Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Návrh a implementace softwarového nástroje pro porovnání multikriteriálního filtrování nad strukturovanými daty

Bc. Roman Kučera

Diplomová práce

2018

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2017/2018**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Roman Kučera**
Osobní číslo:	**I16229**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Návrh a implementace softwarového nástroje pro porovnání multikriteriálního filtrování nad strukturovanými daty**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

V teoretické části práce budou popsány vybrané nástroje a přístupy pro multikriteriální filtrování dat se zaměřením na jejich výkonnost. Pozornost bude rovněž věnována popisu, jak tyto nástroje využít v praxi.
Praktická část bude zaměřena na vypracování softwarového nástroje umožňující porovnání vybraných přístupů a nástrojů pro multikriteriální filtrování nad strukturovanými daty. Půjde především o porovnání časové náročnosti filtrovaných dat, a to jak v jazyku JAVA (například pomocí datových proudů) tak s využitím specifických fulltextových databází.

Rozsah grafických prací:

Rozsah pracovní zprávy: **60 stran**

Forma zpracování diplomové práce: **tištěná**

Seznam odborné literatury:

KUĆ, Rafa a Marek ROGOZIŃSKI. Elasticsearch server. S.l.: Packt Publishing Limited, 2013. ISBN 1849518440.
WARBURTON, Richard. Java 8 lambdas. Beijing: O'Reilly, 2014. ISBN 1449370772.

Vedoucí diplomové práce: **Ing. Jan Fikejz, Ph.D.**
Katedra softwarových technologií

Datum zadání diplomové práce: **30. října 2017**
Termín odevzdání diplomové práce: **18. května 2018**

Ing. Zdeněk Němec, Ph.D.
děkan

L.S.

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2017

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 10. 5. 2018

Roman Kučera

PODĚKOVÁNÍ

Chtěl bych poděkovat svým blízkým, kteří mě podporovali po celou dobu mého studia. Dále bych rád poděkoval panu Ing. Janu Fikejzovi Ph.D., vedoucímu mé diplomové práce, za rady během vypracovávání této práce.

ANOTACE

Cílem práce je návrh a implementace softwarového nástroje v programovacím jazyce Java, který bude sloužit k porovnání časové náročnosti multikriteriálního vyhledávaní nad strukturovanými daty. Vyhledávání bude realizováno pomocí specifické fulltextové databáze Elasticsearch a také pomocí datových proudů a predikátů v programovacím jazyce Java.

KLÍČOVÁ SLOVA

Big data, Elasticsearch, Java proudy,

TITLE

Design and implementation of a software tool to compare multi-criteria filtering with structured data

ANNOTATION

The main goal of this thesis is to design and implement software tool in Java programming language, which will provide time consumption comparation for multi-criteria searching in structured data. Searching will be realized with specific full text database Elasticsearch and with Java streams and predicates.

KEYWORDS

Big data, Elasticsearch, Java streams

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM GRAFŮ

SEZNAM ZKRATEK

API	Application programming interface
CRUD	Create, Read, Update, Delete
CSV	Comma-separated values
DBMS	Database-managment system
DSL	Domain Specific Language
DVD	Digital video disc
HDD	Hard disk drive
HTTP	Hypertext Transfer Protocol
IDE	Integrated development environment
IoT	Internet of Things
JDBC	Java Databse Connectivity
JPA	Java Persistence API
JSON	JavaScript Object Notation
NoSQL	non Structured Query Language
ORM	Obejct-relation mapping
REST	Representation State Transfer
SQL	Structured Query Language
SSD	Solid State Drive
USA	United States of America
XML	Extensible Markup Language
YAML	YAML Ain’t Markup Language

ÚVOD

Se stále se zvětšujícím množstvím elektronických zařízení, senzorů a webových aplikací, dochází v posledních letech k velkému nárůstu generovaných dat, které je potřeba nějakým způsobem ukládat a následně v nich efektivně vyhledávat bez ohledu na to o jaký typ dat se jedná. Z toho důvodu dochází k vývoji v databázových systémech a v oblasti vyhledávácích nástrojů, které jsou navrženy pro efektivní ukládání a vyhledávání v těchto datech.

Jakým způsobem docházelo a stále dochází k vývoji v generování množství dat a jejich uchovávání se věnuje začátek teoretické části. Následně jsou představeny tradiční SQL databáze s jejich vlastnostmi a případy použití. Poté je pozornost věnována NoSQL databázím a jejím rozdílům oproti relačním databázím. Na závěr první kapitoly je představena jedna z efektivních technologií, jež umožňuje účinné vyhledávání ve velkém množství dat. Jedná se o Apache Lucene na kterém je vyvíjena celá řada dalších nástrojů a jedním z nich je Elasticsearch, jež je použit v rámci této práce a pozornost mu je věnována i v další kapitole.

Praktická část popisuje samotný návrh a implementaci aplikace pro porovnání časové náročnosti vyhledávacích operací nad strukturovanými daty. Pro vyhledávání jsou použity dva přístupy. Prvním z nich je využití klasické relační databáze ve spojení s datovými proudy a predikáty v programovacím jazyce Java. Druhým přístupem je využití specializovaného nástroje Elasticsearch, do kterého jsou pomocí souvisejících nástrojů importována data z relační databáze. Elasticsearch je technologie vyvinuta nad nástrojem Apache Lucene a poskytuje účinný způsob, pro vyhledávání ve strukturovaných nebo nestrukturovaných datech. Na začátku této části práce jsou popsány použité nástroje a technologie spolu s jejich konfigurací. Následně je popsán vývoj aplikace a její fungovaní včetně demonstrace grafického uživatelského prostředí. V závěru této části je popsána metodika, jakou bylo prováděno porovnání časové náročnosti, spolu s výsledky, které jsou reprezentovány pomocí grafů pro větší přehlednost a jednoznačnost.

1 BIG DATA

Teoretická část se zabývá vývojem v oblasti big data z pohledu množství uchovávaných dat, protože jejich objem se neustále navyšuje. S tím úzce souvisí potřeba efektivního vyhledávání v těchto datech. V druhé části je popsáno, jaké jsou možnosti pro uchování dat a pozornost je zejména věnována přehledu specializovaných nástrojů pro vyhledávání v nich.

1.1 Vývoj v oblasti big data

Nyní bude stručně probrána historie tvorby a ukládání dat, dále dojde k vysvětlení pojmu big data. Na závěr této kapitoly bude uvedeno zastoupení jednotlivých druhů dat, kde jsou data tvořena a ukládána, množství dat vygenerovaných za den a množství serveru dle největších firem v tomto odvětví

1.1.1 Historie dat a informací

Historicky se kolem nás už velmi dlouho dobu vyskytují data v určité formě. Můžeme se například podívat již do období paleolitu, kde došlo k objevům kostí, na kterých byly vyryty rýhy. Tím mohlo například docházet ke značení počtu zásob a na základě toho měli obyvatelé možnost dělat předpovědi, jak jim dlouho vydrží určité jídlo.[1]

Později docházelo k uchování dat v podobě různých zápisků případně knih. Při pohledu do posledních dvou staletí zjistíme, že se postupem času měnil způsob ukládání dat, jejich množství a také v neposlední řadě druh těchto dat. Již v roce 1928 byl navržen způsob uchování pomocí magnetické pásky. Princip digitálního uchovávání dat se zachoval až do současnosti, kde jsou pro tento účel používány pevné disky případně SSD disky.[1]

Jeden z prvních pokusů o vyčíslení množství informací, jež jsou produkovány byl proveden v Americe roku 1944. Knihovník Fremont Rider došel k závěru, že pro uchování veškeré akademické práce a vše co má nějakou informační hodnotu by bylo nutné každých 16 let zdvojnásobit kapacitu knihoven v USA.[1]

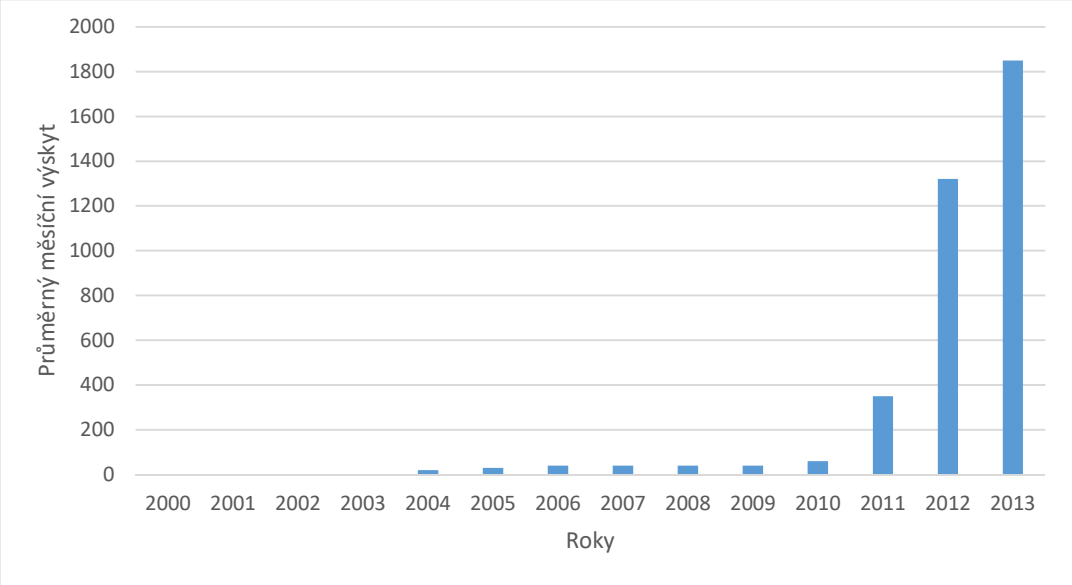
Okolo roku 1965 došlo k vybudování prvního datového centra. Bylo zřízeno vládou USA a sloužilo pro archivaci 742 milionů záznamů o daních a 175 milionů otisků prstů na magnetických páskách.[1]

Pravděpodobně k prvnímu použití pojmu „big data“ došlo v devadesátých letech minulého století. [1]

1.1.2 Co jsou to big data

Na začátek je ještě potřeba definovat pojem big data a co si pod tímto výrazem představit. K rozšíření výrazu došlo relativně před krátkou dobou v roce 2011, za iniciativy IBM a dalších technologických firem, jak lze vidět na Graf 1.1. [2]

Stejně jako docházelo k vývoji v oblasti big data tak stejný vývoj se odehrával i ve změnách samotné definice. Některé vysvětlení se zaměřovali, na co to vlastně jsou, mezitím jiné se snažili odpovědět co dělají.[2]



Graf 1.1 Četnost výskytu pojmu Big data v databázi knihovny ProQuest Zdroj: [2]

V každém případě velikost je první charakteristika, jež nás napadne při položení otázky „Co jsou big data?“. Nicméně jsou zde i další charakteristiky, například objem (volume), různorodost (variety) a rychlost (velocity). Tyto tři dimenze byly přijaty jako rámec pro popis big data. Například společnost Gartner definovala big data jako: „Big data jsou velko-objemové, velmi variabilní a rychlé sady informací, které vyžadují inovativní a cenově efektivní zpracování pro rozšířené analýzy a rozhodování“. [2]

Nyní si podrobněji vysvětlíme jednotlivé dimenze.

Objem

Jak se dá již z názvu poznat, objem představuje množství dat. Nejčastěji se v big data používají jednotky terabyte a petabyte. Definice na základě množství také záleží na dalších faktorech jako je například čas. To, co se dnes považuje jako big data nemusí platit již v budoucnu s příchodem větších úložných zařízení. [2]

Různorodost

Představuje rozmanitost ukládaných dat. Firmy pracují s celou řadou různých typů. Můžeme je rozdělit na strukturovaná, nestrukturovaná a na půl strukturovaná data. Nejmenší zastoupení představují data strukturovaná, jež jsou například data uspořádané v tabulkách v relačních databázích.

Obrázky, videa, zvuky nebo prostý text jsou zástupci z kategorie nestrukturovaných a typickým příkladem na půl strukturovaných dat jsou informace v podobě XML souboru.

Firmy ukládají velmi rozmanitá data, ať už se jedná o jejich shromažďování z různých senzorů, internetového obchodu či sociálních sítí a podobně. Vznikla tak potřeba pokročilé analýzy nad těmito daty, která následně firmám umožní dle výsledků upravit jejich business procesy. Další využití je v oblasti propagování produktů, nebo zjištění jakým způsobem se uživatelé pohybují na webových stránkách.[2]

Rychlost

Dimenze rychlosti znamená, s jakou četností jsou data vytvářena a s jakou rychlostí by měla být analyzována. Stále zvětšující se počet digitálních zařízení, ať už chytrých mobilních telefonů nebo dalších zařízení, které obsahují různé senzory a čidla vede k nevídané četnosti vzniku nových dat.

Za těchto podmínek je však potřeba provádět real-time analýzu takto sbíraných dat a získané informace použít pro přizpůsobení nabídek konkrétnímu uživateli na základě místa bydliště, či demografických informací.[2]

Tradiční úložiště dat nejsou schopny zvládat takovou četnost a množství generovaných informací, a proto se k těmto účelům používají technologie specializované na big data.

Kromě výše zmíněných tří základních dimenzí se také často uvádí ještě věrohodnost (veracity). Můžeme se setkat také ještě s dalšími kritérii jako je variabilita a složitost (variability and complexity), kterou představila společnost SAS. Firma Oracle představila ještě kritérium hodnoty (value). [2]

1.1.3 Množství generovaných dat a jejich původ

Velmi rychle se blížíme do nové části digitální věku. Svět kolem nás se mění, ať už z pohledu autonomních vozidel, robotů nebo inteligentních osobních asistentů pro chytrá domácí zařízení. Tento vývoj ovlivňuje a mění způsob jakým vedeme naše životy, jak pracujeme a jak se bavíme. S tím souvisí i neustále větší množství generovaných dat, které se je nutné nějakým způsobem uchovávat.[3]

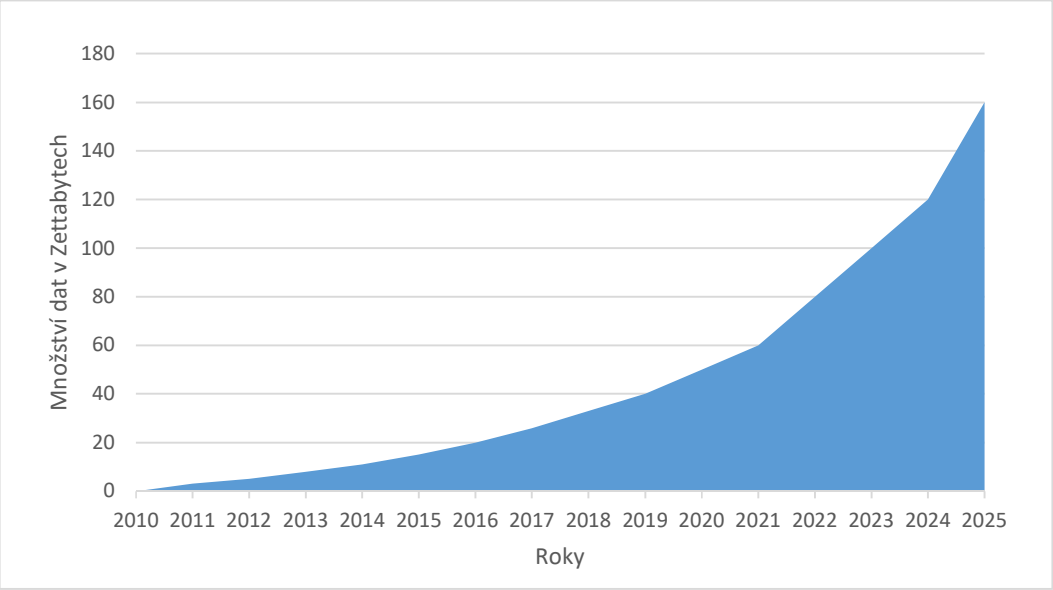
Vývoj ve způsobu ukládání dat

Způsob ukládání dat prošel a stále prochází vývojem. Před rokem 1980 byla většina dat uložena ve speciálních datacentrech, jež byla pro tento účel vystavěna. Jejich hlavní využití bylo v oblasti podnikání a přistup k nim byl možný pouze z místa kde se nacházela.[3]

Druhé období bylo v rozmezí let 1980 až 2000 a představovalo rozšiřování osobních počítačů a taktéž bylo možné posílat data po pomalé, ale stále se zvětšující internetové síti ke koncovým zařízením. Tato zařízení tak získala možnost ukládat a zpravovat data pro osobní použití běžnými uživateli. Díky tomuto pokroku se objevilo odvětví digitální zábavy, do kterého můžeme zařadit hudbu, filmy, nebo hry.[3]

Třetí a poslední období je od roku 2000 po současnost. S rozvojem bezdrátové sítě a vznikem a rozšíření rychlé internetové sítě začalo docházet k přesunu dat do takzvaných cloudových úložišť. Datacentra se taktéž přesouvají do cloudové infrastruktury skrze řešení od Amazonu, Googlu, či Microsoftu. Distribuce výpočetní síly se stále zvětšuje hlavně z důvodu čím dál větší popularity mobilních telefonů, chytrého nositelného příslušenství (hodinky, náramky) a konzolí pro hraní her. Tyto koncová zařízení ještě společně s tradičními osobními počítači stále potřebují pro svou funkčnost data, ale díky přesunu informací do cloudových služeb jsou nároky na velikost lokálního uložiště menší.[3]

V důsledku těchto změn došlo k velkému celosvětovému růstu množství dat. Z Graf 1.2 lze vidět, že dnes máme zhruba 20krát více dat, než bylo v roce 2010 a tento trend bud i nadále pokračovat, pravděpodobně ještě ve větším měřítku. [3]



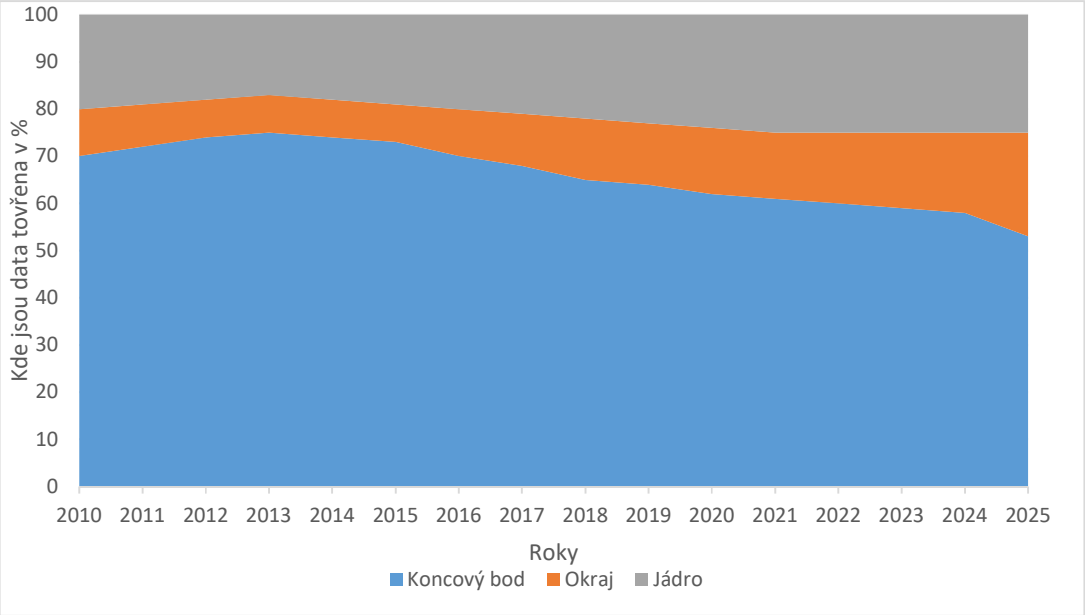
Graf 1.2 Vývoj množství dat v čase Zdroj: [3]

Kde jsou data tvořena

Současně s rostoucím množstvím generovaných dat dochází také ke změnám v rozložení vzniku dat. Vznik můžeme rozdělit do třech základních kategorií:

- Jádro – Jedná se o datacentra v profesionální sféře. Obsahují celou řadu cloudových uložišť (veřejná, osobní). Také sem můžeme zařadit kontrolní střediska, která se starají o elektrickou, či telefonní síť.
- Okraj – Do této kategorie lze zařadit podnikové servery, jež nejsou v hlavních datacentrech, která spadají do jádra.
- Koncový bod – Představuje všechna zařízení, která se nacházejí na konci internetové sítě. Může se jednat o počítače, mobilní telefony, kamery, chytrá auta, chytré nositelné příslušenství, případně o celou řadu senzorů. [3]

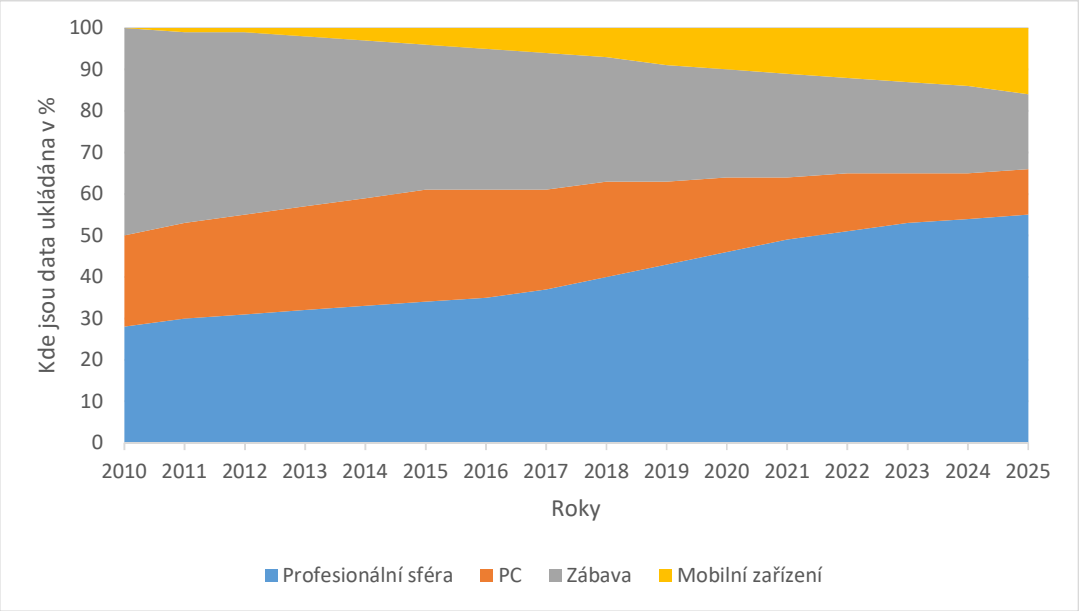
V níže přiloženém Graf 1.3 lze pozorovat, že zhruba od roku 2012 dochází ke zmenšování podílu koncových zařízení na tvorbě dat a předpokládá se, že tento vývoj bude pokračovat i nadále. V průběhu posledního desetiletí zažila koncová zařízení velký rozvoj, nicméně data z těchto zařízení jsou produkovány v malých signálech. Naproti tomu analýzy big data, cloudové aplikace a data reálného času způsobují stále se zvětšující podíl kategorií jádra a okraje. [3]



Graf 1.3 Vývoj místa tvorby dat v čase Zdroj: [3]

Kde jsou data ukládána

Další oblast, ve které můžeme pozorovat změny jsou zařízení na nichž jsou data ukládána. Od roku 1980 až do začátku nového tisíciletí tvořili největší podíl počítače a média sloužící pro zábavu. Se zlepšujícím se internetovým připojením klesá potřeba, aby byla data ukládána lokálně na počítačích, či mobilních zařízeních. Jak můžeme vidět na Graf 1.4 v roce 2012 téměř 50 % dat sloužilo pro zábavu, bylo to způsobeno hlavně velkou distribucí DVD a Blu-ray disků. Postupem času došlo k přesunu těchto dat do podoby online služeb, které nabízí muziku, filmy a podobně. V důsledku toho můžeme pozorovat zvyšující se podíl profesionální sféry a zmenšující se podíl medií a zařízení pro zábavu. [3]



Graf 1.4 Vývoj místa ukládání dat v čase Zdroj: [3]

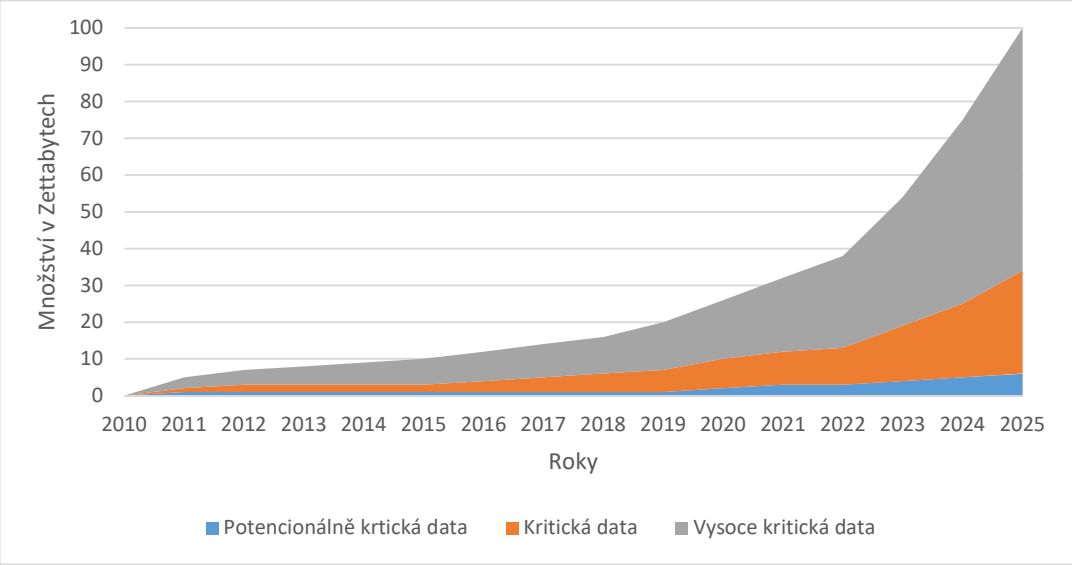
Kritičnost dat

K analýze dat se přistupuje dle jejich úrovně kritičnosti. Samotná úroveň je ovlivněna několika faktory jako zpracování v reálném čase, nízká odezva a také jaký následek by způsobila nedostupnost daných dat. Například výpadek nějaké aplikace používané ve zdravotnictví může mít mnohem větší následky než nedostupnost služby pro streamování hudby či videa.

Jedna věc je ztratit data v osobním počítači způsobené třeba nefunkčním pevným diskem, ale ztráta, či výpadek dat může způsobit i fyzickou újmu například v případě autonomních vozidel. Proto je kladen stále větší důraz na shromažďování dat a jejich analýzu, na vysokou spolehlivost, propustnost a dostupnost systému. [3]

Data můžeme z pohledu kritičnosti rozdělit do tří skupin a jejich zastoupení v čase je zobrazeno na Graf 1.5:

- Potencionálně kritická data, která mohou být důležitá pro pohodlný každodenní život.
- Kritická data, potřebná pro očekávaný průběh každodenního života.
- Vysoce kritická data mají okamžitý následek na zdraví uživatelů (letecká doprava, zdravotnické aplikace, kontrolní systémy). [3]



Graf 1.5 Kritičnost dat v čase Zdroj: [3]

Množství dat vyprodukovaných za den

Pro představu o množství dat v současné době, je vhodné uvést konkrétní příklady na největších firmách na světě.

Google zpracuje každý den 3.5 miliardy dotazů. K tomu je potřeba obrovské množství datových úložišť. Odhaduje se, že Google patří k jedné z největších firem v oblasti big data. Díky množství aplikací a funkcí, které nabízí se v roce 2009 připravoval na uchování jednoho exabytu dat, což je milión terabytu. Odhadem se Google musí nyní vypořádat s desítinásobkem této hodnoty.[4]

Amazon zpracuje data ze 152 miliónů objednávek, aby byl schopen uživatelům doporučit další produkty. Odhadem pracují s množstvím 1 exabytu dat.[4]

Facebook sesbírá 500 terabytů dat denně, která obsahují 2.5 miliardy příspěvků, 2.7 miliard označení „Líbí se mi“ a 300 miliónů obrázků. K roku 2012 firma ukládala 100 petabytů obrázků a videí (1 petabyte je 1000 terabytů).[4]

Množství serverů

Všechna vyprodukovaná data je potřeba ukládat na serverech. Amazon vlastní zhruba 1 400 000 serverů a má prvenství v této kategorii. Za ním jsou firmy Google a Microsoft kde obě mají přes milión serverů. Dalším v pořadí je Facebook, který jich má stovky tisíc.[4]

Další firmy již mají řádově méně serverů, například HP 380 000. Společnosti OVH.COM, Akamai, Yahoo!, SOFTLAYER a rackspace mají okolo 100 000 každá.[4]

2 DATABÁZE A JEJÍ DRUHY

Databáze představuje systém pro uchování záznamů v počítači. Může být považována za repositář pro kolekce digitálních dat. Uživatelé tohoto systému mohou provádět celou řadu operací, například:

- Přidávání a mazání souborů v databázi.
- Vkládání dat do existujících souborů.
- Získávat data ze souborů.
- Mazat a měnit data v souborech.

Jednoduše řečeno databáze představuje systém, jehož smyslem je uchovávat informace. Systém se skládá z několika částí jako jsou data, hardware, software a uživatelé.

Data

Jedná se o informace, které jsou v databázi uloženy a následně nad nimi mohou uživatelé provádět různé druhy operací.

Hardware

Obsahuje dvě hlavní komponenty:

- Sekundární úložné zařízení – typicky se jedná o HDD nebo případně SSD disky, na které jsou data ukládána společně se vstupně/výstupními zařízeními.
- Procesory a operační paměť, které jsou nezbytné k běhu databázového systému.

Software

Mezi fyzickou částí a koncovými uživateli se nachází vrstva softwaru známá jako DBMS (Database managment system). Všechny požadavky na databázi jsou vyřizovány právě díky DBMS. Jedná se o nejdůležitější část celého systému.

Uživatelé

Můžeme rozdělit na tři skupiny podle způsobu jakým přistupují k samotné databázi. První jsou programátoři aplikací, jež jsou zodpovědní za vývoj aplikací, které používají databázi.

Druhý typ jsou koncoví uživatelé přistupující k databázi pomocí nějakého nástroje, který byl například vyvinut uživateli z první skupiny.

Poslední druh uživatelů jsou databázoví administrátoři, jejichž úkol je zabezpečení a správa databáze.[5]

Databáze můžeme rozdělit na dva základná typy, jedná se o takzvané SQL databáze a NoSQL databáze, můžeme se také setkat s pojmem relační a nerelační databáze. Liší se způsobem, jakým jsou navrženy, jaká data obsahují a jakým způsobem je ukládají. Relační databáze obsahují strukturovaná data naproti tomu nerelační jsou dokumentově orientované a distribuované.[6]

2.1 SQL databáze

Nejprve se podíváme na jednu z hlavních vlastností, v které se liší SQL a NoSQL databáze. Jde o způsob, jakou strukturu dat ukládají.

Název SQL (Structured Query Language) je podle jazyku, který je používán pro práci s nimi. Jde o pevnější strukturovanou metodu ukládání dat, například jako telefonní seznam. Vyvinuta byla společností IBM po roce 1970. Relační databáze se skládají z tabulek, kde každá tabulka obsahuje řádky a sloupce. Každý řádek reprezentuje jeden záznam a každý sloupec typ záznamu. Může se například jednat o jméno, telefonní číslo či jiné údaje. Vazba mezi tabulkami a jednotlivými položkami se nazývá schéma. Schéma musí být definované dříve, než se můžou vložit jakékoliv informace.

Aby mohli být relační databáze efektivní, data v nich uložená musí být strukturovaná. Dobře navrhnuté schéma minimalizuje množství zbytečných dat a zabraňuje tabulkám, aby se rozcházely s realitou. To je důležité pro mnoho druhů podnikání, hlavně pro finanční oblast. Naproti tomu při špatně navrženém schématu může dojít k situaci, kdy bude nutné upravit strukturu tabulky později.

SQL je dotazovací jazyk používaný pro návrh relační databáze. V databázích jako MySQL, Oracle a další je použit pro získávání dat a jejich editaci, mazání, přidávání. Jde o lehký a deklarativní jazyk. Jedna z hlavních výhod je použití výrazu JOIN, pomocí něhož je možné získat data uložená napříč několika tabulkami v jednom dotazu.[6]

Přehled populárních relačních databází

- MySQL – Jedna z nejvíce populárních open source databází, vhodná pro použití s webovými stránkami a blogy.
- Oracle – Jde o objektově relační DBMS naprogramovanou v jazyce C++. Nabízí velmi dobrou zákaznickou podporu a celkový servis. Zdarma je dostupná verze s omezenou podporou hardwaru.
- IMB DB2 – Skupina databázových produktů od firmy IBM, navržena pro práci z big data.
- Sybase – Určena hlavně pro komerční využití na operačním systému Unix.
- MS SQL Server – Databáze od společnosti Microsoft pro komerční použití, která podporuje jak SQL, tak NoSQL architekturu.
- Microsoft Azure – Služba dostupná v cloudu, které podporuje jakýkoliv operační systém a umožňuje ukládat, analyzovat a škálovat data na jednom místě.
- MariaDB – Jde o komunitní odnož MySQL databáze.
- PostgreSQL – Komerční, objektově relační databáze, která používá jako doplněk k SQL procedurální programovací jazyky Perl a Python

Proč použít SQL databázi

V databázových technologiích nelze aplikovat metodu, že jedno řešení je nejlepší na všechny případy. Proto mnoho firem používá jak relační, tak nerelační databáze pro různé úlohy. Přesto, že NoSQL databáze získávají stále větší popularitu, v některých případech je lepší a vhodnější použít databázi relační.

U relačních databází probíhá veškeré zpracovaná v transakcích, které zajišťují atomicitu, konzistenci, izolaci a trvalost. Díky tomu se redukuje množství chyb a dochází k ochraně integrity dat v databázi.

Další ze scénářů, kdy je vhodné použít tento druh databáze je v případě strukturovaných a neměnných dat. Pokud nedochází k velkému růstu dat a pracuje se s konzistentními daty, není potřeba přecházet na nerelační databáze, které podporují různé druhy datových typů a vysoké množství přenášených dat.[6]

2.2 NoSQL databáze

Pojem NoSQL databáze zahrnuje široké množství různých databázových technologií, které byly vyvinuty pro potřeby moderních aplikací.

Vývojáři pracují s aplikacemi, které vytváří velké množství dat, jež jsou strukturovaná, na půl strukturovaná a nestrukturovaná, viz kapitola 1.1.2 Co jsou to Big data.

Dříve se při vývoji používal především tzv. vodopádový model, který spočíval ve dvanácti až osmnácti měsíčních cyklech. Nyní je velmi populární agilní vývoj, který je rozdělen na krátké sprinty, opakující se každý týden nebo čtrnáct dní.

Aplikace, jež dříve sloužili konečnému množství uživatelů jsou v dnešních dnech distribuovány jako služba. Na takové služby jsou kladeny požadavky, aby byly neustále dostupné z různých koncových zařízení pro milióny uživatelů po celém světě.

Společnosti se obrací k open source architekturám, protože jsou dobře škálovatelné. Jedná se především o multifunkční servery a služby dostupné v cloudu místo velkých jednoúčelových serverů.

Pokud není dopředu jasná struktura dat nebo je jich velké množství a jedná se o nestrukturovaná data není zde ta možnost vytvořit relační databázový model s jasně definovaným schématem. NoSQL databáze poskytují mnohem větší flexibilitu než databáze relační. Použití je také možné vysvětlit na příkladu ukládaní dat z webové stránky, kde jsou dohromady data o konkrétním článku a k tomu obrázky, počty sdílení na sociální sítě, odkazy a další informace. Z důvodu potřeby analyzovat takováto data došlo ke vzniku NoSQL databázovým systémům.

Jak fungují NoSQL databáze

Oproti relačním databázím, jejichž fungování a princip je vysvětlen v předešlé kapitole 2, NoSQL databáze nepracují s tabulkami nýbrž využívají dokumentově orientovaný přístup. Pomocí této architektury je možné uložit články, fotky, sdílení a video společně do jednoho dokumentu, který je možné jednoduše vyhledat, ale není zde potřeba mít ho zařazený do nějakého pole jako je tomu u databází relačních. Tento způsob je více intuitivní, ale je nutné podotknout, že je také zvýšena náročnost na zpracování a jsou větší nároky i na úložný prostor než v případě organizovaných dat u relačních databází.

Další z výhod, která je užitečná hlavně pro vývojáře je jednoduchost přístupu. Relační databáze je svázána s aplikací, ale u NoSQL je přístup k datům pomocí různých API a vývojář tak nemusí znát SQL jazyk a také nemusí znát architekturu databázového systému.

Druhy NoSQL databází

- Model klíč-hodnota je nejjednodušší NoSQL databáze, Každá položka v databázi je uložena jako klíč společně s hodnotou. Tento přístup používají úložiště Riak a Berkeley DB. Některé systému jako Redis, umožňují mít hodnotu konkrétního typu což přidává na funkcionalitě.
- Sloupcové úložiště jako je například Cassandra nebo HBase jsou optimalizovány pro dotazy nad velkými soubory dat a ukládají sloupce dat dohromady místo řádků.
- Databáze dokumentů páruje každý klíč s komplexní datovou strukturou nazvanou jako dokument. Dokument může obsahovat mnoho rozdílných párů klíč-hodnota, nebo klíč-pole, případně vnořené dokumenty.
- Grafové databáze jsou používány pro ukládání dat o síti, jako jsou propojení na sociálních sítích. Grafové úložiště je například Neo4J nebo Giraph.[7]

Přehled populárních nerelačních databází

- MongoDB – Jedná se o nejpopulárnější NoSQL systém, hlavně mezi startup firmami. Databáze je dokumentově orientovaná a dokumenty jsou ve formátu, který je podobný JSONu. Uloženy jsou v dynamických schématech. Jedná se o open source řešení a má dobrou zákaznickou podporu.
- CouchDB – Za vývojem stojí společnost Apache a je vhodná pro použití na webu. Pro uchování dat je využit formát JSON a jazyk Javascript pro indexaci, kombinaci a transformaci dokumentů. Dále je použita technologie HTTP pro poskytnutí API.
- HBase – Další z projektů firmy Apache, vyvíjena jako součást produktu Hadoop. Jedná se open source databázi, která využívá sloupcový model pro uložení dat. Je vyvinuta v programovacím jazyku Java.
- Oracle NoSQL – Snaha firmy Oracle vstoupit na půdu NoSQL databází.
- Cassandra DB – Ke vzniku došlo ve firmě Facebook, nyní se jedná o další projekt firmy Apache. Jde o distribuovanou databázi, která je dobrá pro správu velkého množství dat napříč servery. Dá se velmi dobře v případě potřeby škálovat. Používají jí například firmy Apple, Spotify, Instagram.
- Riak – Jedná se o open source databázi fungující na principu klíč-hodnota. Pro její vývoj je využit programovací jazyk Erlang.[6]

Proč použít NoSQL databázi

V případě, kdy jsou všechny části serverové aplikace navrhnuty tak, aby byl její běh rychlí, NoSQL databáze zajišťují, aby data nebyla úzkým místem. Big data jsou hlavním důvodem proč se zvyšuje jejich popularita, protože umožňují věci, které standardní relační databáze nedokáži.

Jsou velmi vhodné pro ukládání velkého množství dat, které mají velmi malou nebo žádnou strukturovanost. NoSQL databáze nadávají žádný limit na typ dat, které je možné v nich uchovat a také umožňují přidat nové rozdílné typy podle potřeb. S dokumentově orientovanými databázemi, je možné uložit data bez nutnosti definovat jejich typ.

Pro co nejlepší využití cloudových služeb a úložišť, je potřeba, aby byla data lehce rozdělitelná na více serveru a byla zde tak možnost škálovatelnosti. Databázový systém Cassandra je navržen právě pro toto využití a podporuje škálovatelnost napříč více datových center ve výchozím stavu.

V případě agilního vývoje založeném například na dvoutýdenních iteracích a potřebě dělat úpravy ve struktuře dat a minimalizovat čas kdy nebude služba dostupná hrozí riziko, že relační databáze budou tento cíl ztěžovat.[6]

2.3 Apache Lucene a vyhledávací nástroje

Za vývojem nástroje Apache Lucene stojí firma Apache Software Foundation zkráceně ASF.

Ke vzniku došlo v roce 1999 a jednalo se o charitativní organizaci, placenou z darů a korporátními partnery. V portfoliu této dobrovolnické organizace je přes 350 Open Source projektů, včetně Apache HTTP Server – nejpopulárnější webový server na světě.

ASF poskytuje zavedený rámec pro duševní vlastnictví a finanční příspěvky. Organizace má více než 620 členů a 5 500 přispěvatelů, kteří spolupracují na vývoji volně dostupných programů, jež jsou používány milióny uživateli po celém světě. [8]

2.3.1 Apache Lucene Core

Stejně jako ostatní nástroje vyvíjené společností ASF tak i Apache Lucene je open source projekt, který je možné zdarma stáhnout.

Jedná se o velmi výkonnou, fulltextovou vyhledávací knihovnu vyvinutou pomocí programovacího jazyku Java. K dispozici jsou i implementace v jiných programovacích jazycích. Tato knihovna je vhodná téměř pro každou aplikace, jež vyžaduje fulltextové vyhledávání.

Nabízí velkou možnost škálovatelnosti a vysoký výkon při indexování. Umožňuje zpracovat přes 150 GB dat za hodinu. Je nenáročný na operační paměť RAM, stačí mu po pouze 1 MB na heapu. Přírůstkové indexování je stejně rychlé jako dávkové indexování a velikost indexu je tvořena zhruba 20–30 % z velikosti indexovaných dat.

Dále poskytuje přesný, efektivní a výkonný vyhledávací algoritmus. Mezi jeho vlastnosti patří vracení nejlepšího výsledku na první pozici. Několik druhů vyhledávacích dotazů. Umožňuje řadit dle libovolných položek, spojit výsledky napříč více vyhledávání. Je rychlí a paměťově efektivní.

Jde o multi-platformní nástroj, který je možné použít jak v komerčních, tak open source projektech.[9]

Apache Lucene slouží jako základ pro množství nástrojů, které umožňují vyhledávat ve velkém množství dat. Nyní bude uveden přehled několika vybraných nástrojů.

Apache Nutch

Jedná o vyspělého open source internetového bota, který je připraven pro použití na produkčních prostředích. Je dobře rozšiřitelný a škálovatelný. Je rozdělen na dvě verze:

- Verze 1.x je závislá na datové struktuře Apache Hadoop, která je vhodná pro dávkové zpracování.
- Verze 2.x představuje alternativu k předchozí verzi, inspiruju se taktéž z verze 1.x, ale hlavním rozdílem je úložiště, jež je abstrahováno a není tedy požadována žádná specifická datová struktura. To je zajištěno pomocí nástroje Apache Gora pro správu persistence objektů. Díky tomu je možné implementovat velmi flexibilní model pro ukládání různých typů dat do rozdílných NoSQL databází.

Může běžet jak samostatně, tak být součástí již zmíněného Apache Hadoop, kde právě v tomto spojení má největší potenciál.[10][11]

Apache Solr

Je také open source platforma, sloužící k vyhledávání v datech, jež jsou uložena v systému Hadoop. Solr používá mnoho firem po celém světě, hlavně kvůli velmi dobré schopnosti zajistit full-textové vyhledávání a indexování téměř v reálném čase. Nezáleží, v jakém typu dat se vyhledává, ať už se jedná o text, či data z nejrůznějších senzorů, Solr je schopen poskytnout rychlé vyhledávání mezi nimi. Má dobrou automatickou opravu v případě výskytu nějaké chyby a také je snadno škálovatelný.

Dokumenty jsou do Apache Solr indexovány pomocí XML, JSON, CSV nebo binárních dat, vše přes protokol HTTP. Následně může uživatel vyhledávat v těchto datech s využitím HTTP GET metody. Výsledek je mu vrácen opět v XML, JSON, CSV nebo binární podobě.[12][13]

CrateDB

CrateDB je distribuovaná SQL databáze, jež je postavená na NoSQL. Díky této kombinaci není nutná znalost nových technologií, protože pro dotazy je použit SQL jazyk stejně jako v běžných relačních databázích. Nabízí navíc škálovatelnost a flexibilitu, která souvisí s NoSQL systémy. Stejně jako výše zmíněné produkty je pod licencí open source, nicméně firmy mají možnost zaplatit si zákaznickou podporu.

Mezi její klíčové vlastnosti patří již uvedená škálovatelnost, která je zajištěna pomocí automatického vyvažování dat. Stačí pouze přidat nový server do clusteru a databáze se o zbytek postará sama.

Dále pomocí automatické replikace se snižuje riziko nedostupnosti dat v případě výpadku nějakého prvku v clusteru.[14]

Elasticsearch

Nyní bude stručně představen nástroj Elasticsearch a dále v kapitole 3.1 je podrobnější popis spolu se souvisejícími nástroji, které byly využity v rámci diplomové práce.

Jedná se o distribuovaný, RESTful vyhledávací a analytický engine, jež je schopný pokrýt velké množství případů užití. Umožňuje vyhledávat mezi různými druhy dat, může se jednat o strukturovaná, nestrukturovaná, geografická, či metrická data. Elastisearch obsahuje několik implementací, jak uchovávat data, aby byla zajištěna rychlost jak pro full-text, tak pro číselná a geografická data.

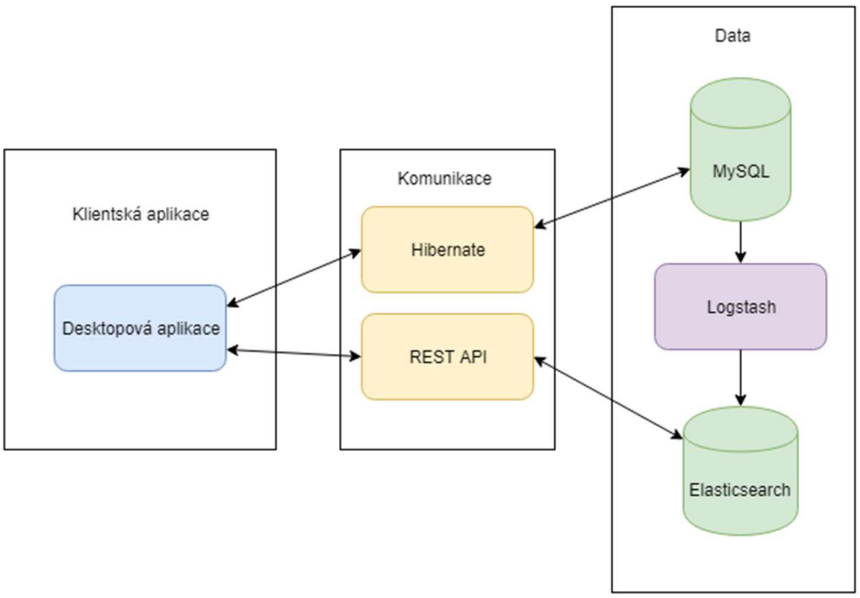
Obdobně jako u předešlých řešení nabízí velmi dobrou škálovatelnost a jeho chování je stejné bez ohledu, zda běží pouze na jednom prvku nebo na stovkách prvcích v clusteru. Automaticky rozhoduje, jak budou dotazy distribuovány mezi jednotlivé prvky, aby byl jeho běh vyvážený a nedocházelo k přetížení určitého prvku.

V důsledku automatické distribuce dat napříč prvky zabezpečuje dobrou dostupnost v případě výpadku.[15]

3 NÁVRH A IMPLEMENTACE

Tato část se věnuje implementaci aplikace v programovacím jazyce Java, která slouží jako vyhledávací nástroj pro strukturovaná data. Postupně bude popsáno, jaké nástroje a technologie byly použity a k čemu slouží. Dojde k uvedení potřebných konfiguračních kroků pro každý nástroj a popisu jakým způsobem je aplikace vyvinuta.

Architektura celé aplikace je rozdělena na tři vrstvy. Pro lepší pochopení je níže přiložen Obrázek 3.1. V datové části je databázový server a nástroj Elasticserach, do kterého jsou data importována z databáze. Prostřední částí je komunikační vrstva, jež obsahuje technologii Hibernate pro přímý přístup do databáze a rozhraní REST pro operace s daty uloženými v Elasticsearch. Poslední částí je klientská aplikace, jež je tvořena desktopovou aplikací v jazyce Java.



Obrázek 3.1 Architektura aplikace Zdroj: Vlastní

3.1 Nástroje a technologie

Pro vývoj byly použity různé nástroje a technologie, které určitým způsobem ulehčují vývoj a konfiguraci prostředí jako je například databázový server, či správa závislostí při vývoji aplikace.

Docker

Docker je nástroj, který je navržen pro zjednodušení tvorby, nasazení a spuštění aplikací za použití kontejnerů. Kontejner umožňuje vývojáři zabalit aplikaci včetně všech jejích částí a distribuovat ji v podobě jednoho balíčku. Tím je zajištěno, že daná aplikace bude mít stejné chování na různých počítačích, nebo serverech.[16]

Obraz kontejneru je tedy samostatný, spustitelný balíček softwaru, jež obsahu vše potřebné pro běh. Může se jednat o zdrojový kód, knihovny potřebné pro běh, systémové nástroje a nastavení. Používat se dá na operačním systému Linux, Windows i Mac. Jednotlivé kontejnery jsou izolované od ostatních aplikací a od systému, na kterém běží. V případě výskytu chyby je postihnut pouze daný kontejner, a ne celý systém. Jednotlivé kontejnery mezi sebou mohou komunikovat. Například v jednom kontejneru mohu mít databázi a z jiného se do ní mohu připojit.[17]

Součástí Docker je takzvaný Docker Compose. Jedná se o nástroj pro definování a spuštění více kontejnerových aplikací. Pro definici je použit soubor typu YAML. Následně lze jedním příkazem vytvořit a spustit všechny služby z konfigurace.[18]

Při vývoji aplikace byl použit právě zmíněný Docker Compose, který obsahuje databázi, vyhledávací nástroj Elasticsearch a s ním souvisejí nástroje Kibana a Logstash. Konfigurace je uvedena v kapitole 3.1.1.

MySQL

MySQL představuje rychlý, více vláknový, více uživatelský a robustní SQL databázoví server. K dispozice je ve dvou licencích, a to open source nebo standardní komerční licenci od společnosti Oracle.

V této databázi jsou uložena veškerá data nad, kterými následně vyvinutá aplikace bude provádět vyhledávání.

Pro vývoj byla použita nejnovější verze databázového serveru s označením 8.0.3

Všechna data jsou strukturovaná to znamená, že jsou uloženy standardně v jednotlivých tabulkách složených ze sloupců a řádků. V databázi je hlavní tabulka Patient, jež představuje záznamy o pacientech a jakékoliv hledání bude navracet právě jednoho, či více pacientů. Další tabulky mají jako unikátní klíč identifikační číslo pacienta, dle kterého jsou záznamy provázány. Jde o tabulky Autopsy, Emd, Ems, Ems_time, Fr_aed, Hospital a Ohca.

Kromě výše zmíněných tabulek databáze obsahuje dále číselníky pro nejrůznější hodnoty, jež jsou dále využívány v aplikace. Z důvodu velmi malého množství poskytnutých dat bylo potřeba vygenerovat tisíce násobky dat, aby porovnání časové náročnosti mělo vypovídající hodnotu. Generátor je součástí implementované aplikace a pozornost mu je věnována v kapitole 3.2.2.

Elasticsearch

První verze byla představena v roce 2010, od té doby Elasticsearch začaly používat firmy jako NASA, Wikipedia nebo GitHub. Je vyvíjen pod open source licencí společností Elastic a v posledních verzích se vývoj nástroje soustředí na funkce, aby bylo možné Elasticsearch používat i pro uchování dat, a ne pouze jako vyhledávací nástroj.

Jak bylo zmíněno dříve, Elasticsearch využívá vyhledávací knihovnu Apache Lucene, jejíž přímé použití není příliš snadné a nenabízí možnost škálovatelnosti. Právě z toho důvodu došlu k vytvoření nástroje Elasticsearch, jehož hlavní funkce jsou:

- Škálovatelnost – V základu je připravena možnost škálovatelnosti. Elasticsearch může běžet jako jedno prvkový cluster na jediném zařízení, ale také je možné v případě potřeby rozšířit tento cluster na stovky, či tisíce prvků bez nutnosti řešení problémů, které přichází s distribuovanými výpočty.
- Vysoká dostupnost – Replikace dat znamená mít více kopií stejných dat v clusteru. Tato vlastnost zaručuje uživatelům vysokou dostupnost k datům, a také prevenci před ztrátou dat v případě poruchy serveru.
- REST – Komunikace je zajištěna pomocí architektury REST a poskytuje API pro CRUD operace přes protokol HTTP. Kromě toho má uživatel možnost provádět určité úkony pro správu clusteru, taktéž přes REST API bez nutnosti restartování. Konfiguraci je tak možné provádět pomocí tohoto rozhraní místo toho, aby bylo nutné na každém prvku v clusturu měnit nastavení v konfiguračním souboru.
- Query DSL (Doman-specific language) – Jedná se o JSON rozhraní, pomocí něhož je možné jednoduše psát a číst dotazy pro Apache Lucene. Vývojáři tak bez znalosti syntaxe knihovny Lucene mohou začít vytvářet komplexní dotazy pro Elasticsearch.
- Bez schématu – Není potřeba specifikovat schéma před nahráním dat. Při prvním zaslání dat se Elasticsearch pokusí „naparsovat“ každé pole a vytvoří schéma automaticky ze zdrojových dat, která mohou být uložena například v relační databázi, či souboru.

Základní termíny, s kterými se uživatel setká při používání nástroje Elasticsearch:

- Node (Prvek) – Představuje jednu instanci spuštěného nástroje Elasticsearch.
- Cluster – Jde o souhrn, pod kterým je seskupen jeden nebo více prvků/instancí Elasticsearch propojených mezi sebou.
- Document (Dokument) – Dokument představuje JSON objekt, jež obsahuje data v podobě páru klíč-hodnota.
- Index – Logický prostor nástroje Elasticsearch, který uchovává data.
- Doc types (Typy dokumentů) – Představuje třídu podobných dokumentů. Typ obsahuje název a mapování, včetně datových typů a nastavení pro každou položku.
- Shard (část) – Jedná se o kontejner, jež může být uložen v jednom nebo více prvcích. Index se může skládat z několika těchto částí pro zajištění škálovatelnosti dat.
- Replication (Replikace) – Duplikovaná kopie dat.[20]

Elasticsearch je v první řadě vyhledávací nástroj, ovšem kvůli velkému množství funkcionalit ho společnosti začaly používat jako NoSQL datové úložiště. Pokud bychom chtěli vysvětlit Elasticsearch v terminologii relačních databází vztahy by byly následující:

- Relační databáze => Elasticsearch
- Databáze => Indexy
- Tabulky => Typy
- Řádky => Dokumenty
- Sloupce => Položky

Každá tabulka, která je uložena v MySQL a je v ní prováděno vyhledávání je pro přehlednost uložena v samostatném indexu. Pro přenos dat z MySQL je použit nástroj Logstash a pro následnou kontrolu dat je využita Kibana. Oba tyto nástroje jsou vyvíjeny stejnou společností jako Elasticsearch.[20][21]

Kibana

Kibana je open source analytický a vizualizační nástroj od společnosti Elastic navržen tak, aby úzce spolupracoval s Elasticsearch. Je velmi využívaná hlavně kvůli velkému množství funkcí a také relativně jednoduchému ovládání.

Pomocí Kibany lze prohledávat, zobrazovat a interagovat s daty uloženými v indexech. Lze jednoduše provádět pokročilé analýzy dat a následně data vizualizovat pomocí různých druhů grafů, tabulek a map.

Všechny funkce jsou dostupné skrze webovou aplikaci s moderním a intuitivním designem.[22][23]

Logstash

Stejně jako ostatní nástroje od firmy Elastic jde o open source nástroj pro sběr dat s možností přesměrování dat v reálném čase. Logstash je schopen dynamicky sjednotit data z různých zdrojů a následně je normalizovat a přeposlat dále.

Původně byl vyvinut pro sběr logovacích souborů, ale jeho možnosti nyní sahají mnohem dále. Jakýkoliv typ akce může být obohacen a přeměněn pomocí širokého spektra vstupních kolekcí, filtrů a výstupních pluginů. K dispozici je přes 200 pluginů a v případě potřeby si může firma, či vývojář vytvořit svůj vlastní pro splnění konkrétních požadavků.

Jak již bylo řečeno Logstash umožňuje sbírat různé typy dat jako jsou:

- Logy – Je zde podpora logů z webového serveru Apache a z aplikačních logů vytvořených knihovnou log4j pro programovací jazyk Java. Kromě toho umí zpracovat i další formáty logovacích souborů jako jsou syslog, síťové a firewall logy a další.
- Web – Transformuje HTTP požadavky na události. Konzumuje například webové služby aplikace Twitter pro sociální analýzy. Dále podporuje „webhooky“ pro nejrůznější aplikace, například GitHub, HipChat nebo JIRA.
- Datová úložiště – Možnost importovat data z jakékoliv databáze podporující rozhraní JDBC.
- Sensory a IoT – Umožňuje sbírat data z nejrůznějších senzorů, mobilních zařízení, chytrých domácností, autonomních vozidel a další.[24]

Hibernate ORM

Poskytuje vývojářům jednoduší cestu, jak psát aplikace, které potřebují čerpat data z relačních databází. Jde o objektově/relační mapovací framework využívající technologii JDBC.

Kromě svého nativního API, také implementuje standardní Java Persistence API (JPA). Díky tomu je možné využít Hibernate v jakémkoli prostředí, jež podporuje JPA.

Pro lepší výkon umožňuje inicializaci až v případě, kdy je potřeba, nabízí několik načítacích strategií, optimistické zámky dat, automatické verzování a časové známky.[25]

Maven

Jedná se o nástroj pro správu projektu. Je založen na konceptu POM (projekt, objekt, model) a představuje jasnou definici z čeho se projekt skládá. Může být využit pro sestavení a správu jakékoliv aplikace založená na programovacím jazyce Java. Hlavním cílem celého projektu je poskytnout:

- zjednodušení procesu sestavení,
- jednotný sestavovací systém,
- informace o projektu,
- návod a „best practices“ pro vývoj a
- aktualizaci na nové verze.[26]

Java streams (proudy v jazyce Java)

Představují novou abstraktní vrstvu dostoupnou od Javy verze 8. Proud je sekvence objektů vniklá ze zdrojové kolekce, jež podporuje agregační funkce. Proud lze vytvořit z různých kolekcí jako jsou pole, či seznamy zavoláním metody stream případně parallelStream pro zajištění více vláknového zpracování. [27][28]

```
String[] names = new String[]{“Tom”,”Peter”};
Stream<String> stream = Arrays.stream(names);
```

Zpřístupnění proudu nad libovolnou kolekcí, jež implementuje rozhraní Collection je přímo dostupné pomocí tečkové notace. [27]

```
List<String> names = new LinkedList<>(Arrays.asList(“Tom”,”Peter”));
Stream<String> stream = names.stream();
```

Java predicate (predikáty v jazyce Java)

Obdobně jako proudy jsou dostupné v programovacím jazyce Java od verze 8. Jedná se o funkční rozhraní, to znamená že v rozhraní je pouze jedná abstraktní metoda. Predikát může být použit všude, kde je potřeba určitý objekt otestovat, zda splňuje danou podmínku a je potřeba aby byla navrácena hodnota typu boolean. Tím, že se jedná o funkční rozhraní může být také použit pro přiřazení lambda výrazu nebo reference na metodu. [29][30][31]

Predikáty lze ve spojení s proudy využít pro filtrování dat v kolekci.

3.1.1 Konfigurace

Před samotným vývojem aplikace bylo potřeba připravit prostředí, na kterém bude spuštěna databáze, Elasticsearch, Kibana a Logstash. Každý z těchto nástrojů lze provozovat různými způsoby. Klasická varianta je instalace každého nástroje samostatně a následně jeho konfigurace. Při tomto přístupu, může nastat problém v případě kdy budeme chtít přenést toto prostředí z jednoho systému na jiný. Po přenosu může dojít k jinému chování z důvodu, že systémy nejsou identické. Z toho důvodu byla vybrána technologie Docker Compose, které nám tuto zmíněnou nevýhodu odstraní, a kromě toho je spouštění a vypínání všech služeb k dispozici pomocí jednoho příkazu „docker-compose up“.

Všechny zmíněné nástroje se nachází ve složce „elk“, jež obsahuje následující pod složky:

- elasticsearch,
- kibana,
- logstash,
- mysql.

Mimo to obsahuje také soubor docker-compose.yml.

Struktura jednotlivých složek

Složka pro Elasticsearch má následující strukturu:

- config – Složka s nastavením pro samotný Elasticsearch. Slouží k tomu soubor elasticsearch.yml. V souboru je nastaven název clusteru a počet prvků.
- data – Zde má Elasticsearch uložen veškerá data.
- Dockerfile – Konfigurační soubor pro Docker. Obsahuje pouze jeden atribut „FROM“, který specifikuje, z jakého obrazu bude tento kontejner vycházet. Firma Elastic poskytuje oficiální obraz pro tento nástroj.

Struktura pro Kibanu je velmi podobná, v podsložce config se nachází soubor kibana.yml, v kterém je potřeba nastavit URL adresu na niž je dostupný Elasticsearch. Dále je také nezbytné mít soubor Dockerfile, který opět obsahuje pouze odkaz na obraz, z jakého bude kontejner vycházet.

Logstash má strukturu:

- config – V souboru logstash.yml je nutné nespecifikovat atribut „path.config“, aby došlo k využití samostatného souboru pipelines.yml. Ten obsahuje nastavení jednotlivých vstupních proudů. Vždy je nastaveno unikátní jméno (id) daného proudu a cesta ke konfiguračním souborům.
- lib – Obsahuje knihovnu pro připojení k MySQL databázi.
- pipeline – Jak název napovídá, v této složce jsou konfigurační soubory pro jednotlivé vstupní proudy. Nastavení pro vstupní proud má dvě základní položky input (vstup) a output (výstup). V části pro vstup je pomocí klíčového slova „jdbc“ řečeno, že chceme čerpat data z nějaké databáze a pod ním jsou samotné atributy, jež specifikují cestu ke knihovně pro připojení k databázi, název třídy pro připojení, adresu pro připojení a uživatelské jméno a heslo. V neposlední řadě je nutné použít atribut „statement“, který obsahuje SQL dotaz dle toho, jaká data chceme přenést do indexu.
  Ve výstupní části nám položka elasticsearch říká, kam chceme data přenést. Uvnitř je nutné uvést adresu, na které je dostupný Elasticsearch, název indexu a také položku „document_id“, jež je nastavena na název primárního klíče, abychom zamezili duplicitám v indexu. V příloze A můžeme vidět konkrétní soubor pro tabulku Autopsy.
- Dockerfile – Kromě specifikace obrazu je zde příkaz „RUN“, jenž zajistí instalaci pluginu pro podporu JDBC.

Poslední složkou je mysql, jež obsahuje podsložku data, uvnitř které si databáze uchovává data. Stejně jako předešlé konfigurace i zde je nutné vytvořit soubor Dockerfile s uvedením cesty k obrazu.

Hlavní konfigurační soubor pro Docker Compose

Obsahuje veškerou konfiguraci pro jednotlivé Docker kontejnery, které budou spuštěny. Pod položkou „services:“ jsou postupně definovány nastavení pro jednotlivé služby.

Nastavení pro jednotlivé služby má mnoho společných atributů, jejichž funkce budou obecně vysvětleny na příkladu nastavení pro Logstash.

Jak lze níže vidět konfigurace obsahuje několik atributů:

- build – cesta ke složce, která obsahuje soubor Dockerfile.
- volumes – slouží pro mapování souborů a složek z hostitelského zařízení do kontejnerů. Pomocí toho je možné mít data a konfigurační soubory uložené na hostitelském zařízení, a ne uvnitř kontejneru.
- ports – umožňuje přesměrování portů z kontejneru na hostitelské zařízení
- environment – lze nastavit proměnné prostředí, např. kolik operační paměti má Java k dispozici.
- networks – nastavení do jaké sítě služba spadá, aby byly kontejnery mezi sebou dostupné.
- depends_on – v případě potřeby, že je nutné určitou službu spouštět až poté co jiná již nastartovala, lze tak nastavit v tomto atributu.

```
logstash:
  build: logstash/
  volumes:
   - ./logstash/config/logstash.yml:/usr/share/logstash/config/logstash.yml
   - ./logstash/config/pipelines.yml:/usr/share/logstash/config/pipelines.yml
   - ./logstash/pipeline:/usr/share/logstash/pipeline
   -./logstash/lib/mysql-connector-java-8.0.8-dmr.jar:/usr/share/logstash/mysql-connector-java-8.0.8-dmr.jar
  ports:
   - "5000:5000"
  environment:
   LS_JAVA_OPTS: "-Xmx256m -Xms256m"
  networks:
   - elk
  depends_on:  ['elasticsearch', 'mysql']
```

3.2 Vývoj aplikace

Nástroj pro vyhledávání byl programován v jazyce Java. Pro vývoj bylo použito vývojové prostředí IntelliJ IDEA od společnosti JetBrains. Vzhledem k tomu, že k sestavení aplikace a správě závislostí je použit Maven neměl by být problém s jejím otevřením v konkurenčních nástrojích jako je například Eclipse nebo Netbeans.

3.2.1 Koncepce vyhledávání

Než dojde k podrobnému popisu struktury aplikace a její implementace, bylo by vhodné objasnit způsoby jakými je vyhledáváno v datech.

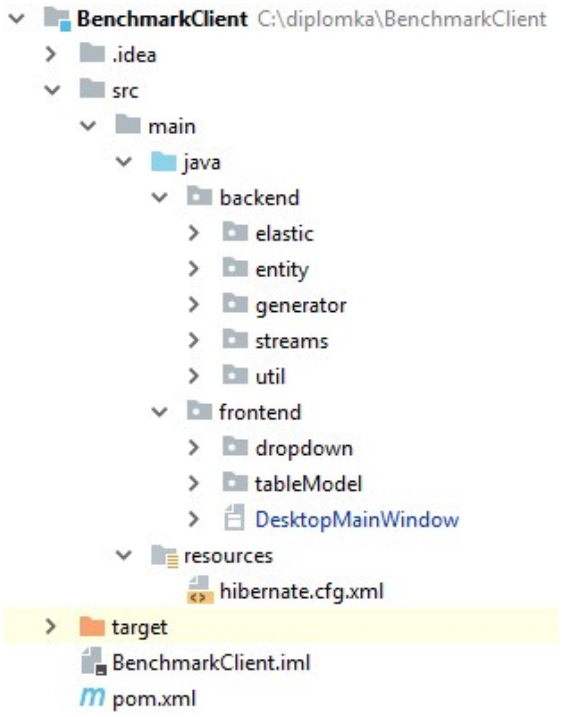
Pro první přístup jsou veškerá data uložena v databázi MySQL a aplikace je získává pomocí frameworku Hibernate. Z databáze jsou vždy získána veškerá data pro danou tabulku, v níž se má vyhledávat a jsou uložena v paměti v samostatných seznamech. Následně je na základě uživatelského vstupu sestaven predikát pro každou tabulku, který je využit ve spojení s datovými proudy v jazyce Java pro filtrování dat. Tím získáme seznamy pro každou tabulku, jež obsahují unikátní identifikátory uživatelů splňující zadané podmínky. Na závěr se najdou společné identifikátory v těchto seznamech, aby bylo zajištěno, že výsledky splnily všechna kritéria.

Druhým způsobem je vyhledávání pomocí nástroje Elasticsearch. Tento přístup je stejný jak pro využití oficiálního klienta, tak pro knihovnu Jest. Na základě uživatelského vstupu se sestaví dotaz, jež je odeslán pomocí protokolu HTTP na REST API nástroje Elasticsearch. Dle dotazu proběhne vyhledávání a opět pomocí REST API dojde k navrácení výsledků zpět do aplikace. Obdobně jako v prvním přístupu i zde je vyhledáváno pro každou tabulku samostatně, a tak je potřeba provést na závěr stejný krok pro získání společných identifikátorů.

3.2.2 Struktura a popis aplikace

Zdrojové soubory jsou rozděleny do balíčků (viz. Obrázek 3.2) podle účelu, aby byla struktura aplikace přehledná a byla umožněna snadná orientace mezi soubory. Nejdříve bude pozornost věnována balíčku backend.entity, který tvoří základ pro celou aplikaci, jelikož představuje objekty (entity) s kterými následně aplikace pracuje. Následně bude probrán balíček backend.generator sloužící ke generování libovolného množství testovacích dat, aby bylo možné následně provést samotné měření časové náročnosti. Dalšími balíčky jsou backend.elastic a backend.streams kde každý obsahuje třídy potřebné pro vyhledávání pomocí nástroje Elasticsearch respektive pomocí Java proudů. Dále v balíčku backend.util jsou pomocné třídy a

v neposlední řadě bude popsán balíček frontend, jež zpřístupňuje veškeré funkce v grafickém uživatelském rozhraní.



Obrázek 3.2 Struktura aplikace ve vývojovém prostředí Zdroj: Vlastní

Konfigurace pro Maven - pom.xml

Aplikace využívá několik knihoven, které nejsou standardní součástí jazyka Java. Díky technologii Maven nebylo nutné stahovat jednotlivé knihovny v podobě balíčku jar, ale všechny jsou automaticky staženy z Maven repozitářu. Aplikace využívá následující knihovny:

- Elasticsearch klient – Oficiální klient poskytovaný společností Elastic. Jsou k dispozici dvě verze, takzvaný nízko úrovňový a vysoko úrovňový klient. První zmíněný klient byl představen s verzí Elasticsearch 5.0. Oproti tomu s příchodem verze 6.0, která je stará pouze několik měsíců poskytli druhou verzi, která představuje obalení nízko úrovňového klienta pro pohodlnější používání. Je možné v HTTP žádostech a odpovědích pracovat na úrovni objektů a odpadá tak nutnost implementovat tyto funkce vlastnoručně. Kromě toho je možné volat každé rozhraní jak synchronně, tak asynchronně. Při použití této oficiální verze je nutné dát pozor, aby verze klienta byla stejná jako hlavní verze Elasticsearch, případně vyšší nebo stejná jako minoritní verze. [32][33]

- Knihovny weblaf a weblaf-ui – Slouží ke změně vzhledu Swingových komponent, aby aplikace působila modernějším vzhledem než výchozí styl komponent v grafickém uživatelské prostředí.[34]
- Hibernate – Knihovna, které zajišťuje podporu pro stejnojmennou technologii pro práci s relační databází. Konfigurace se nachází v balíčku resources v souboru hibernate.cfg.xml, jež bude popsán v další části.
- Mysql-connector – Jelikož je využíván databázový server Mysql je potřeba do aplikace zahrnout knihovnu pro připojení k ní, která je využívána technologií Hibernate.
- Jest – Jde o klienta pro Elasticsearch, který komunikuje skrze HTTP REST webové služby. Představuje malý a nenáročný nástroj pro komunikaci Java aplikace s instancí Elasticsearch. Oproti obyčejnému volání webových služeb nabízí tato knihovna připravené třídy pro vytvoření připojení, přípravu dotazu, zavolání endpointu pro konkrétní index. Stejně tak umožňuje mapovat výsledek na určitou entitní třídu. Celá knihovna je volně k dispozici pod Apache licencí.[35][36]

Celý konfigurační soubor se nachází v příloze B.

Nastavení pro Hibernate - hibernate.cfg.xml

Aby bylo možné automaticky načítat a ukládat data pomocí technologie Hibernate je nejdříve potřeba provést konfiguraci, ke které slouží soubor ve formátu XML. Obsahem je specifikace názvu třídy, jež bude použita pro připojení k databázi, dále je nutné zadat url pro připojení a uživatelské jméno s heslem.

```
<property name="connection.driver_class">com.mysql.cj.jdbc.Driver</property>
<property name=
"connection.url">jdbc:mysql://192.168.0.107:3306/mycold94?useSSL=false
</property>
<property name="connection.username">root</property>
<property name="connection.password">admin</property>
```

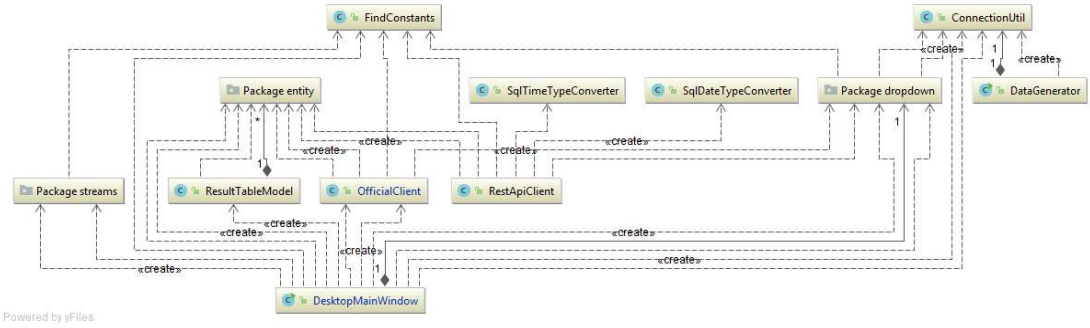
Kromě těchto potřebných položek, bez kterých by nebylo možné se připojit k databázi je možné nastavit například zda chceme vypisovat SQL příkazy do logu.

```
<property name="show_sql">true</property>
```

Dále už obsahuje konfigurační soubor pouze mapování Entitních tříd, které mají být automaticky spravovány.

<mapping class="backend.entity.search.Patient"/>

Jak již bylo řečeno aplikace je rozdělena do následujících balíčků, pro lepší představu je přiložen Obrázek 3.3, na kterém je UML diagram celé aplikace.



Obrázek 3.3 UML diagram aplikace Zdroj: Vlastní

backend.entity

Obsahuje třídu SearchDTO, které slouží k udržení informací o nalezeném záznamu z tabulky Patient dle kritérií a tyto informace jsou použity v grafickém uživatelské prostředí. Atributy je křestní jméno, příjmení, pohlavní a unikátní identifikátor.

Další třídy jsou ještě rozděleny do balíčků:

- dropdown – Aplikace pracuje dohromady s celkem třiceti číselníky, kde každá třída reprezentuje přesně jeden číselník. Pro zjednodušení další práce byla vytvořena abstraktní třída AbstractDropdown s jednou metodou getDropdownId. Každý číselník implementuju tuto třídu a přepisuje danou metodu.
- search – Hlavní je zde abstraktní třída Entity, jež obsahuje pouze jednu metodu k navrácení unikátního identifikátoru. Třídy Autopsy, Emd, Ems, EmdTime, FrAed, Hospital, Ohca a Patient dědí od abstraktní třídy Entity a přepisují metodu pro navracení unikátního identifikátoru. Kromě toho, každá z nich obsahuje mnoho dalších atributů specifických pro danou třídu a stejnojmennou tabulku v databázi.

backend.generator

Jde o velmi důležitou součást celé aplikace. Aby bylo možné provést měření časové náročnosti vyhledávání a následnou analýzu bylo potřeba mít k dispozici určité množství dat a ideálně provést měření na různě velkém množství dat.

Generátor je samostatná aplikace, kterou lze spustit bez nutnosti pouštět hlavní program. Aplikace nepodporuje žádný uživatelský vstup a veškeré parametry jsou nastaveny přímo ve zdrojovém kódu. Taktéž neobsahuje grafické uživatelské prostředí a vypisuje informace do konzole. Předpokladem pro správné fungování je dostupná databáze Mysql, aby bylo možné vygenerovaná data vkládat přímo do konkrétních tabulek. Veškeré funkce jsou zajištěny třídou DataGenerator, kromě samostatných metod pro jednotlivé entity, obsahuje pomocné metody, jež generují hodnoty:

- generateNumber – Vrátí náhodné číslo v zadaném rozmezí
- generateFirstName – Slouží k vybrání náhodného křestního jména ze seznamu anglických jmen, které jsou načteny ze souboru firstName.csv.
- generateOption – Náhodně vybere některou hodnotu, která je dostupná pro daný číselník.
- generateLastName – Obdobně jako metoda pro tvoření křestního jména, tak i zde je příjmení náhodně vybráno ze seznamu téměř 90 000 anglických příjmení.
- generateText – Vygeneruje libovolně dlouhý textový řetězec.
- generateTime – Návratovou hodnotou je náhodný časový údaj ve dvaceti čtyř hodinovém formátu.
- generateDate – Poskytuje stejnou funkci jako předchozí metoda, pouze místo času dojde k vygenerování data.

Vkládání do jednotlivých tabulek je spouštěno v samostatných transakcích.

backend.elastic

Uvnitř tohoto balíčku jsou dvě třídy. Každá z nich představuje trochu jiný přístup k instanci Elasticsearch. V rámci aplikace jsou využity pro zaslání dotazu a navrácení záznamů, jež vyhovují zadaným kritériím.

Nejdříve bude pozornost věnována třídě RestApiClient, která využívá knihovnu Jest. Inicializace klienta je velmi jednoduchá a je nutné vytvořit pouze několik objektů, jež poskytuje zmíněná knihovna. Nutností je zaslat správnou url adresu jako parametr metody HttpClientConfig.Builder. Adresa musí obsahovat i port, který je pro Elasticsearch standardně 9200. Dále lze klientovi nastavit vlastní instanci Gson knihovny, jež slouží k parsovaní odpovědí. Jak lze vidět v kódů níže při vytváření objektu Gson registrujeme dva druhy adaptérů pro Datum a Čas.

```
JestClientFactory factory = new JestClientFactory();
Gson gson = new GsonBuilder()
    .registerTypeAdapter(Time.class, new SqlTimeTypeConverter())
    .registerTypeAdapter(Date.class, new SqlDateTypeConverter()).create();
factory.setHttpClientConfig(new HttpClientConfig.Builder("http://192.168.0.107:9200")
    .gson(gson)
    .multiThreaded(true)
    .build());
JestClient client = factory.getObject();
```

K dispozici jsou dvě metody pro vyhledávání:

- search – Uvnitř metody dojde k sestavení dotazu ve formátu JSON. Následně se uvede, v jakém indexu se má vyhledávat. Nyní je možné získat výsledky, které lze převést na objekty pro zjednodušení další práce. Jedním z parametrů je třída, která dědí od abstraktní třídy Entity, tím je zajištěno, že pro každou takovou třídu bude správně sestaven výsledný objekt pro získání unikátního identifikátoru. Metoda navrací seznam unikátních identifikátorů pacientů, jež splnili vyhledávací kritéria.
- searchPatientDetails – Plní velmi podobnou funkci jako výše zmíněná metoda s tím rozdílem, že navrací seznam objektů typu SearchDTO.

Ukázka jednoduchého dotazu ve formátu JSON může obsahovat atributy „from“ který udává kolik najitých hodnot bude přeskočeno a „size“, jímž je možné omezit počet najitých záznamů. Pokud neuvedeme zmíněné parametry dojde k navrácení omezeného počtu výsledků, jež je nastaven v nástroji Elasticsearch. Dále už je obsahem samotný dotaz, kde je možné řetězit více parametrů pomocí logických operátorů.

```
{
  “from“: 0, “size“: 30000,
  “query“: {
    “query_string“: {
      “query“: “fname: Josef AND lname: Novak“
    }
  }
}
```

Druhou třídou je OfficialClient, jež využívá knihovnu oficiální klienta pro Elasticsearch, jedná se o takzvaný vysoko úrovňový REST klient (High Level REST Client). Inicializace je také velmi jednoduchá a je opět potřeba specifikovat url adresu a port, na kterém je Elasticsearch dostupný.

```
RestHighLevelClient client = new RestHighLevelClient(RestClient.builder(new Htt-
pHost("192.168.0.107", 9200, "http));
```

K dispozici jsou stejné metody jako u předchozího klienta, pouze se liší způsob, jakým je sestavován a volán dotaz. Není potřeba ručně sestavovat JSON, ale je k tomu použita metoda

QueryBuilders.queryStringQuery které je předán pouze samotný dotaz, například:
fname: Josef AND lname: Novak

backend.streams

Druhým přístupem pro vyhledávání v aplikace jsou proudy s využitím predikátů. V balíčku je rozhraní IGeneralPredicate, které poskytuje metodu find s návratovou hodnotou Predicate a parametry je název dle čeho se má hledat a hodnota pro toto pole. Tohle rozhraní je následně implementováno v těchto třídách:

- AutopsyPredicate,
- EmdPredicate,
- EmsPredicate,
- EmsTimePredicate,
- FrAedPredicate,
- HospitalPredicate,
- OhcaPredicate,
- PatientPredicate.

V každé třídě je přepsána metoda find, jež rozhoduje dle prvního parametru, jaký predikát bude využit. Níže lze vidět predikát pro vyhledání v tabulce Patient dle křestního jména. Pro ostatní atributy a tabulky jsou vytvořeny analogicky další predikáty.

```
public Predicate findByFirstName(String firstName) {
  Predicate<Patient> predicate = p -> p.getFname().equals(firstName);
  return predicate;
}
```

backend.util

Třídy v tomto balíčku slouží k nejrůznějším pomocným funkcím, které jsou používané napříč celou aplikací.

V příloze C je uvedena třída ConnectionUtil, jež zajišťuje připojení k databázi s využitím technologie Hibernate. Připojení je vytvořeno na základě konfiguračního souboru hibernate.cfg.xml. Mimo to bylo nutné při otevření spojení do databáze specifikovat časové pásmo, protože jinak docházelo ke špatnému načítaní časových údajů z tabulek, což by při vyhledávání způsobovalo uživateli komplikace.

Druhá třída FindConstans uchovává sadu názvů ve statických atributech typu String. Tyto názvy jsou používány v grafickém uživatelském prostředí pro identifikaci komponent do, kterých uživatel zadává hodnoty. Dále jsou také využity ve třídách pro tvorbu predikátů, pro identifikaci a volbu správné metody.

Dalšími třídami jsou SqlDateTypeConverter a SqlTimeTypeConverter, které umožnují konverzi data a času z Elasticsearch do typu java.sql.Date a java.sql.Time .

frontend

Součástí jsou třídy, jež jsou využity pro grafické uživatelské prostředí aplikace. V balíčku dropdown je třída DropDownHolder. Její úkolem je načtení a držení všech číselníků po celý běh aplikace. Dalším podřazeným balíčkem je tableModel s třídou ResulTableModel. Tato třída dědí od AbstractTableModel, jež je součástí java.swing. Při vytváření instance je nutné zaslat v parametru seznam s objekty typu SearchDTO, které jsou následně vykresleny do tabulky.

Hlavní třída, která vykresluje prostředí, obsluhuje uživatelský vstup a zprostředkovává vyhledávání je DesktopMainWindow. Uživatelské prostředí využívá technologii Java Swing a k jeho návrhu byl využit grafický návrhář obsažen ve vývojovém prostředí IntelliJ IDEA.

V příloze D je přiložen obrázek, na němž je možné vidět, jak celá aplikace vypadá. Ve vrchní části jsou seskupeny ovládací prvky pro nastavení předefinovaných hodnot do formulářů a tlačítka pro vyhledávání za využití nástroje Elasticsearch a Java streamů. Vedle těchto tlačítek je pole pro zadání kolikrát bude vyhledávání opakováno, výchozí hodnota je 1. Pod ovládacími prvky se nachází 8 záložek, kde každá reprezentuje jednu tabulku, v které se bude vyhledávat. Výsledkem jakéhokoliv hledání je seznam pacientů. Tento výsledek je zobrazen v tabulkách ve spodní části aplikace podle toho, jakou metodou bylo vyhledáváno. Spolu s výsledky je zobrazen i čas a počet záznamů.

Nyní budou popsány jednotlivé metody této třídy:

- main – Hlavní metoda celé třídy, slouží ke spuštění aplikace. Uvnitř je nastaven vzhled a rozměr okna. Dochází také k vytvoření instance samotné třídy DesktopMainWindow.
- DesktopMainWindow – Konstruktor třídy, v němž dochází k postupné přípravě komponent před spuštěním aplikace. Jsou naplněny mapy komponent pro každý panel kde klíčem je název komponenty, který musí být nastaven v atributu name a hodnotou je typ komponenty (text, time, dropdown, date, year). Pomocí typu je v dalších metodách rozlišeno, jakým způsobem budou získána vyplněná data. Také jsou volány pomocné metody prepareElasticAtribute, prepareSpinnes, prepareDropdowns a createComponentMap.

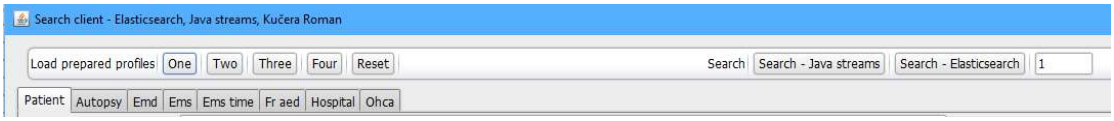
- prepareElasticAttributes – Vytvoří mapu, v které je klíčem název komponenty a hodnotou název položky v příslušném indexu v Elasticsearch. Jelikož v indexu jsou rozdílné názvy než v databázi.
- prepareSpinners – Číselné hodnoty, jako jsou například čas jsou zadávány za využití komponenty JSpinner a pro nastavení minimální, maximální hodnoty a velikosti kroku je použit takzvaný model. Pro všechny komponenty tohoto typu jsou zde nastaveny příslušné modely.
- prepareDropdowns – Aplikace využívá velké množství číselníků, jak již bylo řečeno dříve. V uživatelském rozhraní jsou zobrazeny pomocí komponenty JComboBox a zde dochází k nastavení hodnot z číselníku do jednotlivých komponent.
- prepareComponent – Pomocná metoda, jež je volána z prepareDropdowns. Obsluhuje nastavení hodnot do komponent.
- createComponentMap – Vytvoří mapu všech komponent, z kterých je nutné později získat hodnotu. Klíčem je její název a hodnotou objekt typu Component.
- onSearchElastic – K jejímu zavolání dojde po stisku tlačítka „Search - Elasticserach“ a obsluhuje proces od přípravy dotazu až po vypsání nalezených hodnot. Celý tento proces je měřen a následně zobrazen v uživatelském prostředí. Nejprve je nutné pro každý index vytvořit dotaz na základě uživatelského vstupu. K tomu je využita metoda appendQuery. Po sestavení dotazu je zavolána metoda search ze třídy RestApiClient respektive OfficialClient. Tím získáme seznam unikátních identifikátorů, jež splnily daná kritéria. Vyhledávání probíhá v každém indexu zvlášť, proto jsou i výsledky s identifikátory v separátních listech. Pro zobrazení správného výsledku je nutné vložit tyto seznamy do jednoho a najít společné identifikátory, tím dojde k zajištění, že zbylé identifikátory splnily všechna požadovaná kritéria. Nyní již stačí sestavit dotaz pro získání dat z indexu Patient dle unikátních identifikátorů a předat jej metodě searchPatientDetail z klientských tříd pro Elastisearch. Na závěr jsou výsledky vypsány do tabulky s využitím třídy ResultTableModel.

- onSearchStreams – Funkčně se jedná o velmi podobnou metodu jako byla výše zmíněná. Rozdílem je, že k jejímu volání dochází po stisknutí tlačítka „Search – Java strams“. Celá metoda je spuštěna v jedné databázové transakci a opět je měřen čas, jak dlouho trvá její průběh. Nejdříve jsou vytvořeny seznamy pro jednotlivé entity, v nichž se bude vyhledávat a jsou naplněny všemi daty z databáze. Dále je potřeba sestavit pro každou entitu predikát, dle kterého se budou data filtrovat. K tomuto účelu je dostupná metoda preparePredicate. Následně je již možné získat seznamy unikátních identifikátorů pro každou entitu, jež splňuje kritéria s pomocí metody filterData. Na závěr se opět provede seskupení a najití společných hodnot, které jsou využity pro sestavení finálního predikátu na získaní dat z tabulky Patient a jejich zobrazení do tabulky.
- getValueByClass – Vrátí unikátní identifikátor vybrané položky v komponentě JComboBox.
- preparePredicate – Slouží k sestavení predikátů. V cyklu projde komponenty pro jednotlivé záložky a získá z nich zadané hodnoty, z kterých je postupně tvořen výsledný predikát. Zdrojový kód je přiložen v příloze E.
- getValue – Obstarává získání hodnoty z komponenty dle jejich typu (text, time, dropdown, date, year)
- filterData – Na základě poskytnutého predikátu a seznamu hodnot provede jejich filtraci. Celá metoda je k vidění v příloze F.
- getCommonElements – Pomocná metoda, jež najde a vrátí společné položky z libovolného počtu seznamů.
- getComponentByName – Najde a vrátí komponentu dle jména.
- appendQuery – Podobně jako metoda preparePredicate v cyklu projde komponenty dle záložek a ze získaných hodnot sestaví dotaz pro Elasticsearch. Postup sestavní je k vidění v příloze G.
- getElasticAttributeName – Z mapy atributů pro Elasticsearch vrátí hodnotu dle klíče.

4 POPIS APLIKACE

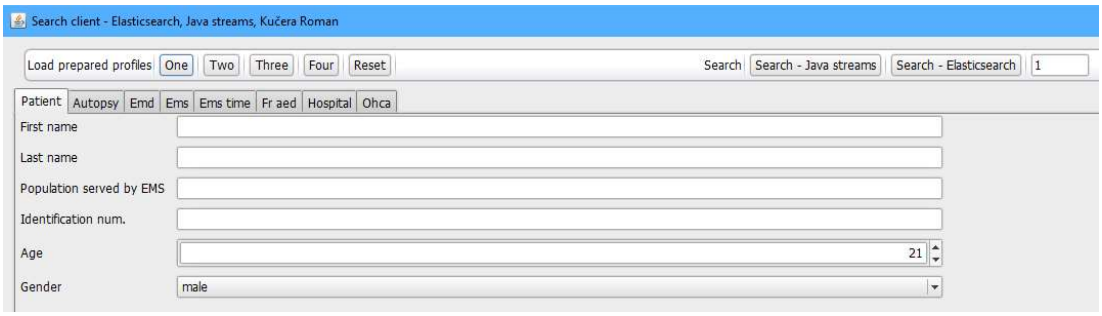
Nyní bude ještě popsán průchod aplikací z uživatelského pohledu pro jeden konkrétní scénář, jež je použit v následující kapitole 5.

Po startu aplikace dojde k zobrazení okna aplikace viz. příloha D. Následně si vybere jeden z předem definovaných scénářů. Tím je zajištěno, že dojde k nalezení nějakých výsledků, jelikož aplikace nabízí v jednotlivých panelech celou řadu atributů. K využití přednastavených profilů jsou určeny tlačítka „One, Two, Three, Four“. Další tlačítko s popisem „Reset“ slouží k navrácení hodnot, jež byly nastaveny některým profilem na výchozí hodnoty. Panel s tlačítky pro ovládání je zobrazen na Obrázek 4.1



Obrázek 4.1 Nastavení parametrů pro vyhledávání Zdroj: Vlastní

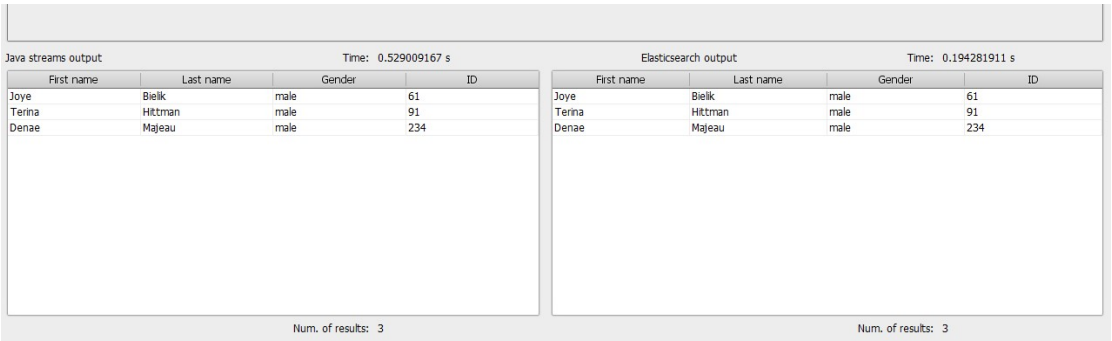
Po zvolení prvního profilu pomocí tlačítka „One“ dojde k vyplnění věku a pohlaví na kartě Patient viz. Obrázek 4.2.



Obrázek 4.2 Vyplněné hodnoty na kartě Patient Zdroj: Vlastní

Nyní je možné spustit vyhledávání stiskem tlačítka „Search – Java stream“ nebo „Search - Elasticsearch“. Jak již název napovídá, každé z tlačítek obsluhuje jeden přístup, kterým jsou data prohledávána. Pole s hodnotou 1 vedle těchto tlačítek necháme nezměněné, aby se vyhledávání provedlo pouze 1x.

Následně v dolní části aplikace dojde k zobrazení výsledků (Obrázek 4.3). Pro každou metodu je k dispozici samostatná tabulka, u které je uvedena délka hledání a celkový počet nalezených záznamů. V tabulce je uvedeno jméno a příjmení pacienta spolu s jeho pohlavím a unikátním identifikátorem.



Obrázek 4.3 Nalezené záznamy Zdroj: Vlastní

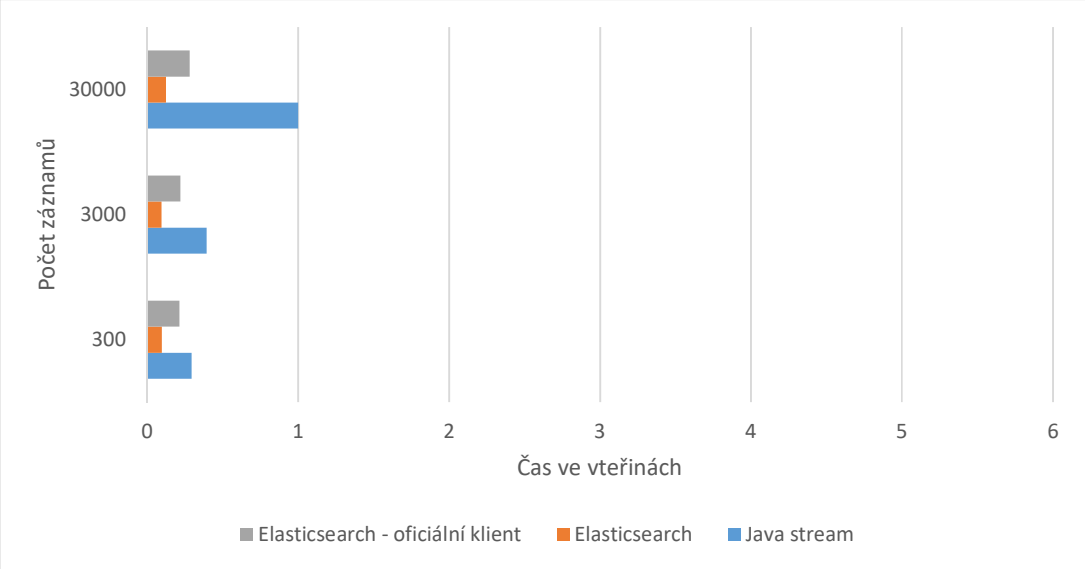
5 EXPERIMENTY

Po vytvoření aplikace a vygenerování dat bylo možné začít provádět testování časové náročnosti. Celkem byly vytvořeny 4 testovací scénáře a 3 sady dat, kde každá sada se lišila množstvím uložených dat. Nejmenší z nich obsahovala 300 záznamů v každé tabulce, druhá 3000 a poslední 30 000 záznamů.

Testování bylo prováděno ve 100 replikacích pro každou sadu dat a pro každý profil nad touto sadou. Následně byl spočítán kumulativní průměr, z něhož byly tvořeny výsledné grafy. V každém grafu osa x představuje čas, jak dlouho trvalo zobrazení výsledků do tabulky od stisku tlačítka. Na ose y je počet záznamů, které byly v každé tabulce. Pro všechny sady dat byl využit i další přístup pro vyhledávání pomocí oficiálního klienta pro Elasticsearch, který byl vždy o něco pomalejší než knihovna Jest.

Scénář 1

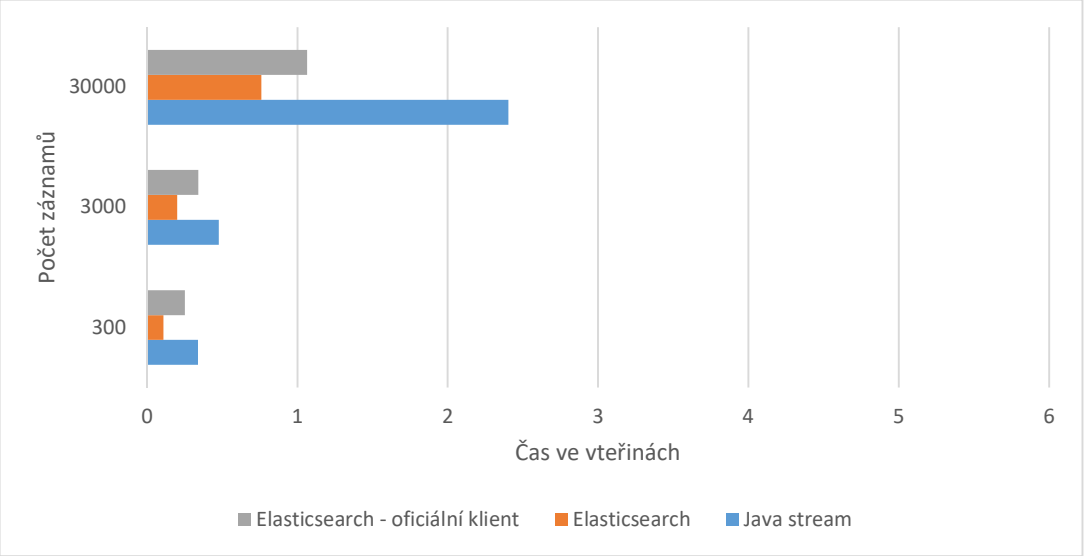
První testování bylo prováděno na základě dvou parametrů pouze v jedné tabulce. Vyhledávali se všichni pacienti, jež jsou muži a jejich věk je 21 let. Z výsledného Graf 5.1 lze vyčíst, že, při velmi malém množství dat lze pozorovat nejmenší rozdíl, ale i zde byl Elasticsearch přibližně 3x rychlejší. S přibývajícím množstvím dat se rozdíl mnohonásobně zvětšoval a na největší testovací sadě byl až 10x rychlejší Elasticsearch. Oficiální klient dosahoval pouze o něco horší výsledek kvůli větší režii oproti odlehčení knihovně Jest.



Graf 5.1 Porovnání časové náročnosti pro scénář č. 1 Zdroj: Vlastní

Scénář 2

Tento scénář vyhledává dle parametrů ve třech tabulkách (Patient, Autopsy, Emd). Výsledkem jsou všichni pacienti, co jsou muži, projevovali známky života a jméno dispečera je „Kuder“. Na základě Graf 5.2 můžeme konstatovat, že v nejmenším množství dat byl opět časový rozdíl nejmenší a velmi podobný předchozímu scénáři. S větším množstvím dat lze už ovšem pozorovat navyšovaní časové intervalu, jak pro datové proudy, tak pro Elasticsearch. Rozdíl je, ale pořád znatelný a datové proudy jsou zhruba 3 - 4x pomalejší. Oficiální klient je opět pouze o něco málo pomalejší než knihovna Jest.



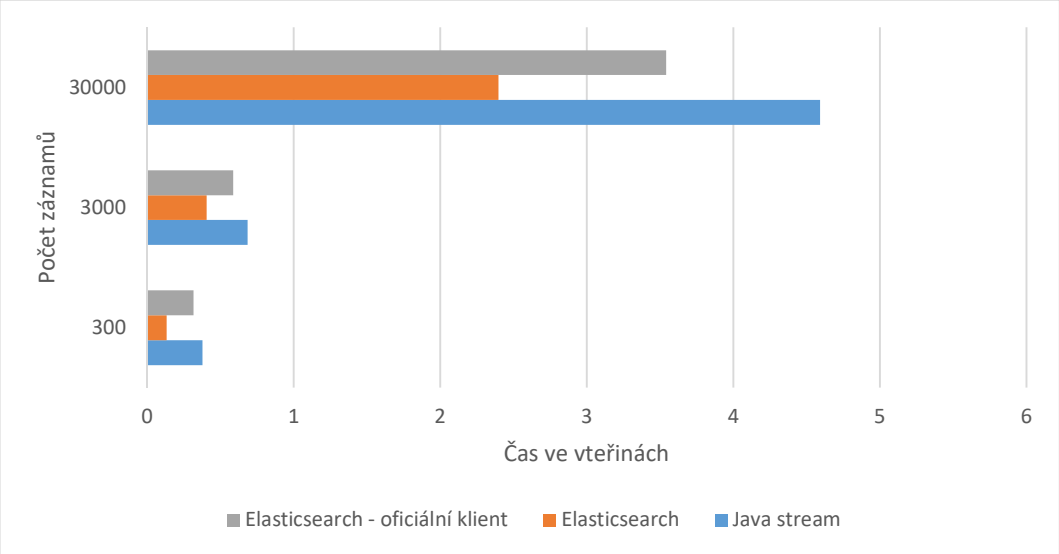
Graf 5.2 Porovnání časové náročnosti pro scénář č. 2 Zdroj: Vlastní

Scénář 3

Oproti předchozí variantě bylo hledání opět rozšířeno do dalších tabulek. Nyní se celkem vyhledávalo v 5 tabulkách (Patient, Autopsy, Emd, Ems, Fr_Aed). Byly nastaveny následující atributy:

- Záložka Patient – atribut „Gender“ na hodnotu „male“
- Záložka Autopsy – atribut „Vital signs“ na hodnotu „yes“
- Záložka Emd – atribut „DA CPR is not provided“ na hodnotu „non-co caller“
- Záložka Ems – atribut „Collapse witnessed“ na hodnotu “yes, bystender“
- Záložka Fr aed – atribut „Lay – responder activation who“ na hodnotu „both EMD/APP“

Na výsledném Graf 5.3 lze pozorovat podobný vývoj jako v přechozích případech. Dochází k nárůstu časové prodlevy u všech způsobů vyhledávání a rozdíl datovými proudy a nástrojem Elasticsearch je menší. Při největším množství dat je použití Elasticsearch zhruba 2x rychlejší.



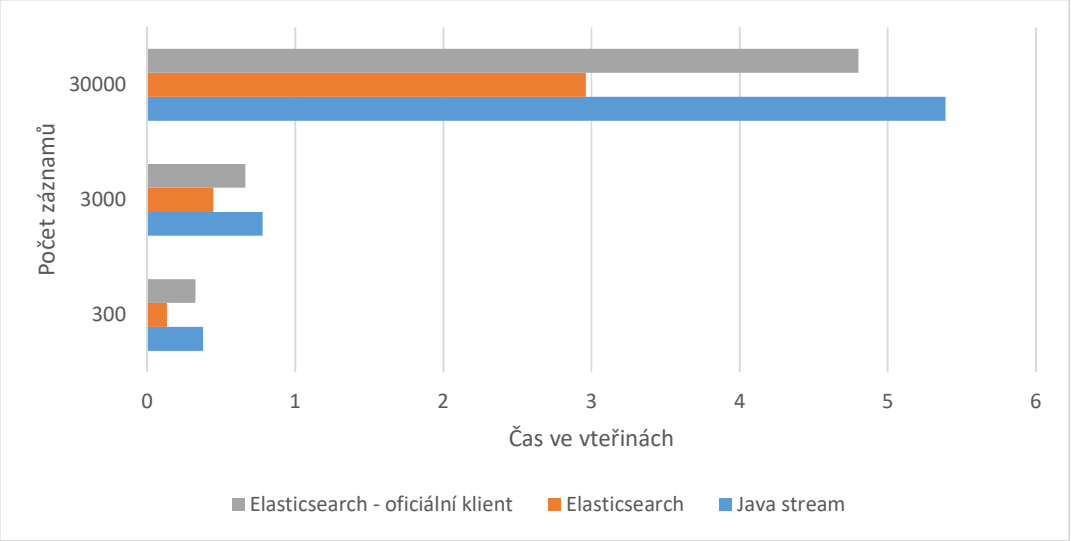
Graf 5.3 Porovnání časové náročnosti pro scénář č. 3 Zdroj: Vlastní

Scénář 4

Poslední varianta vyhledává napříč 7 tabulkami (Patient, Autopsy, Emd, Ems, Fr_Aed, Hospital, Ohca). Nastavení pro vyhledávání je stejné jako v předešlém případě pouze přibyly tyto atributy:

- Záložka Hospital– atribut „Admitted to hospital neurological outcome at hospital discharge“ na hodnotu „CPC 4“
- Záložka Ohca – atribut „CPR attempted“ na hodnotu „no“

Na Graf 5.4 lze pozorovat, že opět dochází ke stejnému vývoji jako v ostatních scénářích. Výsledky jsou nejvíce podobné scénáři č. 3 a i zde jsou datové proudy zhruba 2x pomalejší pro největší sadu dat.



Graf 5.4 Porovnání časové náročnosti pro scénář č. 4 Zdroj: Vlastní

ZÁVĚR

Hlavním cílem této práce bylo navrhnout a implementovat aplikaci, která poskytne možnost porovnat časovou náročnost vyhledávání pomocí datových proudů a standardní relační databáze se specializovaným nástrojem, jež je přesně pro tyto účely navržen. Pro vyhledávání bylo použito filtrování dat s využitím datových proudů a predikátů v jazyce Java. Druhým přístupem bylo využití databázového nástroje Elasticsearch.

Na základě provedených testů, jež jsou podrobněji popsány v předchozí kapitole, lze konstatovat, že ve všech případech bylo vyhledávání pomocí specializovaného nástroje rychlejší. V případě použití oficiálního klienta pro Elasticsearch nebyl výsledný rozdíl až tak znatelný, protože odeslání dotazu a obdržení výsledku má velkou režii. Z toho důvodu byla použita nenáročná knihovna navržena přímo pro Elasticsearch, jež zapouzdřuje volání REST API.

Vyvíjená aplikace byla navržena tak, aby vždy vrátila všechny nalezené výsledky, to má za následek, že při větším množství dat (tisíce až desetitisíce záznamů v každé tabulce) by byl uživatel takové aplikace nucen čekat přibližně 3 vteřiny na navrácení výsledků, v případě, kdy vyhledává přes více tabulek. Taková prodleva by v dnešní době nebyla akceptovatelná, protože uživatelé požadují co nejrychlejší odezvu veškerých aplikací. Jsou k dispozici tedy další způsoby, jak by se časová prodleva dala minimalizovat. Při využití datových proudů, které provádějí filtrování nad celou sadou dat a pokaždé jí musí nejdříve načíst z databáze do paměti by bylo možné provést načtení například pouze při startu aplikace a následně by se provádělo pouze již filtrování, které samo o sobě trvá maximálně vteřinu. Pokud by měl uživatel aplikaci spuštěnou delší dobu bylo by nutné v určitých časových intervalech provést znovu načtení všech dat, aby byla zajištěna jejich aktuálnost. Pro urychlení vyhledávání přes nástroj Elasticsearch by bylo vhodné použít takzvané „scroll API“, což je vlastně stránkování výsledků. Případně lze stránkovat pomocí specifikace atributů „from“ a „size“ v dotazu, jejichž význam byl dříve vysvětlen.

Při vývoji jakékoliv aplikace, je dobré již na samotném začátku provést analýzu s jakým množství dat se bude potencionálně pracovat a dle toho vybrat správný nástroj pro vyhledávání. Přístup vyhledávání pomocí datových proudů by se dal označit za jednodušší na implementaci a aplikace by se tak dala vyvinout v kratším časovém úseku. S využitím specializovaného nástroje je spojena instalace a konfigurace dalších nástrojů. V případě, kdy vývojář s žádnou podobnou technologií nepracoval bude vývoj aplikace prodloužen o nutné nabrání znalostí.

POUŽITÁ LITERATURA

[1] A brief history of big data everyone should read. *World Economic Forum* [online]. [cit. 2018-05-09]. Dostupné z: https://www.weforum.org/agenda/2015/02/a-brief-history-of-big-data-everyone-should-read/

[2] GANDOMI, Amir a Murtaza HAIDER. Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management* [online]. **2015**(35), 137-144 [cit. 2018-05-09]. ISSN 0268-4012. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0268401214001066

[3] Total WW Data to Reach 163ZB by 2025. *StorageNewsletter* [online]. 2017 [cit. 2018-05-09]. Dostupné z: https://www.storagenewsletter.com/2017/04/05/total-ww-data-to-reach-163-zettabytes-by-2025-idc/

[4] INFOGRAPHIC: HOW MUCH DATA IS PRODUCED EVERY DAY?. *Cloud Influencer Resource Portal - Connecting The Cloud* [online]. 2015 [cit. 2018-05-09]. Dostupné z: https://cloudtweaks.com/2015/03/how-much-data-is-produced-every-day/

[5] C. J. DATE. *An introduction to database systems*. 8. ed., international ed. Boston, Mass. [u.a.]: Pearson, Addison-Wesley, 2004. ISBN 03-211-8956-6.

[6] SQL vs. NoSQL Databases: What's the Difference?. *Upwork, the world's largest online workplace* [online]. [cit. 2018-05-09]. Dostupné z: https://www.upwork.com/hiring/data/sql-vs-nosql-databases-whats-the-difference/

[7] NoSQL Databases Explained. *MongoDB for GIANT Ideas | MongoDB* [online]. [cit. 2018-05-09]. Dostupné z: https://www.mongodb.com/nosql-explained

[8] Foundation Project. *Welcome to The Apache Software Foundation!* [online]. [cit. 2018-05-09]. Dostupné z: http://www.apache.org/foundation/

[9] Apache Lucene Core. *Apache Lucene - Welcome to Apache Lucene* [online]. [cit. 2018-05-09]. Dostupné z: https://lucene.apache.org/core/

[10] What is Apache Nutch?. *Nutch Wiki* [online]. [cit. 2018-05-09]. Dostupné z: https://wiki.apache.org/nutch/FrontPage#What_is_Apache_Nutch.3F

[11] Nutch Tutorial. *Nutch Wiki* [online]. [cit. 2018-05-09]. Dostupné z: https://wiki.apache.org/nutch/NutchTutorial

[12] Apache Solr - Features. *Apache Solr* [online]. [cit. 2018-05-09]. Dostupné z: http://lucene.apache.org/solr/features.html#top

[13] Apache Solr. *Data Management Platform, Solutions and Big Data Analysis | Hortonworks* [online]. [cit. 2018-05-09]. Dostupné z: https://hortonworks.com/apache/solr/

[14] CrateDB: The Open Source SQL Database for Machine Data | Crate.io. *CrateDB: Real-time SQL Database for Machine Data & IoT | Crate.io* [online]. [cit. 2018-05-09]. Dostupné z: https://crate.io/products/cratedb/

[15] Elasticsearch: RESTful, Distributed Search & Analytics | Elastic. *Open Source Search & Analytics · Elasticsearch| Elastic* [online]. [cit. 2018-05-09]. Dostupné z: https://www.elastic.co/products/elasticsearch

[16] What is Docker?. *Homepage | Opensource.com* [online]. [cit. 2018-05-09]. Dostupné z: https://opensource.com/resources/what-docker

[17] What is a Container. *Docker - Build, Ship, and Run Any App, Anywhere* [online]. [cit. 2018-05-09]. Dostupné z: https://www.docker.com/what-container

[18] Overview of Docker Compose. *Docker Documentation | Docker Documentation* [online]. [cit. 2018-05-09]. Dostupné z: https://docs.docker.com/compose/overview/

[19] Chapter 1 General Information. *MySQL :: MySQL Documentation* [online]. [cit. 2018-05-09]. Dostupné z: https://dev.mysql.com/doc/refman/5.7/en/introduction.html

[20] ROGOZINSKI, Marek, Bharvi DIXIT, Rafal KUC a Saurabh CHHAJED. *Elasticsearch: A Complete Guide*. Birmingham B3 2PB, UK: Packt Publishing, 2017. ISBN 978-1-78728-854-6.

[21] KUĆ, Rafał a Marek ROGOZIŃSKI. *Elasticsearch server.* S.l.: Packt Publishing Limited, 2013. ISBN 1849518440.

[22] Introduction | Kibana User Guide [6.2]. *Open Source Search & Analytics · Elasticsearch | Elastic* [online]. [cit. 2018-05-09]. Dostupné z: https://www.elastic.co/guide/en/kibana/current/introduction.html

[23] What is Kibana?. *Amazon Web Services (AWS) - Cloud Computing Services* [online]. [cit. 2018-05-09]. Dostupné z: https://aws.amazon.com/elasticsearch-service/kibana/

[24] Logstash Introduction. *Open Source Search & Analytics · Elasticsearch | Elastic* [online]. [cit. 2018-05-09]. Dostupné z: https://www.elastic.co/guide/en/logstash/current/introduction.html

[25] Hibernate ORM. *Hibernate. Everything data. - Hibernate* [online]. [cit. 2018-05-09]. Dostupné z: http://hibernate.org/orm/

[26] Maven – Introduction. *Welcome to The Apache Software Foundation!* [online]. [cit. 2018-05-09]. Dostupné z: https://maven.apache.org/what-is-maven.html

[27] Java 8 - Streams. *Tutorials Point* [online]. [cit. 2018-05-09]. Dostupné z: https://www.tutorialspoint.com/java8/java8_streams.htm

[28] WARBURTON, Richard. *Java 8 lambdas*. Beijing: O'Reilly, 2014. ISBN 14-493-7077-2.

[29] Java 8 Predicate with Examples. *GeeksforGeeks | A computer science portal for geeks* [online]. [cit. 2018-05-09]. Dostupné z: https://www.geeksforgeeks.org/java-8-predicate-with-examples/

[30] Predicate (Java Platform SE 8). *Overview (Java Platform SE 8)* [online]. [cit. 2018-05-09]. Dostupné z: https://docs.oracle.com/javase/8/docs/api/java/util/function/Predicate.html

[31] Java 8 java.util.function.Predicate tutorial with examples. *JavaBrahman* [online]. [cit. 2018-05-09]. Dostupné z: https://www.javabrahman.com/java-8/java-8-java-util-function-predicate-tutorial-with-examples/

[32] Overview. *Open Source Search & Analytics · Elasticsearch | Elastic* [online]. [cit. 2018-05-09]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/client/java-rest/current/java-rest-overview.html

[33] Compatibility. *Open Source Search & Analytics · Elasticsearch | Elastic* [online]. [cit. 2018-05-09]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/client/java-rest/6.x/java-rest-high-compatibility.html

[34] GitHub - mgarin/weblaf: WebLaF - Java Look and Feel library for cross-platform Swing applications. *GitHub* [online]. [cit. 2018-05-09]. Dostupné z: https://github.com/mgarin/weblaf

[35] Jest/jest at master · searchbox-io/Jest · GitHub. *GitHub* [online]. [cit. 2018-05-09]. Dostupné z: https://github.com/searchbox-io/Jest/tree/master/jest

[36] Java Clients for Elasticsearch. *Open Source Search & Analytics · Elasticsearch | Elastic* [online]. [cit. 2018-05-09]. Dostupné z: https://www.elastic.co/blog/found-java-clients-for-elasticsearch

[37] Overview. *Open Source Search & Analytics · Elasticsearch | Elastic* [online]. [cit. 2018-05-09]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/client/java-rest/current/java-rest-overview.html

PŘÍLOHY

PŘÍLOHA A – PIPELINE PRO INDEX AUTOPSY

```
input {
  jdbc {
    jdbc_driver_library => "/usr/share/logstash/mysql-connector-java-8.0.8-dmr.jar"
    jdbc_driver_class => "com.mysql.jdbc.Driver"
    jdbc_connection_string => "jdbc:mysql://mysql:3306/mycold94"
    jdbc_user => "root"
    jdbc_password => "admin"
    # our query
    statement => "SELECT * from autopsy"
  }
}
output {
  # stdout { codec => json_lines }
  elasticsearch {
    "hosts" => "elasticsearch:9200"
    "index" => "autopsy"
    "document_id" => "%{[id_patinet]}"
  }
}
```

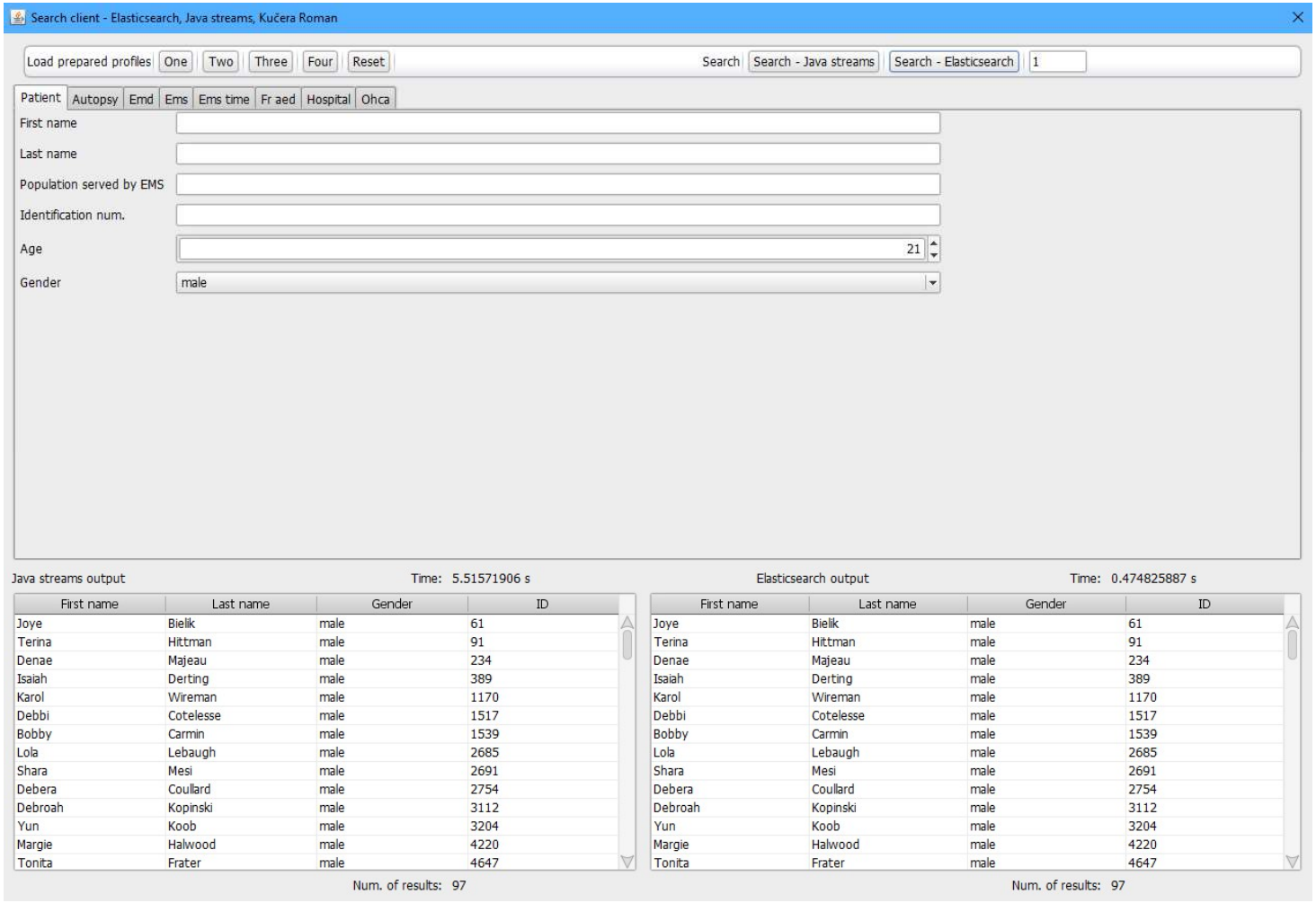
PŘÍLOHA B – KONFIGURAČNÍ SOUBOR POM.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apa-
che.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>cz.kucerea.benchmark.client</groupId>
    <artifactId>BenchmarkClient</artifactId>
    <version>1.0-SNAPSHOT</version>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>

    <dependencies>
        <dependency>
            <groupId>org.elasticsearch.client</groupId>
            <artifactId>elasticsearch-rest-high-level-client</artifactId>
            <version>6.0.0</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/de.sciss/weblaf -->
        <dependency>
            <groupId>de.sciss</groupId>
            <artifactId>weblaf</artifactId>
            <version>2.1.3</version>
            <type>pom</type>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>de.sciss</groupId>
            <artifactId>weblaf-ui</artifactId>
            <version>RELEASE</version>
        </dependency>

        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>5.2.12.Final</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.8-dmr</version>
        </dependency>
        <dependency>
            <groupId>io.searchbox</groupId>
            <artifactId>jest</artifactId>
            <version>5.3.3</version>
        </dependency>
    </dependencies>
</project>
```

PŘÍLOHA C – ZDROJOVÝ KÓD MODELU PRO TABULKU S VÝSLEDKY HLEDÁNÍ

```
public class ResultTableModel extends AbstractTableModel {

    private List<SearchDTO> results = new ArrayList();
    private String[] columnNames = "First name", "Last name", "Gender", "ID"
 };

    public ResultTableModel(List<SearchDTO> list){
        this.results = list;
    }

    @Override
    public String getColumnName(int columnIndex){
        return columnNames[columnIndex];
    }

    public int getRowCount() {
        return results.size();
    }

    public int getColumnCount() {
        return 4;
    }

    public Object getValueAt(int rowIndex, int columnIndex) {
        SearchDTO result = results.get(rowIndex);
        switch (columnIndex) {
            case 0:
                return result.getFirst_name();
            case 1:
                return result.getLast_name();
            case 2:
                return result.getRegion();
            case 3:
                return result.getCity();
        }
        return null;
    }

    @Override
    public Class<?> getColumnClass(int columnIndex){
        switch (columnIndex){
            case 0:
                return String.class;
            case 1:
                return String.class;
            case 2:
                return String.class;
            case 3:
                return String.class;
        }
        return null;
    }
}
```

PŘÍLOHA D – UŽIVATELSKÉ PROSTŘEDÍ



PŘÍLOHA E – ZDROJOVÝ KÓD METODY PRO SESTAVENÍ PREDIKÁTU

```
private Predicate preparePredicate(Map<String, String> items, Predicate predicate,
IGeneralPredicate predicatesHolder) {
   for (Map.Entry entry : items.entrySet()) {
      String key = (String) entry.getKey();
      String value = "";
      value = getValue(entry, value);
      if (!value.equals("")) {
         if (predicate == null) {
            predicate = predicatesHolder.find(key, value);
         } else {
            predicate = predicate.and(predicatesHolder.find(key, value));
         }
      }
   }
   return predicate;
}
```

PŘÍLOHA F – ZDROJOVÝ KÓD METODY PRO FILTROVÁNÍ DAT DLE PREDIKÁTU

```
private List<String> filterData(Predicate predicate, List results) {
   List<String> ids = new LinkedList<>();
   if (predicate != null) {
      results.stream().filter(predicate)
            .forEach(object -> {
               String id = "";
               if (object instanceof Entity) {
                  id = String.valueOf(((Entity) object).getEntityID());
               }
               ids.add(id);
            });
   }
   return ids;
}
```

PŘÍLOHA G – ZDROJOVÝ KÓD METODY PRO SESTAVENÍ DOTAZU PRO ELASTICSEARCH

```
private String appendQuery(String query, Map<String, String> items) {
   for (Map.Entry entry : items.entrySet()) {
      String key = (String) entry.getKey();
      String value = "";
      value = getValue(entry, value);
      if (!value.equals("")) {
         if (!query.isEmpty()) {
            query += " AND " + getElasticAttributeName(key) + ": " + value.trim();
         } else {
            query = getElasticAttributeName(key) + ": " + value.trim();
         }
      }
   }
   return query;
}
```