové databáze a transformace může probíhat paralelně s načítáním. Vytváří historický archiv pro generování reportů v rámci BI. Cíle a strategie společnosti se v průběhu mění a BI týmy mohou znovu analyzovat surová data a provádět nové transformace s využitím komplexních datových sad. Dále poskytuje lepší zpracování semi-strukturovaných a nestrukturovaných dat [29].

Nevýhody Ve většině případů je nutné využívat i ETL proces a s tím spojené dodatečné náklady za provoz tohoto systému.

2.5 FAIR data

V roce 2016 byl definován pojem FAIR (*Findable, Accessible, Interoperable, Reusable*), a to časopisem *Scientific Data* v rámci článku *The FAIR Guiding Principles for scientific data management and stewardship*[8]. Cílem článku je definovat pravidla pro sdílení a opětovné využití dat počítači a lidmi. Zkratka FAIR popisuje, že data musí být *nalezitelná*, *dostupná*, *interoperabilní*[9] a *opětovně využitelná*. Tyto principy jsou podporovány i Evropskou komisí. Autoři článku také formulovali 15 principů FAIR dat, které jsou níže popsány. V souvislosti s FAIR data se často setkáváme s pojmem *Open data*, který zahrnuje veškeré volně dostupná data. Tato data je možné opakovaně využívat, kombinovat s jinými datovými sadami a dále je distribuovat. I *Open data* by měla splňovat principy FAIR (v ideálním případě), ale ve skutečnosti jen část *Open dat* toto splňuje (viz obr. 2.6). Naopak FAIR data nemusí splňovat principy *Open dat*, protože např. z důvodů ochrany osobních údajů nemusí být volně dostupná [6].

2.5.1 15 principů FAIR

Níže je popsáno 15 principů FAIR definovaných v článku zmíněném v podkapitole 2.5.

Nalezitelnost Základem je opětovná využitelnost dat a z tohoto důvodu je nutné zajistit, aby je cílová skupina lidí či počítačů mohla najít. V tomto případě jsou klíčová *strojově čitelná metadata*. V rámci *nalezitelnosti* platí následující principy [6]:

[8] `https://www.nature.com/articles/sdata201618`

[9] Interoperabilita je schopnost různých systémů vzájemně spolupracovat, poskytovat si služby, dosáhnout vzájemné součinnosti. Termín se používá nejen pro spolupráci na technické úrovni, ale i pro sociální komunikaci, politickou spolupráci a spolupráci služeb.

1. data/metadata mají jednoznačný a perzistentní primární klíč,
2. data obsahují dostatečné množství metadat,
3. je zajištěno, že data/metadata jsou indexována v prohledávaných zdrojích,
4. metadata specifikují identifikátor.

Dostupnost Pro získaná data musí být zajištěn *otevřený přístup* ideálně pomocí repozitáře. Jak již bylo zmíněno, k datům může být *omezený přístup* a nutnost autentizace, ale poté musí být přístupná alespoň metadata. V rámci *dostupnosti* platí následující principy [6]:

1. data/metadata je možné získat pomocí jejich primárních klíčů při využití standardního API,
2. API je otevřené a volně k dispozici,
3. přístup k datům může být omezený, a proto API musí umožňovat autentizaci a autorizaci,
4. metadata musí být dostupná i v případě nedostupnosti dat samotných.

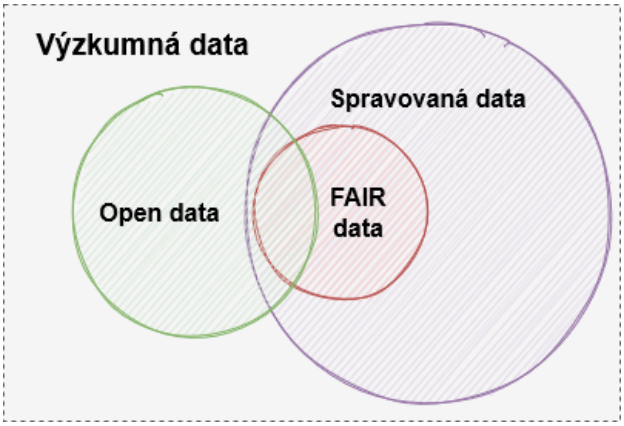
Interoperabilita Cílem je zajistit, aby data bylo možné integrovat s jinými datasety. Tím bude zajištěna i schopnost komunikace různých informačních systémů pomocí standardizovaného formátu. V rámci *interoperabilita* platí následující principy [6]:

1. data/metadata používají pro popis znalostí unifikovaný a široce používaný jazyk,
2. data/metadata používají slovníky (číselníky) řídící se zásadami FAIR,
3. data/metadata obsahují reference (cizí klíče) na další data/metadata.

Opětovná využitelnost *Opětovná využitelnost* je jedním z hlavních bodů principů FAIR, protože nad daty je možné provádět opětovné transformace a analýzy. Data musí být dostatečně popsána a sdílená pod licencí s malým množstvím restrikcí. Potenciální uživatelé musí znát vznik dat, co popisují a jak s daty mohou pracovat. V rámci *opětovné využitelnosti* platí následující principy [6]:

1. data/metadata obsahují relevantní atributy,

2. data/metadata jsou zveřejněna pod dostupnou licencí,
3. u dat/metadat je zmíněn způsob jejich vzniku,
4. data/metadata splňující standardy vědecké komunity v daném oboru.



Obrázek 2.6: Diagram zobrazující závislost FAIR a Open dat [6].

3 Současný stav

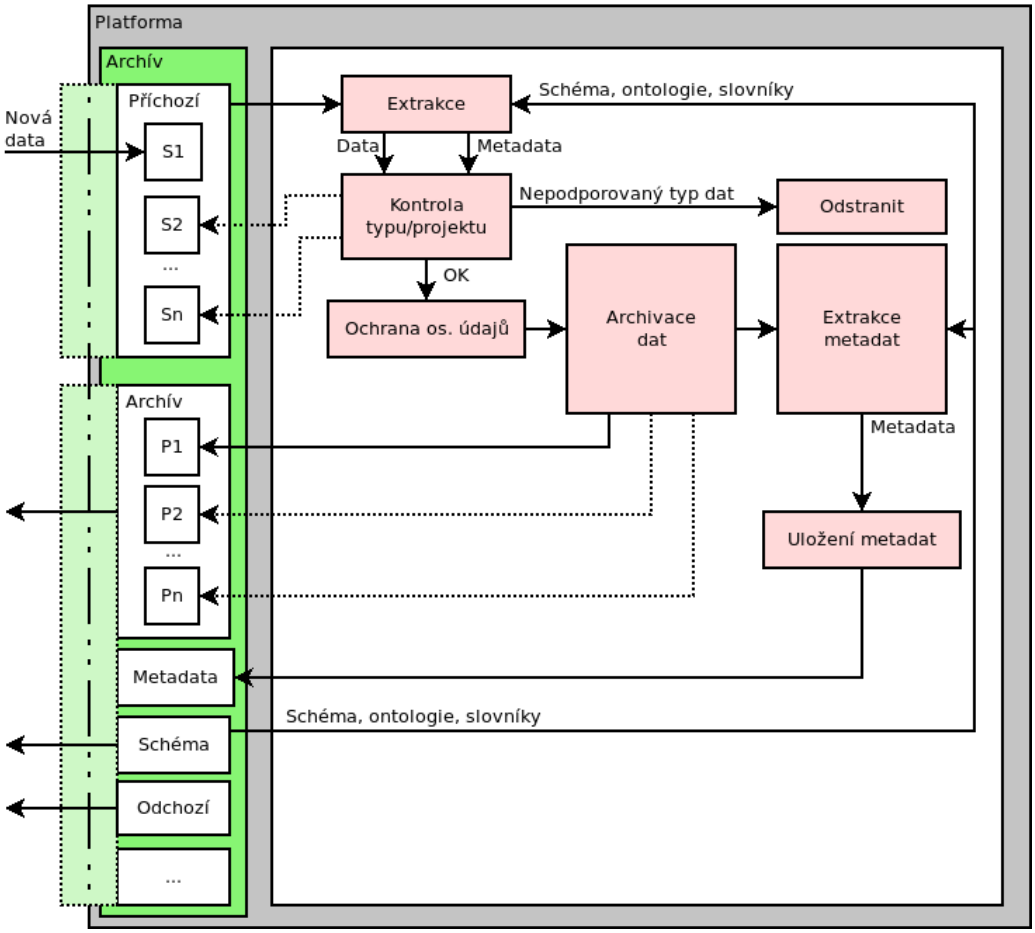
V současné době funguje na *Fakultě aplikovaných věd* (FAV) *Západočeské univerzity v Plzni* (ZČU) komplexní MRE (*Medical Research & Education*) platforma zpracovávající medicínská data (primárně ve formátu DASTA a DAICOM). Část softwarového vybavení je i na straně zdravotnického zařízení, protože nově příchozí data musí být předzpracována před dalším použitím pro výzkumné účely na fakultě.

3.1 Nově příchozí data

Na diagramu 3.1 přichází nová data do vstupního bodu `S1` a následuje první krok procesu nazvaný *Extrakce*, kde jsou získávána metadata pro kontrolu dat. Dalším krokem procesu je *Kontrola typu/projektu*, který může nevalidní data odstranit (např. nepodporovaný formát, verze, poškozená nebo nekonzistentní data) nebo je poslat do karantény, kde následně odpovědný uživatel rozhodne o dalším postupu s těmito daty. Pokud data projdou přes tyto filtry, tak mohou být zařazena do různých projektů, protože nelze předpokládat, že by zdroj znal správné členění. Po *Kontrole* následuje blok *Ochrana osobních údajů*, který produkuje *sekundární data* sloužící pro výzkumné účely a uchovává je v archivu k dalšímu využití [41].

3.2 Ochrana osobních údajů

Po úspěšné *Kontrole* přichází na řadu nezbytný blok *Ochrana osobních údajů*, který zobrazuje diagram 3.2. Celý tento proces běží v rámci zdravotnického zařízení a jsou zde používány metody pro *anonymizaci*, *de-identifikace* nebo *pseudonymizaci dat* v závislosti na požadavcích. Základem je nepoškodit data včetně anatomických struktur a zachovat vazby v původních datech. Podle typu dat je posouzeno, zdali se jedná o obrazová data (DICOM). V těchto datech je hledán výskyt textu pomocí software *PureImage*. Pokud jsou v datech detekovány osobní údaje, tak jsou uloženy do karantény. U strukturovaných a semi-strukturovaných dat nedochází jen k detekci, ale i k modifikaci dat pomocí software *AnonMend*. Výstupem jsou *sekundární data* podléhající pravidlům pro ochranu osobních údajů [41].



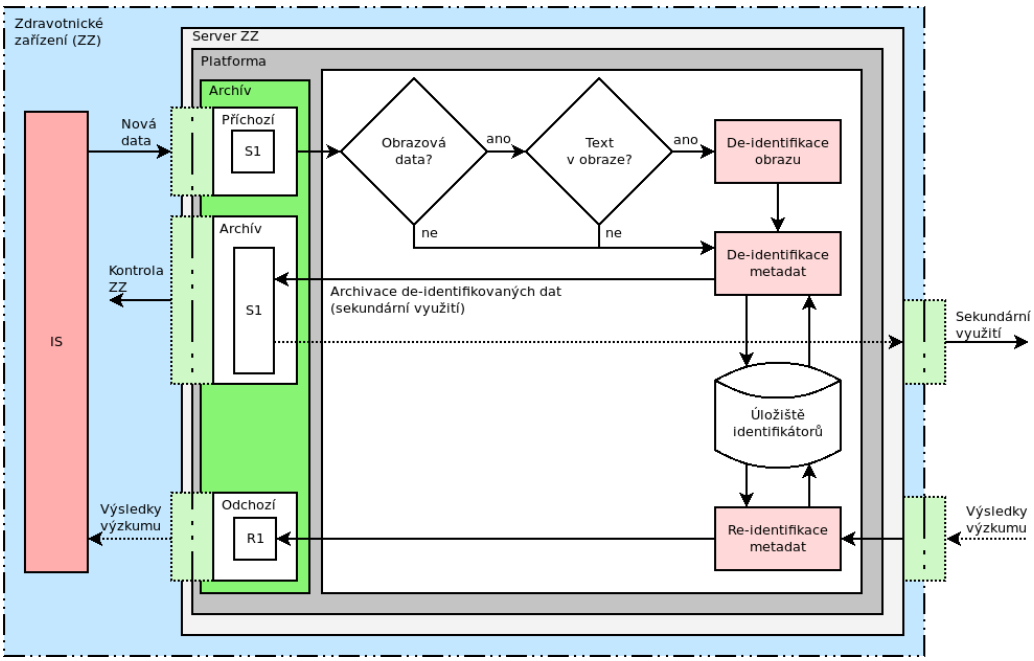
Obrázek 3.1: Proces zpracování nově příchozích dat do MRE platformy [41].

3.2.1 AnonMed

Tato aplikace vznikla v roce 2010 jako neinteraktivní konzolová a pravidly řízená aplikace pro zajištění ochrany osobních údajů. Citlivé údaje ve strukturovaných a semi-strukturovaných datech jsou odstraněny nebo nahrazeny. V závislosti na konfiguraci může sloužit pro tyto účely: nástroj pro akceptaci dat, de-identifikaci, anonymizaci dat a vytváření slepých kopií dat (např. pro klinické studie). Podporováno je množství operací dat s metadaty, ale cílem je odstranění přímých a nepřímých identifikátorů osob v podporovaných formátech.

3.2.2 PureImage

Aplikace *PureImage* slouží ke klasifikaci DICOM snímků na základě vypálených osobních údajů v obrazové informaci. K tomu využívá *adaptivně-iterativní metodu* založenou na optickém rozeznávání snímků. Pokud na ně-



Obrázek 3.2: Proces ochrany osobních údajů [41].

jakém snímků dojde k detekci textového řetězce, tak v další iteraci jsou aplikována přísnější pravidla. Snímky jsou klasifikovány do těchto tříd: čisté bez textu, s textem a s osobními údaji. Lepšího výsledku lze dosáhnout analýzou metadat obsažených v hlavičce DICOM a *PureImage* umí také s těmito daty pracovat. V praxi je vhodné zařadit *PureImage* před *AnonMed* (viz diagram na obr. 3.2). Aplikace má implementovanou i podporu pro odstranění citlivých údajů, které překryje černým obdélníkem. Tento proces de-identifikace však není v produkčním prostředí použit, protože je nutné vyhodnotit kvalitu a přesnost, aby nedošlo ke znehodnocení jiných relevantních textových informací ve snímcích. Dále je nutné posoudit míru poškození zobrazovaných anatomických struktur. Z těchto důvodů je DICOM snímek v karanténě (ve všech případech kromě *čisté bez textu*) až do doby posouzení pověřeným uživatelem [41].

3.3 ETL proces

Po procesu *Ochrana osobních údajů* následuje *Extrakce metadat*. Tento proces již může probíhat mimo zdravotnické zařízení, protože data neobsahují citlivá osobní data [41].

Software *MetaMed* zajišťuje celý ETL proces s využitím ontologií a umí provádět konverze RDF (*Resource Description Format*) dat nebo vykoná-

vat SPARQL (*SPARQL Protocol and RDF Query Language*) dotazy. Aby bylo možné vykonávat dotazy, tak jsou RDF soubory virtualizovány pomocí databáze *OpenLink Virtuoso*.

MetaMed podporuje tyto vstupní formáty pomocí následujících pluginů [41]:

- *MetaMed Extractor Generic* – výchozí plugin pro získání obecných metadat,
- *MetaMed Extractor DASTA* – podporuje formát DASTA ve shodě s *DASTA Ontology* včetně číselníků *DASTA Code-lists*,
- *MetaMed Extractor DICOM* – z DICOM souborů extrahuje metadata uložená v hlavičce,
- *MetaMed Extractor DirectoryStructure* – automaticky rekurzivně generuje hierarchii odpovídající adresářové struktuře od zadaného umístění v úložišti,
- *MetaMed Extractor Miscelaneous* – rozšíření generického modulu *MetaMed Extractor Generic* pro tyto formáty souborů: FLAC, MPEG, EXE, Java, PDF, XML a ZIP,
- *MetaMed Extractor Stroke* – podporuje formát *StrokeXML* a produkuje metadata v shodě se *Stroke Ontology*,
- *MetaMed Extractor Image* – plugin získávající metadata z obrázků ve formátech PNG, JPEG a TIFF. Využívá následující ontologie: *Nepomuk EXIF Ontology* s prefixem `nexif`, *Image Ontology* s prefixem `image` a *Image Ontology Mapping* s prefixem `imm`. Jeho autorem je *Ing. Martin Kryl* z FAV ZČU.

Po úspěšně *Extrakci metadat* je již možné pracovat se zdravotnickými záznamy. Uložená metadata pro každý záznam slouží jako index, resp. jako URI (*Uniform Resource Identifier*) do sdíleného úložiště spravovaného pomocí softwarového nástroje *Samba*[1]. Zde funguje obousměrná komunikace se zdravotnickým zařízením, takže výsledky výzkumu mohou být nahrávány na toto úložiště a lékaři s nimi mohou dále pracovat.

[1]Software umožňující sdílení souborů, tiskáren a dalších zařízení v síti.

3.4 DASTA

Datový standard využívající formátu XML (*Extensible Markup Language*) pro předávání dat mezi informačními systémy zdravotnických zařízení. Vydavatelem standardu DASTA (*Český národní datový standard pro výměnu informací ve zdravotnictví*) je *Ministerstvo zdravotnictví ČR*. Koncepci standardu vytváří pracovní skupina datových standardů nazývající se *Česká společnost zdravotnické informatiky a vědeckých informací* ve spolupráci s ostatními subjekty podílejícími se na tvorbě standardu podle potřeb *Národního zdravotnického informačního systému*. Vývoj standardu nepodléhá jen státním organizacím, ale zapojují se do něj i další veřejné a privátní subjekty, které dodávají informační systémy pro zdravotnické účely [9].

3.4.1 Historický vývoj

Vývoj standardu začal podnětem dodavatelů informačních systémů, kteří do této chvíle museli každé dva informační systémy propojovat individuálně. Dokonce jeden z informačních systému této doby implementoval třicet různých protokolů a číselníků. Tento stav byl dále neudržitelný, a proto v roce 1992 vznikla iniciativa ze strany *Ministerstva zdravotnictví*, IPZV (*Institut postgraduálního vzdělávání ve zdravotnictví*) a již zmiňovaných dodavatelů vytvářejících informační systémy. Prvotním vzorem byla norma EDIFACT (*Electronic Data Interchange for Administration, Commerce and Transport*)[2] nebo např. HL7. Nicméně ani jeden z již existujících standardů nebyl využit a začal vývoj vlastního standardu s orientací na evropské standardy, normy a nomenklatury [3].

První verze standardu nevyužívala žádný formát a vystačila si s prostým textem (TXT soubory). Až od druhé verze vychází standard z formátu XML [3].

Verze DS 1.0 První publikovaná verze nesla název *Datový standard ministerstva zdravotnictví verze 1* a vzniklo pro ni označení *DS 1.0* či *DASTA*. Tato verze určila koncepci standardu, ale byla poměrně jednoduchá a neřešila veškerou problematiku [3].

Verze DS 1.1 V roce 1997 byla vydána druhá verze, která doplnila potřebné datové bloky a komplexní řešení pro komunikaci s laboratorním informačním systémem [3].

[2]Mezinárodní standard pro elektronickou výměnu dat (EDI).

Verze DS 2.01 Velmi zásadní verze, která přinesla velké množství datových bloků a poskytla komplexní řešení pro obousměrnou formalizovanou komunikaci s laboratorním informačním systémem. Je zde využíván jazyk DTD (*Document Type Definition*), který definuje strukturu datového standardu (schéma dokumentu) [3].

Verze DS 3.01 Jedná se o mírně doplněnou verzi *DS 2.01*, která přinesla rozsáhlé datové bloky a číselníky NZIS (*Národní zdravotnický informační systém*). Opět je zde využíván jazyk DTD pro definici schéma datového standardu [3].

Verze DS 4.01 Tato verze přinesla významný pokrok, protože zavedla filozofii klinických událostí, jejich formalizaci a jednoznačnou identifikaci. Dále přinesla další významné datové bloky a k definici schéma datového standardu již nepoužívá DTD dokument [3]. Používá jazyk XSD (*XML Schema Definition*), který navíc podporuje datové typy, jmenné prostory a syntakticky je stejný jako jazyk XML.

3.5 DICOM

DICOM (*Digital Imaging and Communications in Medicine*) je mezinárodní standard pro medicínské snímky a související data o pacientovi. Umožňuje výměnu medicínských obrazových dat mezi informačními systémy v požadované kvalitě pro klinické použití. Standard DICOM je nyní implementován v těchto přístrojích: radiologické, kardiologické zobrazovací a radioterapeutické přístroje jako je CT (*Computed Tomography*), MRI (Magnetic Resonance Imaging), ultrazvuk, rentgen apod. Stále častěji se také objevuje v medicínských oblastech jako je stomatologie a oftalmologie. Díky tomu, že se v lékařském prostředí denně používají snímky k určování diagnóz, tak se tento standard stal jedním ze světově nejrozšířenějších standardů ve zdravotnictví. Je uznán i *Mezinárodní organizací pro normalizaci* jako norma *ISO 12052* [4].

Soubor DICOM obsahuje kromě obrazových snímků i metadata, která obsahují jméno a příjmení pacienta, základní demografické informace, modalitu a účel vyšetření, diagnózu, dávku záření apod. Nemusí obsahovat jen statické snímky, ale dokonce nahrané EKG (*Elektrokardiogram*).

3.5.1 Historický vývoj

V 70. letech 20. století bylo zavedeno CT a další způsoby diagnostického zobrazování, a proto vznikla potřeba vytvoření standardu pro přenos snímků a souvisejících dat o pacientovi mezi různými zařízeními. Do této chvíle většina zařízení používala svůj proprietární standard, a tak se instituce ARC (*American College of Radiology*) a NEMA (*National Electrical Manufacturers Association*) rozhodly pro unifikaci standardu. V roce 1983 vytvořily výbor pro tvorbu standardu, který položil následující myšlenky:

- podpora přenosu snímků (komunikace) bez ohledu na výrobce zařízení,
- usnadnit vývoj a rozšíření systému pro archivaci a komunikaci snímků, které mohou být propojeny s dalšími informačními medicínskými systémy,
- umožnit vytváření databází diagnostických informací s možností širokého sdílení, do kterých lze nahlížet prostřednictvím různých zařízení nezávisle na jejich geografické poloze [24].

ARC/NEMA První standard s názvem *ARC/NEMA 300* byl uvolněn v roce 1985, ale ukázalo se, že není úplně jasný a obsahuje vnitřní rozpory. Druhá verze *ARC/NEMA 2.0* přinesla podporu příkazů pro zobrazovací zařízení a nové schéma k identifikaci obrazu. Přinesla také nové prvky pro zlepšení popisu obrazu [24].

DICOM 3.0 Od této verze je standard již znám jako *DICOM*. Byl zveřejněn v roce 1993 výborem *DICOM Standard Committee* a od tohoto okamžiku je tento standard rozšiřován na základě vznikajících potřeb z lékařského prostředí [24]. Téměř každým rokem je zveřejněna nová verze – v roce 2022 to byla verze *3.22* a takto jsou označeny i předchozí verze, kde číslo za desetinou tečkou určuje rok vydání.

3.6 HL7 CDA

HL7 (*Health Level Seven International*) je organizace založená v roce 1987 zabývající se vývojem nejpoužívanějších akreditovaných standardů ANSI (*American National Standards Institute*). Poskytuje kompletní framework pro výměnu, integraci, sdílení a vyhledávání elektronických zdravotnických služeb. HL7 podporuje více než 1600 členů z více než 50 zemí a více než 500

veřejných a privátních subjektů. HL7 považuje následující standardy za své primární (nejčastěji implementované a používané) [12]:

CDA (*Clinical Document Architecture*) představuje hlavní datový standard pro výměnu informací mezi informačními systémy zdravotnických zařízení [15].

C-CDA (*Consolidated Clinical Document Architecture*) je jediný zdroj pro vyhledávání CDA šablon pro dvanáct typů dokumentů. Umožňuje obchodním analytikům a manažerům získat základní znalosti o jejich používání při použití v různých případech [15].

FHIR (*Fast Healthcare Interoperability Resources*) je standard popisující datové formáty, zdroje a API pro výměnu elektronických dokumentů [15].

CCOW (*Clinical Context Object Workgroup*) definuje standardy pro komunikaci mezi aplikacemi založenými na grafickém uživatelském rozhraní, které běží na zdravotnických pracovních stanicích [5].

Datový standard CDA využívá formátu XML pro výměnu dokumentů mezi zdravotnickými zařízeními a pacienty, resp. pro výměnu mezi informačními systémy zdravotnických zařízení jako u formátu DASTA (viz podkapitola 3.4). Pro zachování maximální interoperability specifikuje strukturu a sémantiku dokumentů a dokument se řídí následujícími šesti charakteristikami: trvalost, správnost, potenciál pro ověřování, kontext, celistvost a srozumitelnost pro člověka. Dokument může obsahovat jakýkoliv typ klinického obsahu. Mezi typické dokumenty obsahující CDA patří obrazová zpráva, příjmová a lékařská zpráva, patologická zpráva, propouštěcí souhrn atd.

V rámci standardu je zaveden standard *přírůstkové sémantické interoperability*. V praxi to znamená, že je povolen určitý rozsah složitosti a na straně implementace je nutné stanovit úroveň shody. Minimální úroveň shody tvoří minimální počet metadat kódovaných v XML (např. identifikátor dokumentu, název poskytovatele, typ dokumentu) a v těle dokumentu poté může být libovolný binární soubor (např. naskenovaný obrazový soubor, PDF nebo Microsoft Word dokument) [21].

3.6.1 Historie

Počátkem roku 1996 se začala scházet skupina lékařů mimo organizaci HL7 a diskutovala o možnostech strukturovaného značení medicínských dokumentů. Nejstarší návrh byl nazván *Kona Architecture* a vznikl v roce 1997,

kdy se skupina lékařů připojila k organizaci HL7. Poté na datovém standardu pracovalo mnoho lidí a byly rozvíjeny základní položené myšlenky společně s rámcem HL7 ve verzi 3 a referenčním informačním modelem. Původní skupina se přejmenovala na *HL7 Structured Documents Work Group* a dnes je zodpovědná za vývoj CDA a dalších typů dokumentů.

Nejnovější CDA (verze 2) nyní slouží jako referenční standard všech dokumentací a v současné době se pracuje na verzi 3 [21].

4 Technologie pro budování Data Lakehouse

V době psaní diplomové práce existovaly tři nejlepší open-source projekty pro budování úložiště *Data Lakehouse*. Jedná se o *Delta Lake*, *Apache Hudi* a *Apache Iceberg* a dále v textu budou zmiňovány jen jako *formáty tabulek*. Každý z projektů implementuje klíčové vlastnosti popsané v podkapitole 2.3, a proto nejsou jednotlivé projekty rozebírány zvlášť. Výběr správného formátu tabulky je zásadním kritériem pro ukládání dat, protože toto rozhodnutí může významně ovlivnit výkon, použitelnost a kompatibilitu systému. Dalším kritériem je kompatibilita formátu s dalšími službami, které poskytují výkonnou sadu nástrojů pro snadný vývoj a správu úložiště [36, 42].

4.1 ACID

ACID je jeden ze základních principů relačních databází, který je vyžadován i u tohoto úložiště. Každý z výše uvedených formátu umí pracovat s transakcemi ACID, ale odlišně pracuje s metadaty.

Delta Lake Sleduje metadata ve dvou typech souborů [36]:

- *Delta logs* – záznamy o změnách v tabulce,
- *checkpoints* – shrnutí všech změn v tabulce do jednoho okamžiku s výjimkou transakcí, které se navzájem ruší.

Podporuje dotazovací jazyk SQL pro vytváření tabulek, vkládání, aktualizace, slučování, mazání záznamů a samozřejmě klasické klauzule pro vyhledávání záznamů podle kritérií [36].

Apache Hudi Organizuje všechny transakce podle různých akcí probíhajících v čase. K tomu využívá adresářový přístup s datovými soubory opatřenými časovými značkami a log soubory sledující změny záznamů v těchto datových souborech. Volitelně je možné zapnout metadatovou tabulku pro optimalizaci dotazů. Ta sleduje seznam souborů, které lze použít pro plánování dotazů namísto souborových operací. S tímto přístupem se lze vyhnout úzkým místům při práci s velkými datasety. *Apache Hudi* podporuje následující SQL klauzule: `CREATE TABLE`, `INSERT`, `UPDATE`, `DELETE` a standardní vyhledávací dotazy [36].

Apache Iceberg Definuje tabulky pomocí tří kategorií metadat:

- *metadatové soubory* – definují tabulku,
- *seznam manifestů* – definují snapshot[1] tabulky,
- *manifesty* – definují skupiny datových souborů, které mohou být součástí jednoho nebo více snímků.

Pomocí těchto vrstev metadat je možné izolovat každý snapshot, podporovat ACID, umožnit optimalizovat dotazy a poskytnout další funkce. Při vykonávání dotazu použije *Apache Iceberg* nejnovější snapshot (pokud není uvedeno jinak) a při zápisu do tabulky vždy vytvoří nový snapshot, který nemá vliv na souběžné dotazy. Ty jsou řešeny pomocí *optimistické souběžnosti*[2], kdy první zápis vytvoří úspěšně nový snapshot a ostatní pokusy jsou opakovány.

4.2 Dělení tabulek – *partition evolution*

Dělení tabulek na menší celky je vhodným nástrojem pro optimalizaci dotazů, protože je sníženo množství prohledávaných dat. Uspořádání dat do logických celků umožňuje efektivní vyhledávání. Schéma rozdělení tabulky se může v průběhu času měnit a ve starších řešeních (např. *Apache Hive*) se jednalo o velmi náročný proces, který vyžadoval přepsání celé tabulky. Pro efektivnější správu tabulek je vhodné mít možnost aktualizovat schéma rozdělení tabulky bez nutnosti přepisování všech předchozích dat – tento postup je známý jako *partition evolution* [36].

Delta Lake Nepodporuje dělení tabulek, ale nabízí alternativu v podobě generovaných sloupců. To umožňuje vytvářet další sloupce, které jsou odvozené z již existujících a potenciálně to může zvýšit efektivitu dotazů, které v projekci zahrnují tyto sloupce. Tuto funkcionalitu *Databricks* umožnil jen v předběžném přístupu a v rámci open-source řešení není plně podporována [36].

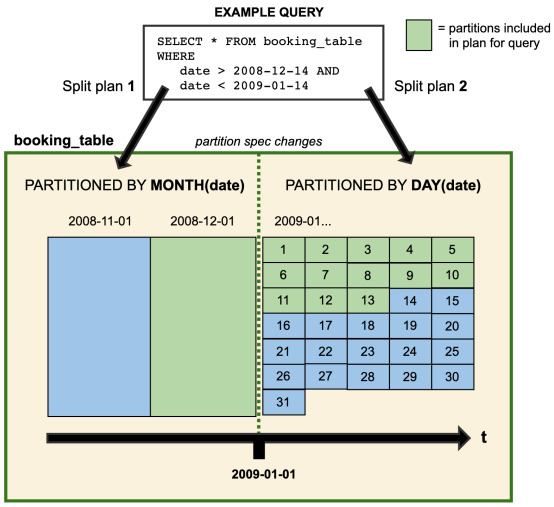
Apache Hudi Nepodporuje dělení tabulek ani skryté rozdělení. Podobného výsledku lze docílit funkcí pro *přeskakování dat*, kdy jsou přeskakovaná

[1] V informatice je to stav systému k danému okamžiku.

[2] Při optimistické souběžnosti nejsou při čtení z databáze uzamykány řádky jako je tomu u *pesimistické souběžnosti*. Pokud má dojít k aktualizaci řádku, tak aplikace musí detekovat, zdali nedošlo ke změně od jeho přečtení a v opačném případě pokus opakovat. Použití je vhodné v systémech s nízkou kolizí dat [11].

nepotřebná data a tím je potenciálně zvýšen výkon při vykonávání dotazu. V současné době je tato funkce podporovaná pouze v režimu optimalizovaném pro čtení [36].

Apache Iceberg V současnosti jediné řešení, které podporuje *partition evolution*. Nespoléhá se na konkrétní hodnoty ve sloupcích, ale sleduje oddíly na základě sloupce oddílu a transformace, která na něj byla aplikovaná (např. transformace časového razítka na den nebo rok). Ve standardním řešení by toto znamenalo vytvoření dalších sloupců a nutnost tyto sloupce někde evidovat, aby bylo známo, že se podle nich může uskutečnit projekce. *Iceberg* toto řeší automaticky, a když jsou data filtrována pomocí SQL podle časového razítka, tak může využít části rozdělené podle roku, ale i podle měsíce. Výhodou je, že stará data zůstávají zachována ve dřívější specifikaci, nová data jsou zapsána pomocí nové specifikace v novém rozložení a metadata jsou pro každou verzi uchovávána samostatně [36]. Na obr. 4.1 je možné vidět, že údaje z roku 2008 jsou rozdělena podle měsíců a až od roku 2009 je tabulka aktualizovaná tak, že jsou údaje rozděleny podle dnů. Dále má *Iceberg* implementovanou speciální funkci *hidden partitioning*, která eliminuje potřebu vytvářet další sloupce rozdělení, které potřebují explicitní filtrování. Stačí jen vykonat dotaz pro výběr potřebných dat a *Iceberg* automaticky přeskočí ta nepotřebná [1]. Toto řešení je vhodné hlavně pro SQL dotazy, které ve filtru neobsahují klíče (sloupce) rozdělených oddílů.



Obrázek 4.1: Rozdělení tabulky pomocí funkce partition evolution [1].

4.3 Vývoj schéma – *schema evolution*

Jedna z nejdůležitějších funkcí, protože v průběhu času se mění data i potřeby společností. Na tyto změny je nutné rychle reagovat a může být nutné přejmenování sloupců, změna datových typů nebo přidávání nových sloupců. Všechny zmíněné projekty podporují vývoj schéma, ale každý na jiné úrovni. Při aktualizaci schéma nemusí být přepisovaná celá tabulka (časově i finančně náročné operace). V rámci *schema evolution* požadujeme následující operace [38]:

- `ADD` – přidání nového sloupce do tabulky nebo vnořené struktury,
- `DROP` – odstranění existujícího sloupce z tabulky nebo vnořené struktury,
- `RENAME` – přejmenování existujícího sloupce nebo pole ve vnořené struktuře,
- `UPDATE` – změna datového typu, struktury, klíče mapy, hodnoty v mapě nebo prvku v seznamu,
- `REORDER` – změna pořadí sloupců nebo polí ve vnořené struktuře.

4.4 Cestování v čase – *time-travel*

Funkce *time-travel* umožňuje vykonávat dotazy nad tabulkou v definovaném časovém okamžiku (snapshot). Každý z těchto snímků obsahuje soubory, které jsou s ním spojeny. Tato funkce je velmi užitečná pro obnovení analýzy, opravu problémů a získání historický dat. Dále může být užitečná pro algoritmy strojového učení, protože mohou být testovány nad stejnou množinou dat jako v předchozích testech. Udržovat velké množství snímků je paměťově náročné, takže je důležité mazat ty staré a nepotřebné [1, 36]. Všechny tři formáty tabulek implementují tuto funkcionalitu a k různým verzím tabulek lze přistupovat i pomocí SQL dotazů.

Delta Lake Umožňuje zobrazit předchozí verze tabulek ve dvou režimech. První z nich je se zaměřením na konkrétní soubory *Delta*, které představují změny provedené v tabulce od předchozí verze souboru. Druhou možností je zobrazení verze tabulky pomocí kontrolních bodů. Ve výchozím stavu uchovává historii tabulky za posledních 30 dní, ale tuto hodnotu lze upravit pomocí nastavení. Pro správu verzí existuje nástroj *Vacuum*, který slouží

pro čištění starých datových souborů, aby bylo možné efektivněji cestovat v čase.

Apache Hudi K zobrazení předchozích verzí tabulek využívá souborového systému. Toho může být docíleno z důvodu, že *Hudi* využívá adresářového přístupu s datovými soubory opatřenými časovými značkami a log soubory sledující změny záznamů v těchto datových souborech, jak již bylo zmíněno v odstavci 4.1. Pro čištění starých verzí se zde používá nástroj *Hoodie Cleaner* [36].

Apache Iceberg V systému se při každé aktualizaci tabulky vytvoří nový snapshot. Pro zobrazení dané verze tabulky stačí zadat ID snímku nebo časové razítko a dotazovat se na data v daném okamžiku. Opět je důležité pravidelně čistit staré a nepotřebné snímky, které jsou zde mazány procedurou `expireSnapshots` [36].

4.5 Podpora otevřených formátu

V tabulce 4.4 jsou uvedeny podporované formáty pro všechny open-source projekty (formáty tabulek) poskytující vrstvu pro *Data Lakehouse*. Všechny řešení podporují dnes nejpoužívanější binární sloupcový formát *Parquet*. Mohou však nastat případy, kdy bude potřeba použít formát ORC nebo AVRO [38].

Formát tabulky	**Podporované formáty**
Delta Lake	Parquet
Apache Hudi	Parquet, ORC
Apache Iceberg	Parquet, ORC, AVRO

Tabulka 4.1: Tabulka podporovaných otevřených formátů [38].

4.5.1 Apache Parquet

Apache Parquet je open-source sloupcově orientovaný formát datových souborů pro efektivní ukládání a získávání dat. Poskytuje účinnou kompresi dat a schémata kódování se zvýšeným výkonem pro hromadné zpracování složitých dat. Je také vhodný pro dávkové i interaktivní úlohy. Podobá se

jiným sloupcovým formátům dostupných v úložišti *Hadoop*[3] (např. RCFile a ORC) [17].

Struktura Svobodný, otevřený a jazykově nezávislý formát. Soubory jsou uspořádány *podle sloupců* a nikoliv *podle řádků*, jak je vidět na ukázce kódu 4.1. `N` sloupců je rozděleno do `M` skupin řádků – je to způsob organizace dat, kdy jsou jednotlivé hodnoty ukládány do sloupců namísto řádků. Metadata souboru obsahují reference všech počátečních metadat sloupců a jsou zapisovány až pod datech. V případě čtení jsou nejdříve přečtena metadata a následně načteny jen relevantní oddíly sloupců. Formát je navržen tak, aby odděloval metadata od dat. To umožňuje rozdělení sloupců do více souborů a také je možné, aby jeden soubor obsahující metadata odkazoval na více souborů *Parquet*. Při dotazování ve sloupcovém úložišti umí velmi rychle přeskočit nerelevantní data a díky tomu umožňuje efektivnější provádění operací, které pracují s celými sloupci jako např. *agregace* nebo *filtrování*, protože nevyžadují přístup ke všem datům v tabulce. V rámci formátu jsou dále podporovány vnořené datové struktury a je optimalizován pro velké objemy dat v řádu gigabajtů pro každý soubor [10, 17].

```
4-byte magic number "PAR1"
<Column 1 Chunk 1 + Column Metadata>
<Column 2 Chunk 1 + Column Metadata>
...
<Column N Chunk 1 + Column Metadata>
<Column 1 Chunk 2 + Column Metadata>
<Column 2 Chunk 2 + Column Metadata>
...
<Column N Chunk 2 + Column Metadata>
...
<Column 1 Chunk M + Column Metadata>
<Column 2 Chunk M + Column Metadata>
...
<Column N Chunk M + Column Metadata>
File Metadata
4-byte length in bytes of file metadata
4-byte magic number "PAR1"
```

Listing 4.1: Struktura uložení dat [10].

Porovnání s CSV formátem CSV je jednoduchý souborový formát určený pro výměnu tabulkových dat. Oproti formátu *Parquet* jsou zde data

[3]Open-source framework, který umožňuje distribuované zpracování velkých datových sad.

uložena do řádků a hodnoty odděleny čárkou. Používá se v mnoha nástrojích jako je Excel, Google Sheets apod. Již v odstavci 4.5.1 bylo zmíněno, že řádkové rozložení dat není efektivní a společnosti poskytující datové služby tento aspekt zohledňují. Používání CSV formátu je zpoplatněno více než používání sloupcových formátu jako je *Parquet* nebo ORC. V tabulce 4.2 lze vidět, že používání formátu *Parquet* snižuje nároky na úložiště o 87 %, vykonávání dotazů nad datasetem je 34× rychlejší a celková cena za datové služby je výrazně nižší.

Dataset	**Velikost na Amazon S3**	**Doba běhu dotazu**	**Cena**
Data uložená v CSV	1 TB	236 s	5,75 $
Data uložená v Parquet	130 GB	6,78 s	0,01 $

Tabulka 4.2: Srovnání používání formátu Parquet s CSV [17].

4.6 Kompatibilita

Samotné úložiště *Data Lakehouse* nemá bez dalších nástrojů téměř žádné využití. Aby implementované úložiště mohlo plnit svůj účel, tak tvoří ekosystém s dalším potřebným software. V tabulce 4.4 jsou uvedeny nejoblíbenější nástroje pro datovou analýzu a jejich vzájemná kompatibilita se zmiňovanými formáty tabulek. Při zhodnocování kompatibility je důležité vzít v potaz, které I/O operace bude nástroj vykonávat. V tabulce 4.4 je znázorněno, že některé nástroje jsou optimalizovány jen pro *zápis* nebo *čtení*. *Apache Hudi* vyvinulo velké úsilí, aby podporovalo velké množství nástrojů. V hodnocení kompatibility je nutné zvážit hledisko, že všechny formáty tabulek běží na platformách poskytující komplexní služby a ty mohou nabízet svá proprietární řešení.

4.7 Komunita

Dalším důležitým aspektem při výběru projektu je komunita kolem projektu. Ta rozhoduje o dynamice vývoje, přijetí ekosystému nebo objektivitě platformy.

Prvním hlediskem mohou být *Github stars* – když uživatel označí repozitář hvězdou, tak si ho přidá na seznam oblíbených a dostává aktualizace o aktivitě v repozitáři. Počet hvězd může být hrubým indikátorem popularity. *Delta Lake* vede v povědomí a popularitě. *Apache Hudi* a *Apache Iceberg* jsou na tom srovnatelně.

Dalším podobným hlediskem může být *Github Watchers*, kdy sledující uživatelé opět dostávají informace o změnách v repozitáři. Jedná se o zainteresované uživatele, které zajímají průběžné informace o projektu a ve srovnání s *Github Stars* se nejedná jen o projev zájmu o projekt. Tento ukazatel má proto větší váhu a na prvním místě se umístil *Apache Hudi*, na druhém místě *Apache Iceberg* a na třetím místě s těsným rozdílem *Delta Lake*.

Stejné pořadí kopíruje i počet *Github Forks*. Jedná se o počet uživatelů, kteří si vytvořili kopii repozitáře daného projektu za účelem integrace do vlastního projektu. V tomto čísle jsou zahrnuty i kopie autorů, kteří přispívají svými úpravami do oficiálních repozitářů. Samostatně je toto číslo vyjádřeno jako poslední řádka tabulky 4.3, kde opět vyhrává *Apache Hudi* s téměř 90 unikátními přispěvateli.

	Delta Lake	**Apache Hudi**	**Apache Iceberg**
Github Stars	5 600	3 800	3 700
Github Watchers	213	1 200	144
Forks	1 300	1 700	1 400
Github Contributors	26	86	39

Tabulka 4.3: Srovnání formátu v rámci komunity [42].

4.8 Výběr vhodného řešení

Všechny tři formáty tabulek mají své výhody/nevýhody a výběr správného formátu je zásadní pro výkonnost a údržbu úložiště. Výběr konkrétního řešení vždy záleží na poskytovali (např. AWS, Databricks, Gooogle), na požadovaných funkcích a potřebných datových analytických nástrojích.

V rámci diplomové práce byl výběr formátu závislý na následujících aspektech:

- *koncept Data Lakehouse* – musí splňovat základní koncepty úložiště,
- *otevřenost projektu* – nezávislý projekt s otevřeným kódem, aby implementace, následné rozšiřovaní a používání nebylo limitující,
- *kvalitní dokumentace* – detailní popis implementace je velice důležitý, protože se jedná o novou problematiku a zatím není velké množství volně dostupných projektů, které by demonstrovaly implementaci a používání,

- *programovací jazyk* – preferována je *Java*, protože medicínské projekty na FAV ZČU jsou vyvíjeny právě v tomto jazyce.

V rámci diplomové práce byl pro implementaci vybrán formát *Delta Lake* provozovaný na platformě od společnosti *Databricks*, která popsala základní koncepty úložiště. Jedná se o nezávislý projekt s otevřeným kódem, který není pod kontrolou jedné společnosti a kromě *Databricks* se k projektu připojila řada velkých společností (např. Adobe, Atlassian, Apple, Amazon, Microsoft, ...). Aby projekt zdůraznil svou nezávislost, tak se ještě v roce 2019 připojil k organizaci *Linux Foundation*. K projektu existuje detailní dokumentace popisující implementaci hned v několika programovacích jazycích. Dokumentace dále obsahuje správu tabulek doplněnou o příklady, ukázky SQL dotazů a mnoho dalšího.

	Delta Lake	**Apache Hudi**	**Apache Iceberg**
Apache Spark	čtení/zápis	čtení/zápis	čtení/zápis
Apache Flink	čtení/zápis	čtení/zápis	čtení/zápis
Presto	čtení	čtení	čtení/zápis
Trino	čtení/zápis	čtení	čtení/zápis
Hive	čtení	čtení	čtení/zápis
DBT	čtení/zápis	čtení/zápis	nepodporuje
Kafka Connect	proprietární řešení	zápis	nepodporuje
Kafka	zápis	zápis	nepodporuje
Pulsar	zápis	zápis	zápis
Debezium	zápis	zápis	zápis
Kyuubi	nepodporuje	čtení/zápis	čtení/zápis
ClickHouse	čtení	čtení	nepodporuje
Apache Impala	nepodporuje	čtení/zápis	čtení/zápis
AWS Athena	čtení	čtení	čtení/zápis
AWS EMR	čtení/zápis	čtení/zápis	čtení/zápis
AWS Redshift	čtení	čtení	nepodporuje
AWS Glue	čtení/zápis	čtení/zápis	čtení/zápis
Goole BigQuery	nepodporuje	čtení	čtení
Goole DataProc	čtení/zápis	čtení/zápis	čtení/zápis
Azure Synapse	čtení/zápis	čtení/zápis	nepodporuje
Azure HDInsight	čtení/zápis	čtení/zápis	nepodporuje
Databricks	čtení/zápis	čtení/zápis	čtení/zápis
Snowflake	čtení	nepodporuje	čtení/zápis
Vertica	čtení	čtení	nepodporuje
Apache Doris	nepodporuje	čtení	čtení
Starrocks	v přípravě	čtení	čtení
Dremio	čtení s omezeními	nepodporuje	čtení/zápis s omezeními

Tabulka 4.4: Tabulka zobrazující kompatibilitu s s dalšími nástroji [38].

5 Implementace

V kapitole 3 bylo popsáno, že v současné době existuje *MRE platforma* zpracovávající medicínská heterogenní data. V rámci této platformy musely být řešeny problémy, které nové úložiště *Data Lakehouse* implicitně pokrývá. Jednou z hlavních věcí je extrakce metadat (viz podkapitola 3.3), kdy jsou v rámci ETL procesu extrahována relevantní metadata, která jsou následně ukládána do grafové databáze v podobě RDF souborů. Databáze poté zajišťuje meta-transakční vrstvu (resp. transakční zpracování ACID), protože uložená metadata slouží jako index do sdíleného úložiště. Medicínské záznamy jsou zde uloženy v jejich nativní podobě a lze k nim přistupovat pomocí dotazovacího jazyka SPARQL.

Implementací úložiště *Data Lakehouse* by měly být odstraněny následující problémy, kterým současná platforma musí čelit:

Extrakce metadat je poměrně náročný proces citlivý na kvalitu a konzistenci dat. V současném řešení jsou extrahována jen data, která jsou definována v konfiguraci systému. V praxi to znamená, že např. z medicínského záznamu DASTA (soubor v XML formátu) jsou extrahovány jen definované prvky nebo atributy. Pokud dojde ke změně schématu medicínských záznamů určitého standardu, tak je nutné změnit statickou konfiguraci systému.

Transakční zpracování je důležitou vlastností, kterou známe z klasických relačních databází. Pokud k úložišti přistupuje více uživatelů zároveň, tak je téměř nemožné udržet konzistentní stav. Požadujeme, aby každá transakce měla ACID vlastnosti, a proto je v současném řešení nutné budovat vlastní meta-transakční vrstvu.

SPARQL je dotazovací jazyk pro data (metadata) uchovávána v RDF souborech. Oproti dotazovacímu jazyku SQL má nějaké limity a pro uživatele není tolik přirozený v porovnání s SQL. To má za následek, že některé operaci nejsou možné nebo sestavení dotazu je příliš složité.

Díky výše zmíněným úskalím se přistoupilo k implementaci nového řešení, které využívá úložiště *Delta Lakehouse*. Nově vzniklý software je *Java* aplikace využívající *Spring Boot* framework a pro zajištění vrstvy úložiště *Delta Lakehouse* je využit open-source projekt *Delta Lake*, který byl vybrán z důvodů popsaných v podkapitole 4.8.

5.1 Použité technologie

Veškeré použité technologie byly vybrány v návaznosti na současné řešení, aby bylo potenciálně možné nový software implementovat do *MRE platformy*. Dále byl kladen důraz na možnost dalšího rozšíření v budoucnu, životaschopnost technologií v následujících letech a v neposlední řadě na kvalitu poskytovaných dokumentací.

Pro *back end* byly vybrány následující technologie:

- *Delta Lake 3.3.0* – open-source framework umožňující budování úložiště *Delta Lakehouse* a využívající formát *Delta* pro tabulky (dále v textu uváděno jen jako *Delta tabulky*),
- *Apache Spark 3.3.0* – open-source framework pro distribuované zpracování velkých datových souborů poskytující velké množství analytických nástrojů (v projektu použit modul *Spark SQL* a *Spark XML*),
- *Java 11* – vysokoúrovňový, objektově orientovaný programovací jazyk,
- *Project Lombok* – *Java* knihovna, která automaticky generuje bloky kódu nebo metody (např. gettery a settery) v rámci *Java bytecode* do zkompilovaných `.class` souborů na základě přidaných anotací,
- *Spring Boot 2.7.6* – *Java* framework, který je nadstavbou standardního *Spring* frameworku, ale má navíc servlet kontejner *Tomcat* (odpadá nutnost konfigurace webového serveru) a podporuje návrhový vzor *Dependency Injection*,
- *Spring Data JPA 2.7.6* – jedná se o sadu rozhraní umožňující přistupovat a uchovávat relační data mezi *Java aplikací* a *relační databází* za pomocí ORM (*Object Relational Mapping*),
- *MySQL (MariaDB)* – open-source relační databázový systém,
- *OpenAPI 3.0* – specifikuje API, aby bylo možné integrovat nově vzniklou aplikaci do dalších služeb (v projektu využita jen vizualizace API v rámci modulu *SpringDoc OpenAPI UI*),
- *BaseX 9.0* – open-source databázový systém s dotazovacím jazykem XQuery, který je navržen pro ukládání, správu a vykonávání dotazů nad velkým množstvím dat uložených v XML formátu,
- *Další Java knihovny* – kromě výše zmíněných technologií byly použity i tyto Java knihovny: *MySQL Connector Java*, *Commons Lang*, *Commons FileUpload* a další uvedené v souboru `pom.xml`.

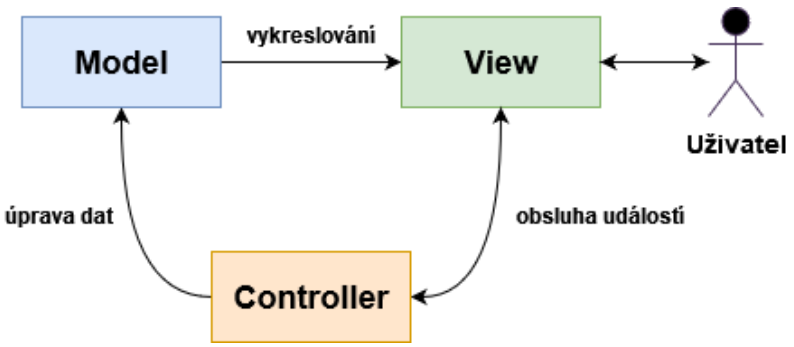
Z pohledu *front endu* byly použity následující technologie:

- *Thymeleaf 3.0.11* – šablonovací nástroj běžící na straně serveru, který generuje HTML (*Hypertext Markup Language*) zobrazitelné ve všech moderních prohlížečích (syntaxe šablony téměř srovnatelná s HTML),
- *Bootstrap 5.2.3* – open-source framework poskytující mnoho užitečných webových prvků a komponent (např. formuláře, menu, tlačítka atd.) a rozsáhlý gridový systém umožňující vývoj responzivních layoutů,
- *Font Awesome 6.2.0* – sada piktogramů použitelná ve webových aplikacích a frameworku *Bootstrap*,
- *jQuery 3.6.1* – javascriptová knihovna kladoucí důraz na interakci mezi Javascriptem a HTML a je kompatibilní se všemi moderními prohlížeči.

5.2 Architektura aplikace

Nově vzniklá aplikace splňuje vlastnosti třívrstvé architektury nazývané MVC (*Model-View-Controller*). Tato architektura dělí aplikaci na tři logické části, aby každá z nich mohla být upravována samostatně bez dopadu na ostatní. Jedná se o následující vrstvy [30]:

- *Model* – vrstva pracující s daty/databází a součástí je i business logika aplikace,
- *View* – za pomocí *Thymeleaf* šablon zobrazuje (reprezentuje) data z vrstvy *Modelu*,
- *Controller* – obsluhuje události vzniklé na vrstvě *View*.



Obrázek 5.1: Komunikace vrstev architektury.

Na obr. 5.1 lze vidět komunikaci jednotlivých vrstev architektury. Pokud uživatel přistoupí skrze URL (*Uniform Resource Locator*) na určitou sekci v administraci, tak vrstva *View* vykreslí *Thymeleaf* šablonu zobrazující data z vrstvy *Model*. Uživatel může v rámci šablony vykonávat různé akce, které mohou měnit i datový model aplikace. Nově vzniklé události jsou obsluhovány vrstvou *Controller*, která modifikuje nebo přidává nová data obsluhována vrstvou *Model*. Tato datová vrstva je v aplikace zastoupena pomocí jednotlivých služeb, které pracují primárně s databázovou vrstvou skrze JPA zajišťující objektově relační mapování mezi Java aplikací a relační databází.

5.2.1 Modulárnost aplikace

Aplikace je logicky rozdělená na dvě části. První část tvoří *administrace*, která poskytuje kompletní správu uživatelů a obsahuje sekce zajišťující správu úložiště *Data Lakehouse*. Druhou část tvoří *úložiště* a skrze REST API poskytuje veškeré funkce pro jeho správu. Tyto dvě části jsou zatím součástí jedné aplikace, aby bylo jednodušší nasazení v testovacím prostředí. Dále to zjednodušuje testování a rozšiřitelnost aplikace. V budoucnu však není problém tyto části od sebe oddělit a nasadit v odlišných prostředích. Již při návrhu aplikace se počítalo s budoucí separací těchto částí, takže jsou oddělené služby sloužící pro část *administrace* a *úložiště*. Všechny *Controllery* v *administrační části* jsou pro tento případ také připraveny, a proto využívají REST klient, který přistupuje k části *úložiště* skrze API namísto služeb. Tímto přístupem je simulováno reálné chování po rozdělení aplikace a snaha eliminovat potenciální chyby.

Administrace První část poskytuje kompletní správu uživatelů. Pokud uživatel přistoupí do administrační sekce, tak se může *přihlásit* nebo *registrovat* jako nový uživatel. V případě *zapomenutí hesla* jej může obnovit a po zadání správné odpovědi zadat nové heslo. Změnu hesla nebo údaje libovolného uživatele může po úspěšném přihlášení provést také uživatel s administrátorskými právy. Všechny tyto uživatelské operace poskytuje i skrze REST API (viz obr. 5.2). K rozhraní přísluší jeden transportní objekt s názvem `UserDTO`. Administrace obsahuje následující sekce pro správu úložiště:

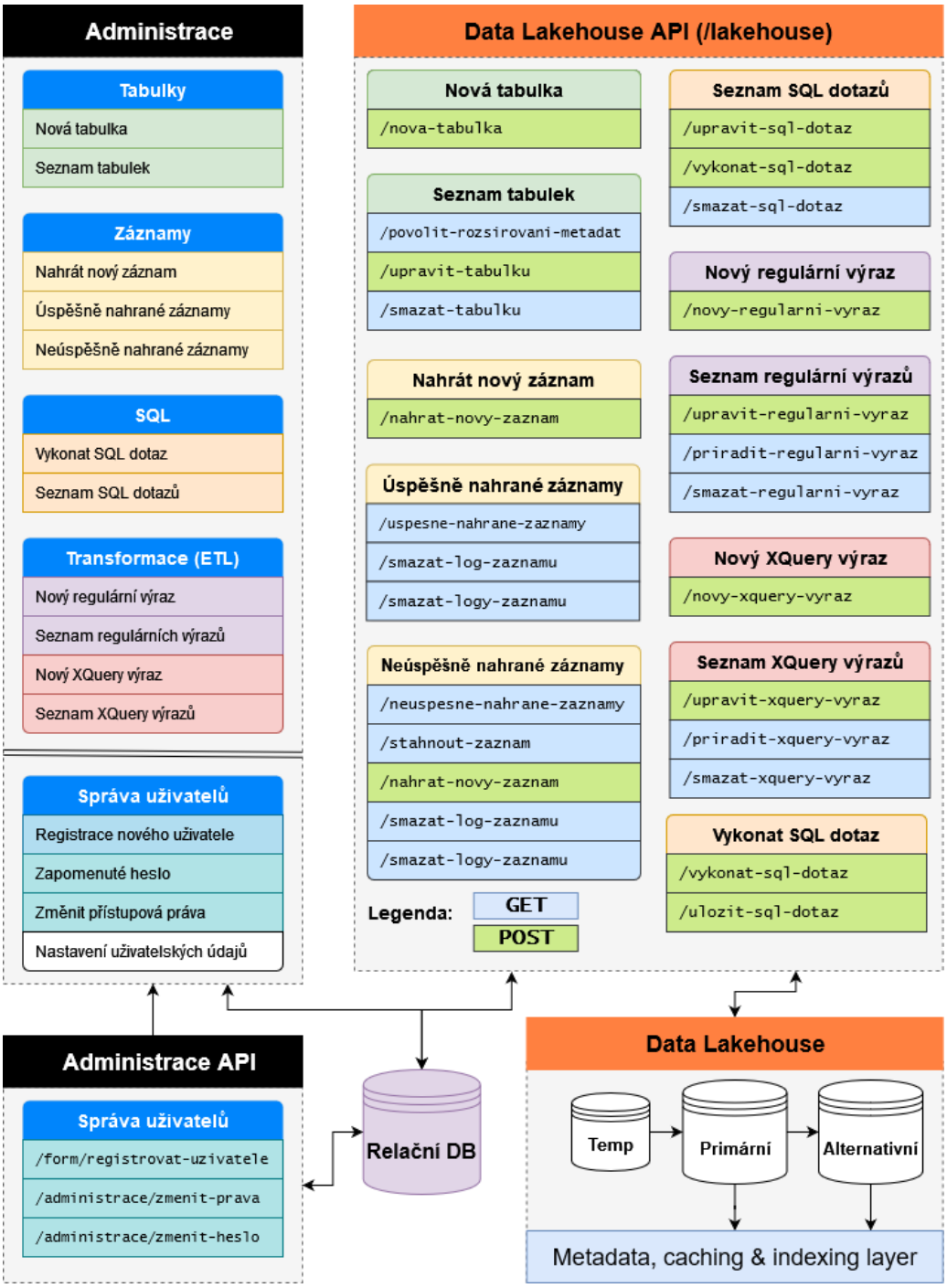
- *Nová tabulka* – založení nové tabulky typu *Delta*,
- *Seznam tabulek* – kompletní seznam existujících tabulek s možností danou tabulku upravit nebo ji smazat,
- *Nahrát nový záznam* – nahrání nového medicínského záznamu do úložiště,

- *Úspěšně nahrané záznamy* – seznam úspěšně nahraných medicínských záznamů,
- *Neúspěšně nahrané záznamy* – fronta neúspěšně nahraných záznamů poskytující možnosti pro opětovné nahraní záznamu, aby se úspěšně uložil do úložiště,
- *Vykonat SQL dotaz* – vykonávání SQL dotazů nad úložištěm s možností zadání dodatečných parametrů pro ovlivnění výsledného pohledu,
- *Seznam SQL dotazů* – seznam SQL dotazů s možností jejich úpravy nebo opětovného vykonání,
- *Nový regulární výraz* – vytvoření nového regulárního výrazu pro ETL proces (nahrávání nových záznamů),
- *Seznam regulární výrazů* – seznam regulárních výrazů s možností daný výraz upravit nebo jej smazat,
- *Nový XQuery výraz* – vytvoření nového XQuery výrazu pro ETL proces (nahrávání nových záznamů),
- *Seznam XQuery výrazů* – seznam XQuery výrazů s možností daný výraz upravit nebo jej smazat.

Úložiště Druhou část tvoří úložiště *Data Lakehouse*, které poskytuje veškeré funkce pro jeho správu. K jeho správě poskytuje REST API, které obsahuje následující endpointy:

- `nova-tabulka` – založení nové tabulku typu *Delta*,
- `povolit-rozsirovani-metadat` – povolení rozšiřování metadat (schéma) tabulky na základě nahraných záznamů,
- `upravit-tabulku` – úprava tabulky typu *Delta*,
- `smazat-tabulku` – smazání tabulky typu *Delta*,
- `nahrat-novy-zaznam` – nahrání nového medicínského záznamu do úložiště,
- `uspesne-nahrane-zaznamy` – vrací mapu s logy o úspěšně nahraných záznamech,
- `neuspesne-nahrane-zaznamy` – vrací mapu s logy o neúspěšně nahraných záznamech,

- `smazat-log-zaznamu` – smazání logu o úspěšném/neúspěšném nahrání záznamu,
- `smazat-logy-zaznamu` –hromadné smazání logů o úspěšném/neúspěšném nahrání záznamů,
- `stahnout-zaznam` – stažení neúspěšně nahraného medicínského záznamu do úložiště (např. nevalidní schéma),
- `vykonat-sql-dotaz` – vykonání SQL dotazu ovlivněného zadanými parametry,
- `ulozit-sql-dotaz` – uložení nového SQL dotazu,
- `upravit-sql-dotaz` – úprava již existujícího SQL dotazu,
- `smazat-sql-dotaz` – smazání již existujícího SQL dotazu,
- `novy-regularni-vyraz` – vytvoření nového regulárního výrazu pro ETL proces (nahrávání nových záznamů),
- `upravit-regularni-vyraz` – úprava regulárního výrazu,
- `priradit-regularni-vyraz` –přiřazení regulárního výrazu k dané *Delta* tabulce,
- `smazat-regularni-vyraz` – smazání regulárního výrazu,
- `novy-xquery-vyraz` – vytvoření nového XQuery výrazu pro ETL proces (nahrávání nových záznamů),
- `upravit-xquery-vyraz` – úprava XQuery výrazu,
- `priradit-xquery-vyraz` – přiřazení XQuery výrazu k dané *Delta* tabulce,
- `smazat-xquery-vyraz` – smazání XQuery výrazu.



Obrázek 5.2: Schéma architektury aplikace.

Na obr. 5.2 lze vidět schéma aplikace z pohledu API. Po úspěšné autentizaci jsou uživateli zobrazeny (na základě autorizace) sekce popsané v odstavci 5.2.1, které jsou sdruženy do těchto logických celků: *Tabulky*, *Záznamy*, *SQL*, *Transformace (ETL)* a *Správa uživatelů*. Každá z těchto sekcí

má vlastní *View* a *Controller*, který využívá `REST client` k volání jednotlivých endpointů na straně úložiště (Data Lakehouse API). Endpointy jsou sdruženy podle názvu sekcí a barevné pozadí názvu sekce na straně administrace odpovídá barevnému pozadí názvu na straně úložiště. Barevně odlišeny (viz legenda na obr. 5.2) jsou i HTTP (*Hypertext Transfer Protocol*) metody GET a POST pro posílání dat mezi klientem serverem – API je určeno primárně pro komunikaci s úložištěm *Data Lakehouse*. Pro správu úložiště je nutná i relační databáze (sloužící jako datový slovník), která uchovává informace o Delta tabulkách, logy o nahraných záznamech, informace o uživatelích atd. I administrační část má své API (Administrace API), ale zahrnuje jen část funkcí pro správu uživatelů.

5.3 Moduly

Již v podkapitole 5.2.1 bylo zmíněno, že je aplikace rozdělena na dva velké celky (administrace a úložiště), které obsahují moduly pro správu úložiště a uživatelů. Každý modul zahrnuje implementaci, administrační sekci a příslušné API endpointy. Tyto moduly jsou nezávislé (díky architektuře MVC), takže mohou být modifikovány bez dopadu na ostatní a z pohledu autorizace mohou být některé nepřístupné pro uživatele s nedostatečnými právy. Všechny níže uvedené případy jsou popsány primárně s použitím administrace, ale samozřejmě je možná i varianta jen s použitím API.

5.3.1 Založení nové tabulky

Po inicializaci aplikace je založení nové tabulky prvním krokem, který uživatel musí udělat. Pokud je autentizace úspěšná, tak může vstoupit do sekce *Nová tabulka*, která je dostupná všem uživatelům nezávisle na přidělených právech. Následně je nutné vyplnit povinná vstupní pole. Jako první je *Název* sloužící jen pro administrativní účely a *Název tabulky* odpovídající názvu v úložišti *Data Lakehouse*. Pole *Root tag* označuje nejvyšší úroveň (tag), podle které bude rozeznáván vstupní XML soubor a pole *Popis tabulky* pro popis potenciálních dat v tabulce.

Dále je důležité vybrat XSD soubor obsahující strukturu metadat (schéma) tabulky. Pomocí tohoto souboru dojde k založení tabulky včetně zachování všech struktur. Dále je možné zvolit možnost *Povolit rozšiřování metadat tabulky na základě vstupních dat*, kdy se schéma tabulky rozšiřuje na základě vstupních dat – tímto způsobem se tabulka „učí“ a struktura metadat se rozšiřuje. Možná je i kombinace obou způsobů a tím je zajištěna větší kontrola nad daty. Tabulka se nejdříve vytvoří podle definovaného schéma,

které musí vstupní data splňovat. Pokud některý medicínský záznam obsahuje metadata nad rámec definovaného schéma, tak dochází k rozšiřování struktury metadat tabulky.

Posledním důležitým aspektem je volitelný výběr *Alternativní tabulky*. V případě, že medicínský záznam nemůže být nahrán (např. některá metadata mají rozdílný datový typ) do primární tabulky (právě zakládaná tabulka), tak je aplikace zkusí nahrát do alternativní tabulky. Pokud z nějakých důvodů nemůže být nahrán ani do alternativní tabulky, tak je uložena standardně na disk do adresáře `upload`. Tento proces je znázorněn na obr. 5.2. Neúspěšně uložený záznam si uživatel může stáhnout, modifikovat jej a následně zkusit znovu nahrát.

Komunikace s úložištěm Pro založení nové tabulky je potřeba vytvořit požadavek, který má dva parametry. Prvním z nich transportní objekt `LakehouseTableDTO` obsahující veškeré informace o nově zakládané tabulce a druhý s názvem `xsdFile` obsahuje XSD schéma (viz ukázka kódu 5.1). Po odeslání požadavku je volána metoda `createTable(...)` a pokud je tabulka úspěšně založena, tak přijde odpověď v podobě transportního objektu `CommonResponseDTO` s parametrem `success = true`. V tomto případě je tabulka vytvořena v úložišti *Data Lakehouse* a paralelně jsou uloženy informace o tabulce do tabulky `lakehouse_table` v relační databázi. Uložení dat do relační databáze slouží jen pro správu tabulek, ale na úložiště *Data Lakehouse* nemá žádný vliv.

```
# API endpoint: ../lakehouse/nova-tabulka
# HTTP metoda: POST

{
  "lakehouseTableDTO": {
    "id": 0,
    "title": "string",
    "tableName": "string",
    "rootTag": "string",
    "description": "string",
    "alternativeTableName": "string",
    "isEnableMetadataExtension": true
  },
  "xsdFile": "string"
}
```

Listing 5.1: Požadavek pro založené nové tabulky.

```
{
  "success": true,
  "message": "string",
  "result": "string",
  "json": "string"
}
```

Listing 5.2: Obecná odpověď pro většinu REST volání.

5.3.2 Správa tabulek

Správa tabulek je v administraci dostupná každému uživateli pod sekcí s názvem *Seznam tabulek*. Po vstupu do sekce jsou vypsány všechny existující tabulky. První možností je *povolení rozšiřování metadat*. Dále uživatel může přistoupit k úpravě tabulky nebo ji smazat. Při procesu mazání dochází k odstranění tabulky z úložiště *Data Lakehouse* i z relační databáze, a to z tabulky `lakehouse_table`.

Komunikace s úložištěm Při *povolení rozšiřování metadata* je volán endpoint se dvěma parametry v URL (`string nazevTabulky` a `boolean povolit`) a následně metoda `allowMetadataExtension(...)`. Pokud uživatel přistoupí k úpravě tabulky, tak se zobrazí *View*, které je velmi podobné jako při zakládání nové tabulky. Nelze však upravovat *Název tabulky*, protože ten je vázán na úložiště *Data Lakehouse*. Dále není možné nahrát soubor s XSD schéma, protože tabulka je již inicializována. Požadavek je téměř totožný (viz ukázka kódu 5.1) jako v případě zakládání nové tabulky, ale chybí zde parametr `xsdFile`. Po odeslání tohoto požadavku je volána metoda `updateTable(...)`. Poslední možností je smazání tabulky. Pro tento případ je volán endpoint s jedním parametrem (`string nazevTabulky`) v URL a následně metoda `deleteTable(...)`. Tato metoda označí danou tabulku jako *neaktivní* v relační databázi a smaže ji ze souborového systému, resp. z úložiště *Data Lakehouse*.

5.3.3 Nahrání nového záznamu

Po úspěšném založení nezbytných tabulek může začít docházet k nahrávání medicínských záznamů. Po vstupu do sekce uživatel vybere jeden nebo více záznamů ze souborového systému. Následně může manuálně nastavit název cílové tabulky nebo zvolit možnost *Automaticky rozeznat záznam a uložit ho do příslušné tabulky*. V případě zvolení první možnosti je nutné, aby všechny záznamy byly stejného typu. U druhé možnosti může jít o heterogenní data,

protože každý záznam je identifikován a ukládán do příslušné tabulky. Pro správnou identifikaci je nutné, aby při zakládání tabulky bylo správně vyplněno pole s názvem *Root tag*, podle kterého je záznam rozeznáván.

Komunikace s úložištěm Po výběru záznamů a volbě dalších parametrů pro nahrání se vytvoří požadavek obsahující záznamy a transportní objekt `RecordDTO` (viz ukázka kódu 5.3), jehož součástí jsou uživatelem zvolené parametry. Po odeslání požadavku je volána metoda `uploadNewRecord(...)`, která v případě nutnosti rozezná cílovou Delta tabulku u každého záznamu. Následně jsou v rámci tohoto ETL procesu aplikovány *regulární* a *XQuery výrazy* na daný záznam a po těchto transformacích je záznam uložen na dočasné konvenční úložiště. Poté se z relační databáze načtou veškeré informace o Delta tabulce, záznam se převede do sloupcové struktury `DataFrame` a následuje pokus o uložení do této tabulky. V ideálním případě dojde k uložení záznamu v otevřeném sloupcovém formátu *Parquet* do úložiště *Data Lakehouse*. Pokud nedojde k uložení (např. záznam neodpovídá schématu tabulky), tak se načtou informace o alternativní Delta tabulce a následuje pokus o uložení na toto místo. Může nastat případ, že ani ukládání do alternativní tabulky se nezdaří a záznam zůstává uložen v konvenčním úložišti, aby jej bylo možné modifikovat a zkusit nahrát znovu. O každém nahrávaném záznamu se vytváří log v relační databázi, takže po dokončení lze vidět, zdali se záznam uložil, popř. neuložil a kvůli jaké chybě. Úspěšné a neúspěšné záznamy jsou poté viditelné v příslušných sekcích.

```
# API endpoint: ../lakehouse/nahrat-novy-zaznam
# HTTP metoda: POST

{
  "recordDTO": {
    "tableName": "string",
    "isEnableAutomaticRecognition": true
  },
  "files": "string"
}
```

Listing 5.3: Požadavek pro nahrání záznamu.

5.3.4 Úspěšně nahrané záznamy

Tato sekce poskytuje výpis všech úspěšně nahraných záznamů. Pro větší přehlednost jsou řazeny podle dávek, jak byly nahrávány. Každá dávka je opatřena časovým razítkem a po rozbalení jsou viditelné jednotlivé záznamy, které mohou být smazány. Jedná se pouze o logy v relační databázi, takže

jejich smazání nemá žádný dopad na úložiště *Data Lakehouse*. Pokud má uživatel přidělena administrátorská práva, tak může využít hromadného mazání a jedním stiskem smazat všechny logy o úspěšně nahraných záznamech.

Komunikace s úložištěm Po zaslání bezparametrického HTTP GET požadavku na URL `../lakehouse/uspesne-nahrane-zaznamy` obdržíme všechny logy o úspěšně nahraných záznamech ve formě mapy (`Map<RecordLogBatch, List<RecordLog»`). `RecordLogBatch` a `RecordLog` jsou entity relační databáze, ve kterých jsou uloženy dávky a veškeré logy. Pro jednotlivé mazání těchto logů existuje endpoint, který vyžaduje jako parametr `id` daného log záznamu. Pro hromadné mazání existuje zase jiný endpoint a má opět jeden parametr s názvem `uspesne`. Tento endpoint slouží i pro mazání *neúspěšně nahraných záznamů*, ale v tomto případě bude požadavek HTTP GET zasílán v této podobě na tuto URL `../lakehouse/smazat-logy-zaznamu?uspesne=true`.

5.3.5 Neúspěšně nahrané záznamy

Sekce poskytující výpis neúspěšně nahraných záznamů funguje na stejném principu jako pro ty úspěšně nahrané (viz podkapitola 5.3.4), ale hlavní rozdíl nastává na úrovni jednotlivých logů. U každého logu jsou následující čtyři možnosti: stažení záznamu, výběr záznamu ze souborového systému, opětovné nahrání záznamu do Delta tabulky a v neposlední řadě smazání logu. V praxi uživatel stáhne neúspěšně nahraný záznam, modifikuje jej a opětovně nahraje. Opět nechybí hromadné smazání logů.

Komunikace s úložištěm Po zaslání bezparametrického HTTP GET požadavku na URL `../lakehouse/neuspesne-nahrane-zaznamy` se vypíší všechny logy o neúspěšně nahraných záznamech opět ve formě mapy (`Map<RecordLogBatch, List<RecordLog»`). První možností je stažení neúspěšně nahraného záznamu. K tomu slouží příslušný endpoint, který požaduje `id` logu záznamu jako jediný parametr. Po následné modifikaci záznamu může být znovu vložen a nahrán. Pro opětovné nahrání záznamu slouží stejný endpoint a příslušné metody jako pro *nahrání nového záznamu* (viz podkapitola 5.3.3). Mazání jednotlivých logů funguje stejně jako v podkapitole výše pro *úspěšně nahrané záznamy*. Malý rozdíl je v hromadném mazání logů, protože parametr `uspesne` musí mít rozdílnou hodnotu, tj. URL `../lakehouse/smazat-logy-zaznamu?uspesne=false`.

5.3.6 Vykonání SQL dotazu

Možnost vykonávat SQL dotazy nad úložištěm *Data Lakehouse* je nejvíce stěžejní, protože umožňuje získávat data za pomocí dotazovací jazyka SQL. Pomocí dotazů lze do jisté míry řídit i samotné úložiště z pohledu údržby a optimalizace dat za účelem úspory diskového prostoru a rychlosti vykonávaných dotazů.

Pro vykonání dotazu je nutné vyplnit některá textová pole a další příslušné nastavení. Hlavní textové pole slouží pro definici dotazu. Dále je zde pole ovlivňující maximální počet vypisovaných záznamů, maximální počet znaků ve sloupci a orientaci výsledného pohledu (horizontální nebo vertikální). Pole s názvem *Popis dotazu* slouží jen pro identifikaci ve chvíli, kdy je dotaz uložen. Pokud by byla vyplněna jen výše uvedená pole, tak k jednotlivým Delta tabulkám musíme přistupovat pomocí jejich absolutních adresářových cest (viz ukázka kódu 5.4). Tento přístup není uživatelsky přívětivý, protože je nutná znalost adresářové struktury a v rámci dotazu to není přehledné.

```
SELECT * FROM delta.‘/tmp/lakehouse/dasta‘;
```

Listing 5.4: SQL dotaz s absolutními adresářovými cestami k tabulkám.

Globální pohledy tabulek V nastavení je možnost *Vytvořit nové globální pohledy*. Pokud je tato možnost zvolena, tak *Apache Spark* vytvoří v paměti globální pohledy pro všechny tabulky. K těmto tabulkám se poté přistupuje s prefixem `global_temp` jako je to uvedeno v ukázce kódu 5.5. Již při spuštění aplikace se vytvoří globální pohledy, takže tato možnost slouží hlavně pro případy, kdy je potřeba tyto pohledy přegenerovat.

```
SELECT * FROM global_temp.dasta;
```

Listing 5.5: SQL dotaz s globálními cestami k tabulkám.

Lokální pohledy tabulek Poslední možností v nastavení pohledů je *Vytvořit nové lokální pohledy*. Při tomto nastavení vytvoří *Apache Spark* dočasné pohledy v paměti, ale jen pro danou relaci. Následně k tabulkám lze přistupovat jako u standardního SQL (viz ukázka kódu 5.6).

```
SELECT * FROM dasta;
```

Listing 5.6: SQL dotaz s dočasnými cestami k tabulkám.

Komunikace s úložištěm Po nastavení všech parametrů dotazu se vytvoří požadavek, který obsahuje transportní objekt `SqlQueryDTO`, jehož součástí jsou všechny parametry SQL dotazu. Po odeslání požadavku na příslušný endpoint (viz ukázka kódu 6.5) je volána metoda `executeSqlQuery(...)`, která vykoná dotaz nad úložištěm *Data Lakehouse*. Odpověď tvoří standardně transportní objekt `CommonResponseDTO` (viz ukázka kódu 5.2), který v atributu `result` předává výsledek dotazu. Uživatel má přístup i k výsledku ve formátu JSON (*JavaScript Object Notation*), který může sloužit pro další zpracování výsledku dotazu (uložen v atributu `json` jako součást odpovědi). Další možností je uložení dotazu včetně všech nastavených parametrů. Požadavek využívá stejný transportní objekt, ale je zasílán na odlišný endpoint.

```
# API endpoint: ../lakehouse/vykonat-sql-dotaz
# HTTP metoda: POST

{
  "id": 0,
  "query": "string",
  "numRows": 0,
  "truncate": 0,
  "isVerticalView": true,
  "hasCreateNewGlobalTempView": true,
  "hasCreateNewLocalTempView": true,
  "description": "string"
}
```

Listing 5.7: Požadavek pro vykonání SQL dotazu.

5.3.7 Využití regulárních výrazů

V souvislosti s úložištěm *Data Lakehouse* je nejvíce zmiňován proces ELT a transformace na vstupu se příliš neřeší. V praxi se však ukázalo, že pro zvýšení průchodnosti záznamů do systému je nutné data na vstupu alespoň mírně transformovat a implementovat tedy ETL proces. Regulární výrazy jsou využívány především na „vyčištění“ záznamů od bílých znaků, prázdných a nepárových tagů apod. Hlavním problémem byly tagy s prázdnou hodnotou, protože *Apache Spark* s nimi nepracuje jako s hodnotou `null`, ale vyhodnotí je jako prázdný řetězec. Výsledný `DataFrame` má poté atribut klasifikován jako datový typ `String` a následně se nemusí shodovat se schématem cílové Delta tabulky.

K tomuto účelu vznikla samostatná sekce, kde je možné zakládat nové regulární výrazy a přiřazovat je k Delta tabulkám. Při nahrávání nového

záznamu do dané tabulky jsou aplikovány všechny přiřazené výrazy. Uložené výrazy mohou být také následně modifikovány v příslušné sekci.

Komunikace s úložištěm Pro založení nového regulárního výrazu je nutné vytvořit požadavek (viz ukázka kódu 5.8), jehož součástí je transportní objekt `RegularExpressionDTO`. V transportním objektu může být přiřazena jedna tabulka, na kterou se má výraz aplikovat. Pokud je výraz nutné aplikovat na více tabulek, tak v rámci editace může být přiřazen na více tabulek. V tomto případě je zasílán HTTP GET požadavek na příslušný endpoint se dvěma parametry (`regularExpressionId` a `tableName`). Úprava ostatních parametrů výrazů je téměř srovnatelná se zakládáním a je využíván i stejný transportní objekt.

```
# API endpoint: ../lakehouse/novy-regularni-vyraz
# HTTP metoda: POST

{
  "id": 0,
  "title": "string",
  "expression": "string",
  "lakehouseTableId": 0
}
```

Listing 5.8: Požadavek pro založení nového regulárního výrazu.

5.3.8 Využití XQuery výrazů

Mnoho medicínských standardů využívá formátu XML, a proto byl ETL proces rozšířen o dotazovací jazyk XQuery (*XML Query*), který je navržen pro práci s XML. Jedná se o velice komplexní nástroj a pro pochopení jeho funkčnost jsou níže uvedeny všechny jeho části:

XPath je dotazovací jazyk pro práci s XML, který je navržen pro jednoduché dotazy a jehož výsledkem je posloupnost složená z uzlů nebo primitivních datových typů. Poskytuje funkce pro navigaci ve struktuře dokumentu a pomocí funkce `doc(...)` lze spojovat více XML (více zdrojů) do jednoho dotazu. Poskytuje také základní agregační funkce (`COUNT`, `AVG` a `SUM`). Hlavním problémem je, že výsledkem dotazu nemůže být složitější struktura [34].

FLWOR (*For, Let, Where, Order by, Return*) tvoří množinu výrazů, které do XML dotazování přináší prvky z programování. Jak už akronym napovídá, tak se jedná o následující funkce [34]:

- `FOR` – výběr posloupnosti uzlů,
- `LET` – možnost vytvoření proměnné pro uchování mezivýsledku,
- `WHERE` – filtrování posloupnosti uzlů,
- `ORDER BY` – řazení odfiltrované posloupnosti uzlů vzestupně/sestupně podle nějakého klíče,
- `RETURN` – specifikace výstupu pro odfiltrovanou a seřazenou posloupnost uzlů.

XQuery je dotazovací jazyk navržený primárně pro práci s XML. Je nadstavbou dotazovacího jazyka *XPath*, který je navržen pro jednoduché dotazy a jehož výsledkem je posloupnost složená z uzlů nebo primitivních datových typů. Dále jsou jeho součástí FLWOR výrazy a několik dalších funkcí, které z něj dělají komplexní dotazovací jazyk nad XML datasety [34].

BaseX je framework poskytující nástroje pro práci s XML. Díky tomu, že v rámci aplikace bylo potřeba implementovat jazyk *XQuery*, tak došlo na výběr právě tohoto frameworku. Jedná se o velice komplexní framework nabízející SŘBD pro XML, XQuery procesor, klient/server architekturu, uživatelské rozhraní a mnoho dalšího. Pro účely aplikace byla využita jen Java knihovna pro XQuery procesor, který umožňuje transformovat nahrávané záznamy .V rámci aplikace zatím slouží pro mírné transformace, které zajišťují vyšší průchodnost záznamu do Delta tabulek.

Stejně jako pro regulární výrazy, tak i pro XQuery výrazy vznikla samostatná sekce. Zde je možné zakládat nové výrazy a opět je přiřazovat k Delta tabulkám. Při nahrávání nového záznamu jsou nejdříve aplikovány všechny přiřazené regulární a následně XQuery výrazy. I v tomto případě existuje samostatná sekce pro jejich modifikaci.

Komunikace s úložištěm Pro založení nového XQuery výrazu je nutné vytvořit požadavek (viz ukázka kódu 5.9), jehož součástí je transportní objekt `XQueryExpressionDTO`. V transportním objektu může být opět přiřazena jen jedna tabulka, na kterou se má výraz aplikovat – při editaci výrazu mohou být vybrány i další tabulky. V tomto případě je zasílán HTTP GET požadavek na příslušný endpoint se dvěma parametry (`xQueryExpressionId` a `tableName`). K úpravě parametrů dotazu standardně slouží stejný transportní objekt jako pro zakládání.

```
# API endpoint: ../lakehouse/novy-xquery-vyraz
# HTTP metoda: POST

{
  "id": 0,
  "title": "string",
  "expression": "string",
  "lakehouseTableId": 0
}
```

Listing 5.9: Požadavek pro založení nového XQuery výrazu.

5.3.9 Registrace nového uživatele

Pro přístup do administrace je nutné provést registraci uživatele. Uživatel musí vyplnit jméno, příjmení, e-mail (následně slouží i jako přihlašovací jméno), kontrolní otázku a potřebně silné heslo. Pod heslem se zobrazuje indikátor síly hesla, takže uživatel může heslo přizpůsobit daným požadavkům. Poté jsou zadané údaje přesunuty z registračního formuláře do transportního objektu `UserDTO` (viz ukázka kódu 5.10) a zkontrolovány. Pokud jsou všechny údaje validní, tak dojde k založení nového uživatele a zápisu do relační databáze pomocí metody `addUser(...)`. Nově založený uživatel má roli `USER`, takže může přistupovat ke všem výše zmíněným sekcím, ale nemůže spravovat ostatní uživatele, protože nemá roli `ADMIN` – ta může být přidělena jen existujícím administrátorem.

```
{
  "id": 0,
  "firstname": "string",
  "lastname": "string",
  "email": "string",
  "admin": false,
  "controlQuestion": "string",
  "password": "string",
  "confirmedPassword": "string"
}
```

Listing 5.10: Transportní objekt pro založení nového uživatele.

5.3.10 Nastavení uživatelských údajů

Po přihlášení do administrace může uživatel přistoupit do sekce *Nastavení* a změnit údaje zadané při registraci pomocí metody `changeUserDetails(...)`. Při tomto procesu je využíván stejný transportní

objekt jako při zakládání nového uživatele. Lze změnit všechny údaje vyjma kontrolní otázky pro obnovení hesla.

5.3.11 Změna práv uživatele

Pokud má uživatel roli `ADMIN`, tak je mu po přihlášení přístupná sekce *Administrace*. Zde může měnit práva pomocí metody `grantAdmin(...)` a údaje (včetně hesla) ostatních uživatelů.

5.3.12 Zapomenuté heslo

Z přihlašovacího formuláře je možné přejít na formulář pro zapomenuté heslo. Zde uživatel musí zadat správnou odpověď na kontrolní otázku, nové heslo splňující všechny požadavky a při potvrzení je volána metoda `changePasswordViaQuestion(...)`.

5.4 Management úložiště

Administrační část i Lakehouse úložiště používají ke své činnosti relační databázi navrženou v databázovém systému *MySQL (MariaDB)*, který využívá formát úložiště *InnoDB*. Pro úložiště má relační databáze funkci jako *datový slovník* [1].

Všech jedenáct tabulek je definováno přímo v aplikaci pomocí *Spring Data JPA*. Pro každou entitu je založena nová třída, která má anotaci `@Entity` a vždy obsahuje atribut `id` sloužící jako primární klíč. Takto definovaný atribut musí mít anotaci `@Id` a v tomto případě ještě obsahuje anotaci *GeneratedValue*, která generuje hodnotu jako sekvenci unikátních čísel. Každá třída reprezentující entity obsahuje primární klíč a další potřebné atributy. Datové typy těchto třídních atributů odpovídají sloupcům v *MySQL* databázi. Velká část atributů má anotaci *NotNull*, aby se zabránilo nulové hodnotě v daném záznamu. Pokud je nutné databázový sloupec detailněji definovat, tak se používá anotace `@Column`, která může obsahovat řadu parametrů. Může například obsahovat parametr `name` pro jméno sloupce, pokud je rozdílný s názvem atributu ve třídě. Často používaným parametrem je také `columnDefinition`, který může obsahovat definici DDL (*Data Definition Language*)[2]. Pro definici implicitních hodnot je použita ano-

[1] Zahrnuje seznam všech datových objektů v Data Lakehouse úložišti včetně popisu datových struktur a jejich vztahů.

[2] Sada SQL příkazů používaných k definici schématu databáze (popis databázového schématu a úprava databázových objektů).

tace `@ColumnDefault`, ale implicitní hodnota může být definována i v rámci parametru `columnDefinition`. V několika případech je atribut použit pro časové razítko a pro tento případ je použita anotace `@CreationTimestamp`. V aplikaci je vždy použita s anotací `@Temporal`, která definuje datový typ časového razítka (`DATE`, `TIME` nebo `TIMESTAMP`).

Již bylo zmíněno, že každá tabulka obsahuje primární klíč, který spolu s cizím klíčem může vytvořit vazbu se záznamy z ostatních tabulek. Atribut pro cizí klíč musí mít stejný datový typ jako entita s příslušným primárním klíčem. Dále musí být definovaná vazba mezi primárním a cizím klíčem pomocí následujících anotací: `@ManyToOne` (`M:1`), `@OneToMany` (`M:1`), `@ManyToMany` (`M:N`) a `@OneToOne` (`1:1`). Použitím anotace `@JoinColumn`, `@JoinColumns` nebo `@JoinTable` (pro `M:N` spojení) dojde k vytvoření reference mezi primárním a cizím klíčem.

Na obr 5.3 je zobrazen ERA diagram relační databáze, který zobrazuje jedenáct tabulek a jejich vazby pomocí primárních a cizích klíčů. Datové typy v jednotlivých tabulkách byly zvoleny s ohledem na optimální využití jejich prostoru. Pro primární a cizí klíče byl zvolen datový typ `BIGINT`, který je v současné době naddimenzovaný, ale v budoucnu může mít opodstatnění při velkém množství záznamů. Nějaké tabulky využívají sloupec `active` ve spojení s datovým typem `BIT(1)`, který indikuje, zdali je daný záznam smazaný. Využívá se však i pro další sloupce, kde se očekávají booleovské operace.

V ERA diagramu jsou tabulky rozděleny do tří kategorií. Tabulky s černým pozadím záhlaví jsou využívány úložištěm *Data Lakehouse*, tabulky s oranžovým záhlavím slouží pro administrační a šedivé záhlaví slouží pro automaticky vygenerované tabulky. Jedná se o těchto dvanáct následujících tabulek:

lakehouse_table je primární tabulkou pro část aplikace s úložištěm. Obsahuje veškeré informace o založených Delta tabulkách, bez kterých by správa úložiště *Data Lakehouse* byla značně ztížena a omezena.

regular_expression_list má vazbu na tabulku `lakehouse_table` pomocí cizího klíče `lakehouse_table_id` a uchovává všechny přiřazené regulární výrazy (pomocí jejich `id`) k dané Delta tabulce. Druhý cizí klíč `regular_expression_id` vytváří vazbu s tabulkou `regular_expression`.

regular_expression uchovává samotné regulární výrazy.

xquery_expression_list má vazbu na tabulku `lakehouse_table` pomocí cizího klíče `lakehouse_table_id` a uchovává všechny přiřazené XQuery výrazy (pomocí jejich `id`) k dané Delta tabulce.

xquery_expression uchová samotné XQuery výrazy.

record_log je další důležitou tabulkou pro část tvořící úložiště. Obsahuje veškeré záznamy o úspěšně/neúspěšně nahraných záznamech. Díky těmto záznamů je možné neúspěšně nahrané záznamy stáhnout, upravit a pokusit se je znovu nahrát do Delta tabulek. Obsahuje také cizí klíč `record_log_batch_id` pro vazbu s tabulkou `record_log_batch`.

record_log_batch tvoří jen záznamy obsahující primární klíč a časové razítko. Nový záznam vzniká při nahrávání jednoho nebo více záznamů a v administraci jsou nahrané záznamy seskupeny podle těchto dávek pro větší přehlednost výpisu.

sql_query uchovává SQL dotazy, které mohou být modifikovány nebo vykonány znovu. SQL dotazy slouží i pro správu úložiště, takže je vhodné mít uložené rutinní dotazy. Tato tabulka má černé záhlaví, takže by měla sloužit primárně pro úložiště, ale svým využitím má přesah i do administrační části.

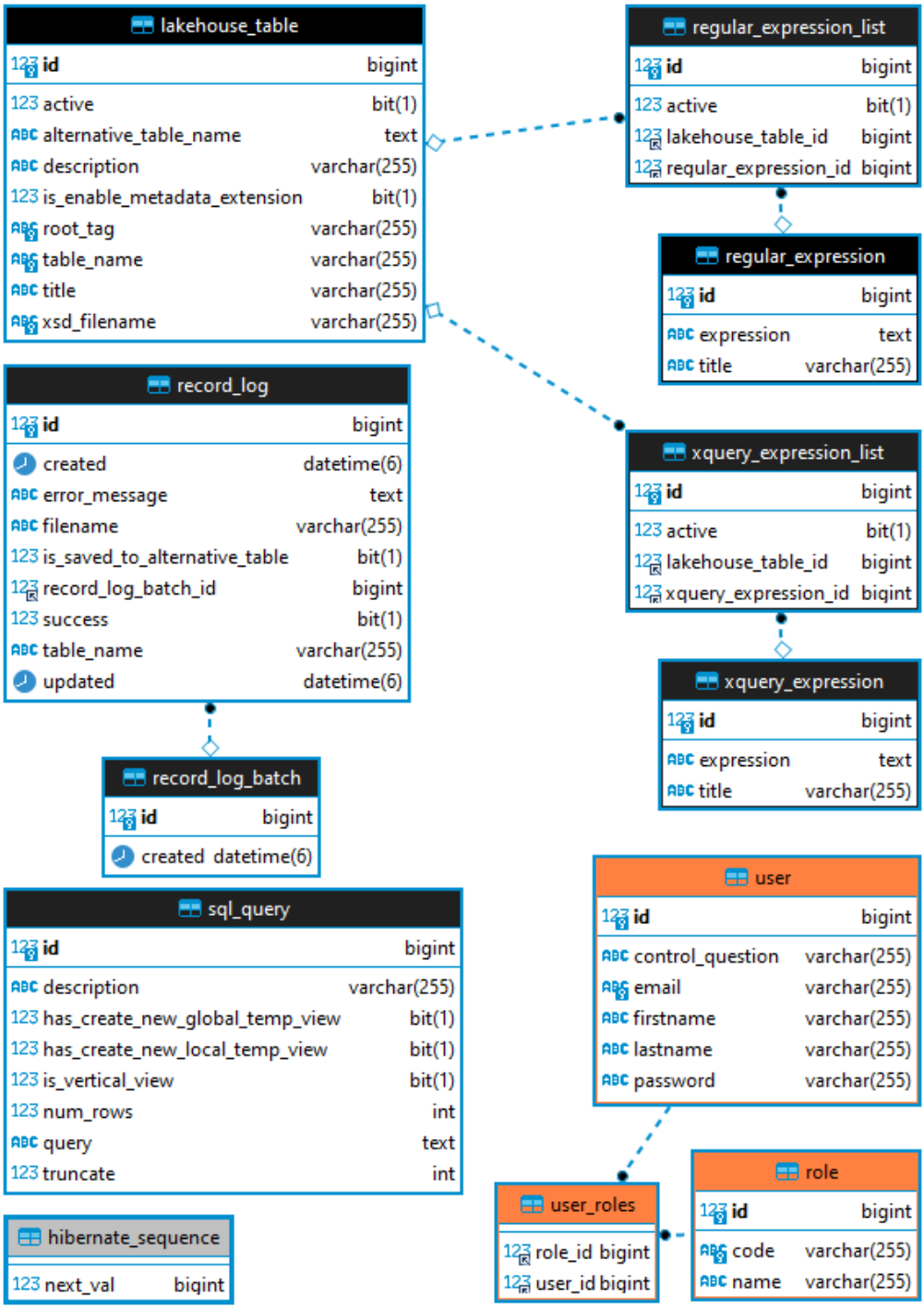
user je primární tabulkou pro administrační část. Po úspěšné registraci se vytvoří nový záznam obsahující veškeré informace o uživateli.

user_roles má vazbu na tabulku `user` pomocí cizího klíče `user_id` a uchovává `id` rolí jednotlivých uživatelů. Druhý cízí klíč `role_id` vytváří vazbu s tabulkou `user_roles`.

role je poměrně neměnná tabulka, ve které se vytvoří záznamy jen při inicializaci aplikace. Záznamy tvoří možné uživatelské role (`USER` nebo `ADMIN`) a popis pracovní skupiny.

hibernate_sequence je automaticky vygenerovaná tabulka uchovávající poslední číslo sekvence pro generování primárního klíče.

5.4.1 ERA diagram



Obrázek 5.3: ERA diagram relační databáze (datového slovníku).

6 Práce s úložištěm Data Lakehouse

Pro demonstraci práce s novým typem úložiště bylo potřeba založit příslušné Delta tabulky a do nich nahrát medicínské záznamy. Veškeré ukázky kódu uvedené níže slouží jen pro přiblížení problematiky a uživatel s nimi nepřijde do styku, protože jsou součástí implementace a pro tyto případy může využít administraci nebo poskytnuté REST API. Příslušné administrační sekce a API endpointy již nejsou zmiňovány, protože byly detailně popsány v kapitole 5.

6.1 Poskytnuté medicínské záznamy

V rámci diplomové práce byly k dispozici medicínské záznamy obsahující informace o pacientech a k některým existovaly záznamy o jejich cévní mozkové příhodě. Záznamy o pacientech využívají standardizovaného formátu DASTA a záznamy o cévní mozkové příhodě mají proprietární formát vycházející z registru RES-Q (*Registry of Stroke Care Quality*)

Lze předpokládat, že do úložiště budou nahrány výše uvedené medicínské záznamy a je potřeba s nimi dále pracovat. K tomuto účelu slouží primárně modul pro *Vykonávání SQL dotazů* (viz podkapitola 5.3.6) a níže jsou popsány některé případy práce s úložištěm *Data Lakehouse*.

6.1.1 RES-Q

Celosvětově využívaná platforma sloužící pro monitorování a zlepšování kvality péče o pacienty s cévní mozkovou příhodou. Za vývojem stojí *Cerebrovaskulární výzkumný program FN u sv. Anny v Brně* ve spolupráci *European Stroke Organisation* (ESA) [22]. Aktuálně není hlavní výzvou vytváření nových a lepších léčebných postupů, ale i zlepšování těch stávajících. Zavádění zlepšení nutně nevyžaduje další zdroje, ale vyžaduje lepší využití stávajících postupů a nástrojů. Změny v RES-Q jsou zaváděny a vyhodnocovány následujícím způsobem [23]:

- sledování cesty péče o pacienty s cévní mozkovou příhodou,
- rozpoznání zásadních výzev, kterým musím čelit každá nemocnice a země,

- zdůraznění oblastí s vysokými potřebami a informování o plánu řešení těchto potřeb,
- sledování realizace těchto změn a následného zlepšení péče o pacienty s cévní mozkovou příhodou.

Nástroj RES-Q nyní využívá více než 1 500 nemocnic ze 72 zemí a v registru je zapojených více než 350 000 pacientů. Díky tomuto systému mohou nemocnice provádět následující úkony [22]:

- monitorování svých procesů souvisejících s cévní mozkovou příhodou,
- srovnávání léčebných výsledků na mezinárodní úrovni,
- zlepšování kvality péče o všechny pacienty s cévní mozkovou příhodou z celého světa,
- shromažďování metrik výkonů souvisejících s kvalitou péče o pacienty podle mezinárodně uznávaných standardů,
- monitorování a prohlížení aktuálních výsledků.

Pro sběr dat je využíván jednoduchý dotazník, který je navržen tak, aby zadání dat o pacientovi netrvalo déle než 5 minut. Jsou požadována pouze data týkající se konkrétní léčby, protože osobní data o pacientech se v rámci registru neshromažďují – nezaměřuje se na sledování pacientů, ale na zlepšování nemocničních procesů. [22]

Sbíraná data musí být smysluplná, a proto RES-Q poskytuje dashboardy, které v reálném čase ukazují klíčové metriky nemocnice v porovnání s ostatními nemocnicemi po celém světě. Kromě dashboardů jsou generovány kvartální reporty obsahující podrobnější statistiky. K dispozici jsou i surová nezpracovaná data pro interní analýzy. Všechna tato data jsou vždy ve vlastnictví dané nemocnice, a proto pro benchmarking lze použít jen souhrnné statistiky, ale nikoli údaje o pacientech. Data o pacientech se nikdy nesdílejí bez výslovného souhlasu nemocnice [22].

6.2 Založení tabulek

V rámci diplomové práce byly k dispozici anonymizované údaje o pacientech ve standardizovaném formátu DASTA. Pro testovací účely byla založena tabulka s názvem *dasta* se základní strukturou metadat vytvořených pomocí XSD schéma, které bylo vytvořeno na základě dodaných záznamů. U tabulky bylo povoleno rozšiřování metadat na základě vstupních dat, takže

se vzrůstajícím počtem záznamů se rozšiřovala i struktura metadat tabulky. Dále byly k dispozici záznamy o cévní mozkové příhodě pocházejí z registru RES-Q a pro tyto záznamy byla založena tabulka s názvem *resq*, u které bylo také povoleno rozšiřování metadat na základě vstupních dat a základní strukturu metadat také obstaralo XSD schéma.

6.2.1 Založení tabulky z poskytnutého schéma

Na ukázce kódu 6.1 je ve zjednodušené formě založení nové tabulky. Jako první je načteno XSD schéma sloužící pro založení základní struktury metadat tabulky. To je následně uloženo na konvenční úložiště kvůli pozdějším úpravám. Z tohoto schéma je následně vytvořen nový `DataFrame`, který je nastaven pro zápis, využívá formátu `delta` a díky módu `overwrite` může přepsat již existující tabulku.

```
StructType xsd = XSDToSchema.read("/upload/schemas/dasta");
Dataset<Row> df = sparkSession.read().schema(xsd).load();
df
  .write()
  .format("delta")
  .mode("overwrite")
  .save("/lakehouse/dasta");
```

Listing 6.1: Založení nové tabulky pomocí XSD schéma.

6.2.2 Manuální založení tabulky

Tabulku lze definovat a založit pomocí klauzule `CREATE` (viz ukázka 6.2). Nejdříve je nutné definovat umístění v souborovém systému a následně název všech sloupců včetně jejich datových typů.

```
CREATE OR REPLACE TABLE delta.‘/tmp/delta/people10m‘ (
  id INT,
  firstName STRING,
  middleName STRING,
  lastName STRING,
  gender STRING,
  birthDate TIMESTAMP,
  ssn STRING,
  salary INT
) USING DELTA
```

Listing 6.2: Manuální založené nové tabulky [18].

6.3 Nahrání záznamů

Po založení tabulek mohu být do úložiště nahrávány medicínské záznamy. Již v kapitole 6 bylo zmíněno, že v rámci diplomové práce byly k dispozici dva druhy záznamů (DASTA a RESQ). Záznamy z registru RES-Q obsahovaly velké množství prázdných XML tagů (např. `<mr_infarkt_obj_pred />`, `<mr_otok_mozku_pred />`, `<jina_specif_lecba />` atd.), a proto bylo nutné definovat regulární výrazy, které tyto tagy identifikují a následně odstraní. Z tohoto důvodu byly definovány všechny formy prázdný tagů (`<EmptyTag/>`, `<EmptyTag></EmptyTag>`, `<EmptyTag>` a `</EmptyTag>`) a následně aplikovány na všechny tabulky. Prázdné tagy jsou kvalifikovány jako datový typ `String` s prázdnou hodnotou, a proto je nutné jejich odstranění, aby nebyly zamítnuty kvůli špatnému schéma. XQuery výrazy nebylo nutné aplikovat, a proto byly využity jen pro testovací účely (např. unifikace hodnot pro rozlišení pohlaví) a jedná se spíše o přípravu pro další typy záznamů.

Na zjednodušené ukázce kódu 6.3 se nejdříve načítá DASTA záznam do `DataFrame`. Ten je poté nastaven pro zápis do tabulky. Hodnota `mergeSchema: true` označuje, že tabulka může rozšiřovat svou strukturu metadat na základě přidání tohoto záznamu. Dále je nastaveno, že s nuluvou hodnotu se má zacházet jako s prázdným řetězcem, formát tabulky `delta` a díky módu `append` je záznam přidán do tabulky.

```
Dataset<Row> df = loadXmlToDataframe("dasta", "/upload/
    records/dasta", "dasta_rnn67903.xml");
df
  .write()
  .option("mergeSchema", "true")
  .option("nullValue", "")
  .format("delta")
  .mode("append").save("/lakehouse/dasta"));
```

Listing 6.3: Nahrání nového záznamu do úložiště Data Lakehouse.

6.4 Výpis informací o pacientovi

Záznamy typu DASTA uchovávají veškeré informace o pacientovi. Výpis těchto informací je možný pomocí SQL dotazu. Přístup k jednotlivým prvkům XML záznamu je možný skrze tečkovou notaci, protože uložená data zachovávají strukturu původního dokumentu (viz ukázka zkráceného záznamu DASTA 6.6). XML prvky v záznamu obsahují také atributy nesoucí další informace a lze k nim přistupovat také pomocí SQL dotazu skrze prefix (`_`),

a to např. `is.ip.z._oznaceni_o`.

```
SELECT DISTINCT
  is.ip.prijmeni, is.ip.dat_dn, is.ip.sex, is.ip.a.psc,
  is.ip.dg.dgz.spec_dg, is.ip.z.nazev,
  is.ip.z._oznaceni_o AS resq_ID
FROM dasta
ORDER BY dat_dn DESC;
```

Listing 6.4: Výpis informací o pacientech.

Na ukázce 6.4 je SQL dotaz, který vybírá určité sloupce (přes tečkovou notaci) z tabulky *dasta* pro výsledný pohled. Pro pozdější účely je do pohledu přidán sloupec s cizím klíčem patřící záznamům RESQ. V tabulce jsou duplicitní záznamy, takže je využita klauzule `DISTINCT`, aby v pohledu byly jen odlišné záznamy. Na obr. 6.1 poté lze vidět výsledný pohled, který je seřazený vzestupně podle data narození pacientů.

```
|prijmeni|        dat_dn|sex|  psc|                      spec_dg|                          nazev|  resq_ID|
+--------+--------------+---+-----+-----------------------------+-------------------------------+---------+
|  PP4655|{1976-05-28, D}|  M| 4925|[Mozkový infarkt způsobený ...|508/000 - Léčba ve speciali...|117025355|
|  PP4766|{1967-12-31, D}|  M|33026|                         null|508/000 - Léčba ve speciali...|118246047|
|  PP4753|{1964-03-22, D}|  M|32600|[Mozkový infarkt způs.tromb...|508/000 - Léčba ve speciali...|118049630|
|  PP4638|{1962-06-27, D}|  M|33011|[Hyperlipidémie, Arteriální...|508/000 - Léčba ve speciali...|116769102|
|  PP4665|{1961-03-21, D}|  M|18000|[Mozkový infarkt způsobený ...|508/000 - Léčba ve speciali...|117074374|
|  PP4769|{1950-05-12, D}|  F|33038|[Mozkový infarkt způsobený ...|508/000 - Léčba ve speciali...|118295148|
|  PP4763|{1946-04-15, D}|  F|31200|[Arteriální hypertenze - ak...|508/000 - Léčba ve speciali...|118202005|
|  PP4768|{1938-01-06, D}|  F|36221|[Mozkový infarkt způsobený ...|508/000 - Léčba ve speciali...|118292782|
|  PP4761|{1937-05-28, D}|  F|33008|[Podezření na COVID-19, Moz...|508/000 - Léčba ve speciali...|118185313|
|  PP4747|{1934-04-30, D}|  F|33401|[Mozkový infarkt způs.embol...|508/000 - Léčba ve speciali...|117966656|
+--------+--------------+---+-----+-----------------------------+-------------------------------+---------+
```

Obrázek 6.1: Výsledný pohled SQL dotazu zobrazující informace o pacientech.

Na zjednodušené ukázce kódu 6.5 lze vidět způsob, jak jsou nad úložištěm vykonávány SQL dotazy. Jako první je vytvořen `DataFrame` pomocí `SparkSession`, resp. pomocí metody `sql(...)`, která má `query` jako jediný parametr. Následně je omezen počet vypisovaných záznamů na 11 a maximální počet znaků ve sloupci na 30. Posledním parametrem metody `showString(...)` je volba orientace pohledu – v tomto případě se jedná o horizontální výpis. Tyto restrikce omezují jen textový výpis dotazu, ale nemají vliv na výsledek v poskytovaném JSON formátu.

```
Dataset<Row> df = sparkSession.sql(sqlQueryDTO.getQuery());
String sqlResult = sqlDataframe.showString(11, 30, false)
```

Listing 6.5: Vykonání SQL dotazu.

```
<?xml version="1.0" encoding="UTF-8"?>
<dasta ...>
  ...
  <is>
    ...
    <ip id_pac="4665">
      <prijmeni>PP4665</prijmeni>
      <dat_dn format="D">1961-03-21</dat_dn>
      <sex>M</sex>
      <a typ="1">
        <psc>18000</psc>
        <mesto>bSlY1V7NAYD5/W82n5kEMQ==</mesto>
      </a>
  </is>
  <z zadost="D" vznik="H" obsah="NL" stav="K" oznaceni_o="
   117074374" dat_ab="2022-09-26T23:20:56">
  ...
  </z>
  ...
</dasta>
```

Listing 6.6: Zkrácená úkazka DASTA záznamu.

6.5 Mazání záznamů

Může nastat případ, že je potřeba odstranit nepotřebné záznamy uložené v tabulce, které zabírají místo v úložišti. Toho lze docílit použitím klauzule `DELETE` a spolu s `WHERE` mohou být efektivně mazány záznamy. Pomocí dotazu na ukázce 6.7 budou smazáni všichni pacienti (záznamy) narozeni před datem *1950-05-12*. Odstranění proběhne na úrovni metadat, takže nedojde k fyzickému smazaní *Parquet* souborů – ty mohou být následně smazány nástrojem *Vacuum*, jehož činnost je níže zmíněna.

```
DELETE FROM delta.‘/tmp/lakehouse/dasta/‘
WHERE dat_dn < ’1950-05-12’
```

Listing 6.7: Mazání pacientů na základě podmínky.

6.6 Aktualizace záznamů

Delta Lake umožňuje aktualizovat záznamy v tabulce standardně pomocí klauzule `UPDATE`. U využití *XQuery výrazů* bylo zmíněno, že mohou sloužit např. na unifikaci hodnot pro rozlišení pohlaví. Toho lze docílit i pomocí SQL (viz ukázka 6.8).

```
UPDATE delta.`/tmp/delta/people-10m` SET gender = 'F' WHERE
   gender = 'Female';
UPDATE delta.`/tmp/delta/people-10m` SET gender = 'M' WHERE
   gender = 'Male';
```

Listing 6.8: Aktualizace záznamů [18].

6.7 Agregace informací o pacientech

Často je potřeba data z různých zdrojů agregovat za účelem generování reportů nebo pro tvorbu souhrnných statistik. Toho je možné docílit pomocí SQL dotazu, protože je standardně dostupná klauzule `JOIN`. Delta tabulky lze tedy spojit pomocí nějakého klíče stejně jako u relačních databází. V rámci dostupných dat bylo vhodné spojit data o pacientovi s jeho cévní mozkovou příhodou a tím otestovat tuto funkcionalitu. Záznam typu RESQ o cévní mozkové příhodě lze ve zkrácené podobě vidět na ukázce 6.9. Tyto záznamy nemají žádnou hierarchickou strukturu, takže na úrovni SQL dotazů se s nimi dobře pracuje.

```
<?xml version="1.0" encoding="UTF-8"?>
<data>
  <klinudal_id>118295148</klinudal_id>
  <kod_typu>508</kod_typu>
  <pacient_id>858063</pacient_id>
  <stav_dokumentace>D</stav_dokumentace>
  <datum_prov>2022-09-30</datum_prov>
  <hospsta_id>2316417</hospsta_id>
  <images>N</images>
  <aspirin_pred>N</aspirin_pred>
  <fibrilace_sini>X</fibrilace_sini>
  <datum_vzniku>2022-09-29</datum_vzniku>
  <teplota_pred>37,7</teplota_pred>
  <odhad_vaha_pred>80</odhad_vaha_pred>
  <eval_fyzioterapeutem>5</eval_fyzioterapeutem>
  <celk_stav_24h>5</celk_stav_24h>
  <pacient_zemrel>9/30/2022 12:30:00 PM</pacient_zemrel>
</data>
```

Listing 6.9: Zkrácená ukázka záznamu RESQ.

Na ukázce 6.2 je SQL dotaz, který vybírá základní informace o pacientech z tabulky *dasta* a pomocí klauzule `JOIN` je spojuje s tabulkou *resq*. Záznamy jsou vzestupně seřazeny podle data provedení a je využita klauzule `LIMIT` pro omezení počtu záznamů ve výsledném pohledu (viz obr. 6.2). I předchozí pohled měl omezený počet záznamů, ale omezení bylo na úrovni

kódu (viz ukázka kódu 6.5). Využitím klauzule `LIMIT` se omezení projeví i ve výsledku reprezentovaném jako JSON. Samotný JSON pohled má několik výhod oproti pohledu v textové podobě. Na jednotlivé záznamy nejsou aplikovány restrikce pro výpis (např. počet znaků ve sloupci) a zachovává hierarchii dat. Díky tomu jsou data a vazby mezi nimi přehlednější a hlavně použitelnější pro další zpracování v porovnání s textovým pohledem, který má *flat* strukturu kvůli 2D zobrazení do tabulky.

```
SELECT
  is.ip.prijmeni, pacient_id, kod_typu, hospsta_id,
  datum_prov, images, aspirin_pred AS aspirin,
  fibrilace_sini AS fibrilace, teplota_pred AS teplota,
  vaha_pred AS vaha
FROM dasta
JOIN resq ON dasta.is.ip.z._oznaceni_o = resq.klinudal_id
ORDER BY datum_prov DESC
LIMIT 11;
```

Listing 6.10: Agregace informací o pacientech a jejich mozkové příhodě.

```
|prijmeni|pacient_id|kod_typu|hospsta_id|datum_prov|images|aspirin|fibrilace|teplota|vaha|
+--------+----------+--------+----------+----------+------+-------+---------+-------+----+
|  PP4766|    245001|     508|      null|2022-09-27|     N|   null|        N|   null|null|
|  PP4753|   1965536|     508|   2313347|2022-09-17|   art|      N|        N|   null|null|
|  PP4753|   1965536|     508|   2313347|2022-09-17|   art|      N|        N|   null|null|
|  PP4753|   1965536|     508|   2313347|2022-09-17|   art|      N|        N|   null|null|
|  PP4753|   1965536|     508|   2313347|2022-09-17|   art|      N|        N|   null|null|
|  PP4753|   1965536|     508|   2313347|2022-09-17|   art|      N|        N|   null|null|
|  PP4753|   1965536|     508|   2313347|2022-09-17|     N|      N|        N|   null|null|
|  PP4747|    645501|     508|   2312448|2022-09-13|     N|      A|        A|   36,5|null|
|  PP4665|    930015|     508|   2301825|2022-07-26|   art|      N|        N|   36,5| 120|
|  PP4665|    930015|     508|   2301825|2022-07-26|   art|      N|        N|   36,5| 120|
|  PP4665|    930015|     508|   2301825|2022-07-26|   art|      N|        N|   36,5| 120|
+--------+----------+--------+----------+----------+------+-------+---------+-------+----+
```

Obrázek 6.2: Výsledný pohled SQL dotazu zobrazující informace o pacientech a jejich mozkové příhodě.

6.8 Aktualizace schéma tabulek

Při zakládání všech tabulek bylo povoleno rozšiřování struktury metadat (schéma) na základě vstupních záznamů. Pokud je při nahrávání identifikován neznámý prvek s hodnotou, tak je schéma rozšířeno o tento prvek s případným atributem.

6.8.1 Manuální přidání sloupce

Může nastat situace, kdy je potřeba přidat nový sloupec do tabulky manuálně. Opět lze využít administraci nebo REST API pro vykonání SQL dotazu a sloupec přidat pomocí dotazu. Na ukázce 6.11 je opět využita standardní syntaxe SQL a malá odlišnost je u definice nového sloupce pomocí klauzule `ADD` a `COLUMN(S)`. Po těchto klauzulích následuje závorka, ve které jsou definovány nové sloupce (název a datový typ).

Na ukázce 6.11 je adresářový přístup k Delta tabulce. Je to z důvodu, že nelze využít dočasný pohled vytvořený *Apache Spark* a k tabulkám přistupovat jen podle jejich jména jako v předchozích případech. Při pokusu s využitím dočasného pohledu *Delta Lake* ihned hlásí chybu a nový sloupec nepřidá. V tomto případě se jedná o správné chování, protože i kdyby *Delta Lake* tuto chybu nehlásil a operaci provedl, tak dojde jen ke změně v dočasném pohledu, ale úložiště zůstane beze změny.

```
ALTER TABLE delta.‘/tmp/lakehouse/dasta/‘
ADD COLUMNS (prvni_sloupec integer, druhy_sloupec string);
```

Listing 6.11: Přidání nových sloupců.

6.8.2 Změna pořadí a úprava komentáře

Pořadí sloupců není příliš důležité, protože ho lze definovat v SQL dotazu, ale *Delta Lake* tuto funkcionalitu podporuje a v některých případech se může hodit (viz ukázka 6.12).

```
ALTER TABLE delta.‘/tmp/lakehouse/dasta/‘
ALTER COLUMN prvni_sloupec AFTER druhy_sloupec;
```

Listing 6.12: Změna pořadí sloupců v tabulce.

Samotný název sloupce spolu s datovým typem nemusí stačit pro pochopení očekávaných metadat. Z tohoto důvodu lze ke každému sloupci přidat, popř. změnit komentář.

```
ALTER TABLE delta.‘/tmp/lakehouse/dasta/‘
ALTER COLUMN prvni_sloupec
COMMENT ’komentar popisujici ocekavana metadata’;
```

Listing 6.13: Přidání nebo úprava stávajícího komentáře.

6.8.3 Přejmenování a odstranění sloupce

K dalším důležitým operacím patří přejmenování a odstranění sloupce. Přejmenování (`RENAME`) a odstranění (`DROP COLUMNS`) sloupců patří k operacím,

které vyžadují vyšší protokol Delta tabulky. Upgrade se provádí na úrovni vlastností dané tabulky (viz ukázka 6.14). Pokud uživatel nemá tuto informaci, tak při pokusu o provádění těchto operací je upozorněn chybovou hláškou, že má nastavit níže uvedené vlastnosti.

```
ALTER TABLE delta.‘/tmp/lakehouse/dasta/‘
SET TBLPROPERTIES (
   ’delta.minReaderVersion’ = ’2’,
   ’delta.minWriterVersion’ = ’5’,
   ’delta.columnMapping.mode’ = ’name’
)
```

Listing 6.14: Změna protokolů tabulky na úrovni jejich vlastností.

Na ukázce 6.15 lze vidět přejmenování sloupce v tabulce.

```
ALTER TABLE delta.‘/tmp/lakehouse/dasta/‘
RENAME COLUMN prvni_sloupec TO prvni_prejmenovany_sloupec;
```

Listing 6.15: Přejmenování sloupce v tabulce.

Na této ukázce 6.16 je naopak smazání již přejmenovaného sloupce. Je však nutné mít na vědomí, že se sloupec odstraní pouze ze struktury metadat, ale v *Parquet* souborech bude uložen stále. To lze považovat za nevýhodu, protože data stále zabírají diskový prostor a při opětovném přidání stejného sloupce se již tato data neberou v potaz.

```
ALTER TABLE delta.‘/tmp/lakehouse/dasta/‘
DROP COLUMN prvni_prejmenovany_sloupec;
```

Listing 6.16: Smazání sloupce v tabulce.

6.8.4 Změna datového typu sloupce

Poslední operací na úrovni sloupce je změna datového typu. Tuto operaci *Delta Lake* zatím nepodporuje, ale údajně je vývojovém backlogu. Jedinou možností je přepsání celé tabulky a tato operace musí proběhnout na úrovni kódu. Na ukázce kódu 6.17 lze vidět, že nejdříve dojde k načtení celé tabulky. Následně je vybrán sloupec `is.ip.a.psc`, který má být přetypován z datového typu `SHORT` na `STRING`. Mód `overwrite` dovoluje přepsat tabulku na úrovni `Parquet` souborů a `overwriteSchema` umožňuje přepsat schéma tabulky.

Jedná se však o výkonnostně náročnou operaci, takže v rámci diplomové práce nebyla tato funkčnost implementována. Alternativním řešením je založení nového sloupce s požadovaným datovým, zkopírováním hodnot do nového sloupce a ponecháním starého sloupce jako zálohy.

```
Dataset<Row> df =
  sparkSession.read().load("/tmp/lakehouse/dasta");
Column col = df.col("is.ip.a.psc");
df
  .withColumn("is.ip.a.psc", col.cast("string"))
  .write()
  .format("delta")
  .mode("overwrite")
  .option("overwriteSchema", "true")
  .save("/tmp/lakehouse/dasta");
```

Listing 6.17: Změna datového typu sloupce v tabulce. [18, 35].

6.8.5 Cestování v čase – *time-travel*

Time-travel je velice užitečná vlastnost známá z datových skladů a aktuálně i z úložiště *Data Lakehouse*. I *Delta Lake* umožňuje cestování v čase, resp. vrátit stav tabulky do určité verze nebo času. Na ukázce 6.18 první dotaz slouží pro výpis posledních deseti verzí a následující dva dotazy vrátí tabulku v čase podle určité verze nebo časového razítka. Výpis verzí je zobrazen na obr. 6.3.

```
// vypis poslednich deseti verzi
DESCRIBE HISTORY delta.‘/tmp/lakehouse/dasta/‘ LIMIT 10;

// vraceni stavu tabulky podle cisla verze
RESTORE TABLE delta.‘/tmp/lakehouse/dasta/‘
TO VERSION AS OF 23;

// vraceni stavu tabulky podle casoveho razitka
RESTORE TABLE delta.‘/data/target/‘ TO TIMESTAMP
AS OF ’2023-05-08 01:52:00’

SELECT * FROM delta.‘/data/target/‘ TO TIMESTAMP
AS OF ’2023-05-08 01:52:00’
```

Listing 6.18: Vrácení stavu tabulky podle čísla verze nebo časového razítka.

Po úspěšném vrácení stavu tabulky do určitého bodu jsou vypsány následující metriky:

- `table_size_after_restore` – velikost tabulky po vrácení do určité verze,
- `num_of_files_after_restore` – počet souborů (záznamů) po vrácení do určité verze,
- `num_removed_files` – počet odstraněných souborů (záznamů),

- `num_restored_files` – počet obnovených souborů (záznamů) díky vrácení,
- `removed_files_size` – celková velikost odstraněných souborů (záznamů) vyjádřená v bajtech,
- `restored_files_size` – celková velikost obnovených souborů (záznamů) vyjádřená v bajtech.

```
|version|          timestamp|userId|userName|
+-------+-------------------+------+--------+
|     27|2023-05-08 02:07:...|  null|    null|
|     26|2023-05-08 02:06:...|  null|    null|
|     25|2023-05-08 01:52:...|  null|    null|
|     24|2023-05-08 01:51:...|  null|    null|
|     23|2023-05-08 01:47:...|  null|    null|
|     22|2023-05-08 01:24:...|  null|    null|
|     21|2023-05-08 01:22:...|  null|    null|
|     20|2023-05-08 01:22:...|  null|    null|
|     19|2023-05-07 23:54:...|  null|    null|
|     18|2023-05-07 23:54:...|  null|    null|
+-------+-------------------+------+--------+
```

Obrázek 6.3: Výpis posledních deseti verzí.

V některých případech není nutné tabulku vracet do určitého bodu, ale zajímají nás pouze historická data. Toho lze docílit jako na ukázce 6.19.

```
// vypis historickych zaznamu podle cisla verze
SELECT * FROM delta.‘/tmp/lakehouse/dasta/‘
TO VERSION AS OF 23;

// vypis historickych zaznamu podle casoveho razitka
SELECT * FROM delta.‘/data/target/‘
TO TIMESTAMP AS OF ’2023-05-08 01:52:00’
```

Listing 6.19: Výpis historických dat podle čísla verze nebo časového razítka.

6.8.6 Integritní omezení

Integritní omezení automaticky ověřují kvalitu a integritu přidávaných dat do tabulky. *Delta Lake* podporuje standardní SQL klauzule pro správu omezení a pokud je omezení porušeno, tak je vyvolána výjimka `InvariantViolationException` [27].

Na ukázce 6.20 je nejdříve založena nová tabulka, ve které je specifikováno pomocí `NOT NULL`, že hodnoty v daných sloupcích nesmí být `null`. Následně je toto omezení u jednoho ze sloupců odstraněno. Dále je přidáno omezení pomocí klauzule `ADD CONSTRAINTS` a `CHECK`, které dovoluje nahrávat jen záznamy splňující určitou podmínkou. Klauzule `CHECK` zastává booleovskou operaci, která jen vyhodnocuje, zdali je zadaná podmínka pravdivá. Následně je opět tato podmínka odstraněna. Všechna přidaná omezení mohou být zobrazena po zadání příkazu `DESCRIBE DETAIL` nebo `SHOW TBLPROPERTIES` [27].

```
// pridani omezeni
CREATE TABLE delta.‘/tmp/lakehouse/test_constraints‘(
  id INT NOT NULL,
  username STRING NOT NULL,
  dateAdded TIMESTAMP
) USING DELTA;


// odstraneni omezeni
ALTER TABLE delta.‘/tmp/lakehouse/test_constraints‘
CHANGE COLUMN username DROP NOT NULL;

// pridani podminky
ALTER TABLE delta.‘/tmp/lakehouse/test_constraints‘
ADD CONSTRAINT dateWithinRange
CHECK (dateAdded > ’1950-01-01’);


// odstraneni podminky
 ALTER TABLE delta.‘/tmp/lakehouse/test_constraints‘
 DROP CONSTRAINT dateWithinRange
```

Listing 6.20: Přidání integritních omezení [27].

6.8.7 Vytváření klonů

Při strojovém učení se může hodit archivace určité verze tabulky, na které byl trénován model. Díky tomu může být ověřena funkčnost natrénovaného modelu na stejné množině trénovacích dat. *Delta Lake* umožňuje vytvářet klony tabulek, ale až od verze 2.3. Vytváření klonů proto nebylo možné otestovat v rámci diplomové práce, protože aplikace využívá verzi 2.1.1, která byla aktuální na začátku vývoje [16].

Shallow klone Na ukázce 6.21 je vytváření mělkých kopií pomocí klauzule `SHALLOW CLONE`. V prvním případě se vytváří klon podle čísla verze tabulky,

v druhém případě podle časového razítka a v posledním případě je vytvářen klon z aktuální verze tabulky. U mělkých kopií dochází pouze ke kopírování metadat a *Parquet* soubory jsou odkazovány z původní tabulky. Zdrojové soubory by měly být vždy k dispozici, aby bylo možné klon používat a pokud dojde k jejich odstranění, tak je klon nepoužitelný. Při aktualizaci záznamů se vytvoří nové soubory v adresáři klonované tabulky a nedojde k ovlivnění zdrojových souborů [32].

```
// vytvareni shallow klonu podle cisla verze
CREATE TABLE delta.‘/tmp/lakehouse/training‘
SHALLOW CLONE delta.‘/tmp/lakehouse/dasta‘
VERSION AS OF 23;

// vytvareni shallow klonu podle casoveho razitka
CREATE TABLE delta.‘/tmp/lakehouse/training_dataset‘
SHALLOW CLONE delta.‘/tmp/lakehouse/dasta‘
TIMESTAMP AS OF ’2023-05-08 01:52:00’

// vytvareni shallow klonu z aktualni verze tabulky
CREATE TABLE IF NOT EXISTS delta.‘/tmp/lakehouse/
   training_dataset‘
SHALLOW CLONE delta.‘/tmp/lakehouse/dasta‘;
```

Listing 6.21: Vytváření shallow klonů [16].

Deep klone Při hlubokém klonování se do cílové tabulky kopírují metadata i zdrojové soubory *Parquet*. Dojde k vytvoření nezávislé kopie a manipulace zdrojových dat nemá žádný vliv na klon. Vytváření těchto klonů je na ukázce 6.22, kde je pro vytvoření klonů použita klauzule `CLONE` a ve všech případech jsou vytvářeny klony podle stejných parametrů jako u mělkých kopií [32].

```
// vytvareni deep kopie klonu cisla verze
CREATE TABLE delta.‘/tmp/lakehouse/training‘ SHALLOW CLONE
   delta.‘/tmp/lakehouse/dasta‘
VERSION AS OF 23;

// vytvareni deep klonu podle casoveho razitka
CREATE TABLE delta.‘/tmp/lakehouse/dasta/‘ SHALLOW CLONE
   delta.‘/tmp/lakehouse/training_dataset‘
TIMESTAMP AS OF ’2023-05-08 01:52:00’

// vytvareni deep klonu z aktualni verze tabulky
CREATE TABLE IF NOT EXISTS delta.‘/tmp/lakehouse/
   training_dataset‘ CLONE delta.‘/tmp/lakehouse/dasta‘;
```

Listing 6.22: Vytváření deep klonů [16].

Po úspěšném vytvoření klonu jsou vypsány následující metriky:

- `source_table_size` – velikost klonované tabulky vyjádřená v bajtech,
- `source_num_of_files` – počet souborů ve zdrojové tabulce.

6.8.8 Údržba úložiště

Zdrojové *Parquet* soubory, na které již Delta tabulka neodkazuje mohou být odstraněny nástrojem *Vacuum*. Tyto soubory nejsou nikdy odstraněny automaticky, ale po spuštění příkazu `VACUUM` jsou smazány soubory, které jsou kandidátem na smazání více než 7 dní (implicitní hodnota). Tato hodnota může být upravena ve vlastnostech tabulky (`delta.deletedFileRetentionDuration = "interval 7 days"`). Na ukázce 6.23 je nejdříve standardní spuštění nástroje. Dále je nástroj spuštěn s hodnotou, která má vyšší prioritu než hodnota uvedená ve vlastnostech tabulek. Poslední příkaz slouží pro výpis kandidátních souborů ke smazání.

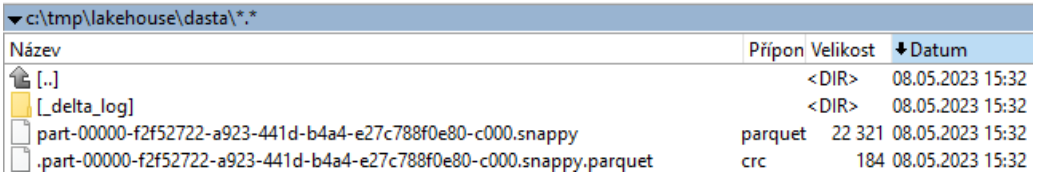
```
VACUUM delta.‘/tmp/lakehouse/dasta‘

// kandidatni soubory starsi nez 120 hodin
VACUUM delta.‘/tmp/lakehouse/dasta‘ RETAIN 120 HOURS

// vypise list souboru pro smazani
VACUUM delta.‘/tmp/lakehouse/dasta‘ DRY RUN
```

Listing 6.23: Spouštení nástroje Vacuum.

Každá tabulka obsahuje adresář `_delta_log` (viz obr. 6.4), který obsahuje protokoly o verzích tabulky. Díky těmto souborům je možné využívat *time-travel*. Ve výchozím stavu se uchovávají protokoly, které nejsou starší než 30 dní (implicitní hodnota). Při zápisu nového kontrolního bodu jsou automaticky smazány protokoly, které jsou starší než interval uchovávání. Interval může být upraven ve vlastnostech tabulky (`delta.deletedFileRetentionDuration = "interval 30 days"`).



Obrázek 6.4: Struktura Delta tabulky.

7 Náměty na zlepšení

V rámci vývoje a hlavně při práci s úložištěm *Data Lakehouse* vyvstalo několik námětů na zlepšení implementace.

7.1 Využití Apache Hive

V současném řešení jsou protokoly tabulek (adresář `_delta_log`) ukládány ve stejném adresáři, jako je umístění tabulek (viz obr. 6.4). To má za následek, že uživatel musí znát adresářovou strukturu, aby mohl k dané tabulce přistoupit. Tento problém je aktuálně řešen pomocí *Apache Spark*, který vytváří pohledy tabulek v paměti.

Lepším řešením by bylo využít službu *Hive Metastore*, která je součástí distribuovaného datové skladu *Apache Hive*. Protokoly tabulek by byly ukládány do vybrané databáze (např. MySQL nebo PostgreSQL) pomocí *Hive Metastore*. Po této změně by i zdrojové soubory *Parquet* mohly být ukládány například do distribuovaného úložiště HDFS (*Hadoop Distributed File System*). Tímto řešením by se zjednodušil přístup k tabulkám a jejich správa.

7.2 Vizualizace struktury metadat tabulky

V *Delta Lake* chybí možnost pro vypsání schéma. Má jen příkaz `DESCRIBE HISTORY`, který po zadání vypíše změny provedené ve struktuře metadat. Aktuálně je schéma vypisováno jen do konzole aplikace při určitých operacích. Další možností je výpis pomocí SQL dotazu.

Ani jedno z výše uvedených řešení není příliš uživatelsky přívětivé, a proto by bylo vhodné tuto funkcionalitu implementovat, rozšířit REST API a vytvořit příslušnou sekci v administraci pro vizualizaci tabulky. K získání struktury by mohl být využit `DataFrame`, který je nyní využíván k výpisu schématu do konzole.

7.3 Optimalizace nahrávání záznamů

Pokud záznamy nesplňují schéma tabulky, tak jsou nahrávány do alternativní tabulky nebo za určitých podmínek může být schéma rozšířeno o určitá metadata. Nemusí však dojít k nahrání ani do jedné tabulky a poté se očekává akce od uživatele. Ke zvýšení průchodnosti dat do úložiště se v rámci

ETL procesu využívají regulární a XQuery výrazy a tím se snižuje množství zásahů od uživatele.

Ke zvýšení průchodnosti by možná přispělo přidání dalších regulárních a XQuery výrazů. Další potenciální možností by bylo rozšíření implementace, které by obnášelo automatické zakládání alternativních tabulek. Pokud by záznam nemohl být nahrán ani do jedné tabulky, tak by se vytvořila tabulka se schématem nahrávaného záznamu. Následně by tabulky pro daný typ záznamu mohly být sloučeny pomocí SQL klauzule `MERGE`, kterou *Delta Lake* podporuje.

7.4 Vytváření klonů

V podkapitole 6.8.7 bylo popsáno, že vytváření klonů nebylo v rámci diplomové práce otestováno, protože v použité verzi *Delta Lake* tato funkcionalita nebyla ještě dostupná.

Bylo by tedy vhodné tuto funkčnost otestovat a popř. ji implementovat do aplikace, vhodně rozšířit REST API a vytvořit příslušnou sekci v administraci.

7.5 Změna datového typu sloupce

V podkapitole 6.8.4 bylo popsáno, že změna datového typu sloupce u Delta tabulky není triviální věc. Tuto funkcionalitu nepodporuje ani *Delta Lake* a v době psaní diplomové práce je jedinou možností přepsání schéma a zdrojových dat tabulky.

Pokud by v budoucnu tato funkcionalita byla dostupná, tak by opět bylo vhodné ji otestovat, implementovat do aplikace, rozšířit REST API a vytvořit příslušnou sekci v administraci.

8 Vhodnost úložiště Data Lakehouse pro oblast medicínských dat

Vhodnost úložiště pro oblast medicínských dat byla ověřena v kapitole 6, kde bylo prováděno ověřování funkcionality prototypu nově vzniklé aplikace. Byla zde ověřena funkčnost samotné aplikace, ale zároveň vhodnost úložiště, protože všechny případy užití byly prováděny na medicínských záznamech. Ve zdravotnických informačních systémech se pracuje s velkým množstvím heterogenních dat, pro které je právě tento typ úložiště vhodný.

Extrakce metadat Aktuálně nasazená *MRE platforma* zpracovávající medicínská data funguje výborně a pokrývá požadovanou funkčnost. Je však nutné zmínit, že extrakci metadat zajišťuje software *MetaMed*, který ukládá relevantní metadata ze záznamu do grafové databáze v podobě RDF souborů. Mapování metadat do RDF souborů je součástí konfigurace systému, takže pokud dojde ke změně schématu medicínských záznamů, tak na to nelze flexibilně reagovat a je potřeba změnit statickou konfiguraci systému. Nová implementace dokáže automaticky rozšiřovat schéma na základě nově přidaných záznamů, takže odpadá nutnost existence konfigurace a její udržování.

Transakční zpracování Je vyžadováno, aby každá transakce měla vlastnosti ACID a *MRE platforma* kvůli tomu vytváří vlastní meta-transakční vrstvu. Tu zajišťuje grafová databáze zmíněná výše a RFD soubory slouží jako indexy do úložiště. V nově vzniklé aplikaci nebylo potřeba řešit mechanismus pro transakční zpracování, protože úložiště *Data Lakehouse* toto implicitně řeší.

Dotazování se nad medicínskými záznamy *MRE platforma* ukládá data do RDF souborů, takže pro výpis informací o pacientech je používán dotazovací jazyk SPARQL, který má některé limity, ale hlavně je méně přirozený pro koncového uživatele. Nové řešení umožňuje získávat informace pomocí dotazovacího jazyka SQL, který poskytuje standardní dotazovací klauzule. Pro většinu uživatelů je přirozenější, a proto jsou schopni vykonávat i složitější operace s daty.

9 Závěr

Cílem práce bylo seznámit se s problematikou úložiště typu *Data Lakehouse*. Nejdříve byl uskutečněn teoretický rozbor, který se zabývá evolucí úložišť pro pochopení tohoto nového konceptu. Jednotlivé typy úložišť byly zevrubně popsány, ale nejvíce prostoru bylo věnováno úložišti *Data Lakehouse*, u kterého byly analyzovány jednotlivé procesy (ETL a ELT) a vrstvy pro různé úrovně kvality dat.

Poté proběhla analýza stávající *MRE platformy* sloužící jako úložiště medicínských dat. Na základě této analýzy byly definovány základní vlastnosti, které nová implementace používající úložiště *Data Lakehouse* musí splňovat. Jednalo se o extrakci metadat, transakční zpracování a možnost používání dotazovacího jazyka pro získávání informací o pacientech. V rámci této části byly popsány i nejpoužívanější medicínské standardy pro ukládání a přenos dat mezi informačními systémy.

Dalším krokem byl výběr správné technologie pro budování úložiště *Data Lakehouse*. Do užšího výběru se dostaly tyto tři open-source projekty: *Delta Lake*, *Apache Hudi* a *Apache Iceberg*. Každý z těchto projektů musel pokrývat výše popsané vlastnosti. Proběhlo důkladné zhodnocení všech projektů v několika aspektech a ve výsledku byl vybrán *Delta Lake*. Po výběru technologie se přistoupilo k vývoji aplikace ve frameworku *Spring Boot* a k implementaci technologie *Delta Lake*. Vznikla komplexní aplikace, která je logicky rozdělena na dvě části. První část tvoří *administrace*, která poskytuje kompletní správu uživatelů a obsahuje sekce zajišťující správu úložiště *Data Lakehouse*. Druhou část tvoří úložiště a skrze REST API poskytuje veškeré funkce pro jeho správu.

Po dokončení vývoje bylo nutné tento prototyp otestovat. V kapitole 6 jsou popsány veškeré případy užití. Testování probíhalo s poskytnutými medicínskými daty, takže tím byla ověřena vhodnost úložiště pro různé typy medicínských dat (doplňující informace viz kapitola 8) a zároveň ověřena základní funkčnost nově vzniklého prototypu. Na základě testování vznikly nové náměty na zlepšení aplikace, které jsou popsány v kapitole 7. Úložiště *Data Lakehouse* je poměrně mladý pojem (první zmínky společností *Databricks* v roce 2019), a proto většina technologií kolem tohoto úložiště prochází progresivním vývojem. Většina nedostatků tedy plyne z toho, že požadovaná funkcionalita od *Delta Lake* v době vývoje nebyla dostupná a v nových verzích již je.

Nově vzniklá aplikace implementující úložiště *Data Lakehouse* je plně

funkční a ukazuje vhodnost použití tohoto úložiště pro ukládání heterogenních medicínských dat. Diplomová práce poukazuje i na zjištěné nedostatky, které mohou být námětem pro pokračování v tomto tématu a rozšiřování nově vzniklé aplikace, která byla vyvíjena způsobem, aby byla v budoucnu snadno rozšiřitelná. Výběr technologií probíhal v souladu se současným řešením, takže by bylo potenciálně možné nový software implementovat do *MRE platformy*.

Všechny cíle diplomové práce byly splněny.

Přehled zkratek

SŘBD	Systém řízení báze dat
SQL	Structured Query Language
CT	Computed Tomography
EKG	Elektrokardiografie
FAV	Fakulta aplikovaných věd
ZČU	Západočeská univerzita v Plzni
ERP	Enterprise Resource Planning
CRM	Customer Relationship Management
IoT	Internet of Things
DW	Data Warehouse
BI	Business Intelligence
AI	Artificial Intelligence
ML	Machine Learning
MPP	Massively Parallel Processing
ETL	Extract Transform Load
NoSQL	Not only Structured Query Language
ODBC	Open Database Connectivity
JDBC	Java Database Connectivity
ACID	Atomocity, Consistency, Isolation, Durability
ORC	Optimized Row Columnar
API	Application Programming Interface
ELT	Extract, Load, Transform
HTTPS	Hypertext Transfer Protocol Secure
CSV	Comma Separated Values
OLTP	On-Line Transaction Processing
OLAP	On-Line Analytical Processing
RDBMS	Relational Database Management System
REST	Representational State Transfer
GPDR	General Data Protection Regulation
FAIR	Findable, Accessible, Interoperable, Reusable
MRE	Medical Research & Education
SPARQL	SPARQL Protocol and RDF Query Language

XML	Extensible Markup Language
RDF	Resource Description Format
URI	Uniform Resource Identifier
DASTA	Český národní datový standard pro výměnu informací ve zdravotnictví
IPZV	Institut postgraduálního vzdělávání ve zdravotnictví
EDIFACT	Electronic Data Interchange for Administration, Commerce and Transport
DTD	Document Type Definition
XSD	XML Schema Definition
NZIS	Národní zdravotnický informační systém
DICOM	Digital Imaging and Communications in Medicine
MRI	Magnetic Resonance Imaging
ARC	American College of Radiology
NEMA	National Electrical Manufacturers Association
HL7	Health Level Seven International
ANSI	American National Standards Institute
CDA	Clinical Document Architecture
C-CDA	Consolidated Clinical Document Architecture
FHIR	Fast Healthcare Interoperability Resources
CCOW	Clinical Context Object Workgroup
IBM	International Business Machines
HL7	Health Level Seven International
ANSI	American National Standards Institute
ORM	Object Relational Mapping
HTML	Hypertext Markup Language
MVC	Model-View-Controller
URL	Uniform Resource Locator
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
XQuery	XML Query
FLWOR	For, Let, Where, Order by, Return
RES-Q	Registry of Stroke Care Quality
DDL	Data Definition Language
ESA	European Stroke Organisation
HDFS	Hadoop Distributed File System

Literatura

[1] *Evolution* [online]. c2023. Dostupné z: https://iceberg.apache.org/docs/latest/evolution/.

[2] *Introduction to data lakes* [online]. Databricks Inc., c2022. Dostupné z: https://www.databricks.com/discover/data-lakes/introduction.

[3] *Vývoj DASTA* [online]. Česká společnost zdravotnické informatiky a vědeckých informací, c2020. Dostupné z: https://dastacr.cz/info-1.html.

[4] *About DICOM: Overview* [online]. The Medical Imaging Technology Association (MITA), a division of NEMA, serves as the DICOM Secretariat, c20. Dostupné z: https://www.dicomstandard.org/about-home.

[5] *Clinical Context Object Workgroup (CCOW)* [online]. Gartner, Inc., c2023. Dostupné z: https://www.gartner.com/en/information-technology/glossary/ccow-clinical-context-object-workgroup.

[6] *FAIR data* [online]. Univerzita Karlova, 2022. Dostupné z: https://openscience.cuni.cz/OSCI-64.html.

[7] *Medallion Architecture* [online]. Databricks Inc., c2022. Dostupné z: https://www.databricks.com/glossary/medallion-architecture.

[8] *Objektové úložiště? Vítejte ve době cloudové!* [online]. 2020. Dostupné z: https://www.3s.cz/cs/odborna-sekce/detail/id/168-objektove-uloziste-vitejte-ve-dobe-cloudove.

[9] *DASTA* [online]. Česká společnost zdravotnické informatiky a vědeckých informací, c2020. Dostupné z: https://dastacr.cz/.

[10] *File Format* [online]. 2022. Dostupné z: https://parquet.apache.org/docs/file-format/.

[11] *Optimistická metoda souběžného zpracování* [online]. Microsoft Corporation, 2023. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/framework/data/adonet/optimistic-concurrency.

[12] *About HL7* [online]. Health Level Seven, Inc., c2023. Dostupné z: http://www.hl7.org/about/index.cfm.

[13] *MPP Database* [online]. c2022. Dostupné z: https://www.sisense.com/glossary/mpp-database/.

[14] *U.S. Census Bureau* [online]. Statistical Atlas of the United States Based on the Results of the Ninth Census, 1895. Dostupné z: https://www.census.gov/history/pdf/1890statisticalcompendium.pdf.

[15] *HL7 Standards - Section 1: Primary Standards* [online]. Health Level Seven, Inc., c2007–2023. Dostupné z: http://www.hl7.org/implement/standards/product_section.cfm?section=1.

[16] *Table utility commands* [online]. c2023. Dostupné z: https://docs.delta.io/latest/delta-utility.html.

[17] *What is Parquet?* [online]. Databricks Inc., c2023. Dostupné z: https://www.databricks.com/glossary/what-is-parquet.

[18] *Table batch reads and writes* [online]. c2023. Dostupné z: https://docs.delta.io/latest/delta-batch.html.

[19] *Best Practice 12.1 - Decouple storage from compute* [online]. Amazon Web Services, Inc., c2022. Dostupné z: https://docs.aws.amazon.com/wellarchitected/latest/analytics-lens/best-practice-12-1.html.

[20] *What Is a Data Warehouse?* [online]. Oracle Corporation, c2022. Dostupné z: https://www.oracle.com/cz/database/what-is-a-data-warehouse/.

[21] *CDA® Release 2* [online]. Health Level Seven, Inc., c2023. Dostupné z: http://www.hl7.org/implement/standards/product_brief.cfm?product_id=7.

[22] *RES-Q: registr kvality iktové péče* [online]. c2020–2022. Dostupné z: https://www.cmp.cz/registr-res-q.

[23] *What is RES-Q?* [online]. 2015. Dostupné z: https://qualityregistry.eu/the-project/about-uk.

[24] *Current Edition* [online]. The Medical Imaging Technology Association (MITA), a division of NEMA, serves as the DICOM Secretariat, c2023. Dostupné z: https://www.dicomstandard.org/current.

[25] *What Is a Lakehouse?* [online]. Databricks Inc., 2020. Dostupné z: https://www.databricks.com/blog/2020/01/30/what-is-a-data-lakehouse.html.

[26] *Fivetran data analyst survey.* [online]. 2020. Dostupné z: https://www.fivetran.com/blog/analyst-survey.

[27] *Constraints* [online]. c2023. Dostupné z: https://docs.delta.io/latest/delta-constraints.html.

[28] Armbrust, M. et al. Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics. 2020.

[29] Bartley, K. *ETL vs ELT: What's the Difference?* [online]. c2023. Dostupné z: https://rivery.io/blog/etl-vs-elt/.

[30] Bernard, B. *Úvod do architektury MVC* [online]. 2009. Dostupné z: https://zdrojak.cz/clanky/uvod-do-architektury-mvc/.

[31] Campbell-Kelly, M. et al. *Computer*. Third edition edition, 2014. ISBN 978-0813345901.

[32] Joshi, U. *Delta Cloning in Azure Databricks* [online]. 2022. Dostupné z: https://medium.com/globant/delta-cloning-in-azure-databricks-7e86d21b2606.

[33] Kava, P. – Gong, C. *Build a Lake House Architecture on AWS* [online]. Amazon Web Services, Inc., 2021. Dostupné z: https://aws.amazon.com/blogs/big-data/build-a-lake-house-architecture-on-aws/.

[34] Kosek, J. *Ukládání a vyhledávání XML dat* [online]. 2021.

[35] Kumpatla, A. *How to change Column type in Delta Table* [online]. 2022. Dostupné z: https://www.projectpro.io/recipes/change-column-type-delta-table.

[36] Nemri, C. *Transactional Data Lakes — a Comparison of Apache Iceberg, Apache Hudi and Delta Lake* [online]. Dostupné z: https://medium.com/geekculture/transactional-data-lakes-a-comparison-of-apache-iceberg-apache-hudi-and-delta-lake-9d7e58fd229b.

[37] Peternel, G. R. *The Fundamentals of Data Warehouse + Data Lake = Lake House* [online]. 2021. Dostupné z: https://towardsdatascience.com/the-fundamentals-of-data-warehouse-data-lake-lake-house-ff640851c832.

[38] Shankar, D. *Data lake Table Formats — Hudi vs Iceberg vs Delta Lake* [online]. 2022. Dostupné z: https://medium.com/mlearning-ai/data-lake-table-formats-hudi-vs-iceberg-vs-delta-lake-2cbf3a695f4e.

[39] Souza, A. *Data Management: From Data to Lakehouse* [online]. 2022. Dostupné z: https://medium.com/blog-do-zouza/gerenciamento-de-dados-dos-dados-ao-lakehouse-6bcef2d7d103.

[40] Stedman, C. *Data Lake* [online]. 2021. Dostupné z: https://www.techtarget.com/searchdatamanagement/definition/data-lake.

[41] Včelák, P. Rozsáhlé informační systémy – standardizace metadat, 2020.

[42] Weller, K. *Apache Hudi vs Delta Lake vs Apache Iceberg - Lakehouse Feature Comparison* [online]. 2023. Dostupné z: https://www.onehouse.ai/blog/apache-hudi-vs-delta-lake-vs-apache-iceberg-lakehouse-feature-comparison.

[43] Žák, K. *Historie relačních databází* [online]. Internet Info, s.r.o., 2001. Dostupné z: https://www.root.cz/clanky/historie-relacnich-databazi/.

Přílohy

A Uživatelská dokumentace

Celý vývoj aplikace probíhal na operačním systému *Windows 10* ve vývojovém prostředí *IntelliJ IDEA*. Výsledkem je multiplatformní *Java* aplikace, takže ji lze spustit v libovolném prostředí. Využívá *Spring Boot* framework, který má integrovaný servlet kontejner *Tomcat*, takže odpadá nutnost konfigurace webového serveru. Ke své činnosti však používá relační databázi, takže je nutná konfigurace a spuštění *MySQL* serveru. O založení databáze a tabulek se stará již samotná aplikace.

Hlavním cílem bylo vytvořit koncept, který ukáže vhodnost úložiště *Data Lakehouse* pro oblast medicínských dat. Z toho důvodu není používání aplikace detailně popisováno. Mnoho informací o používání je již zmíněno v podkapitole 5.3 popisující moduly nebo v kapitole 6, která popisuje práci s úložištěm. Dále každá sekce v administraci obsahuje popis, kde je uživatel seznámen s významem a základním používáním dané sekce. Všechny ovládací prvky (formulářové prvky, tlačítka apod.) jsou řádně popsány a u složitějších prvků nechybí nápověda, která popisuje význam nebo dopad na samotné úložiště.

A.1 Sestavení aplikace

K sestavení aplikace je potřeba *Java 11 (JDK)*, *Maven* a *Docker*. V rootu aplikace je skript `run_docker.sh`, který musí být spuštěn v příkazové řádce (*Bash*). Před samotným spuštěním musí být změněna přístupová práva k tomuto souboru, aby byl skript spustitelný. Změna práv a spuštění skriptu je zobrazeno na ukázce příkazové řádky 1. Pokud dojde ke správnému spuštění skriptu, tak se v *Dockeru* vytvoří nový kontejner `mysqldb`, do kterého je jako první nasazen a následně spuštěn `image` s *MySQL serverem*. Následně je sestavena aplikace, vytvořen `image`, nasazen do nového kontejneru `app` a automaticky spuštěn.

```
1 chmod +x run_docker.sh
2 ./run_docker.sh
```

Listing 1: Spuštění skriptu pro sestavení a spuštění aplikace.

Pokud uživatel chce mít větší kontrolu nad sestavením a následným spuštěním aplikace, tak může spouštět jednotlivé příkazy obsažené ve skriptu

`run_docker.sh` postupně (viz ukázka kódu 2). Pomocí prvního příkazu se v *Dockeru* vytvoří síť `data-lakehouse-network`, kterou následně používá nástroj *Docker Compose*. Následně dojde k vytvoření kontejneru s *MySQL serverem* a k sestavení aplikace. Čtvrtý příkaz vytvoří `image` a následně kontejner pro aplikaci. Definice vytvoření nového `image` je v souboru `Dockerfile`, který se také nachází v rootu aplikace. Poslední příkaz spustí nástroj *Docker Compose*, jehož činnost je definována v souboru `docker-compose.yaml`, který je také umístěn v rootu aplikace. Pomocí tohoto souboru *Docker compose* spustí oba dva kontejnery se správnou konfigurací a umožní komunikaci mezi nimi. Díky tomu může *aplikace* komunikovat s *MySQL databází*.

```
#!/bin/bash
# build and run spring application in docker

echo "START - CREATING NETWORK FOR DOCKER-COMPOSE";
docker network create lakehouse-network
echo "FINISHED - CREATING NETWORK FOR DOCKER-COMPOSE";

echo "START - CREATING CONTAINER WITH MYSQL"
docker pull mysql:8
echo "FINISHED - CREATING CONTAINER WITH MYSQL"

echo "START - BUILDING WEB APPLICATION";
mvn clean install
echo "FINISHED - BUILDING WEB APPLICATION";

echo "START - CREATE DOCKER IMAGE WITH APPLICATION";
docker build -t lakehouse .
echo "FINISHED - CREATE DOCKER IMAGE WITH APPLICATION";

echo "START - RUN APPLICATION WITH MYSQL VIA DOCKER-COMPOSE";
docker-compose up
echo "APPLICATION FINISHED";
```

Listing 2: Bash skript pro sestavení aplikace a spuštění v Dockeru.

A.2 Používání aplikace

Po spuštění aplikace může uživatel přistoupit na URL adresu `http://localhost:8080` a pokračovat přihlášením do aplikace. V případě neexistence uživatelského účtu se musí nejdříve *registrovat*. Na registrační formulář se lze dostat z úvodní obrazovky stejně jako na formulář pro *zapomenuté heslo*. Po úspěšném přihlášení lze aplikaci plně využívat.

Pro testovací účely jsou po inicializaci aplikace založeny čtyři uživatelské

účty (viz tabulka 1) a u všech bylo vyplněno *IntelliJ* jako odpověď na kontrolní otázku. Je připraveno i úložiště *Data Lakehouse*. Byly založeny dvě Delta tabulky (pro záznamy typu DASTA a RESQ) a do nich nahrány příslušné medicínské záznamy. Při automatickém sestavení a spuštění aplikace *Dockerem* je připravené úložiště nahráno do kontejneru s aplikací.

V průběhu přípravy testovacího úložiště bylo potřeba aplikovat regulární výrazy, takže základní sada výrazů se vytváří při inicializaci aplikace a je aplikována na obě tabulky. Pro ověření funkčnosti je připravena i sada XQuery výrazů a nechybí ani sada SQL dotazů, která je dostupná pod sekcí *Seznam SQL dotazů*.

Přihlašovací jméno	**Heslo**	**Role**
lmoucka@students.zcu.cz	qwertzuiop	ADMIN
second.user@students.zcu.cz	qwertzuiop	USER
third.user@students.zcu.cz	qwertzuiop	USER
fourth.user@students.zcu.cz	qwertzuiop	USER

Tabulka 1: Vytvořené uživatelské účty po inicializaci aplikace.

Aplikace využívá vizualizaci poskytovaného API pomocí modulu *SpringDoc OpenAPI UI*. Po přístupu na URL adresu `http://localhost:8080/swagger-ui/index.html` jsou viditelné všechny endpointy a používané transportní objekty. Vizualizace je dostupná jen po autentizace a odkaz je dostupný i skrze administraci.

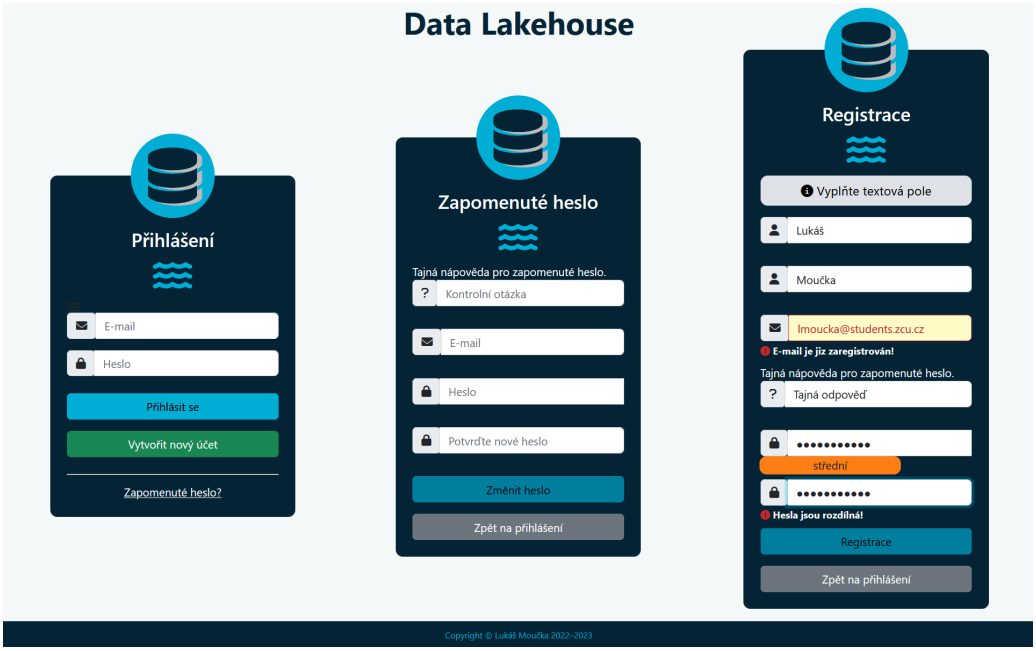
A.3 Spuštění aplikace v lokálním prostředí

Po dobu vývoje a testování byla aplikace spouštěna ve vývojovém prostředí nebo příkazové řádce a *MySQL* databáze běžela v *Dockeru*. V tomto režimu aplikace fungovala dle očekávání, a proto je níže popsán i tento postup.

Nejdříve je nutné změnit cestu k databázi, která je definována v souboru `application.properties` – cesta je zde připravena, takže ji stačí odkomentovat a pokračovat postupem popsaném v podkapitole A.1. Dojde ke spuštění databáze v *Dockeru*, ale kvůli změněné cestě se aplikace v *Dockeru* nespustí. Díky tomu může být aplikace spuštěna ve vývojovém prostředí nebo skrze příkazovou řádkou spuštěním skriptu `run_local.sh`, který je umístěn v rootu aplikace.

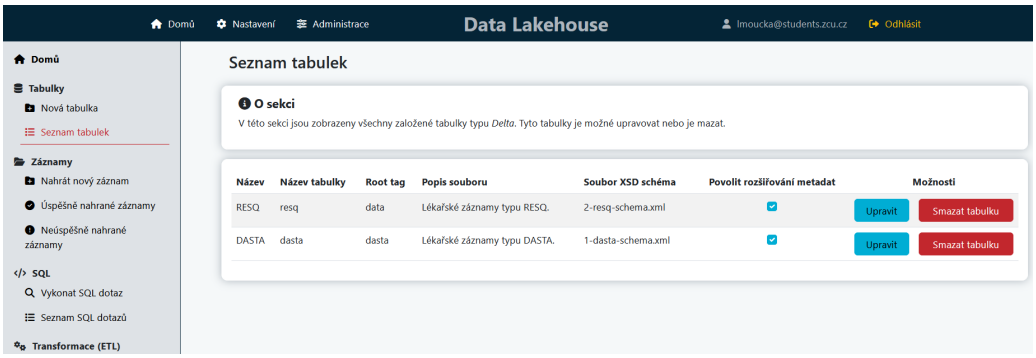
B Obrazová příloha

B.1 Přihlášení, zapomenuté heslo a registrace



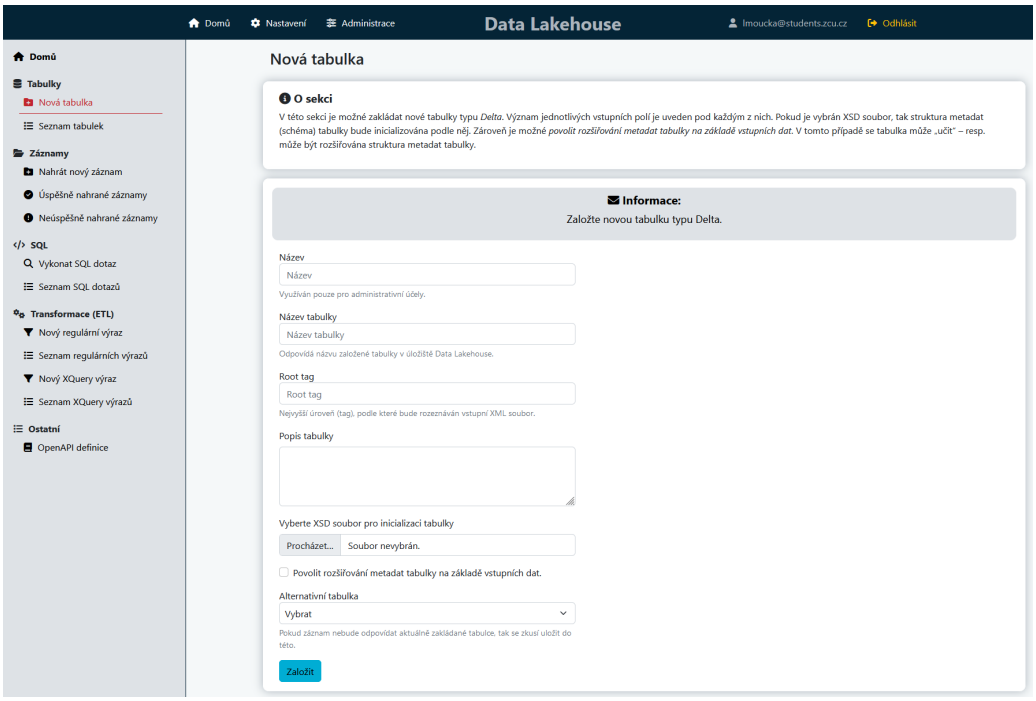
Obrázek 1: Na jednom screenshotu je zobrazeno přihlášení, zapomenuté heslo a registrace nového uživatele. V aplikace je každá z těchto akcí samozřejmě zobrazena samostatně.

B.2 Seznam tabulek



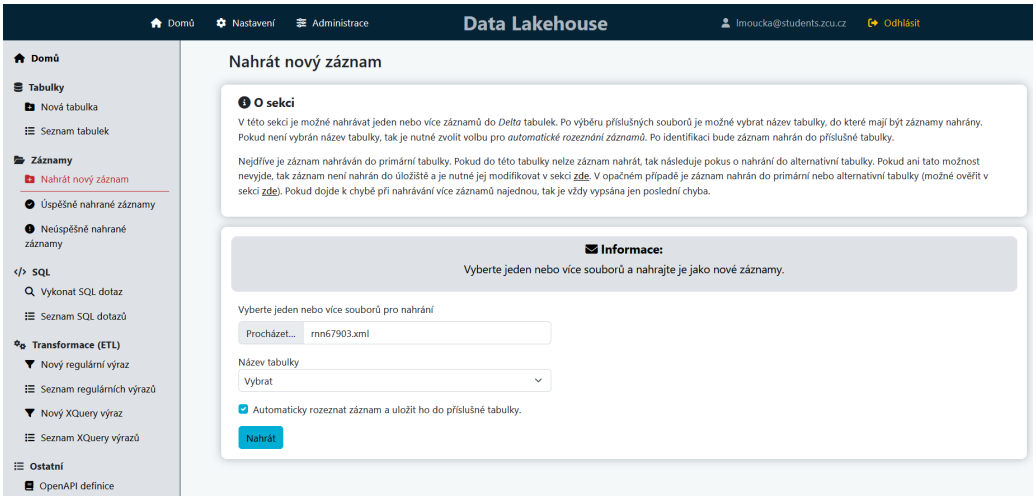
Obrázek 2: Po přihlášení je uživatel přesměrován na přehled založených Delta tabulek. Jednotlivé tabulky může modifikovat nebo je smazat.

B.3 Založení nové tabulky



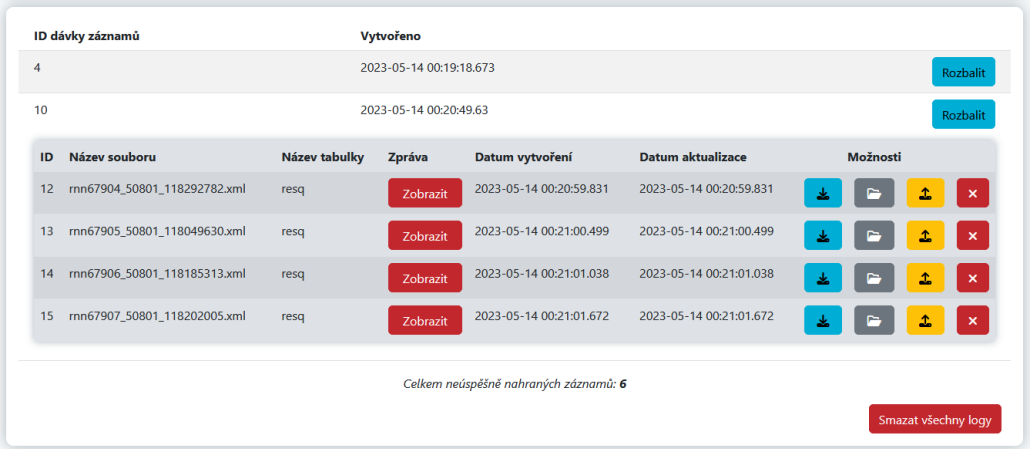
Obrázek 3: Sekce pro založení nové Delta tabulky do úložiště Data Lakehouse.

B.4 Nahrání nového záznamu



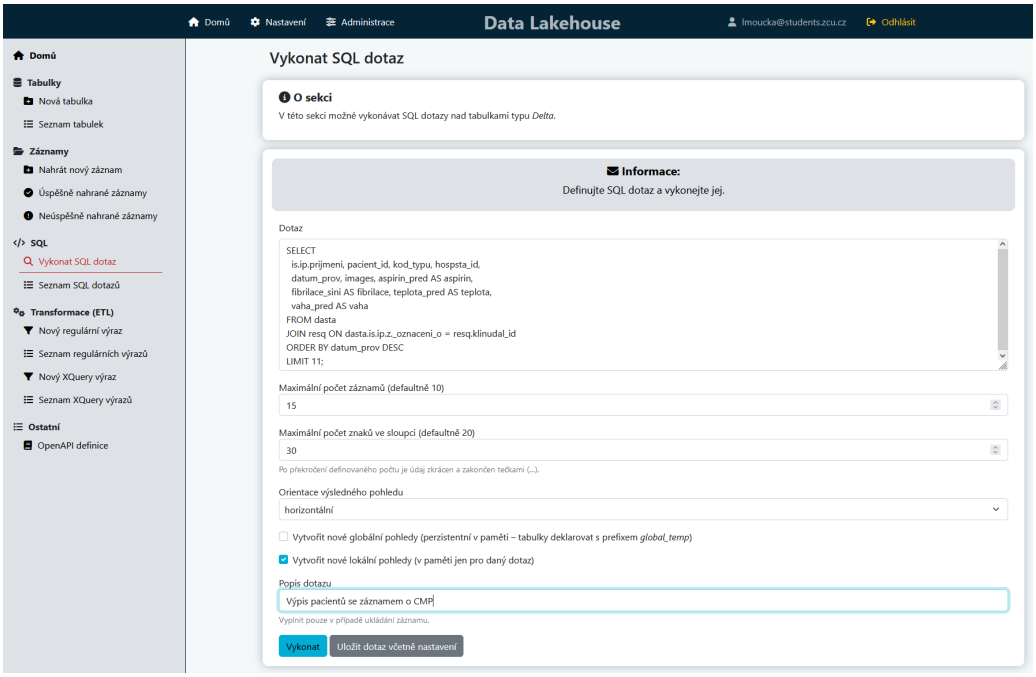
Obrázek 4: Sekce pro nahrávání nových záznamů do Delta tabulek.

B.5 Neúspěšně nahrané záznamy



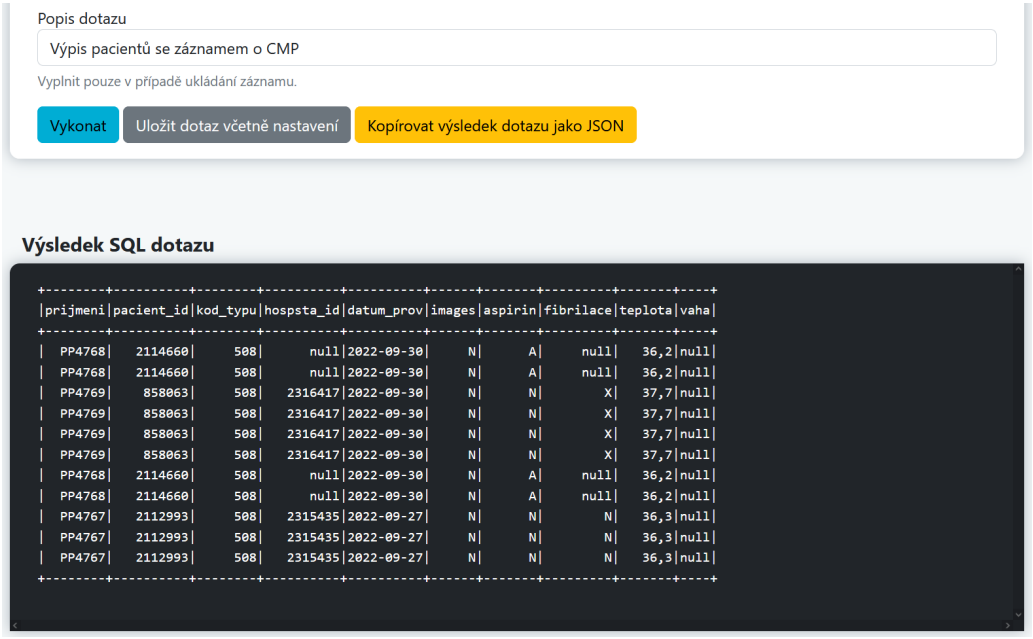
Obrázek 5: Detail neúspěšně nahraných medicínských záznamů (logů) z nahrávané dávky. U každého logu je možnost stažení záznamu a jeho opětovné nahrání po úpravách. Nechybí ani možnosti pro mazání logů.

B.6 Zadání SQL dotazu



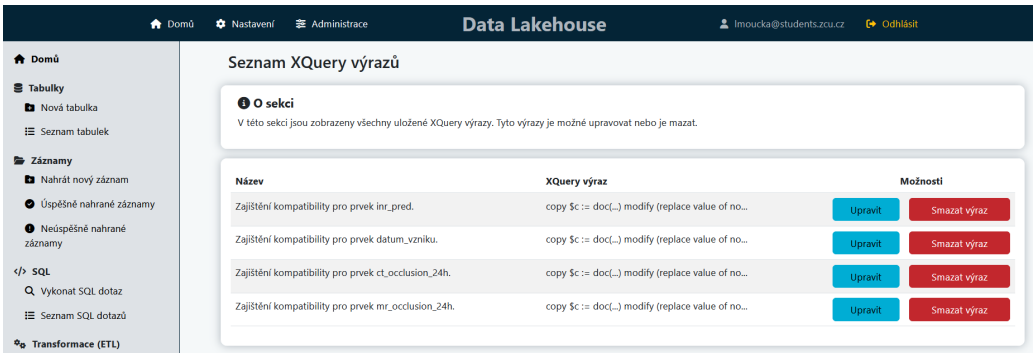
Obrázek 6: Sekce pro zadání SQL dotazu s možností jej uložit pro pozdější využití.

B.7 Výsledek SQL dotazu



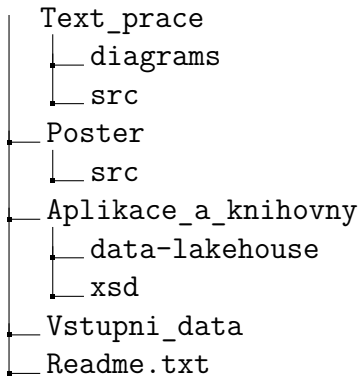
Obrázek 7: Detail výsledku SQL dotazu je uživateli zobrazen ve stejné sekci jako na obr 6. Nechybí možnost pro zkopírování výsledku ve formátu JSON a uložení dotazu pro opětovné vykonání nebo modifikaci.

B.8 Seznam XQuery výrazů



Obrázek 8: Sekce vypisující uložené XQuery výrazy, které jsou při nahrávání aplikovány na záznamy. Stejná sekce je i pro regulární výrazy.

C Struktura odevzdávaného archivu



C.1 Obsah adresářů

Obsah jednotlivých adresářů je následující:

- `Text_prace/diagrams` – zdrojové soubory pro veškeré diagramy vytvořené pro text diplomové práce za pomocí online nástroje *Draw.io*.
- `Text_prace/src` – zdrojové soubory sloužící pro LaTeX kompilátor k sestavení výsledného PDF.
- `Poster/src` – zdrojové soubory včetně použitého diagramu pro vygenerování výsledného PDF posteru.
- `Aplikace_a_knihovny/data-lakehouse` – Aplikace vzniklá v rámci diplomové práce. V adresáři aplikace je dostupný adresář `lakehouse`, který obsahuje založené *Delta* tabulky z přiložených XSD schémat (adresář `xsd` v rootu aplikace). Jedná se o připravené *Data Lakehouse* úložiště, které obsahuje nahrané záznamy z adresáře `Vstupni_data` umístěném v rootu aplikace.
- `Aplikace_a_knihovny/xsd` – XSD schémata pro inicializaci *Delta* tabulek.
- `Vstupni_data` – medicínské záznamy typu DASTA a RESQ (pocházející z registru RES-Q).
- `Readme.txt` – Tento soubor popisující obsah jednotlivých adresářů.